University of Zurich[UZH]

Department of Informatics

s.e.a.l. software evolution & architecture lab

Software Maintenance and Evolution

The Software Change Process: How to deal with a Change Request

Carmine Vassallo
Department of Informatics
University of Zurich

vassallo@ifi.uzh.ch

@ccvassallo

Who am I?

- PhD Student since September 2016
- My current research focus on:
  - Supporting **DevOps** while dealing with **CD** breaks (e.g., build failures)
  - Developing new approaches to prevent stop in the **CD pipeline**
- Interested in the topic?

http://www.ifi.uzh.ch/en/seal/people/vassallo.html

What is Continuous Delivery?

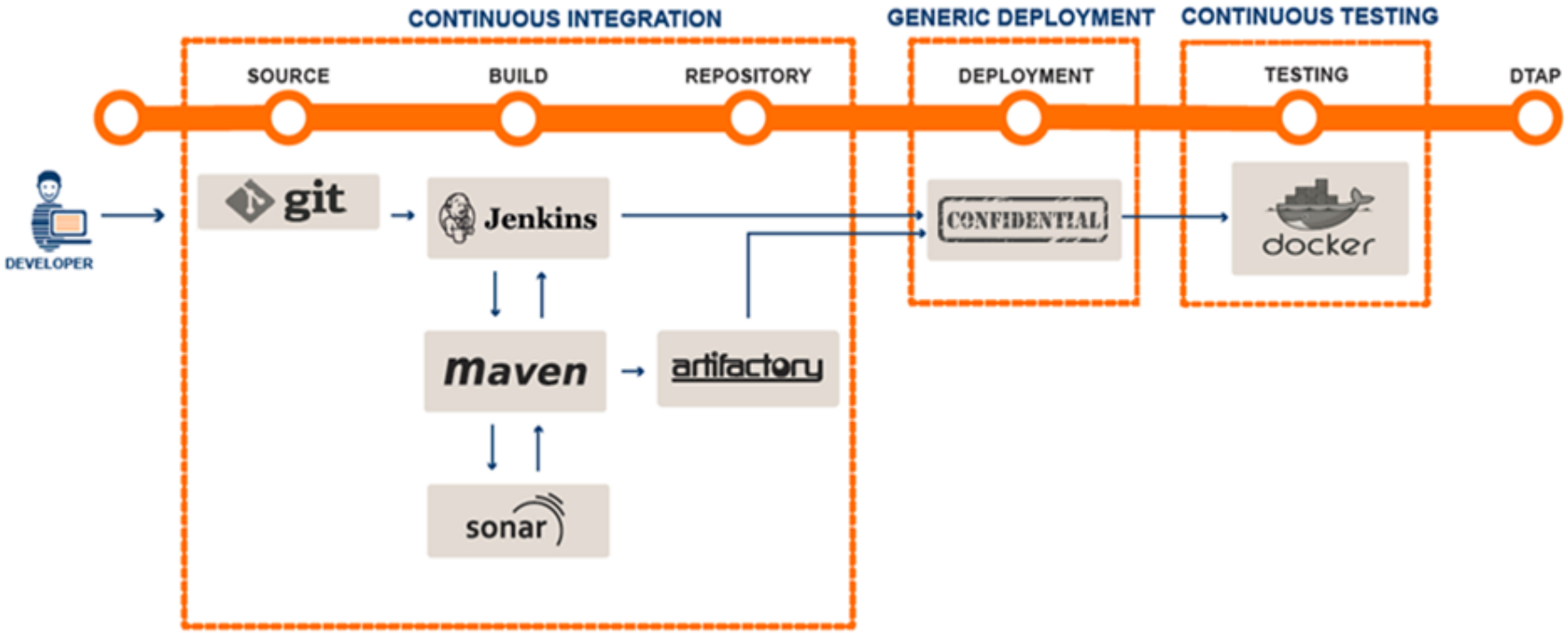
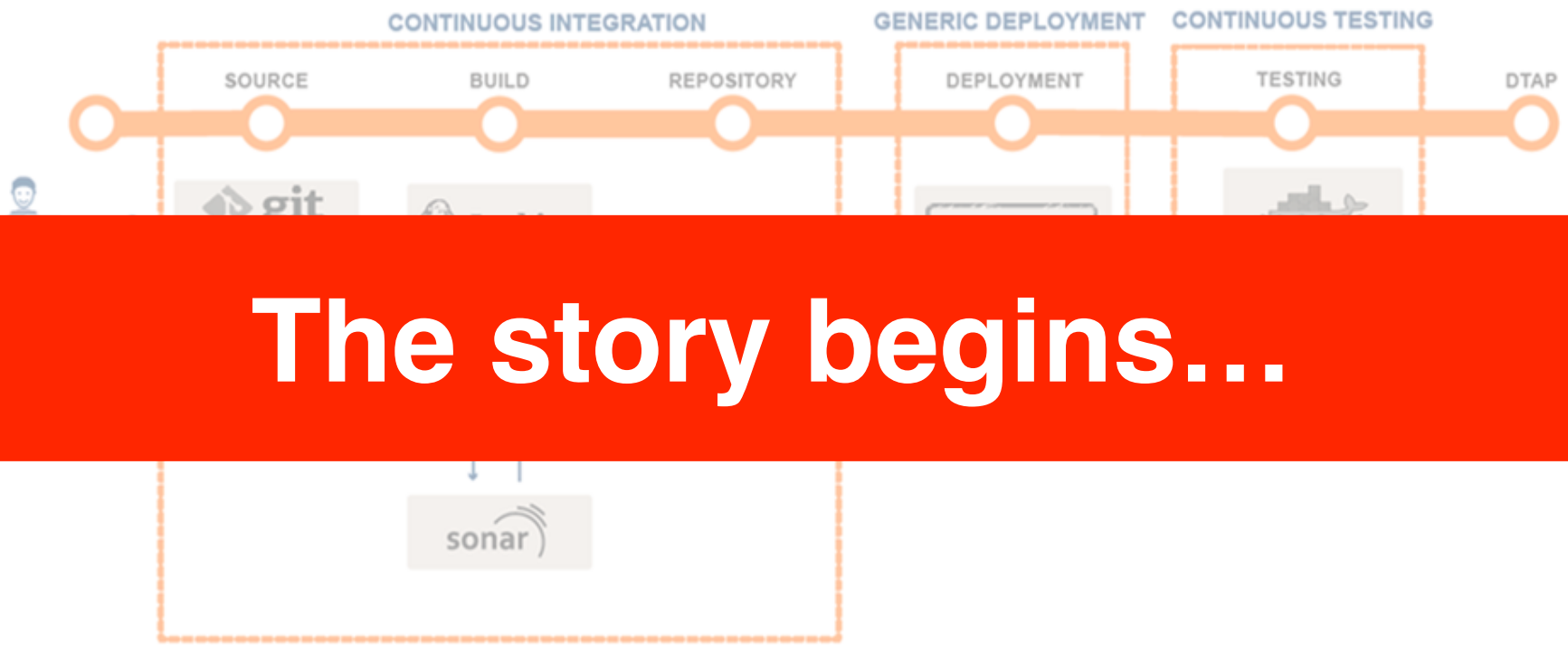


Image from Carmine Vassallo *et al.*, *Continuous Delivery Practice in a Large Financial Organisation*, ICSME 2016

What is Continuous Delivery?



The story begins…

Image from Carmine Vassallo *et al.*, *Continuous Delivery Practice in a Large Financial Organisation*, ICSME 2016

… when the client ask for something

- A change is requested by stakeholders

Change request

- The product owner translates <u>change requests</u> in <u>software changes</u> and put it in the backlog


BACKLOG
TO DO
DOING
DONE
L
M
H
Wen
Sarah
Alison
Skay
Craig
monthly leadership meeting notes
make stickys for next sprint
Green Card
Pick up t-shirts
Finance meeting
Party Planning

Software Change

- Software change is the process of adding new functionality to existing code

- Software change types (Lientz and Swanson):
  - Perfective (~66%)
  - Adaptive
  - Corrective
  - Protective

Part 1

The Software Change process

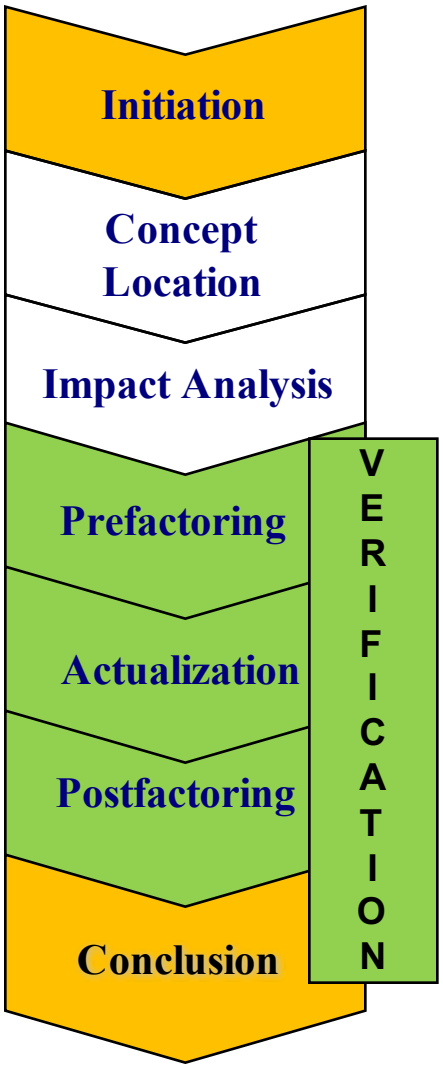
Software Change process

- Phases of Software Change process (SCP)

Interaction with the world

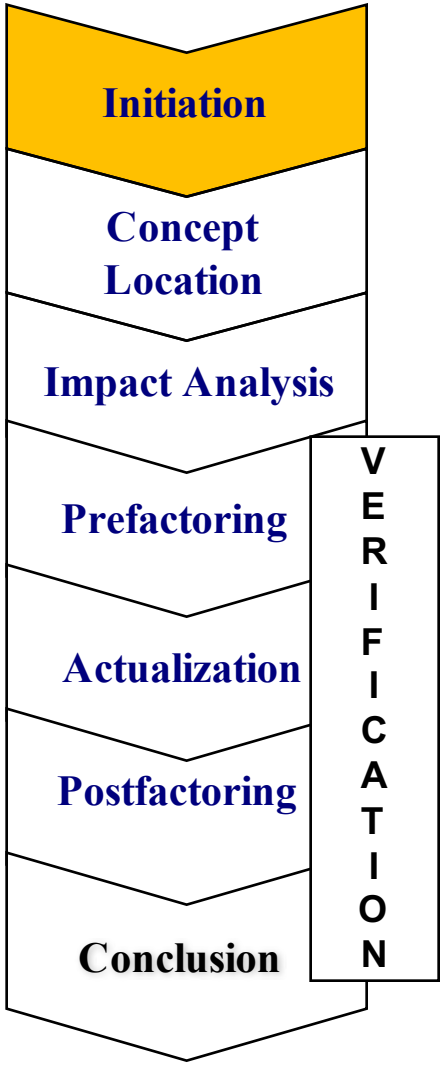
Software Change Design

Software Change Implementation



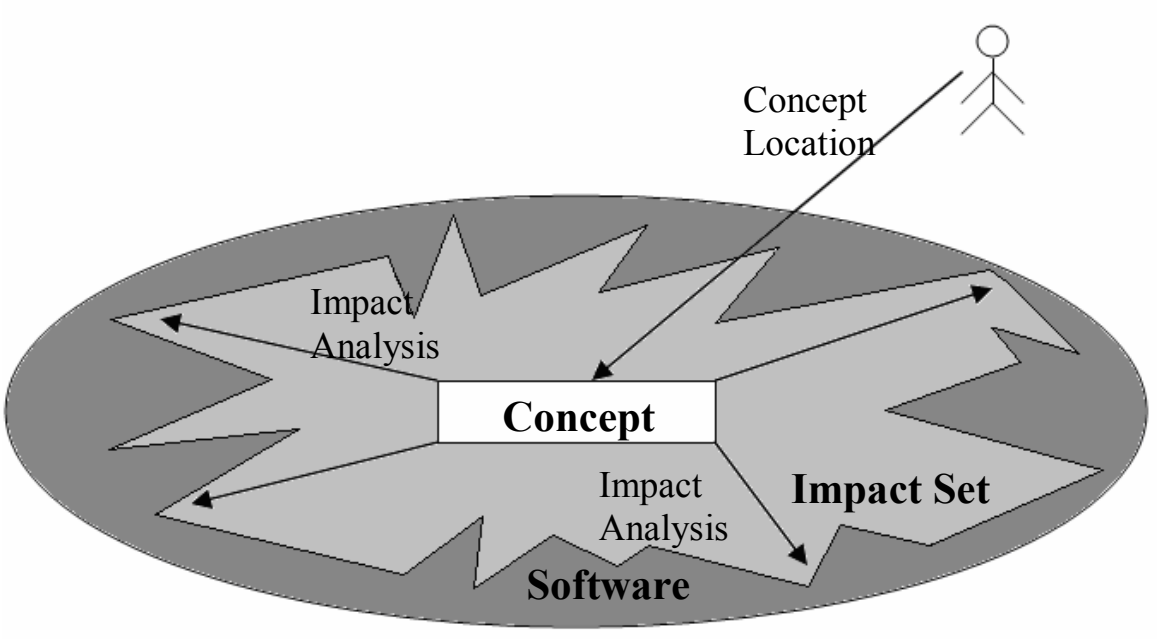
Initiation

- The product owner receives a change request from the stakeholders
  - From change request to software changes
    - Stakeholder to Product Owner: *"I want to be able to print a receipt of a payment"*
    - Product Owner to Developer: *"Add a button Print in the confirmation page of a payment"*, *"The button Print has to print a receipt of the payment when is pressed"*
  - Prioritisation of software changes in the backlog



Software Change Design

- The Software Change Design includes the Concept Location and Impact Analysis phases



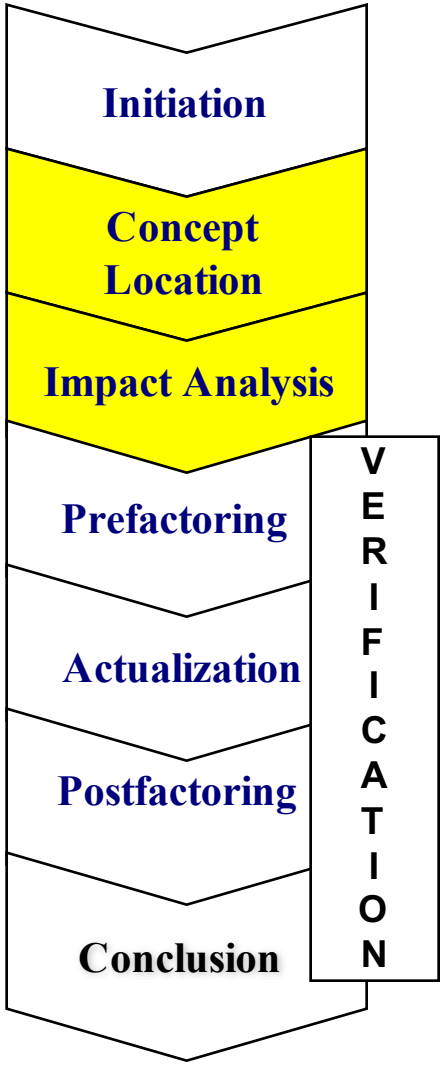
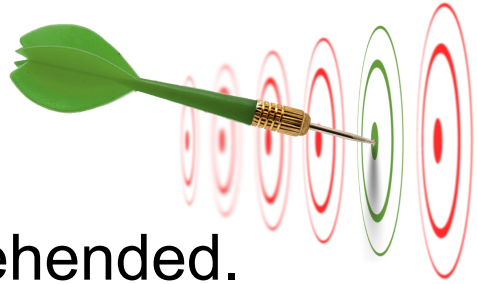


Image from Václav Rajlich, *Software Engineering: The Current Practice*, CRC Press, Boca Raton, FL (2012)

Concept Location



- Large software can't be completely comprehended.
- Concept Location finds code snippet where a change has to be made
- Software changes are often formulated in terms of <u>domain concepts</u>
  - SC1: "Correct error that arises when trying to paste a text"
  - The developer must find in the code the locations where concept "paste" is located to implement SC1

Concept Location: Formulate a Query

- **Extract** the set of concepts used in the software change
- **Delete** the concepts intended for the communication with the developers
- **Delete** the concepts that are unlikely to be implemented in the code
  - concepts related to the things that are outside of the scope of the program
  - concepts that are to be implemented in the future
- **Rank** the remaining concepts by the likelihood that they can be easily located

Concept Location: Formulate a Query

- Example: Our team is currently working at a Point of Sale system
- SC2: “Implement a credit card payment”
- Extract the concepts
  - “Implement”
  - “Credit Card”
  - “Payment”

Concept Location: Formulate a Query

- Example: Our team is currently working at a Point of Sale system
- SC2: “Implement a credit card payment”
- Extract the concepts
  - “Implement”

    communication with developer
  - “Credit Card”
  - “Payment”

Concept Location: Formulate a Query

- Example: Our team is currently working at a Point of Sale system
- SC2: "Implement a credit card payment"
- Extract the concepts
  - "Implement" — communication with developer
  - "Credit Card" — to be implemented…not in the old code
  - "Payment"

Concept Location: Formulate a Query

- Example: Our team is currently working at a Point of Sale system
- SC2: "Implement a credit card payment"
- Extract the concepts
  - "Implement" — communication with developer
  - "Credit Card" — to be implemented…not in the old code
  - "Payment" — Significant concept !!! Find it in the code…

Concept Location techniques

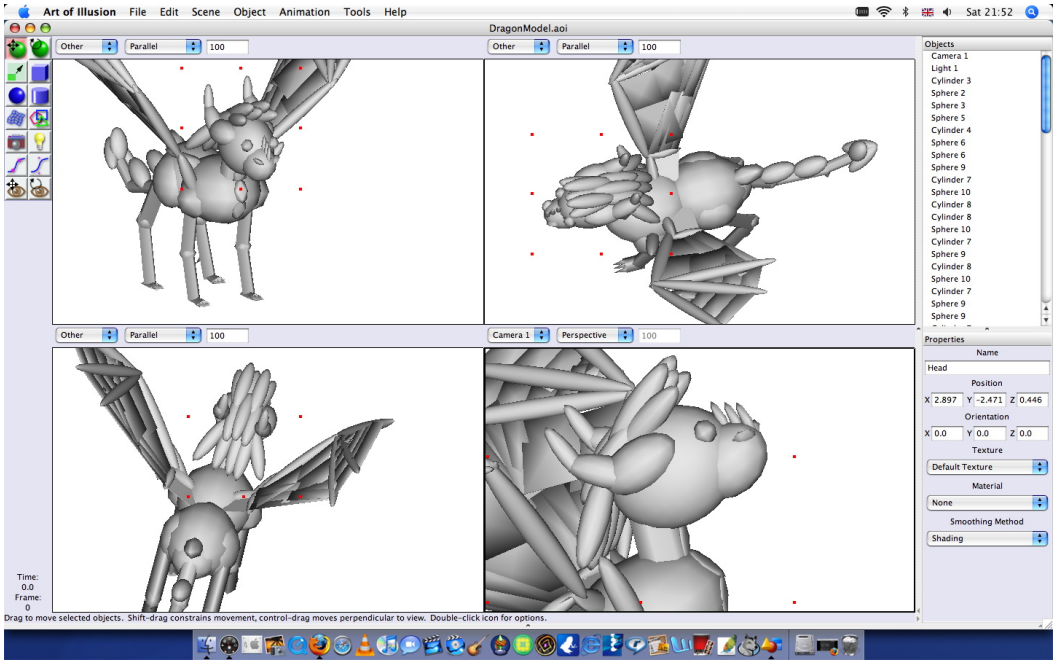
- Once the relevant concepts have been chosen, they have to be located in the source code
- Concept Location techniques:
  - Human knowledge
  - Traceability tools
  - Static search
    - <u>GREP Analysis</u>
    - <u>Dependencies Search</u>
    - Information Retrieval techniques (*already discussed in the previous lecture*)

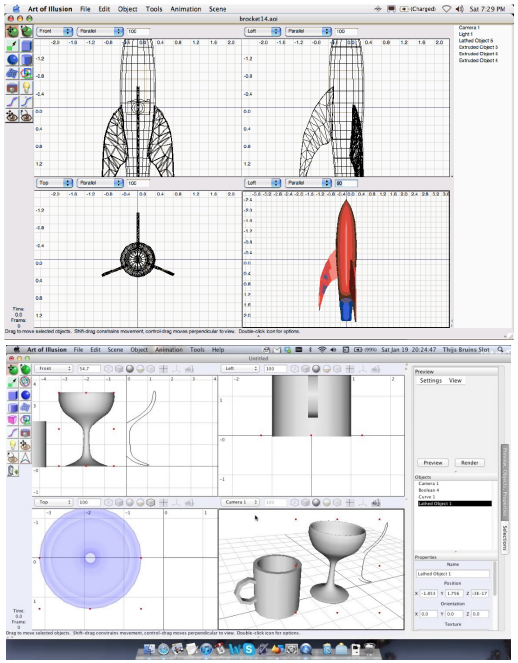
GREP Analysis

- GREP is an acronym for “global regular expression print”
  - GREP prints out the lines that contain a match for a regular expression
  - Developer iteratively formulates search query (based on extracted concepts) and then investigates the results
  - If the results are too big to review, developer either performs further search within these results or reformulates the search query

GREP Analysis: Example

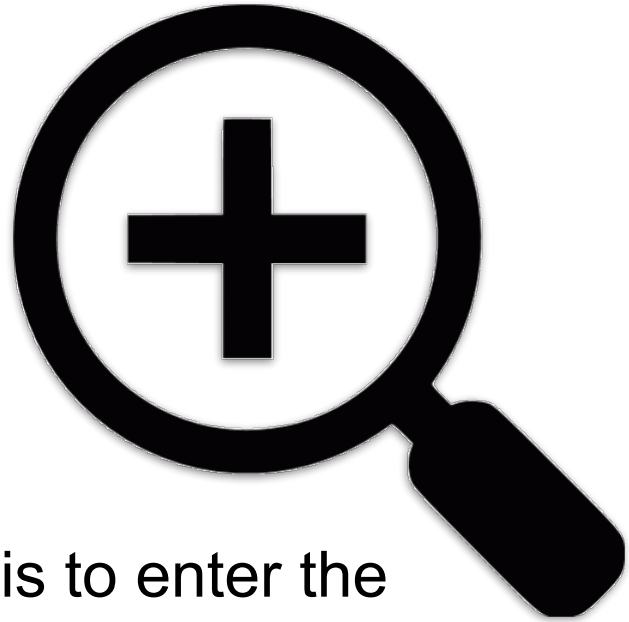
- “Art of Illusion” is a free, open source 3D modelling and rendering studio
- It’s written in Java and contains more than 600 classes





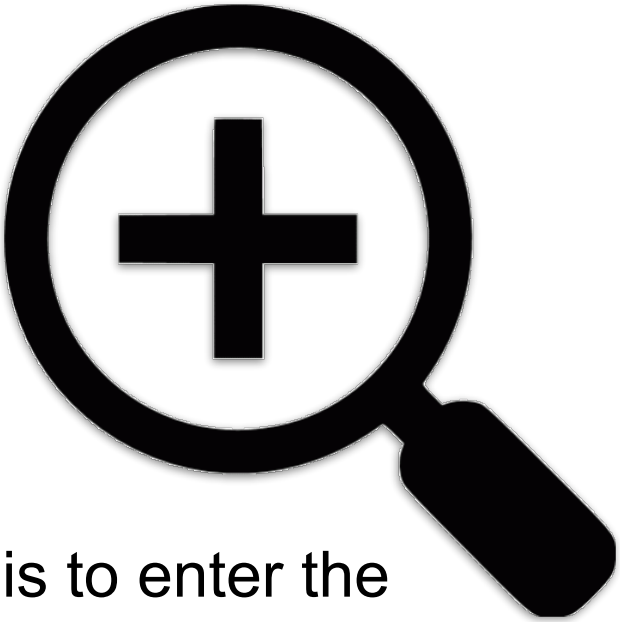
http://www.artofillusion.org

GREP Analysis: Example



- SC3: “Currently the only way to scale is to enter the zooming value into the specific text box (the default value is 100%). Implement zooming control that uses arrow keys”

GREP Analysis: Example



- SC3: “Currently the only way to **scale** is to enter the **zoom**ing value into the specific text box (the default value is **100**%). Implement zooming control that uses arrow keys”

GREP Analysis: Example

- First search: “zoom”
  - The query produced irrelevant 6 lines
- Second search: “scale”
  - Returned in 1,544 lines, too large for inspection
- Third search: “100”
  - default scaling value is 100
  - search the results of the previous search
  - returned 4 lines from the “ViewerCanvas.java” file
- Inspection
  - one of the lines is the location

Dependencies Search

- It’s based on the Class Dependency Graph (CDG)
  - It’s easy to extract from the existing code (using tool as Architexa)
- Navigate the graph searching for the concepts
  - Local Responsibility
  - Composite Responsibility

Local and Composite Responsibility

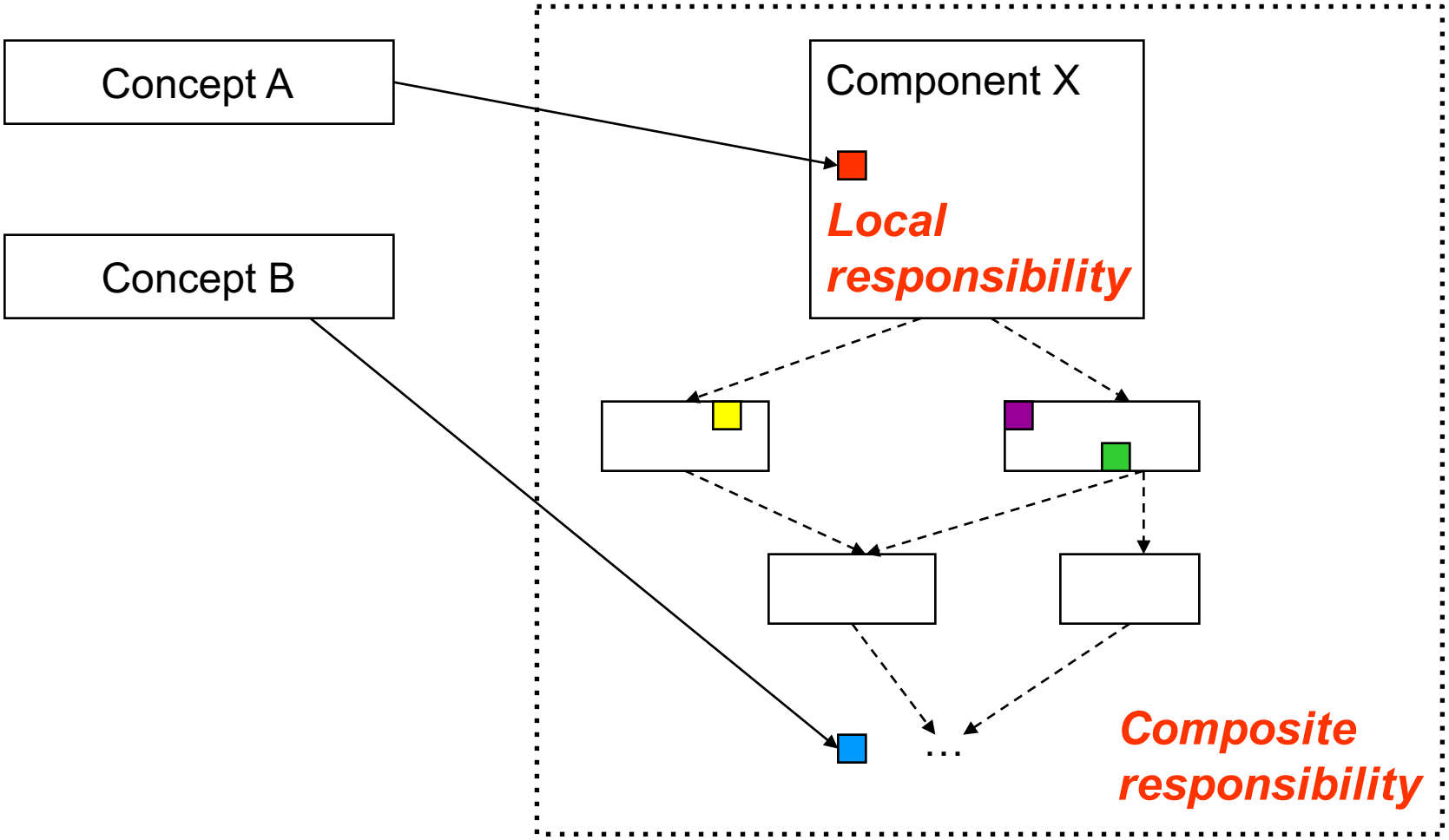


Image from Václav Rajlich, *Software Engineering: The Current Practice*, CRC Press, Boca Raton, FL (2012)

Dependency Search Activity Diagram

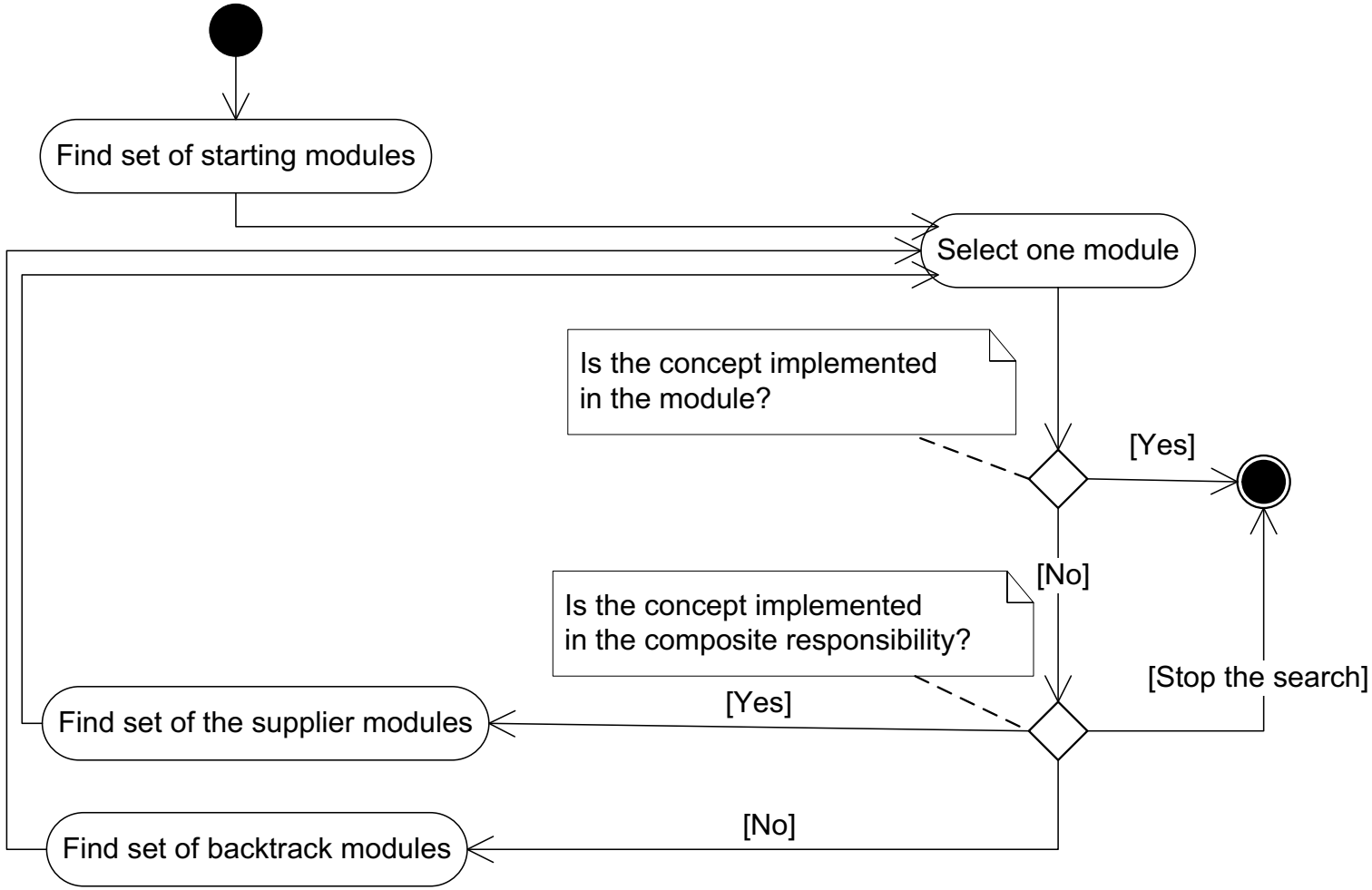
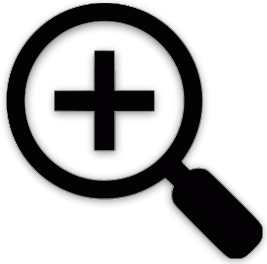


Image from Václav Rajlich, *Software Engineering: The Current Practice*, CRC Press, Boca Raton, FL (2012)

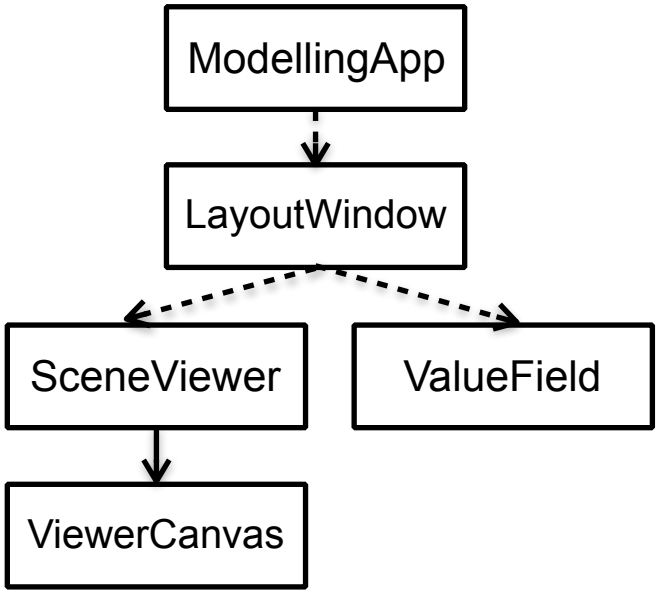
Dependencies Search: Example



- Resume the SC3 related to the system “Art of Illusion”
- SC3: “Currently the only way to **scale** is to enter the **zoom**ing value into the specific text box (the default value is **100**%). Implement zooming control that uses arrow keys”
- First Step: Build the Class Dependencies Graph

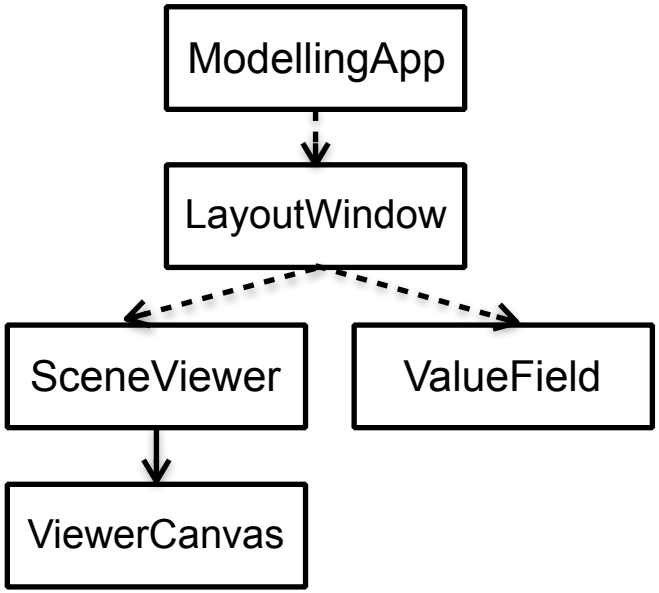
Dependencies Search: Example

- Start at the ModelingApp class
- concept not contained within its local responsibility
- The next step: inspect LayoutWindow
- responsible for constructing the main AOI window
- composite responsibility contains the concept, but the local responsibility does not
- There were clues to search ValueField
- it implements the text box
- concept is not present in the local responsibility
- backtrack to the LayoutWindow class



Dependencies Search: Example

- SceneViewer class
- several functions are responsible for responding to events from the user
- function updateImage() was responsible for repainting the screen
- we determined that the composite responsibility of this function contained the concept
- local responsibility of SceneViewer still did not contain the concept
- ViewerCanvas class
- contains the concept

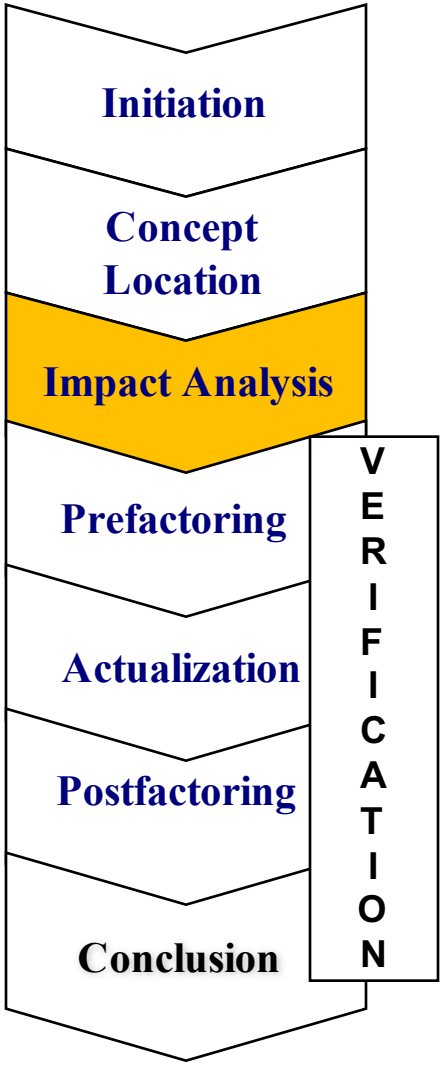


Concept Location: Comparison

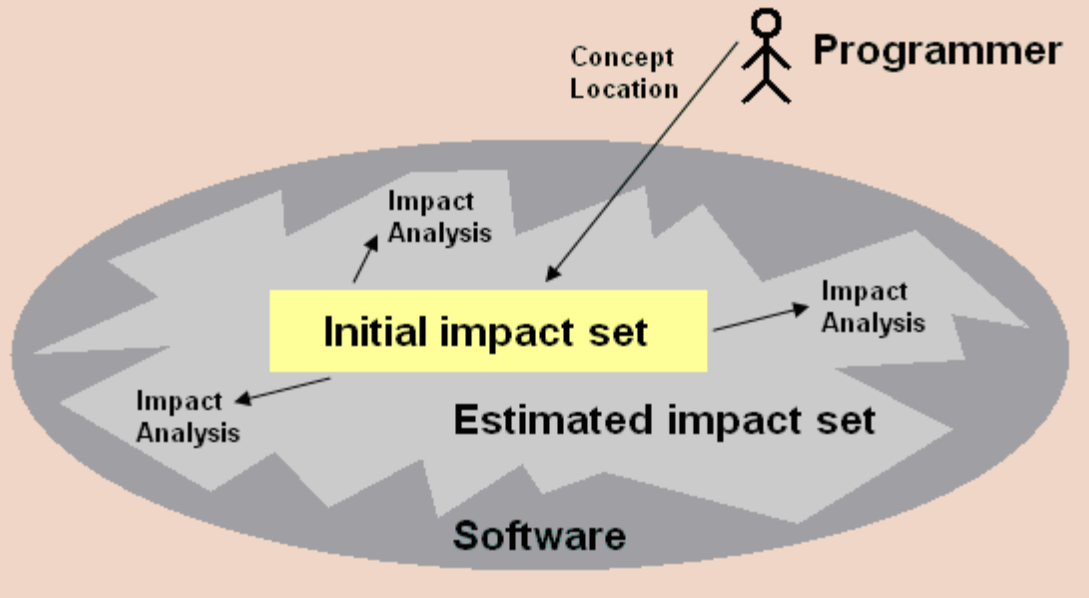
- GREP Analysis
  - depends on the use of naming conventions
  - independent of class structure
  - suitable for explicit concepts only
- Dependencies Search
  - utilizes the class structure
  - needs correct understanding of composite and local functionality
  - suitable for both explicit and implicit concepts

Impact Analysis

- Determines the strategy and impact of change
- Classes identified in concept location are the initial impact set
- Class dependencies are analyzed, and impacted classes are added to the impact set
- Produces estimated impact set



Impact Analysis



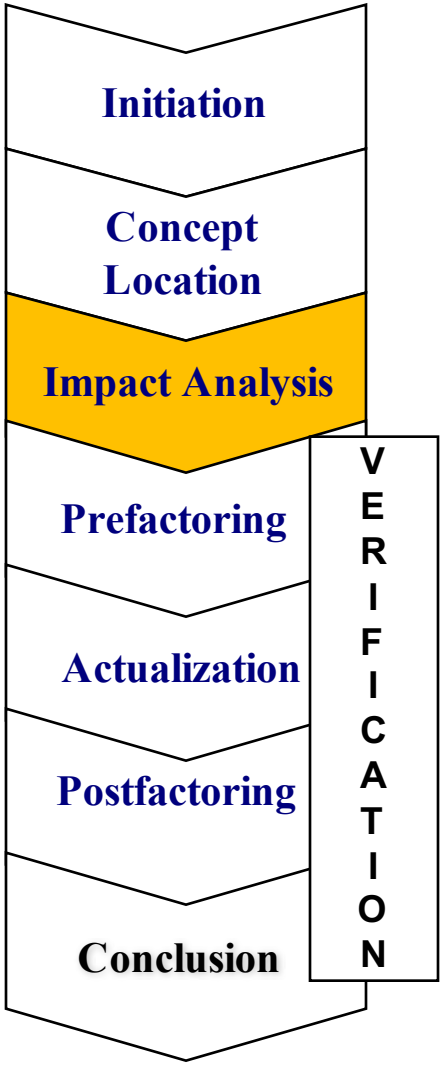


Image from Václav Rajlich, *Software Engineering: The Current Practice*, CRC Press, Boca Raton, FL (2012)

Class Interactions

- Two classes interact if the have something in common
  - One depends on the other
    - There is a contract between them
  - They coordinate
    - They share the same coding, schedule, etc.

- Interactions propagate change
  - In both directions

- From A to B or from B to A

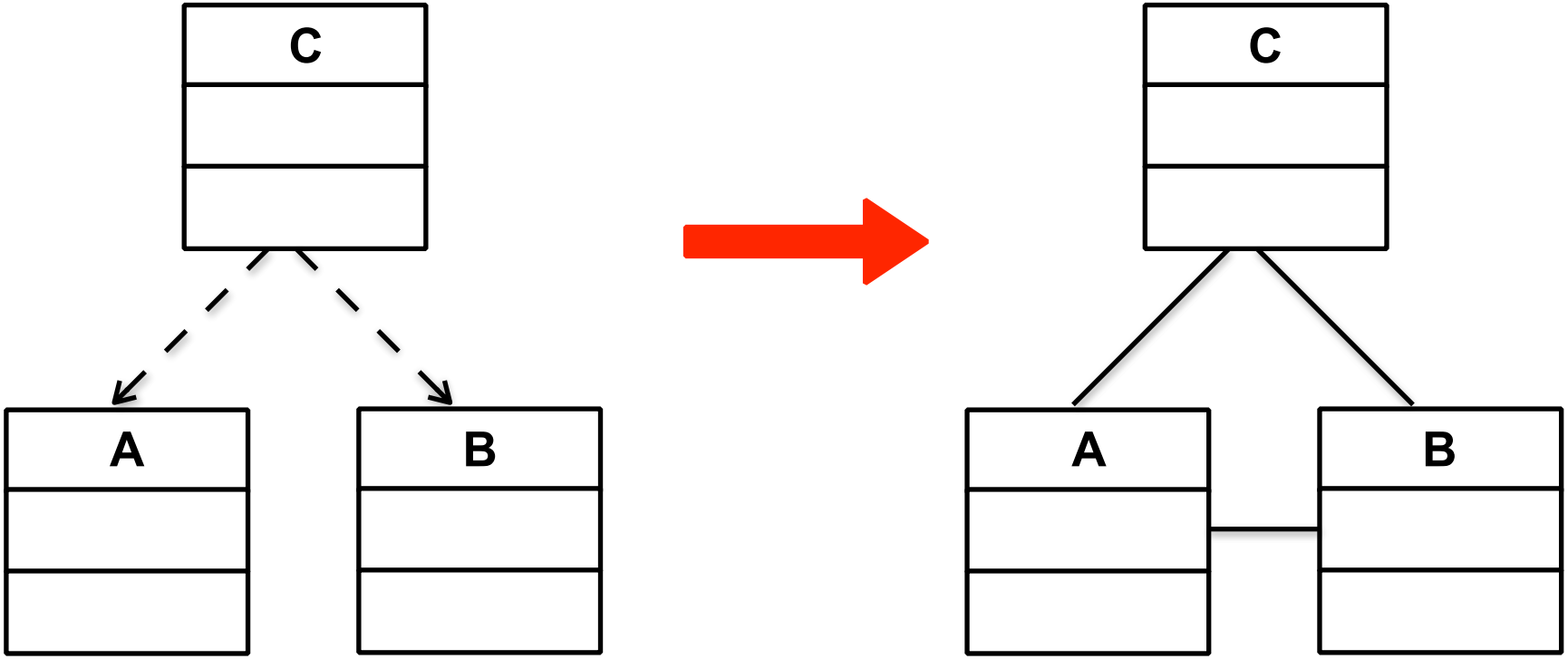
Class Interactions: Example

```
class C
{
    A   a;     //gets the color code
    B   b;         //paints the screen

    void foo()
      {
        b.paint(a.get()); // dataflow between a and b
      }

};
```

From Dependencies to Interactions



Dependency Diagram

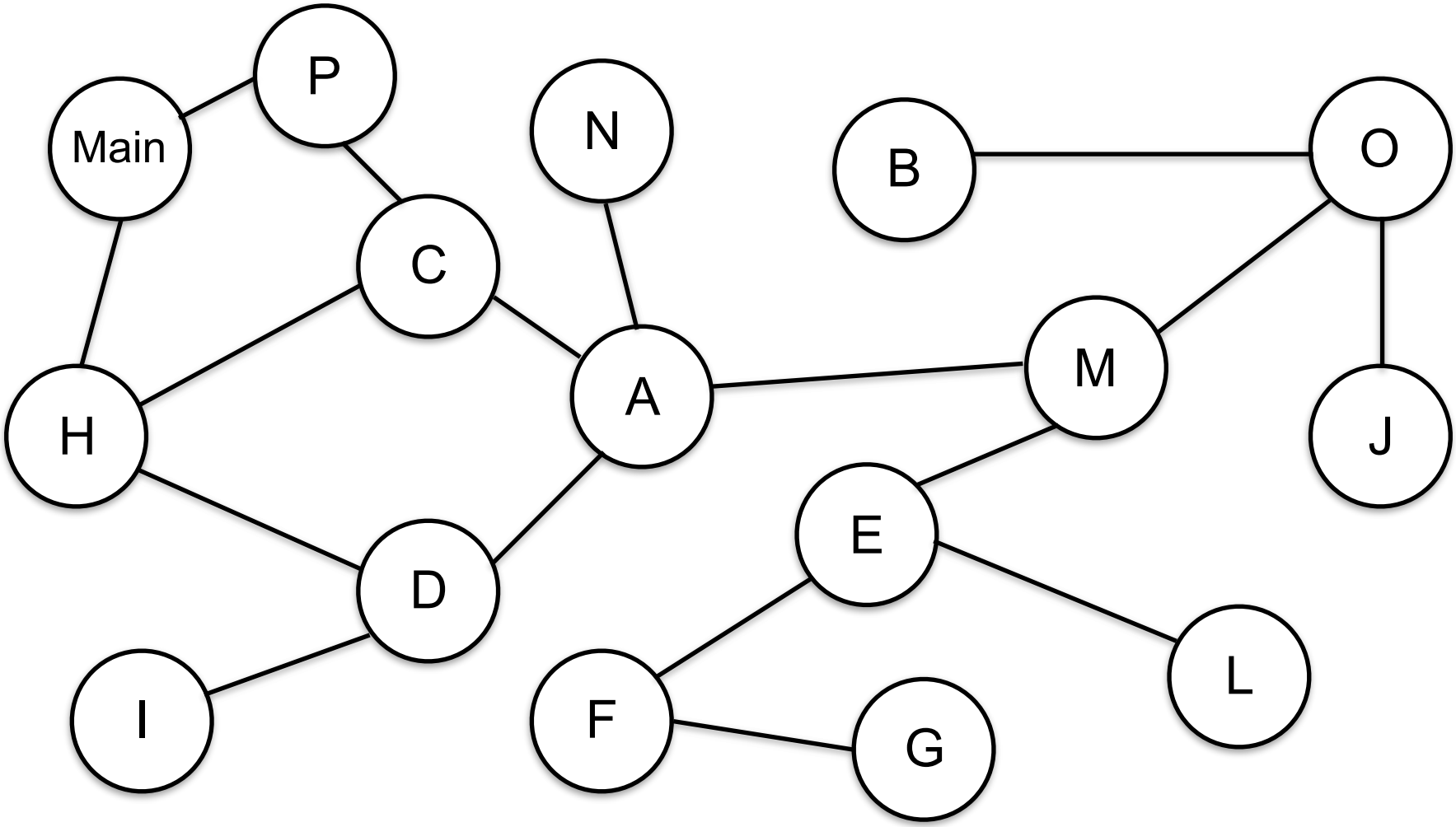
Interaction Diagram

Class Interaction Graph

- $G = (X,I)$
  - X … set of classes
  - I … set of interactions

- Neighborhood of class A
  - $N(A) = \{B \mid (A,B) \in I\}$

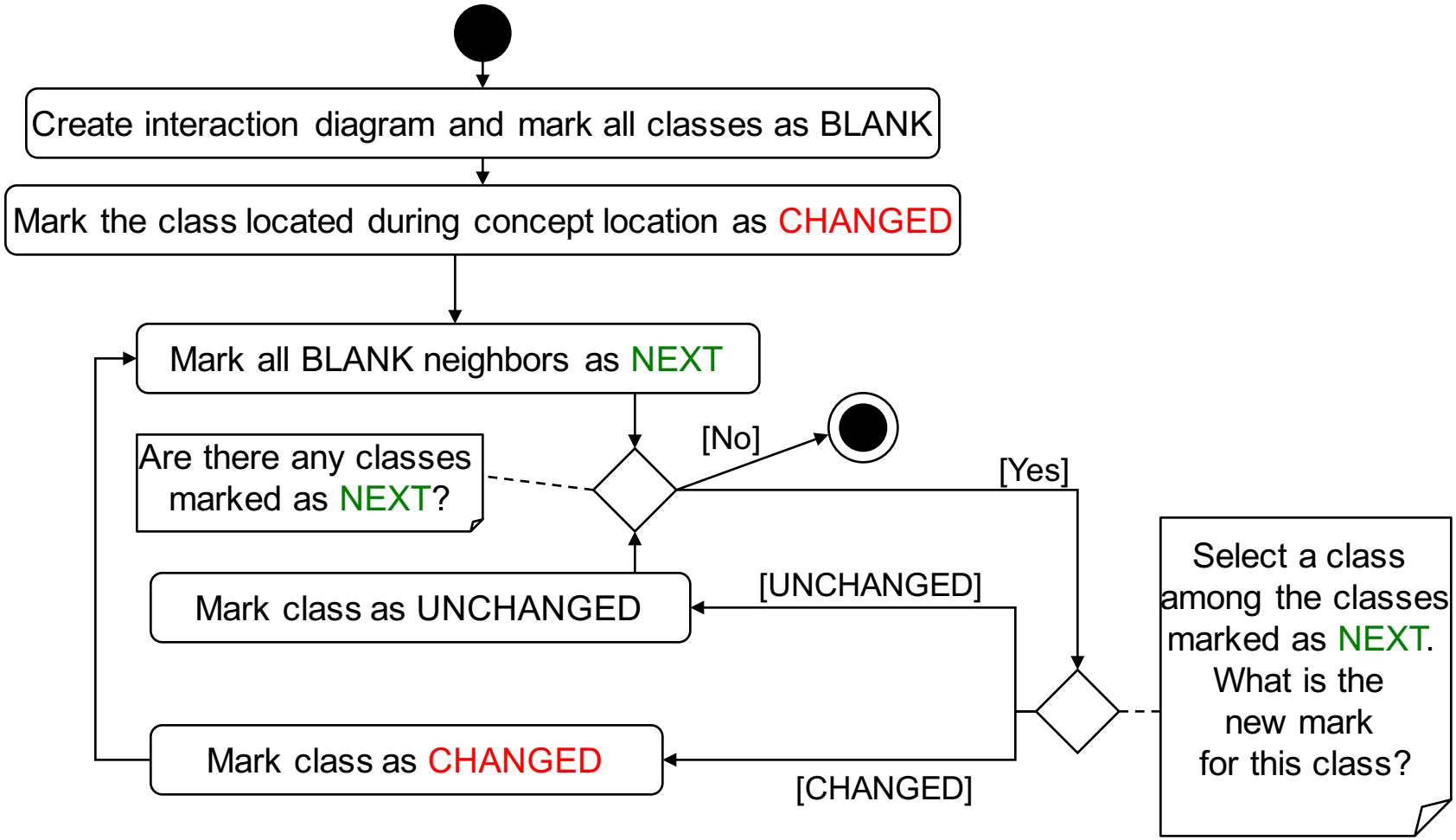
Class Interaction Graph: Example



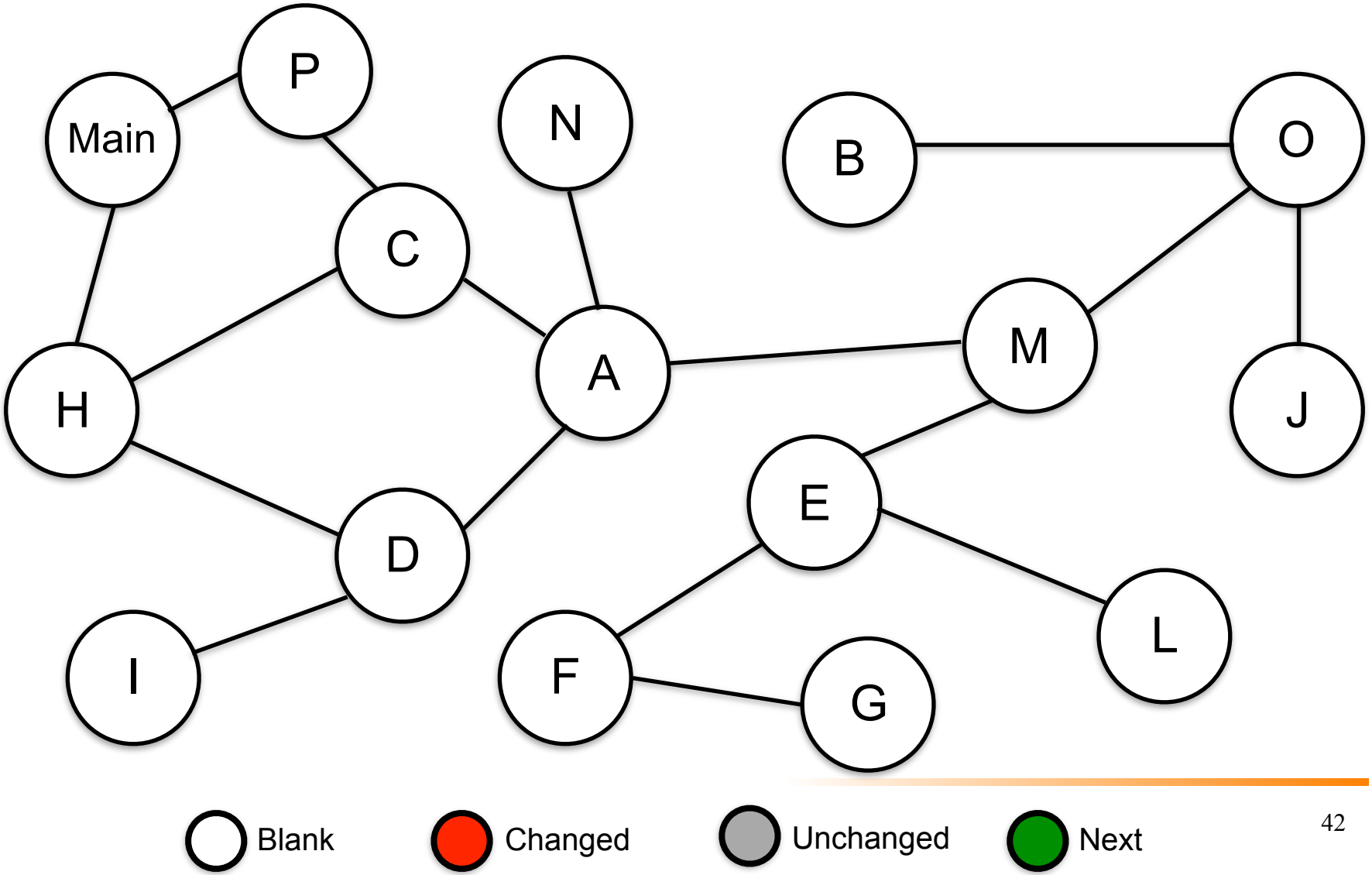
Class Interaction Graph: Node Status

Blank	The class was never inspected and is not scheduled for an inspection.
Changed	The programmers inspected the class and found that it is impacted by the change
Unchanged	The programmers inspected the class and found that it is not impacted by the change.
Next	The class is scheduled for inspection

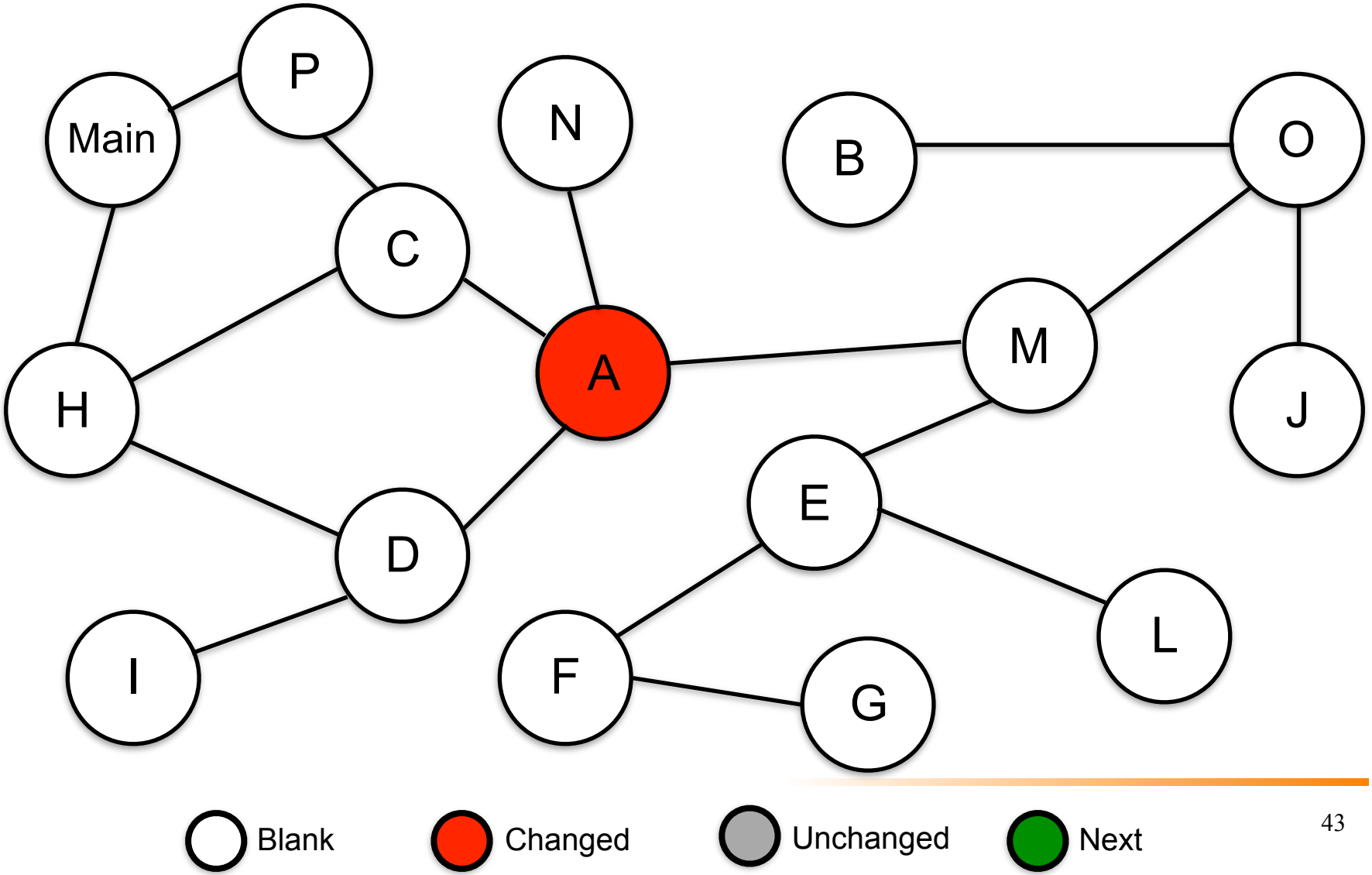
The Impact Analysis Process



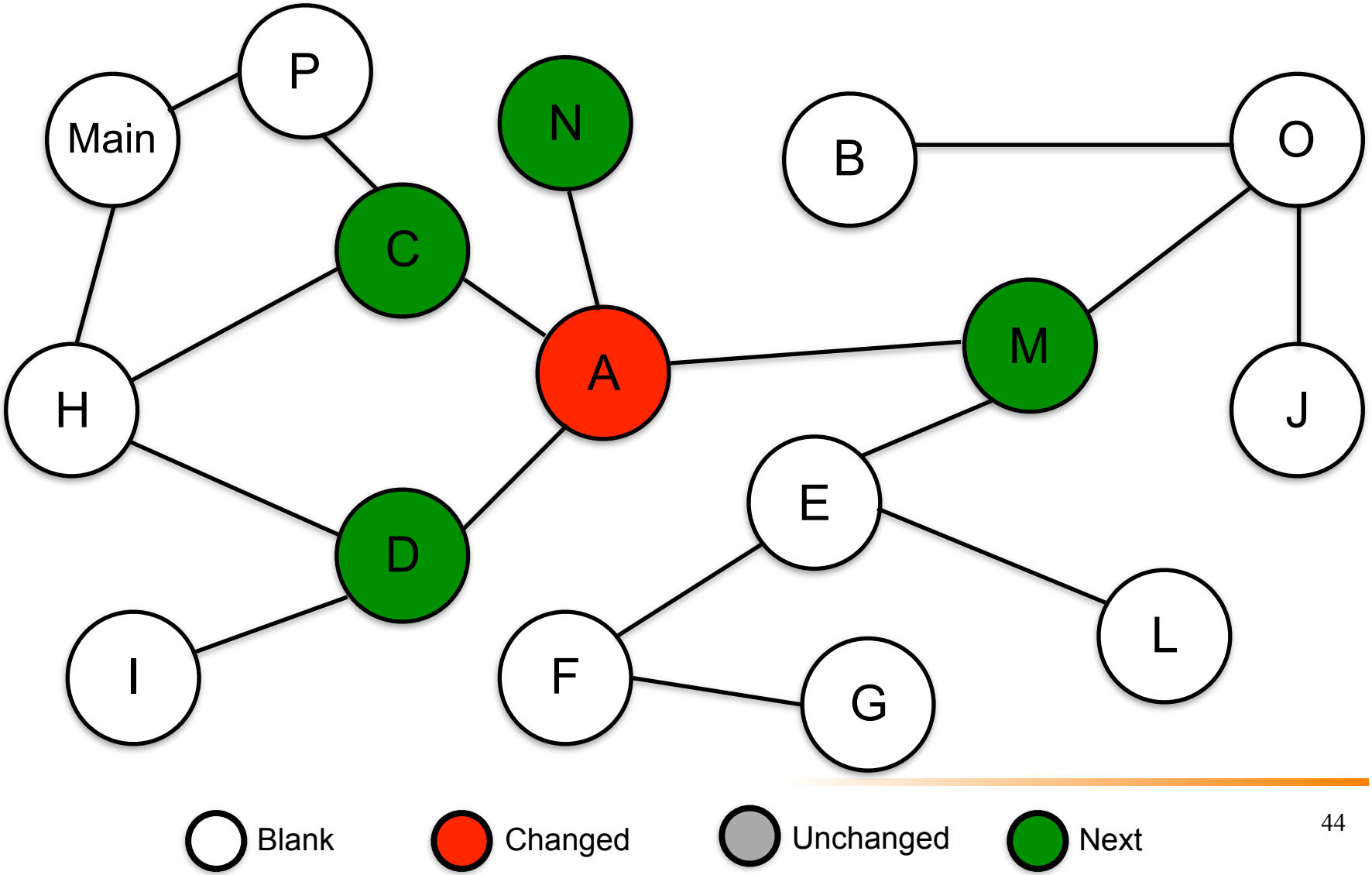
Class Interaction Graph: Example



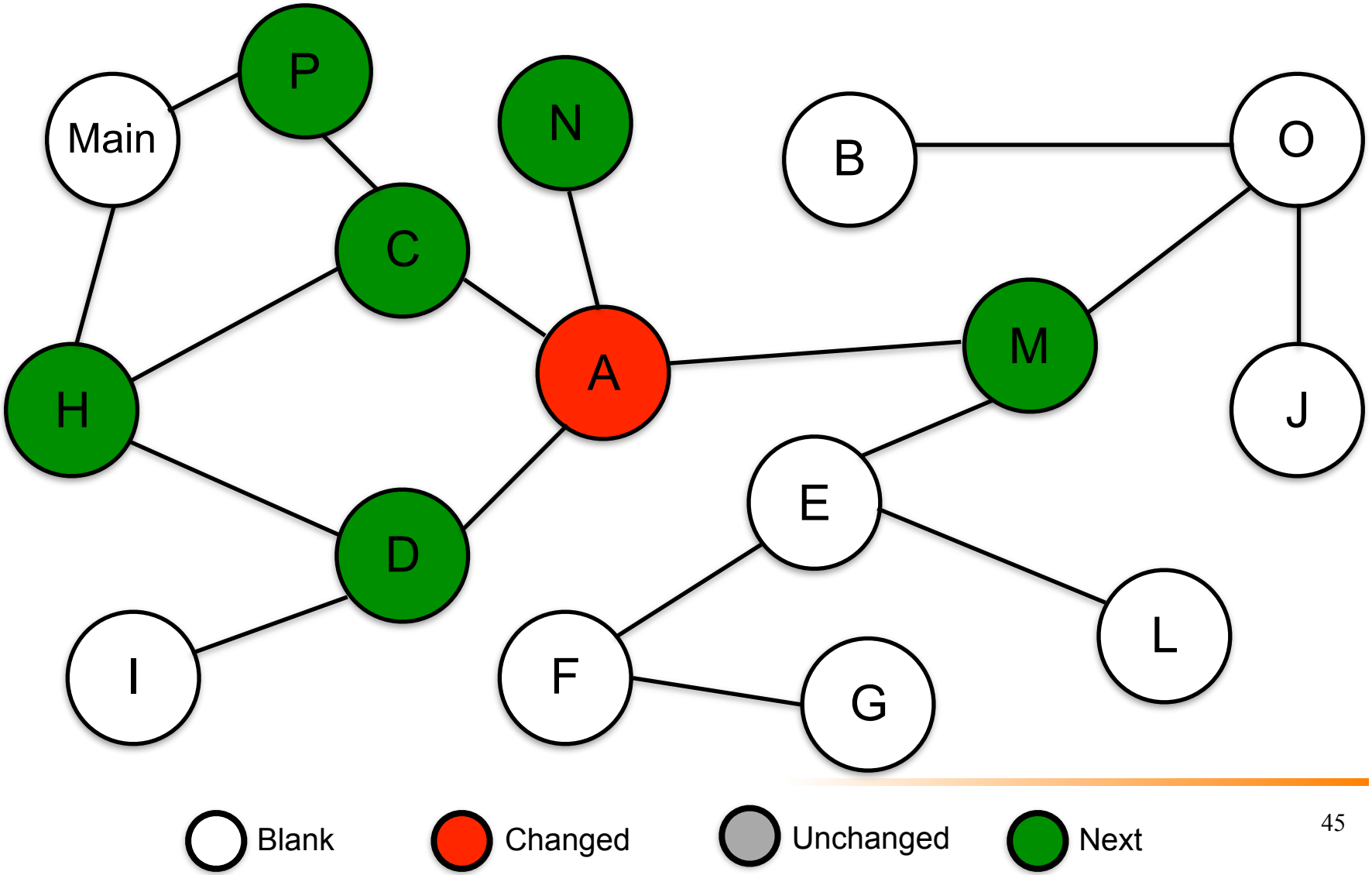
Impact Analysis in action!



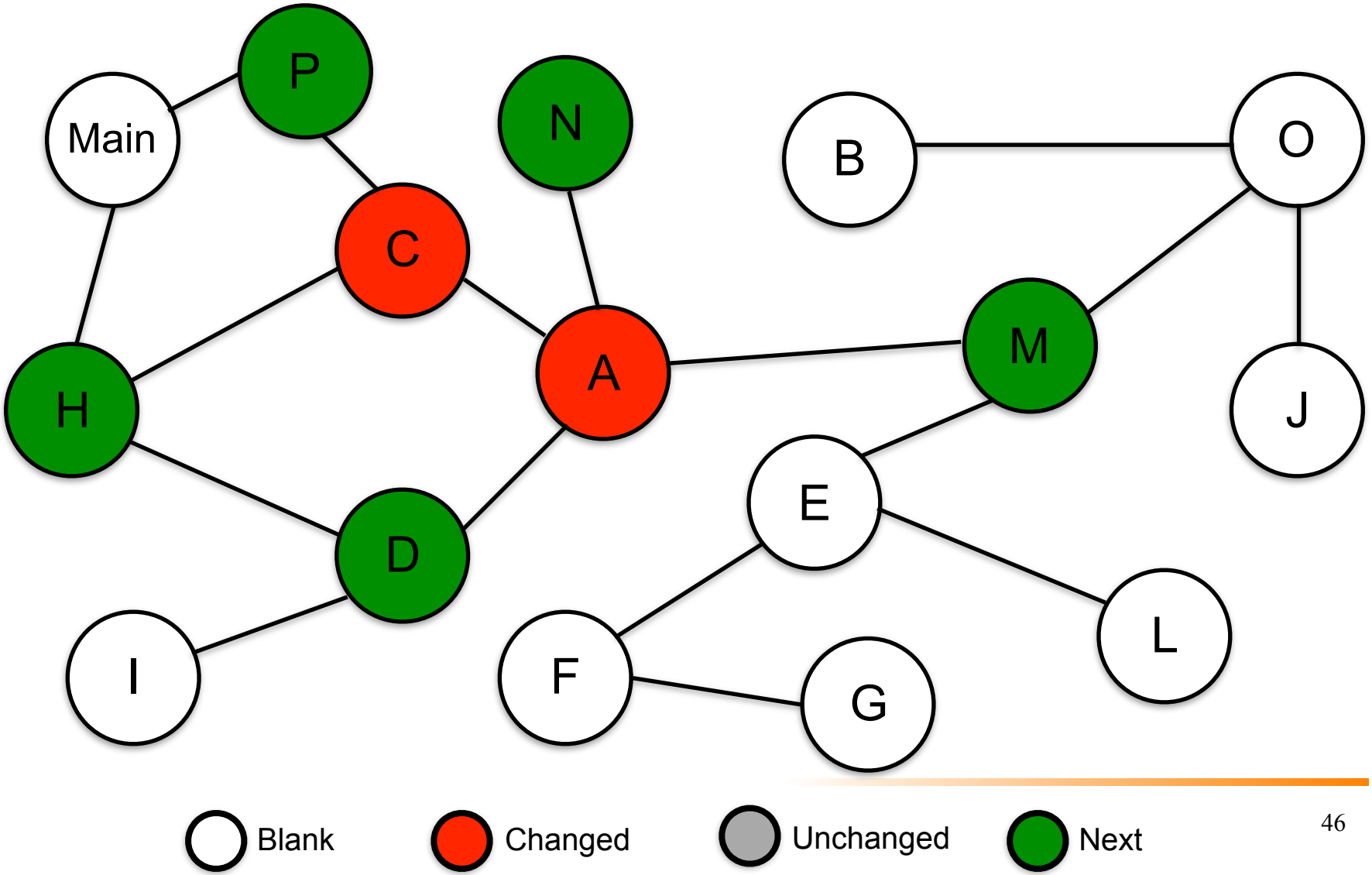
Impact Analysis in action!



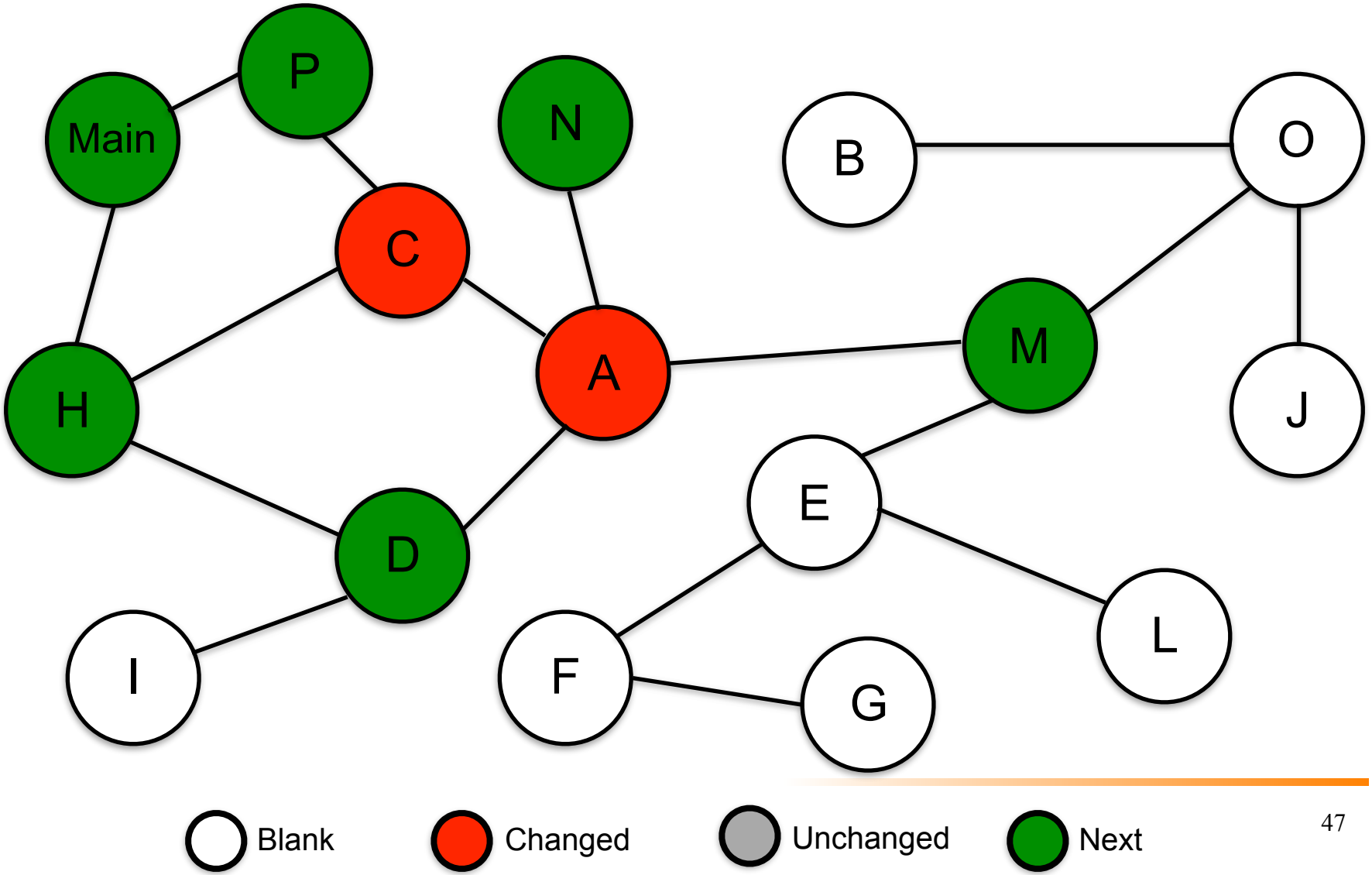
Impact Analysis in action!



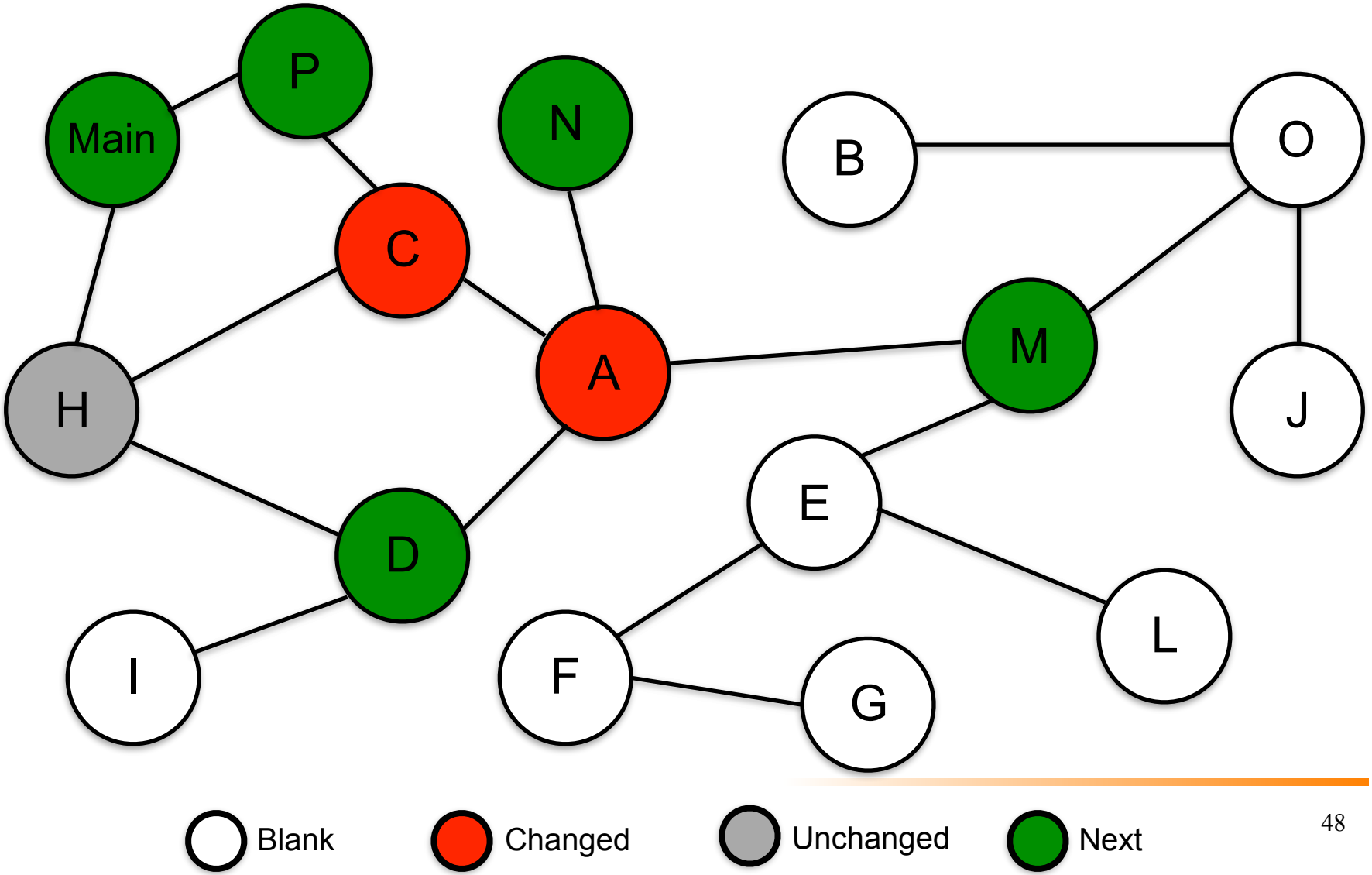
Impact Analysis in action!



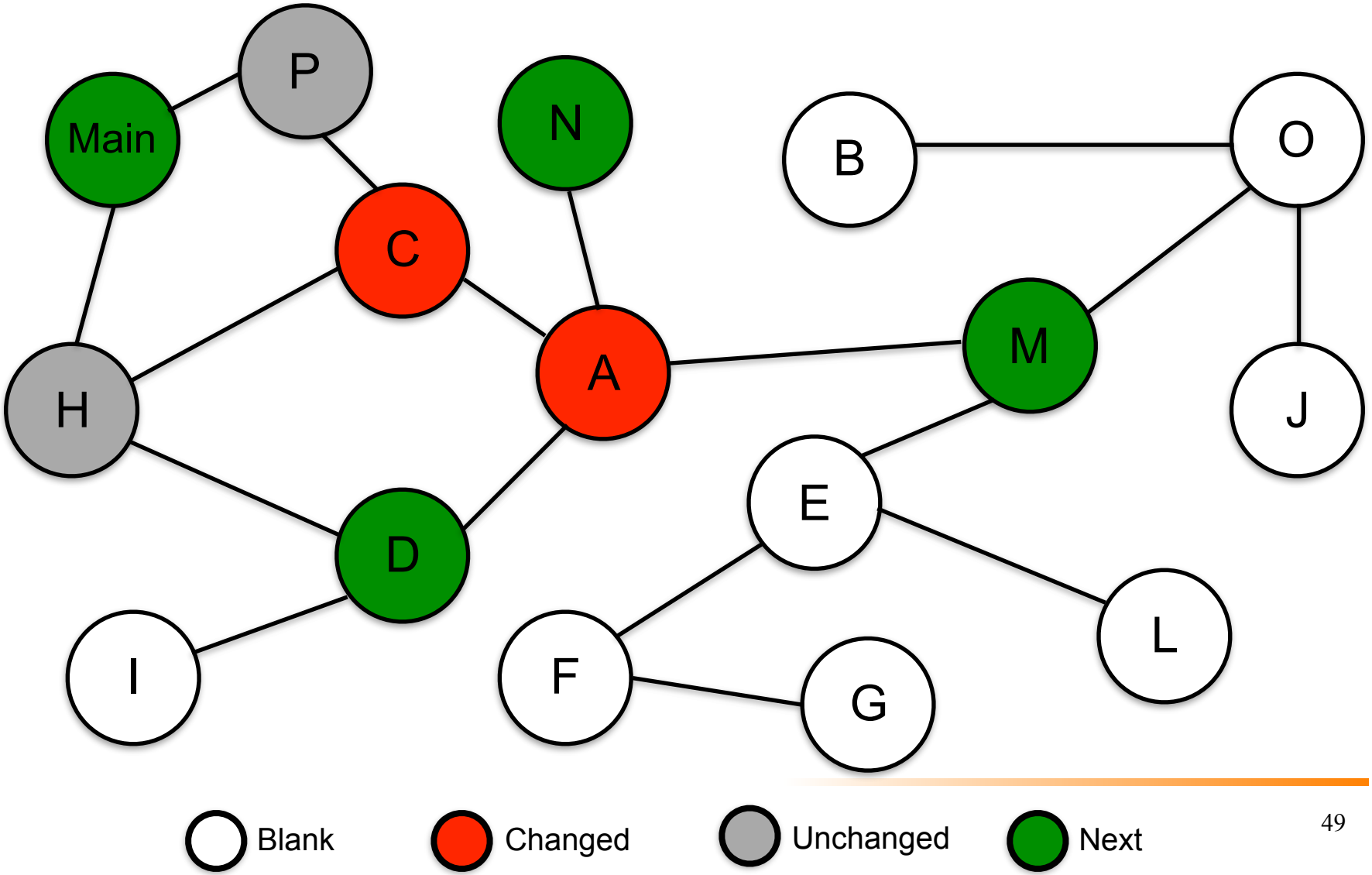
Impact Analysis in action!



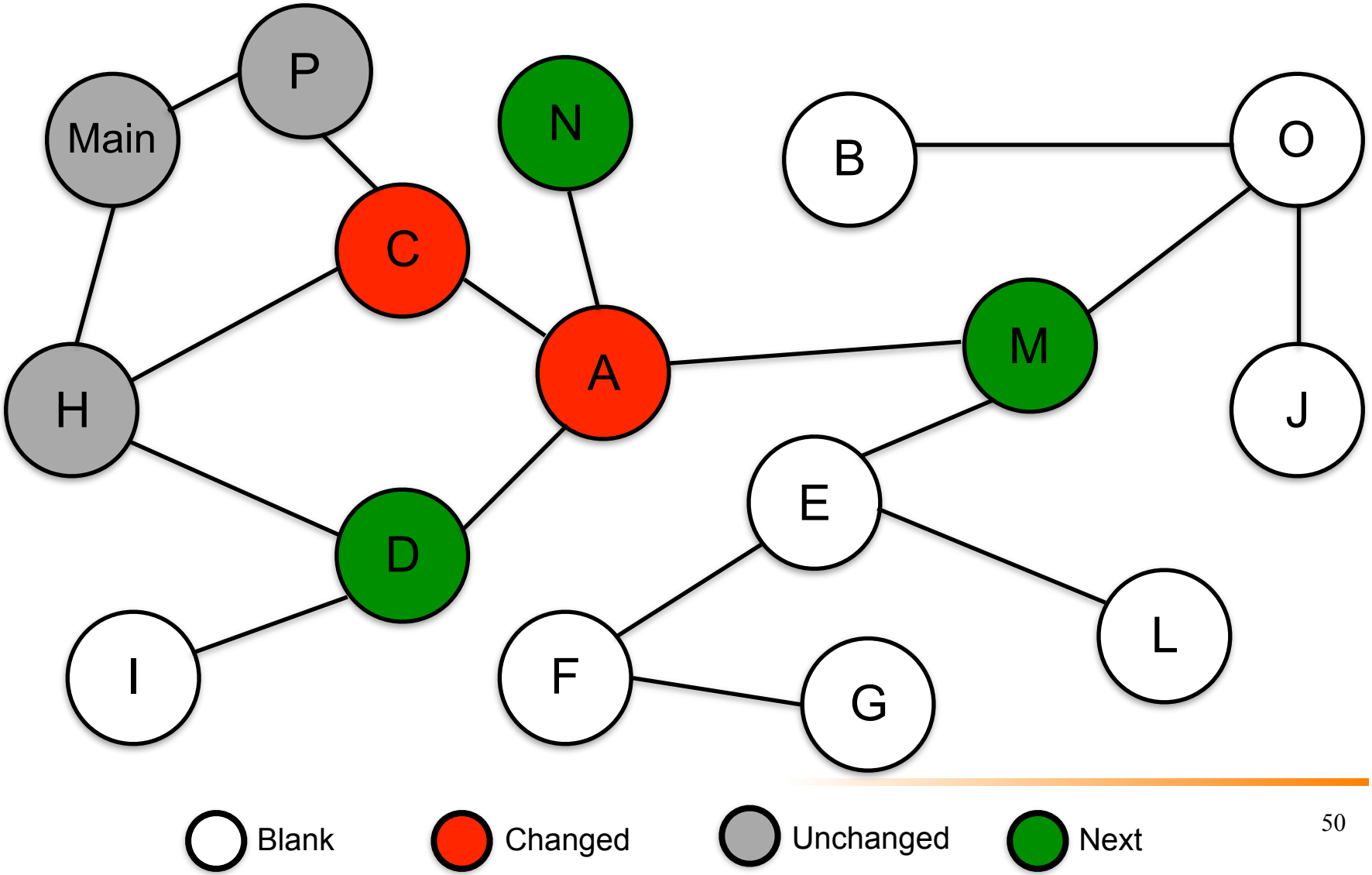
Impact Analysis in action!



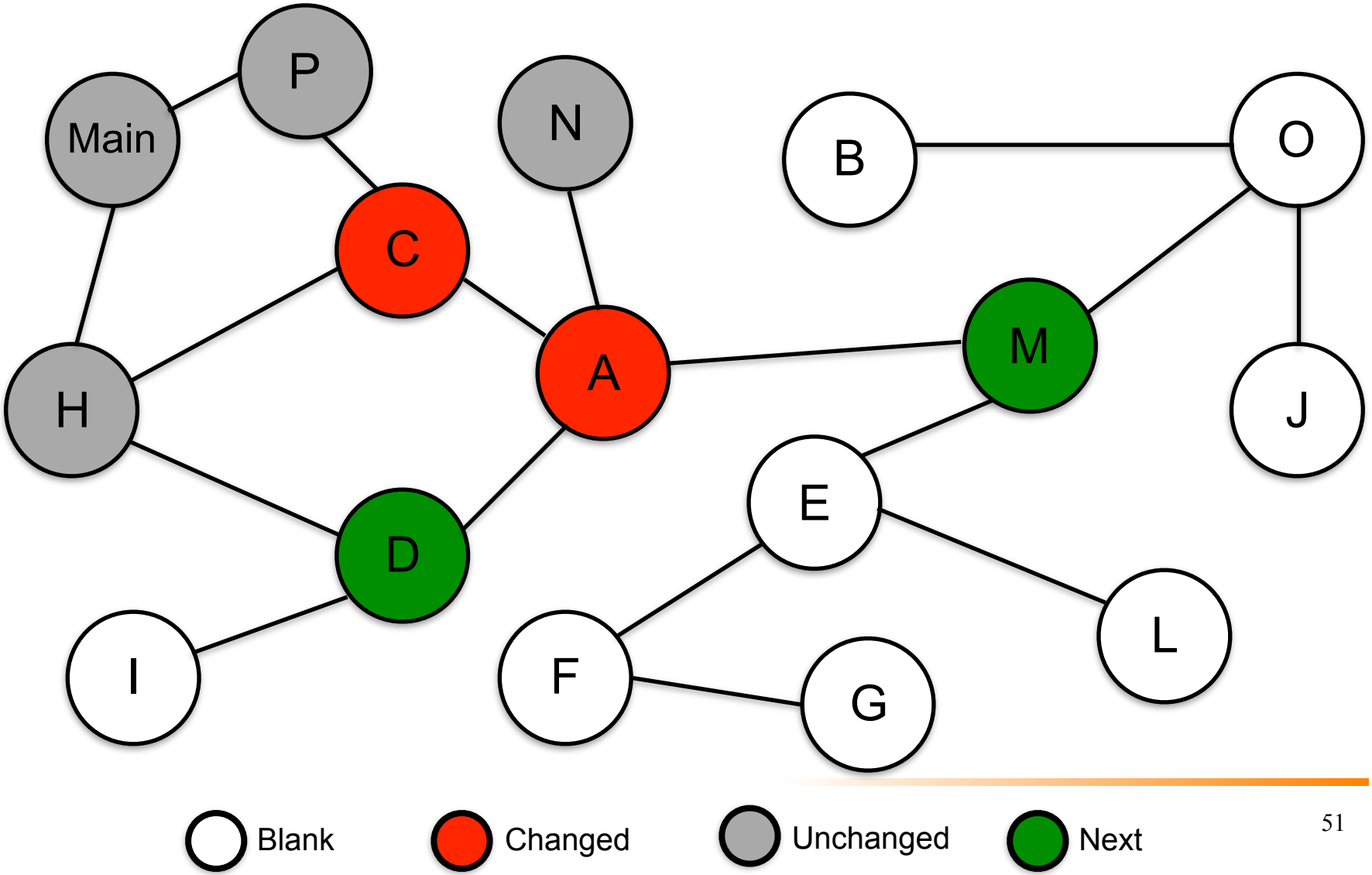
Impact Analysis in action!



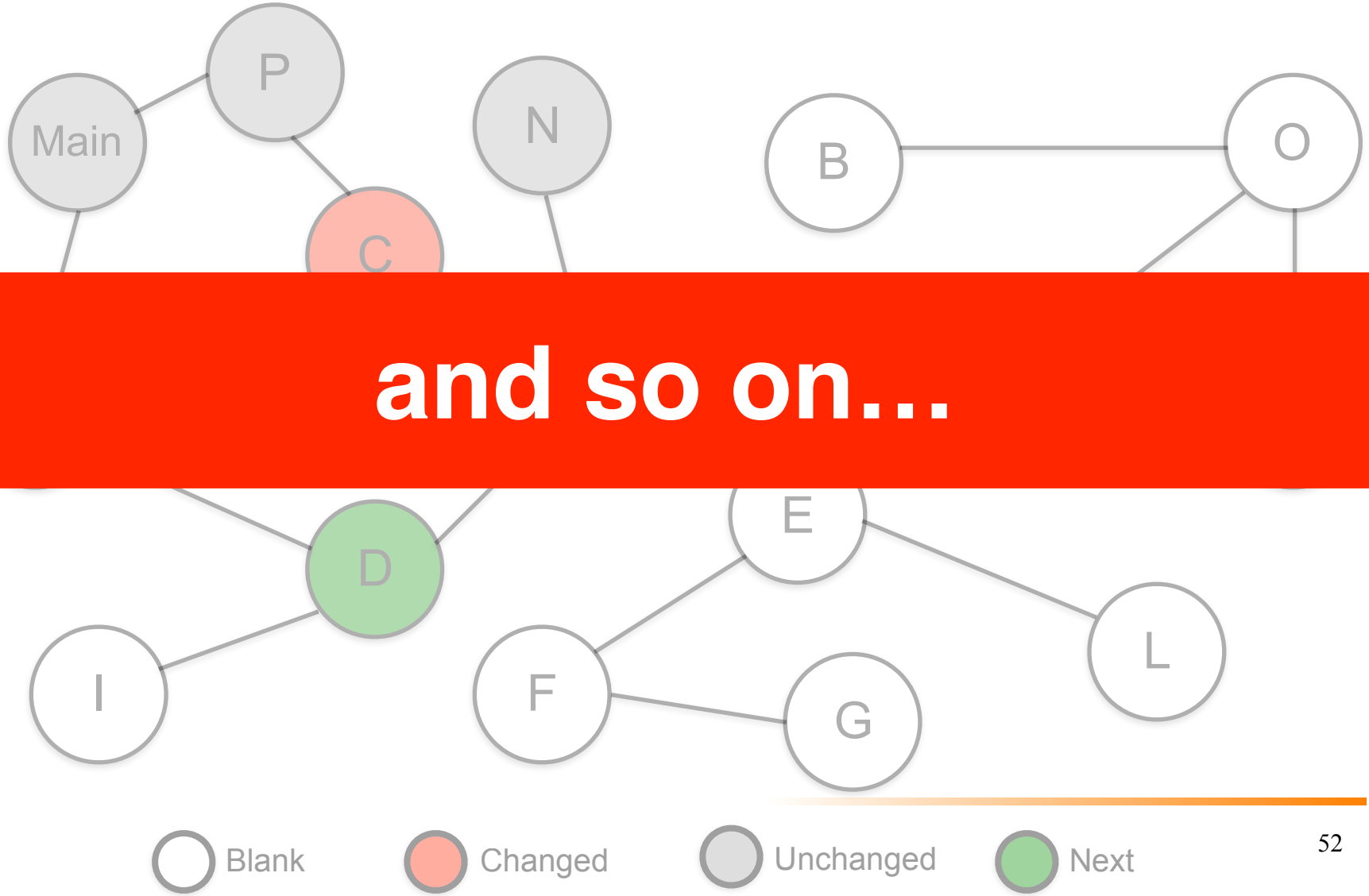
Impact Analysis in action!



Impact Analysis in action!



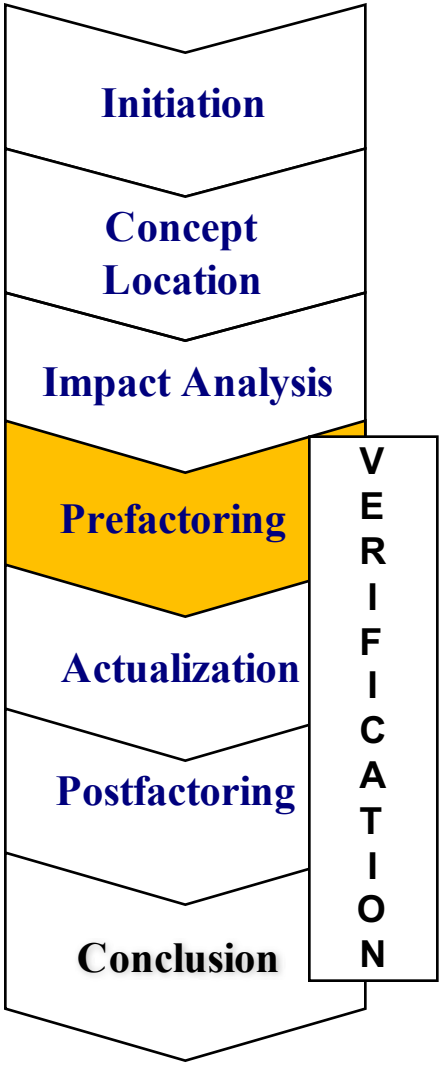
Impact Analysis in action!



and so on…

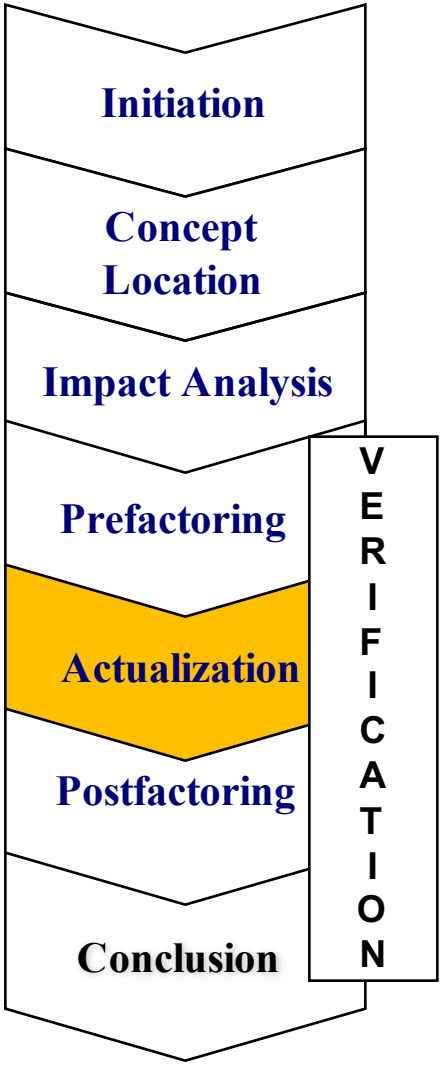
Prefactoring

- Opportunistic refactoring that localizes (minimizes) impact of SC on software
- Extract Class (Fowler)
  - gather fields, methods, and code snippets into a new component class
- Extract Superclass
  - create new abstract class



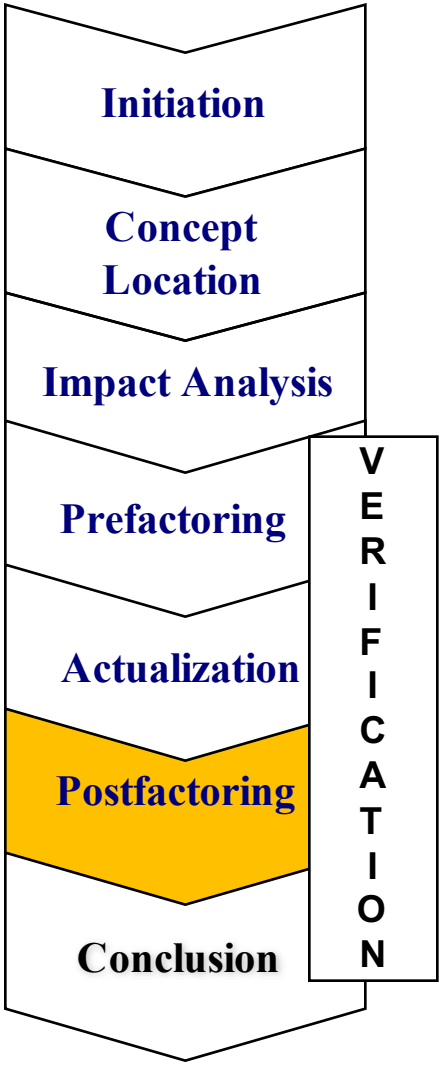
Actualisation

- Creates new code
- Plugs it into the old code
- Visit neighboring classes and update them
  - change propagation
  - ripple effect



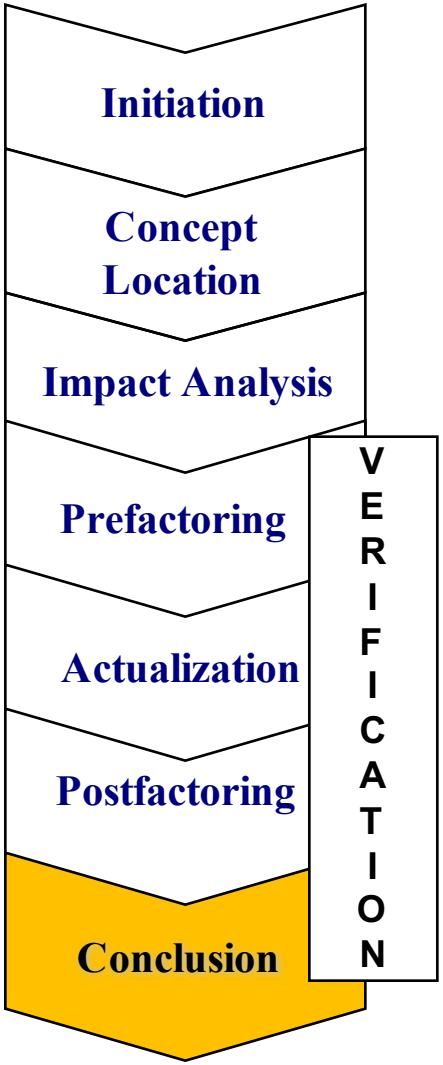
Postfactoring

- Eliminate any anti-patterns that may have been introduced
  - long method
    - after added functionality, some methods may be doing too much
- bloated class
  - after added functionality, a class may be too large



Conclusion

- Commit finished code into version control
- Build the new baseline
- Prepare for the next change



Part 2

Implement a Software Change

Before starting…tooling setup

- We will use *Eclipse Helios* as IDE (pretty old, but it works quite well with the next tools)
  - Go to
    - https://www.eclipse.org/downloads/packages/release/Helios/SR2
  - Select “Helios Packages”
  - Select “Eclipse Classic 3.6.2” and download it (version depends on your machine)

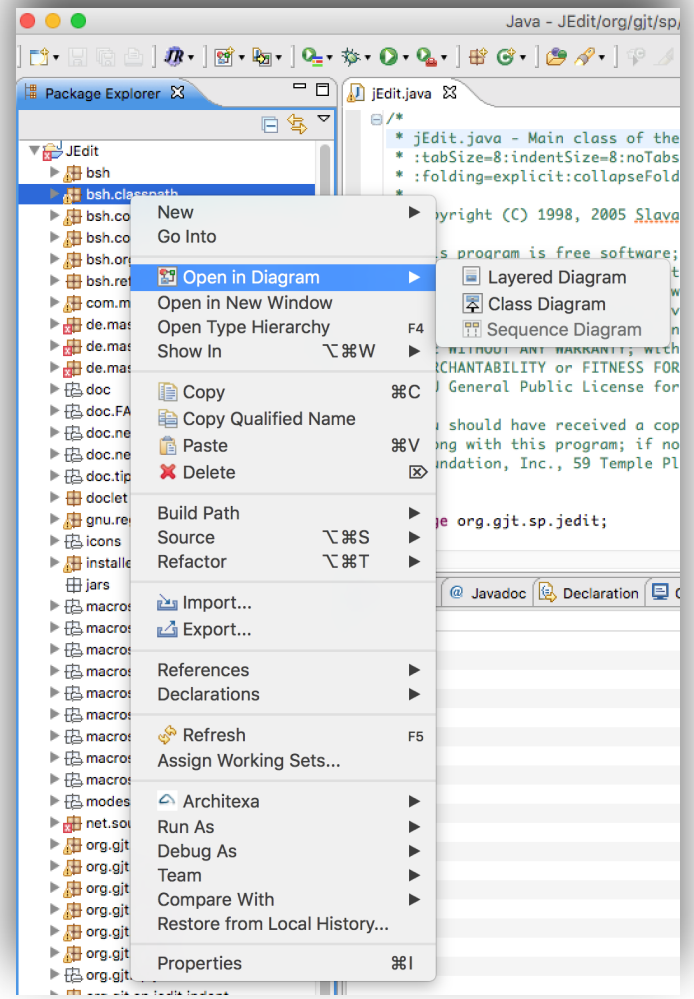
Before starting…tooling setup

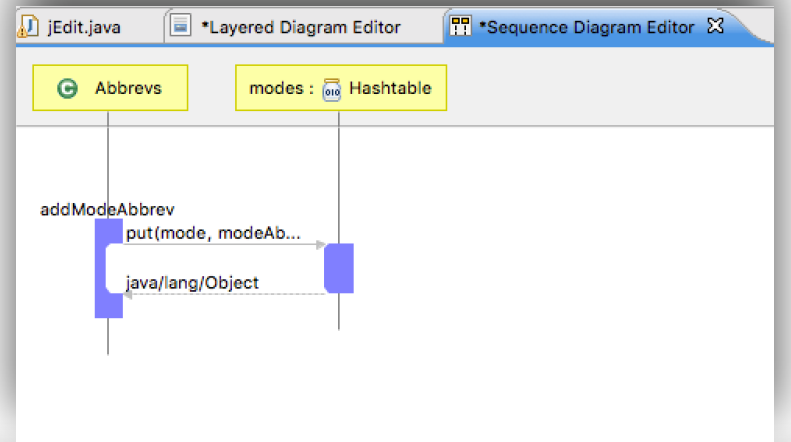
- Concept Location
  - GREP analysis
    - Built-in functionality in Eclipse
  - Dependencies Search
    - Architexa
    - STAN
- Impact Analysis
  - JRipple

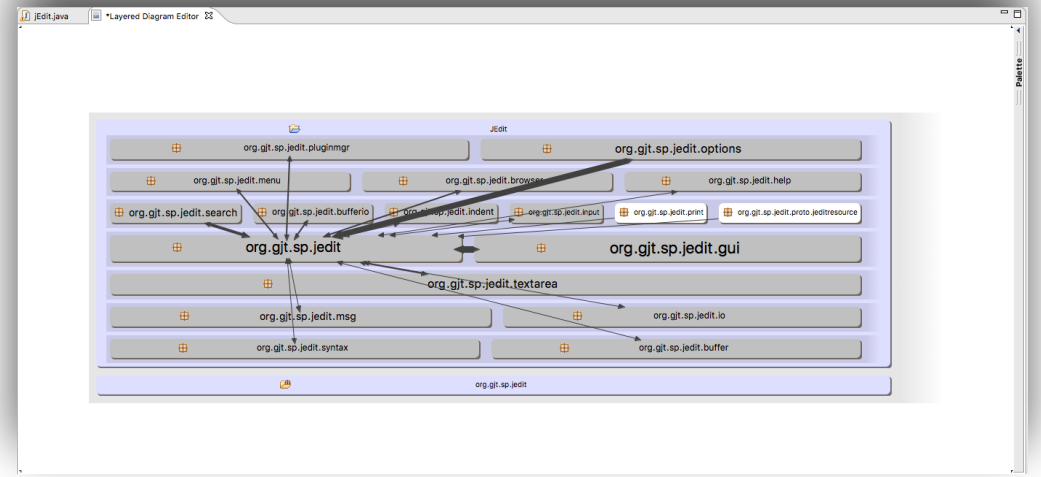
Architexa

- In Eclipse, select **Help > Install New Software**... In the dialog that appears, click on the **Add**... button. Enter the update site URL into the "Location" text box:
  - http://update.architexa.com/client
- Leave the name text box empty (the name will be retrieved from the update site). Click **OK**.
- The main pane should update to show the Architexa RSE Client. Select the checkbox next to it. Click **Next**.
- Review the features that you are about to install. Click **Next**. Accept the license agreements, click **Finish**, and restart Eclipse when prompted.

Architexa in action!



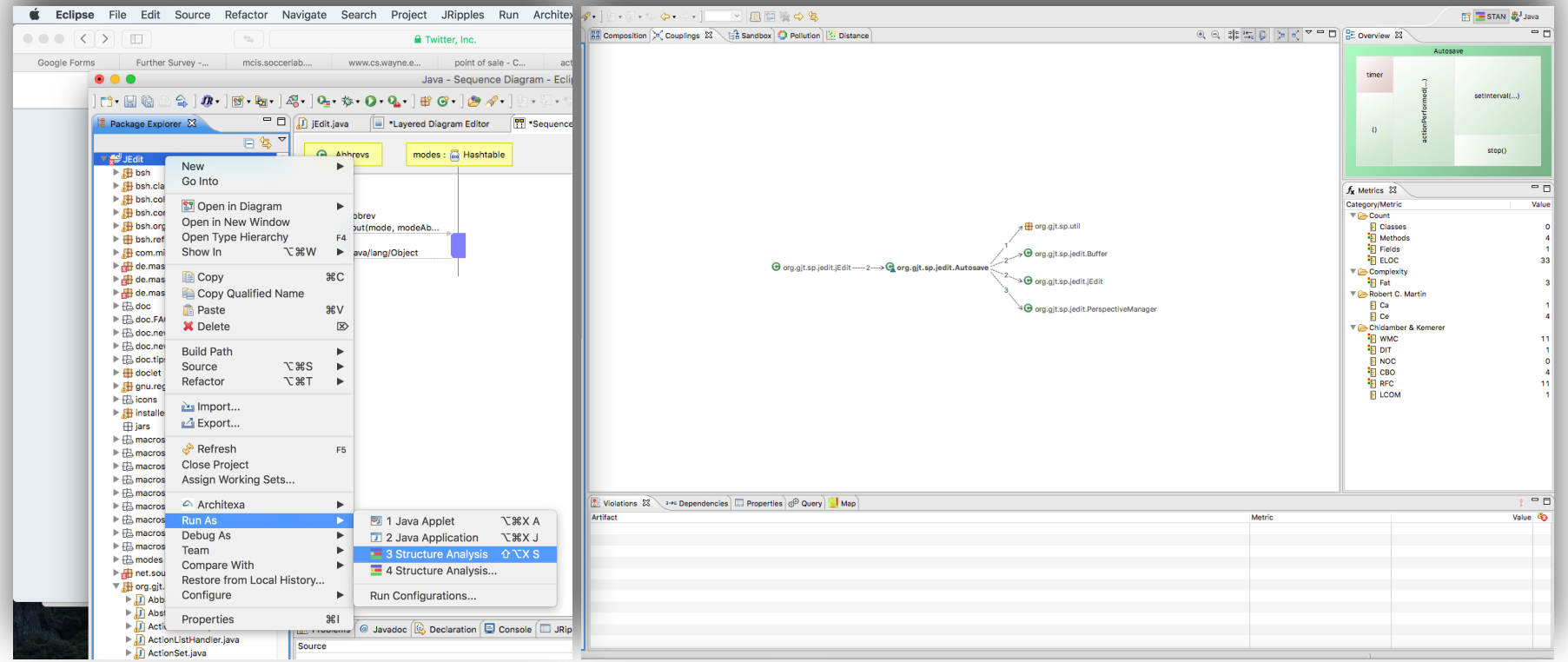




STAN

- In Eclipse, select **Help > Install New Software**... In the dialog that appears, click on the **Add**... button. Enter the update site URL into the "Location" text box:
  - http://stan4j.com/general/download-ide.html
- Type the word “STAN” in the text box called “Name” and Click **OK**.
- Follow the instructions clicking **Next** and **Finish**.

STAN in action!



JRipple

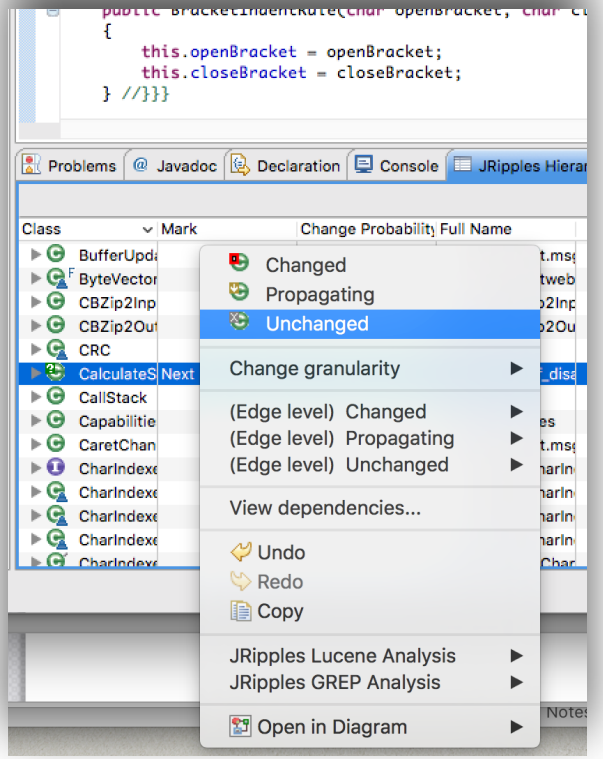
- Download jripples_x_y.jar, where x and y are numbering of a version from http://jripples.sourceforge.net/
- Run Eclipse.
- Choose from the main menu **Help>Install New Software.**
- Click “**Add…**” tab.
- Click "**Archive...**", select the JRipples.jar file you downloaded and press "**Open**".
- Put mark into the box next to "JRipples.jar" in the "Available software" list and press "**Finish**".
- Make sure mark "JRipples Main Components" and "JRipples Feature" are also selected.
- Press "**Install...**" and follow the instructions of the Install Wizard.
- Press "Yes" on the popped-up "Restart workbench" window.
- JRipples menu will appear in Eclipse's menu bar.

JRipple in action!

- After selecting "Propagating" from the right click menu, and marked them "Next".

- The user will then visit each of those classes to determine if the change has propagated into those classes or not.

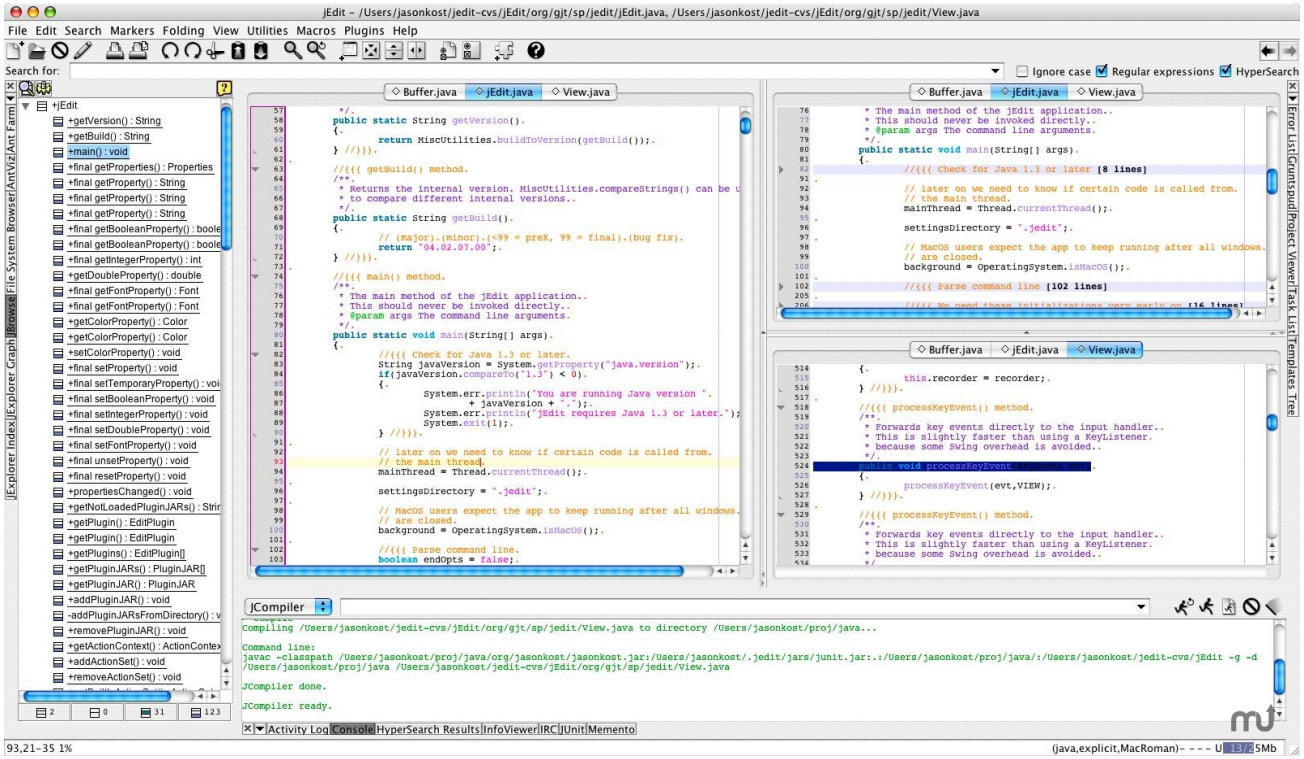
- If the change does not require the class to be modified the class will be marked “Unchanged".

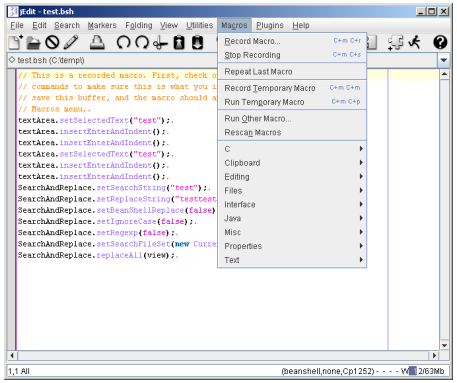


CLICK HERE TO WATCH THE DEMO: https://www.youtube.com/watch?v=fOM0K2bAmm8

Case Study: jEdit

- jEdit is a free text editor
- Written in Java





http://www.jedit.org

Case Study: jEdit

- Dowload jedit4.3pre9source.tar.bz2 from
  - http://sourceforge.net/projects/jedit/files/jedit- devel/ 4.3pre9/
- Unpack the archive in a directory. For reference, let's name the directory JEdit4.3
- Select **File->New->Project...->Java Project**. Hit Next button. Write the Project name, and import the source code browsing to JEdit4.3 directory. Eclipse automatically includes the folders as source folders**
- Run jEdit as a Java Application using the main class
  - org/gjt/sp/jedit/jEdit.java

Software changes

- The Product Owner has already translated the <u>change requests</u> proposed by stakeholders in <u>4 software changes</u>
- You have to implement the software changes following the Software Change Process described in the Part 1
- Deliverables
  1. **Report** (or Presentation) describing which actions you take to implement each change
  2. **Source code** with the implemented changes
- Send the deliverables to me by 26/10

vassallo@ifi.uzh.ch

Software Change 1

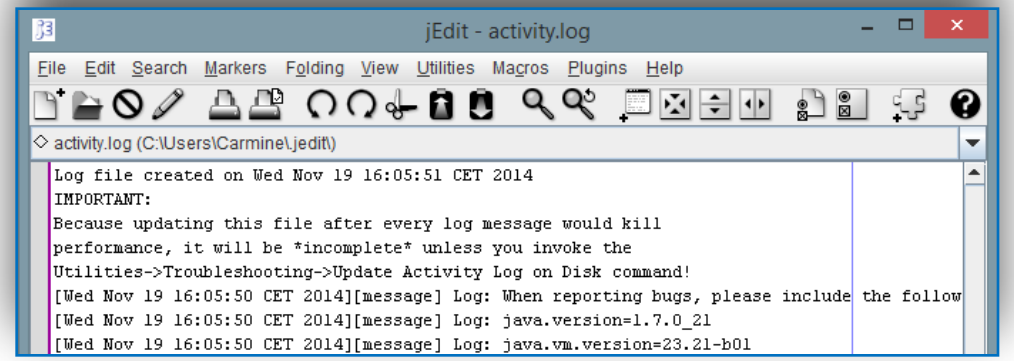
- “Currently the splash window of jEdit is a static picture. Add your name and your email to it.”

Software Change 2

- “Add your name to the text shown in About JEdit dialog”

Software Change 3

- “Locate where the activity.log is. Currently there are no timestamps in the log file. Add timestamps to all kinds of messages.”

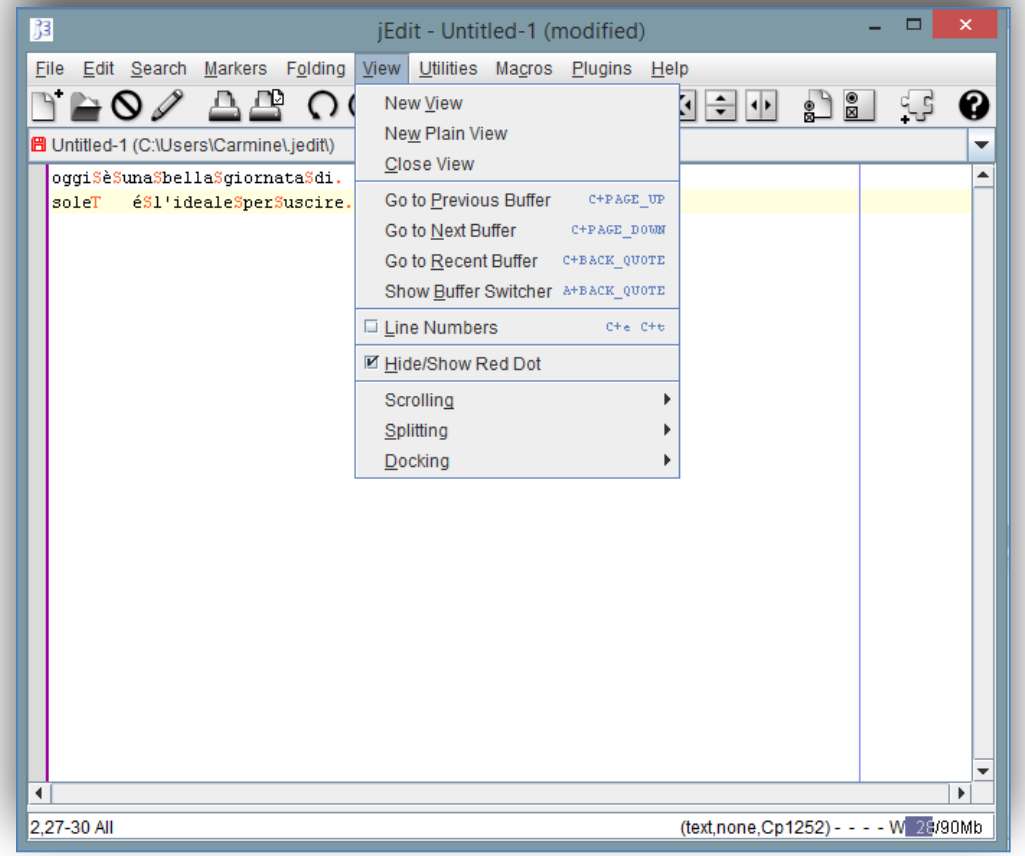


Software Change 4

- “Currently jEdit shows a red dot at the end of every line. Newline is the only whitespace symbol that jEdit shows. Add menu item Show/Hide whitespace under menu View to allow the user to choose whether all whitespace symbols (newlines, blanks, and tabs) will be shown”

Software Change 4

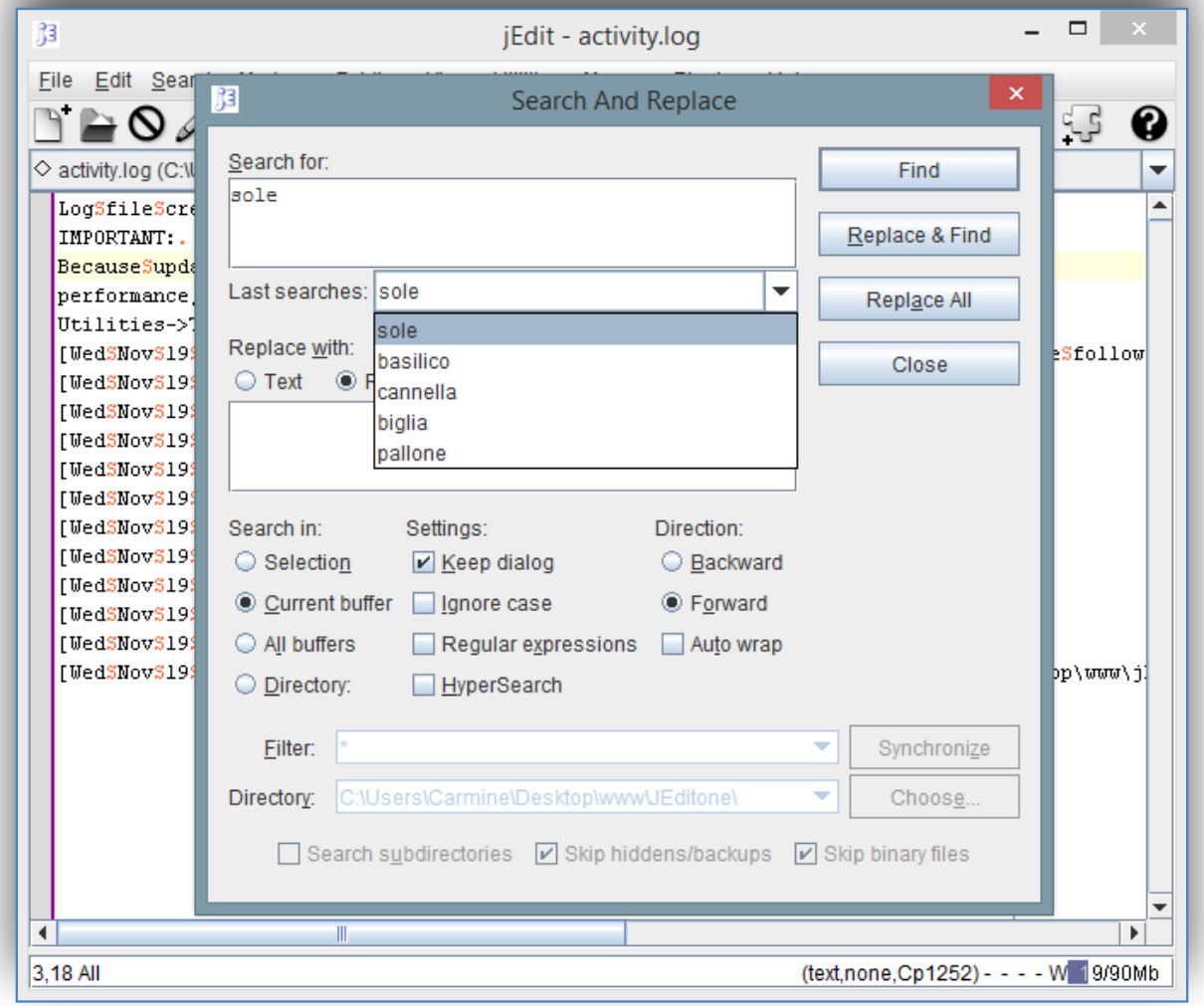
- BLANK = S
- TAB = T
- Nline = . (default)



Software Change 5 (not mandatory)

- “Currently jEdit allows users to access the text that was previously search by pressing page_up or right- click keys in Search Dialog. Display in a listbox the last 5 text fragments that were previously search.”

Software Change 5 (not mandatory)



References

- Václav Rajlich, *Software Engineering: The Current Practice*, CRC Press, Boca Raton, FL (2012)