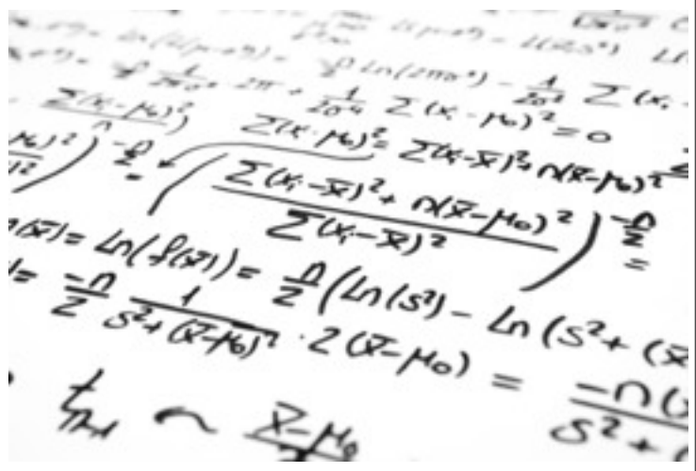
Bug Prediction

SW Wartung und Evolution

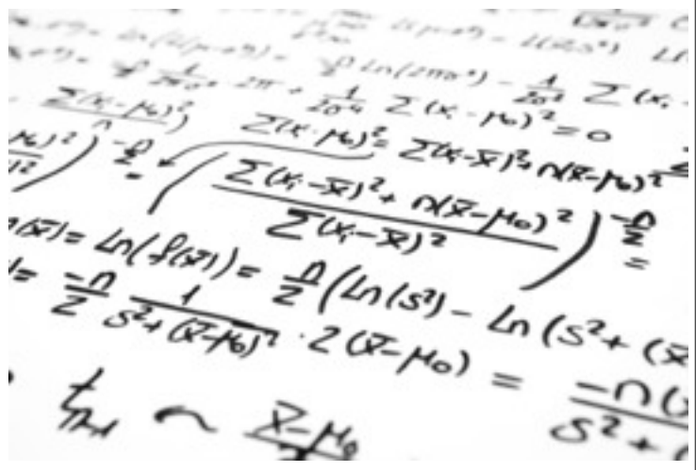
Emanuel Giger

University of Zurich
Department of Informatics

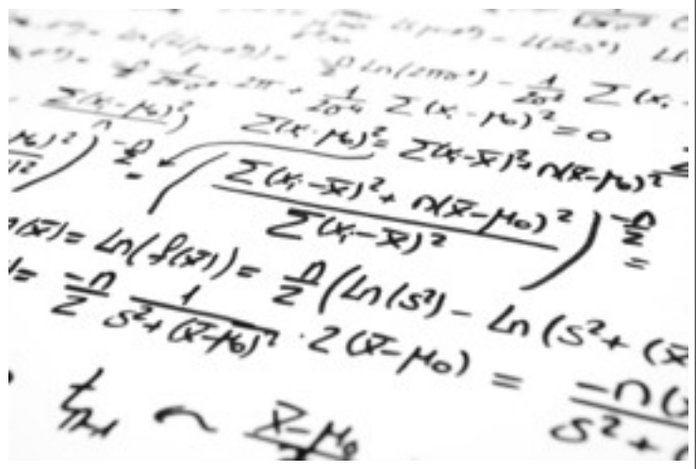
Software has Bugs!

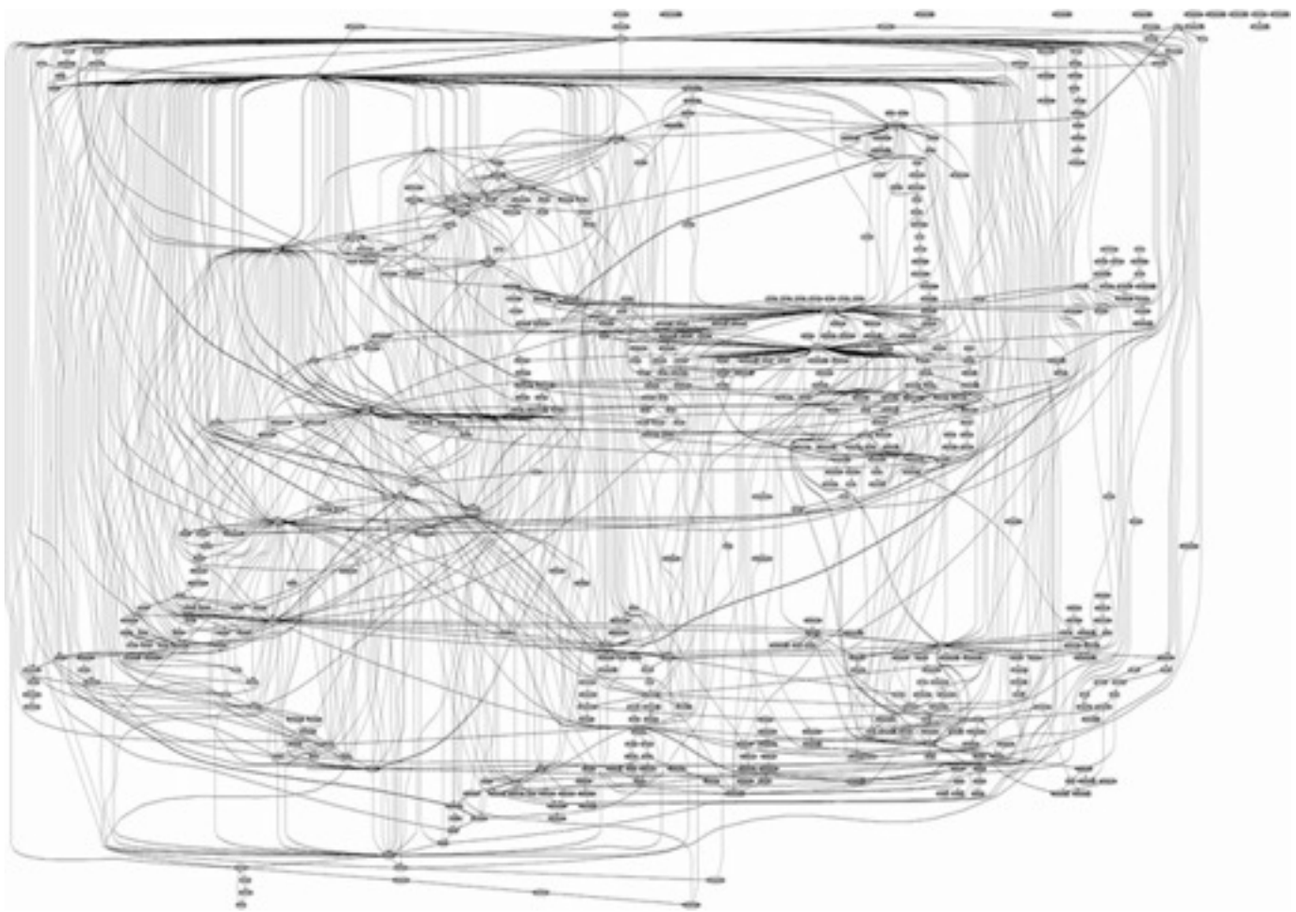


Software has Bugs!

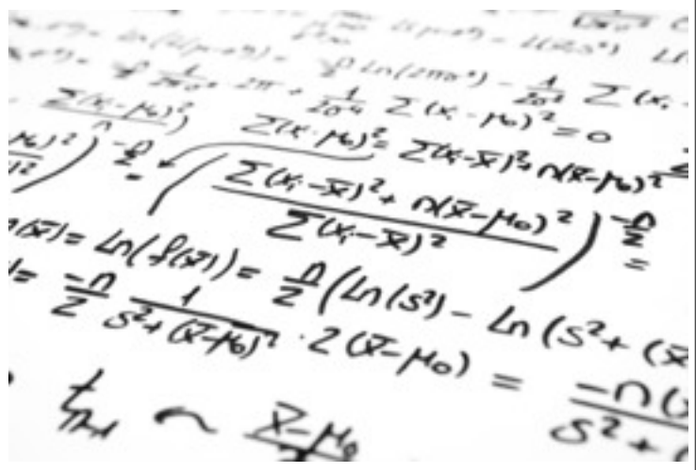


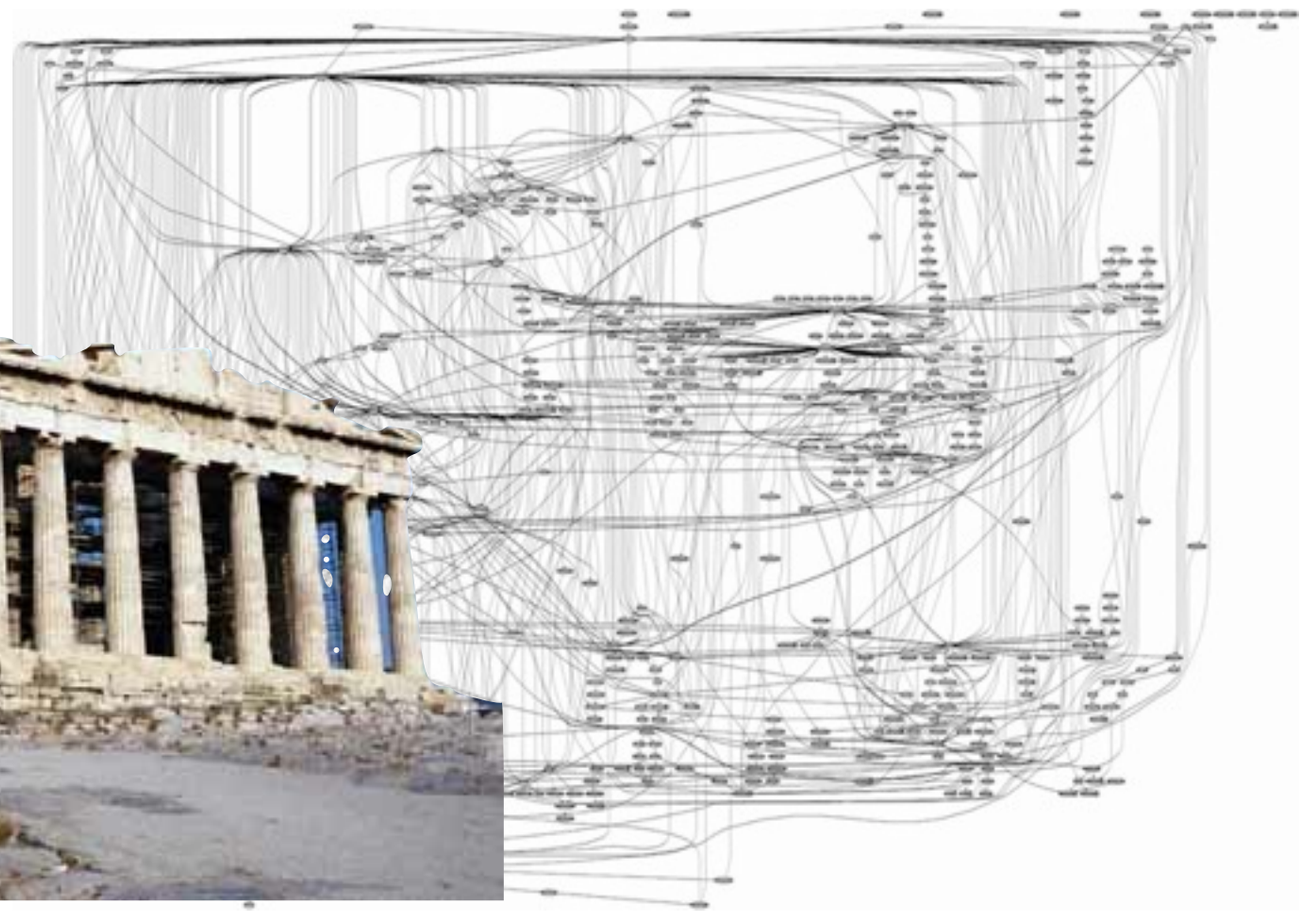
Software has Bugs!



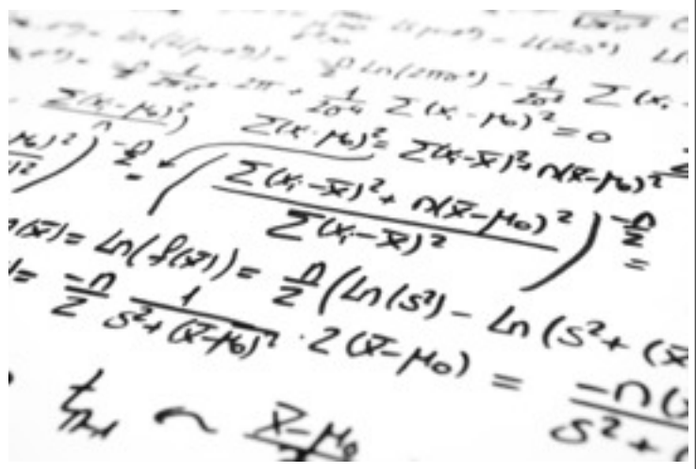


Software has Bugs!

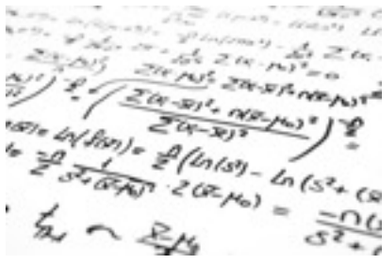


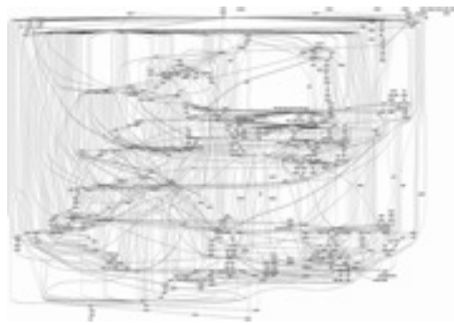


Software has Bugs!



Software has Bugs!





Bugs! Bugs! Bugs! Bugs! Bugs!

Quality Assurance (QA)...

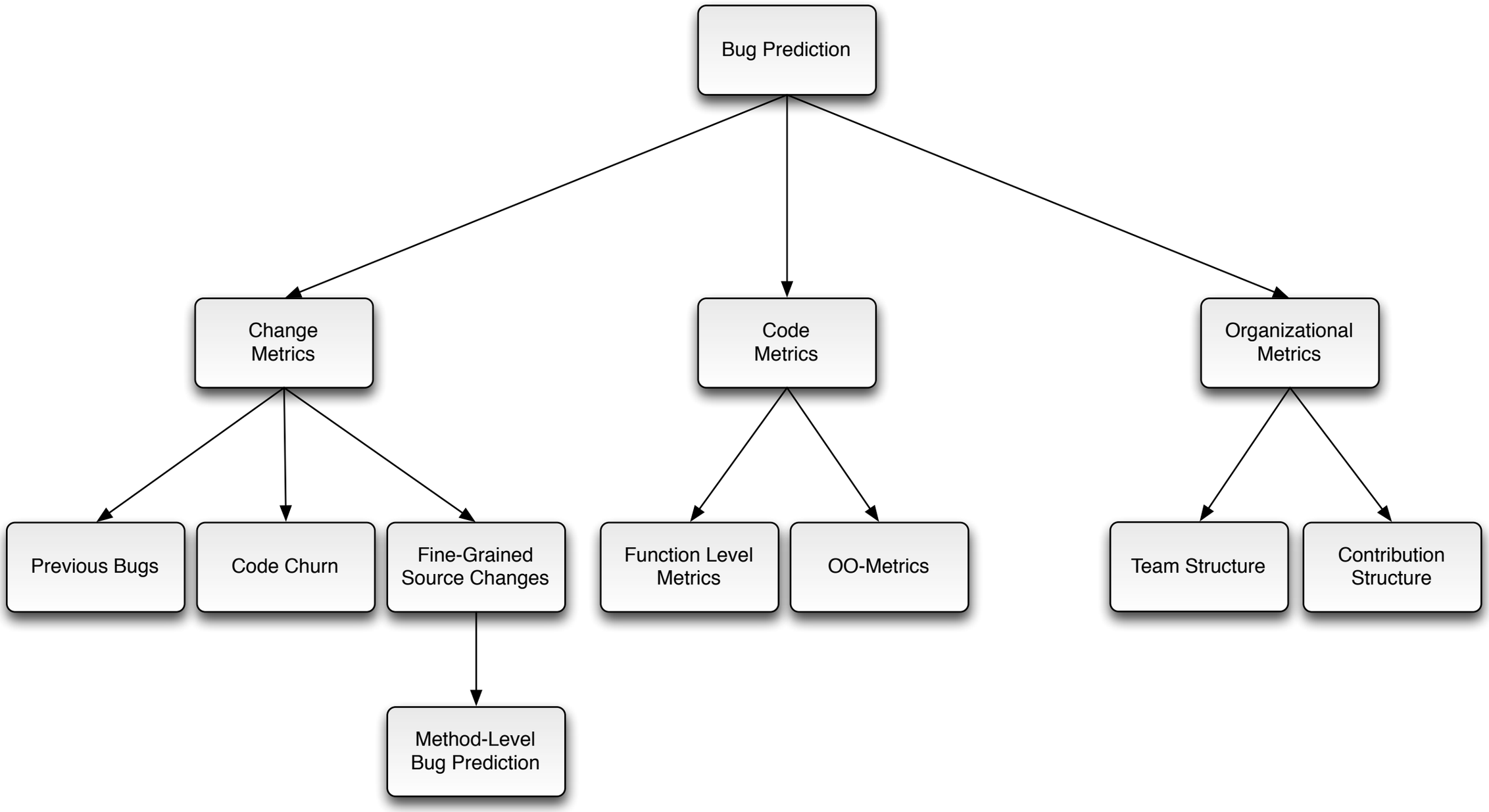
...is limited by time and money

Quality Assurance (QA)...

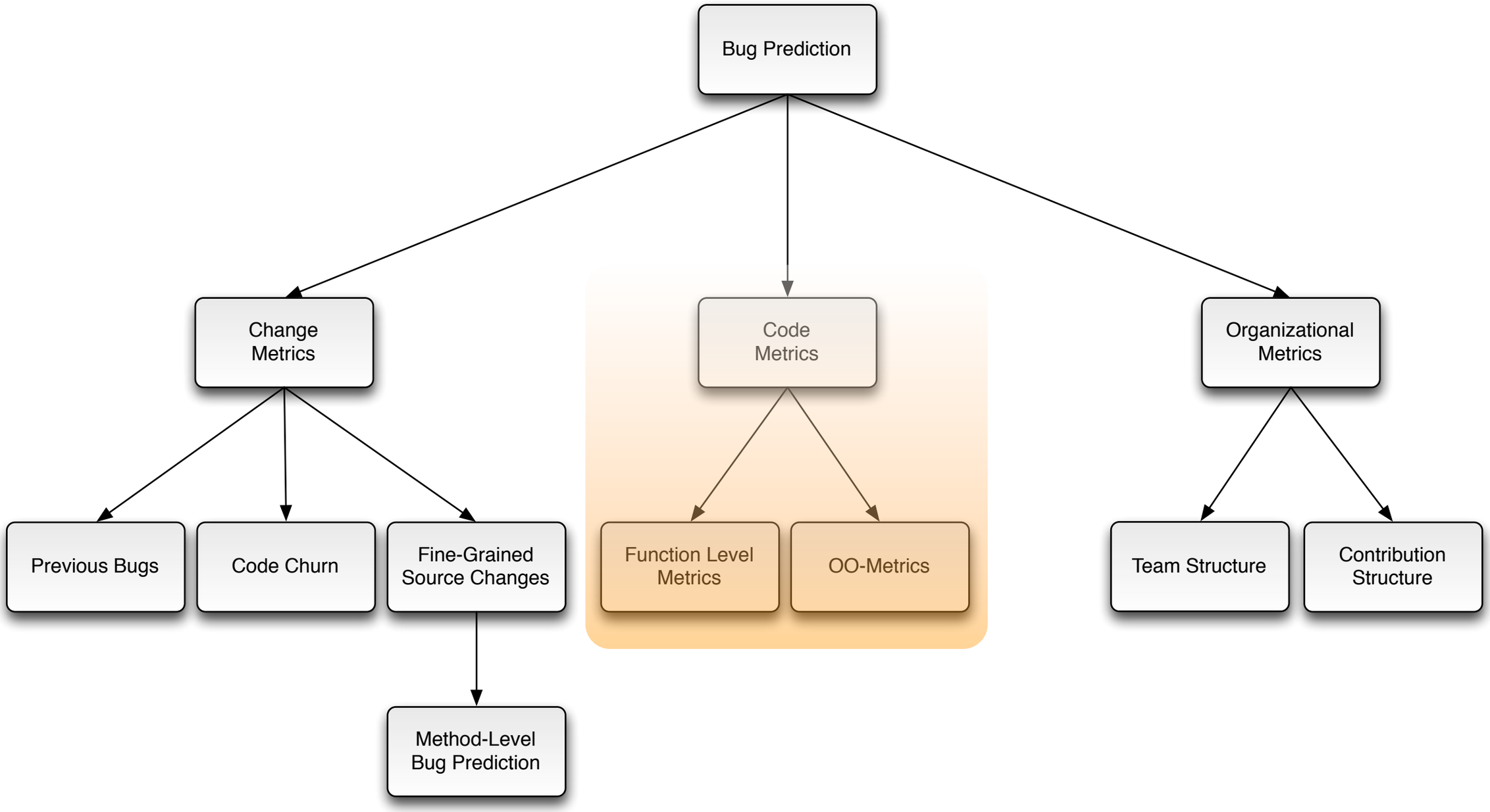
...is limited by time and money

Spend resources with maximum efficiency!
Focus on the components that fail the most!

Bug Prediction Models



Bug Prediction Models

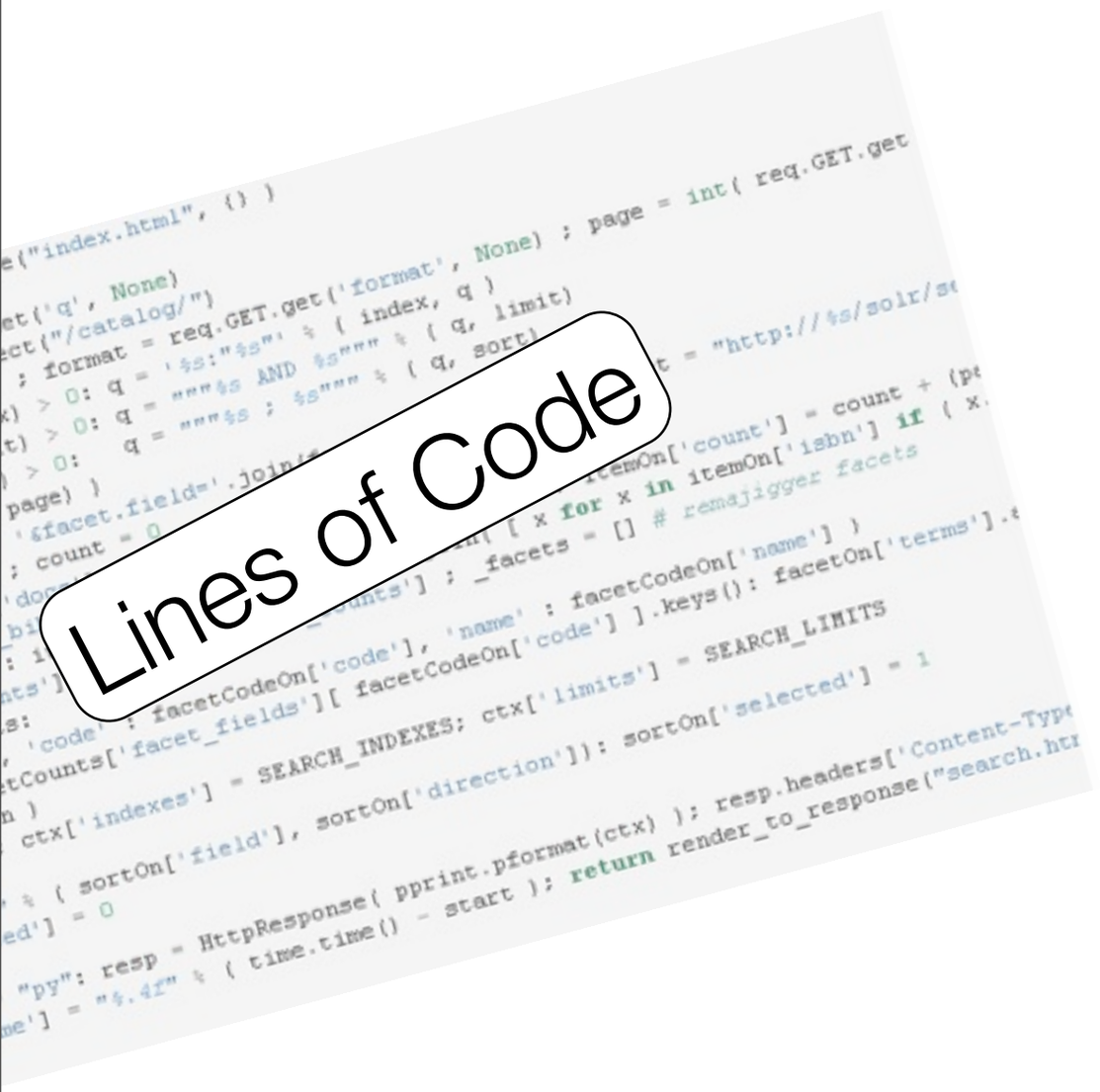


Code Metrics

- Directly calculated on the code itself
- Measure size and complexity of the code

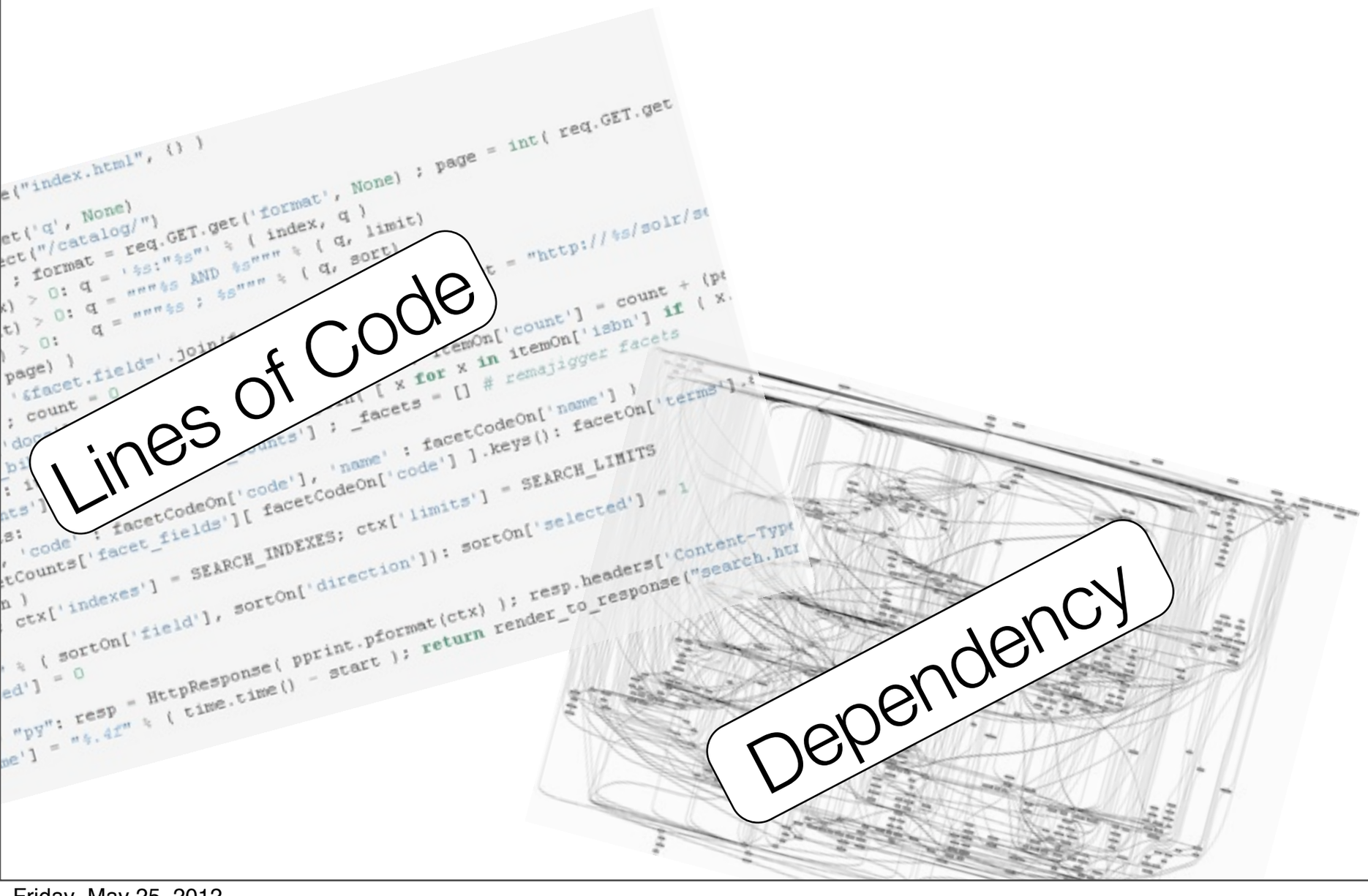
Code Metrics

- Directly calculated on the code itself
- Measure size and complexity of the code



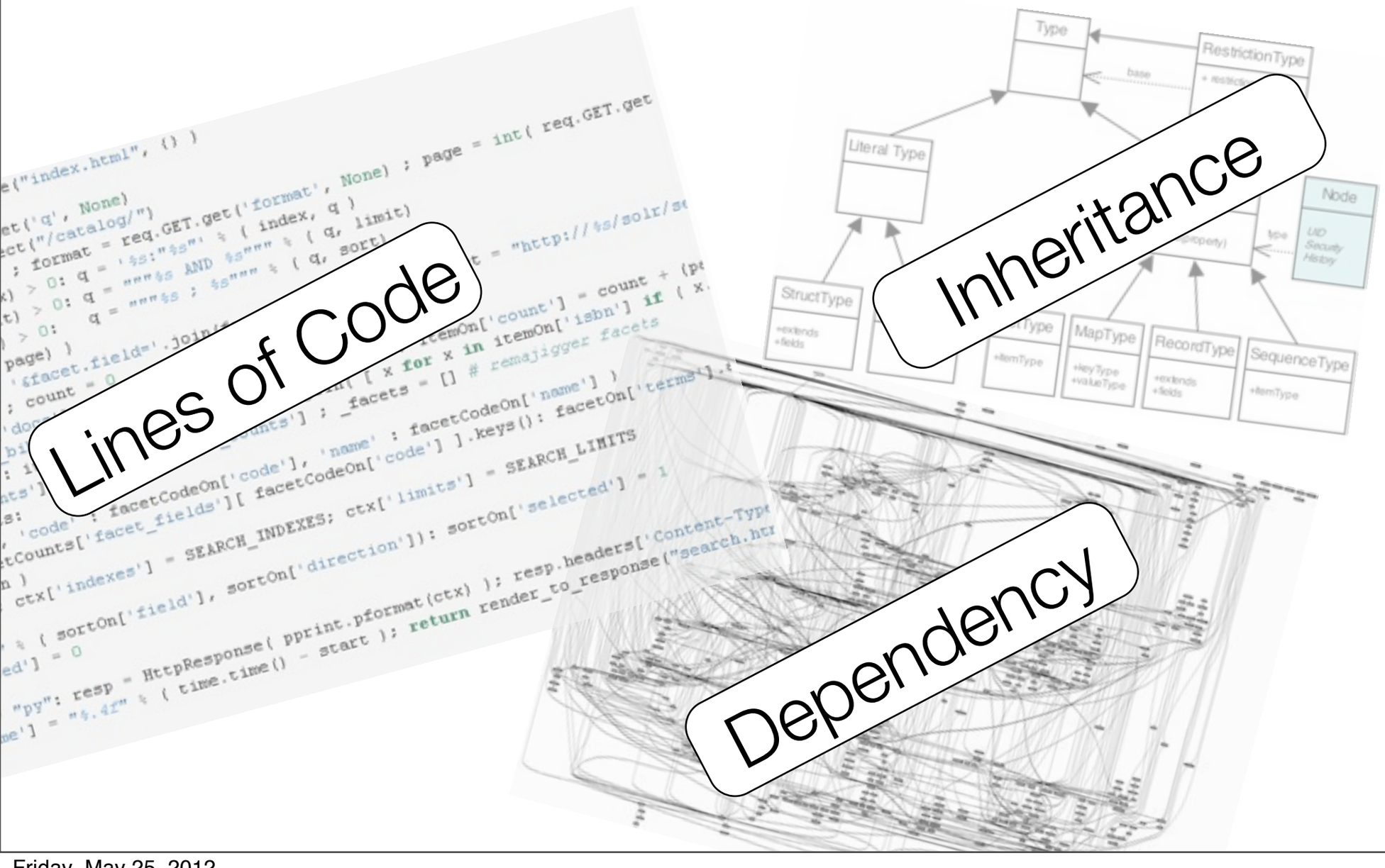
Code Metrics

- Directly calculated on the code itself
- Measure size and complexity of the code



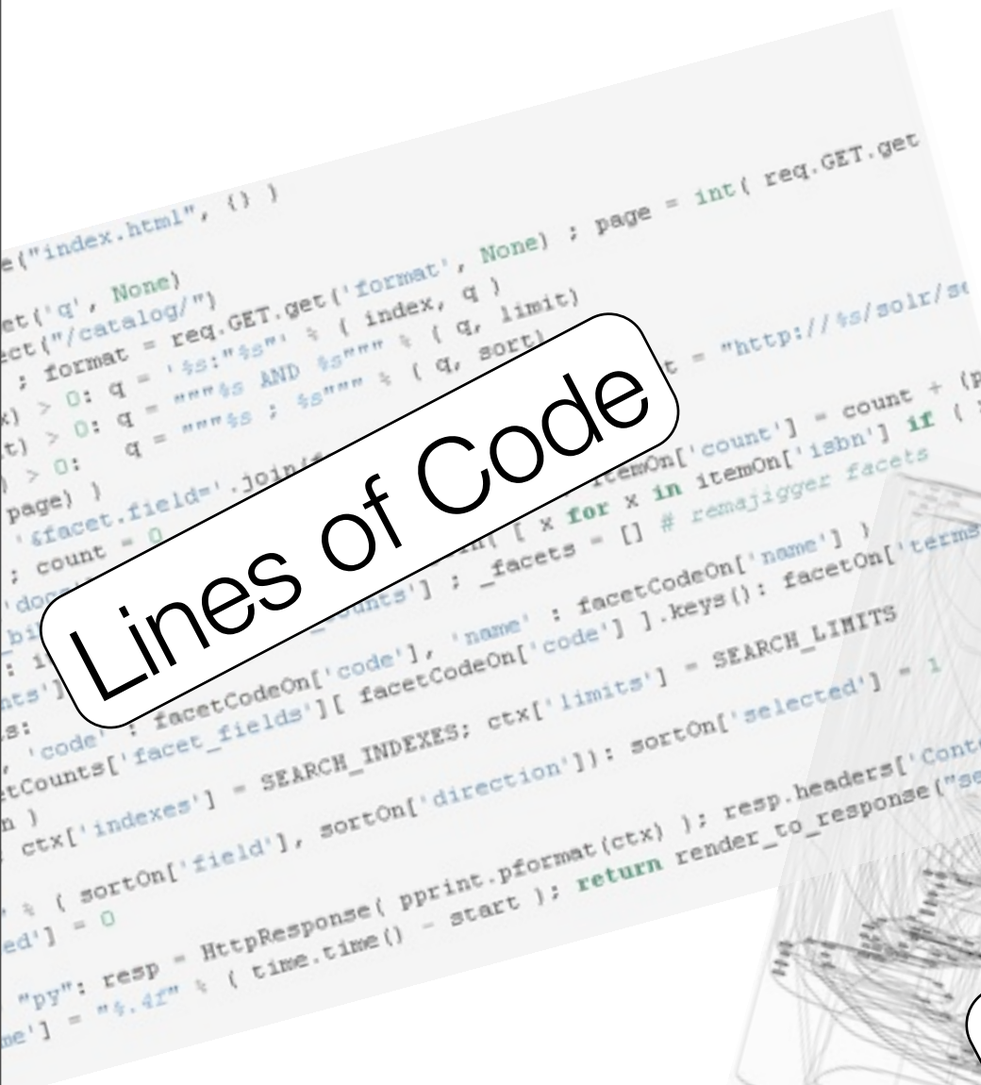
Code Metrics

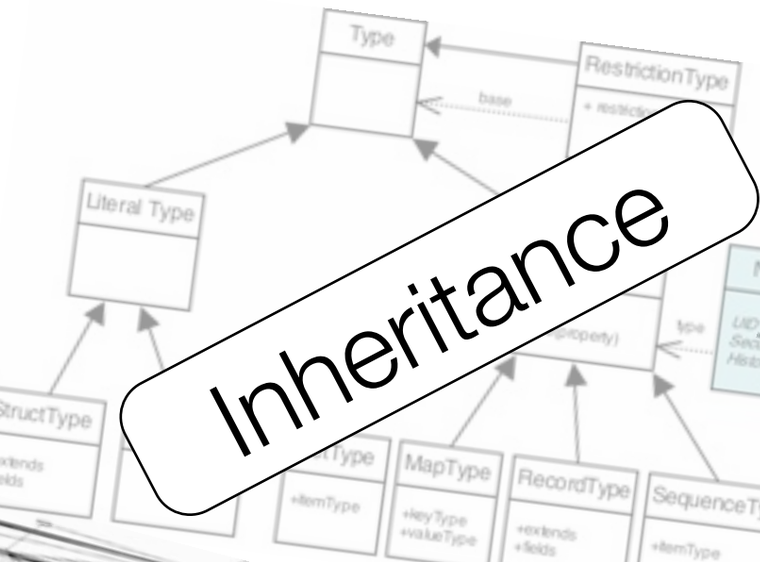
- Directly calculated on the code itself
- Measure size and complexity of the code

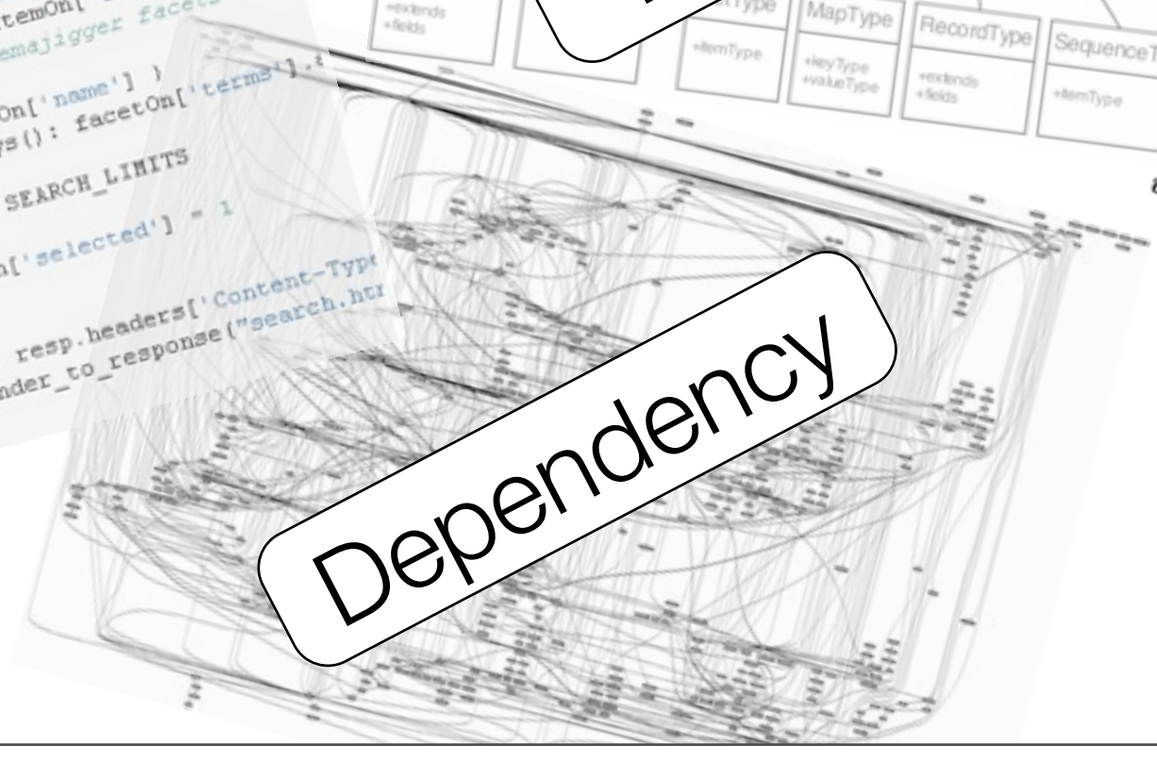


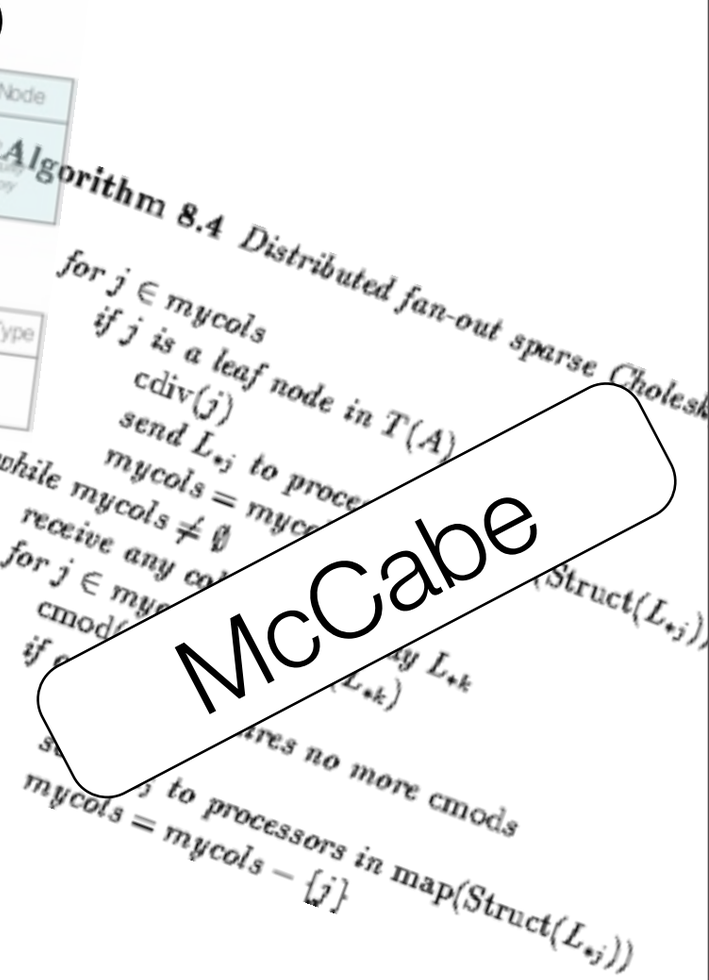
Code Metrics

- Directly calculated on the code itself
- Measure size and complexity of the code





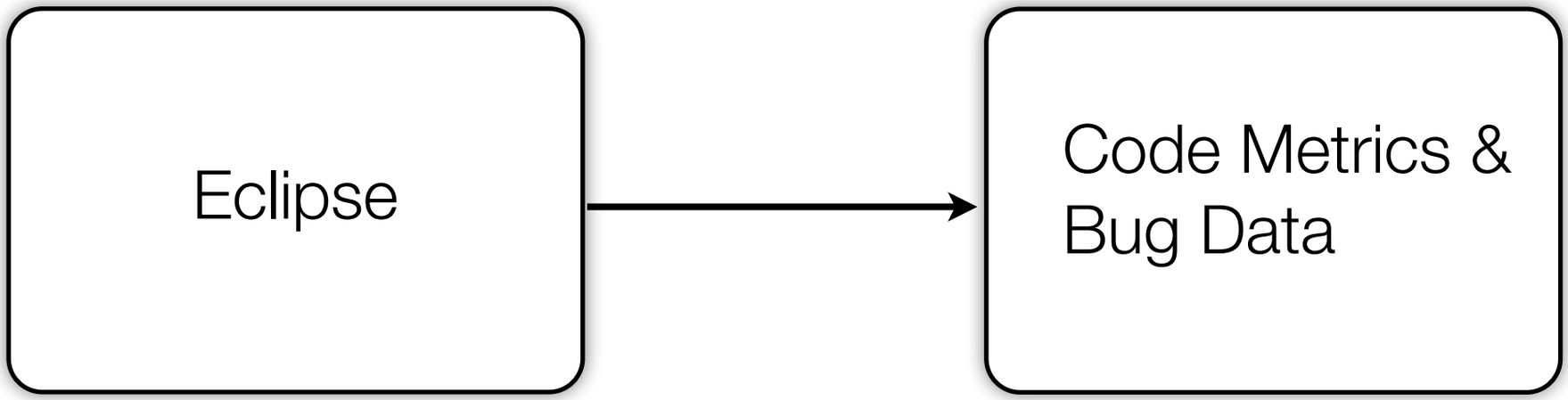




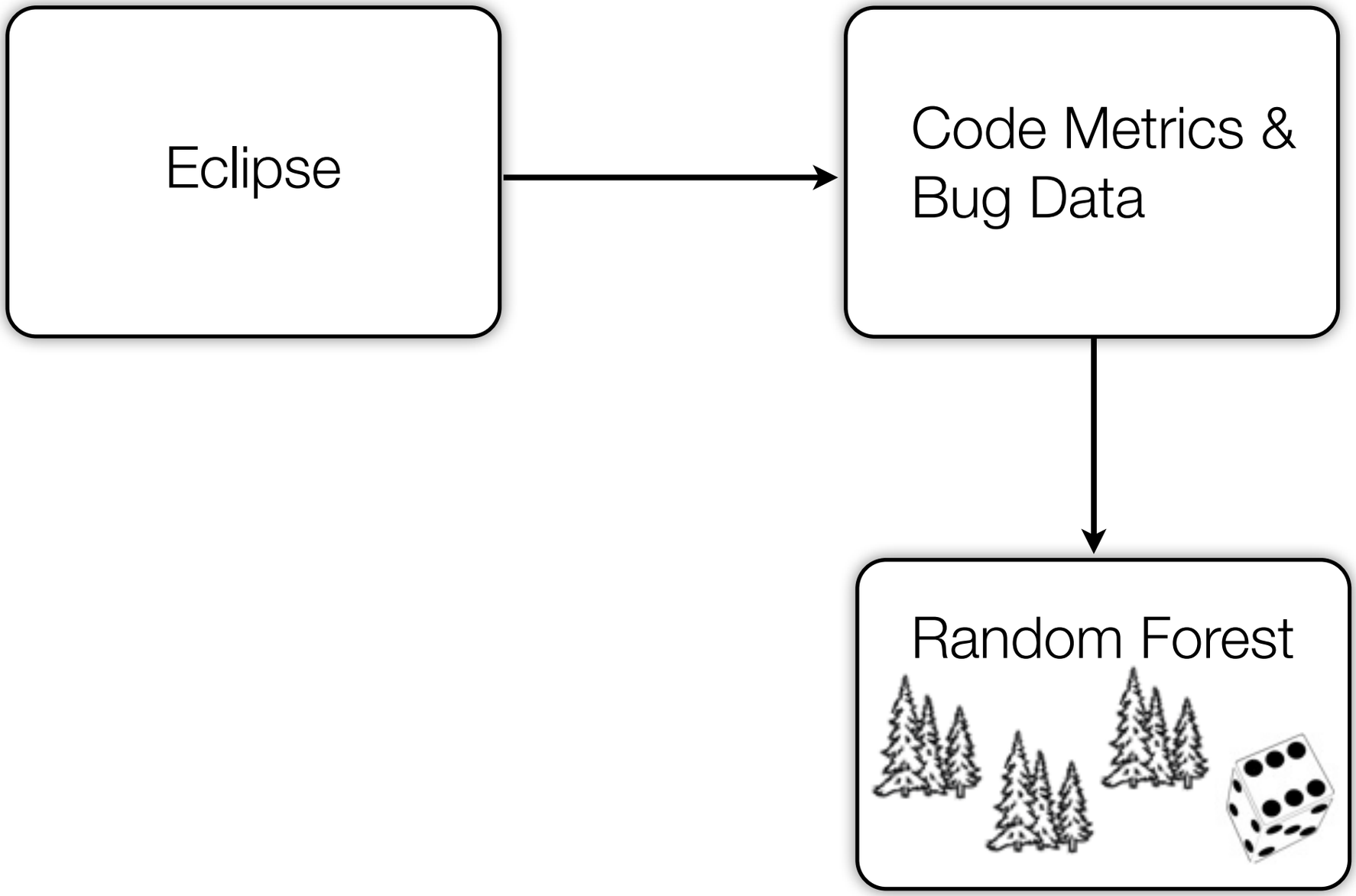
Bug Prediction Setup

Eclipse

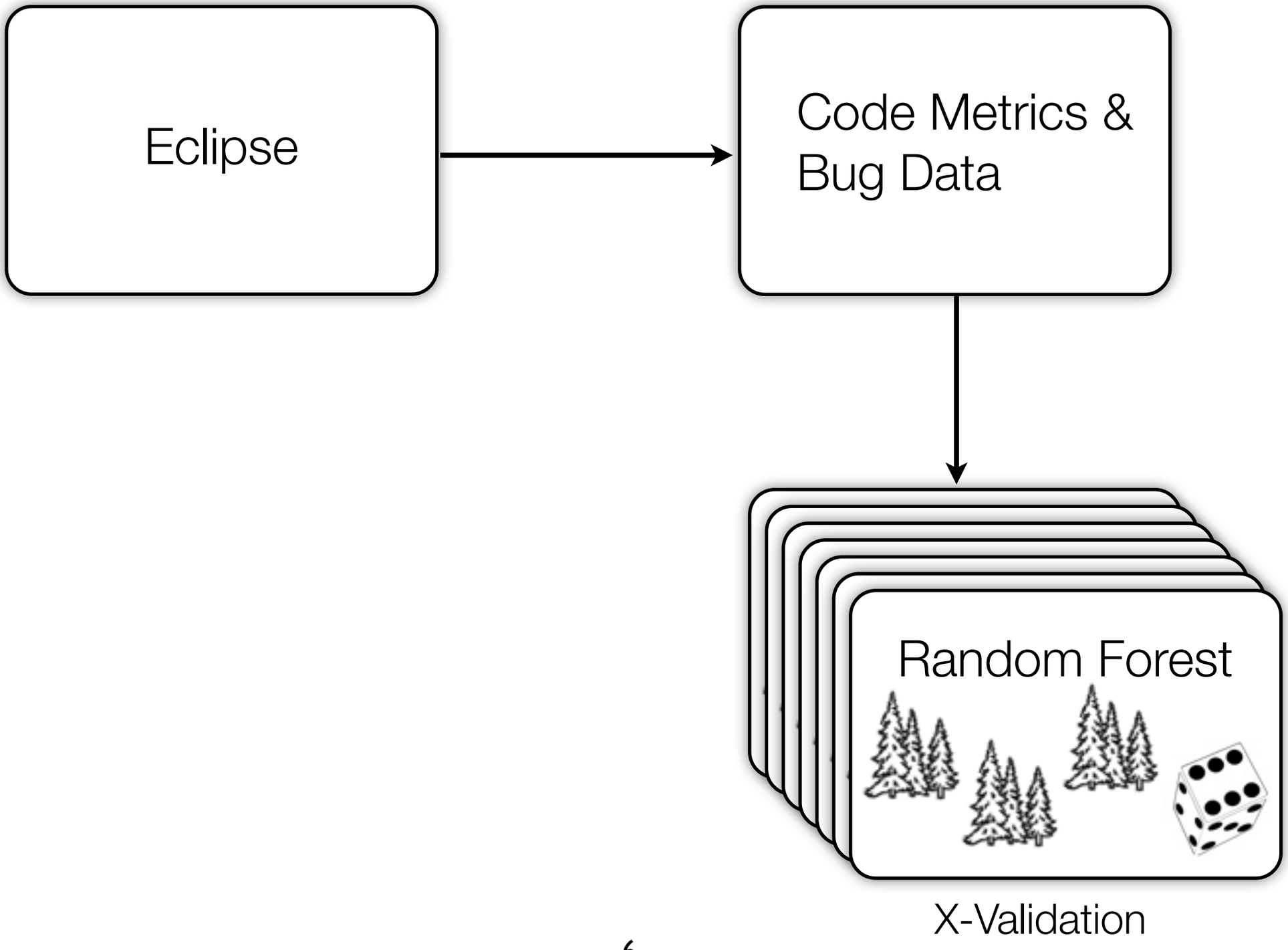
Bug Prediction Setup



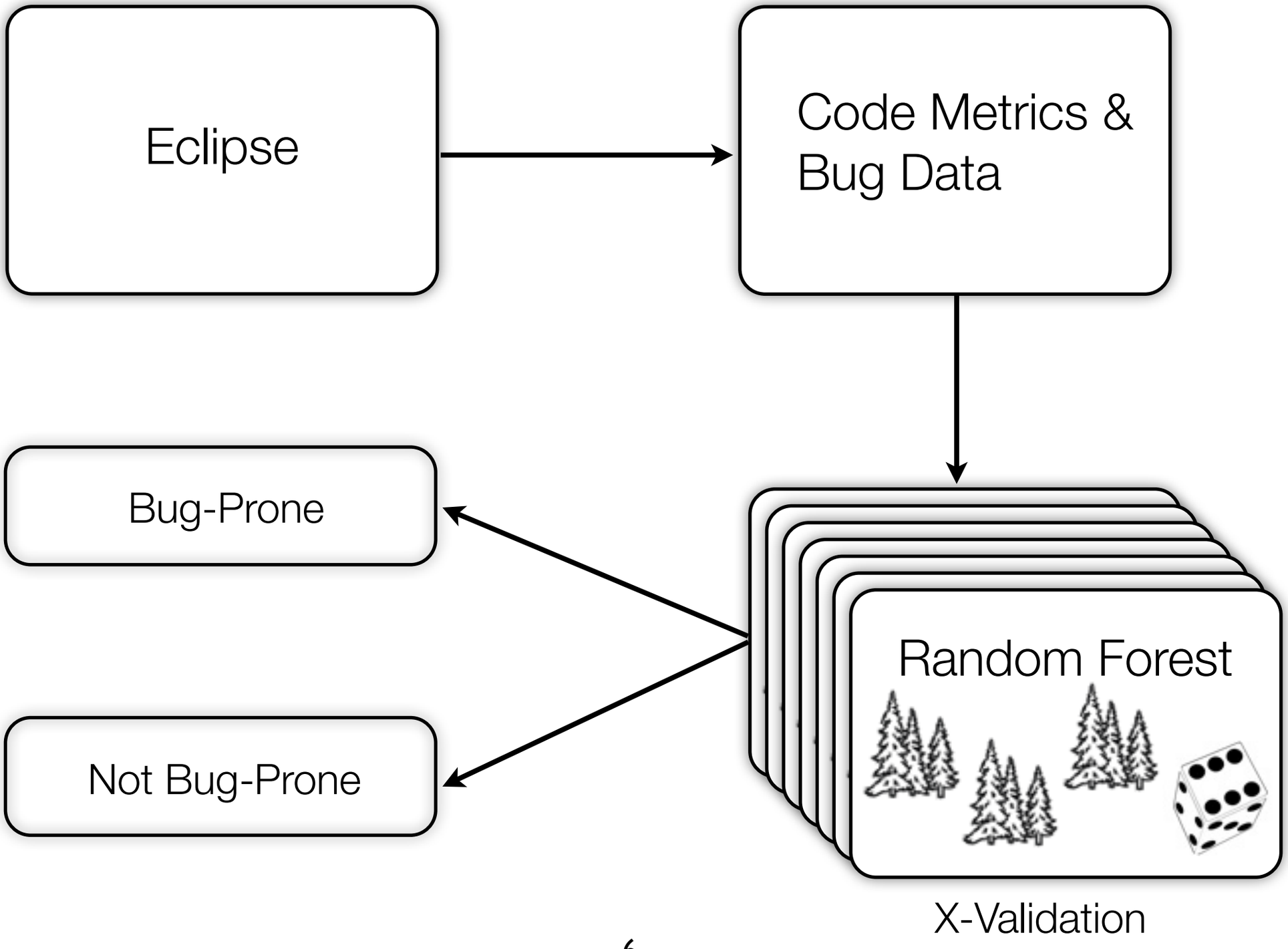
Bug Prediction Setup



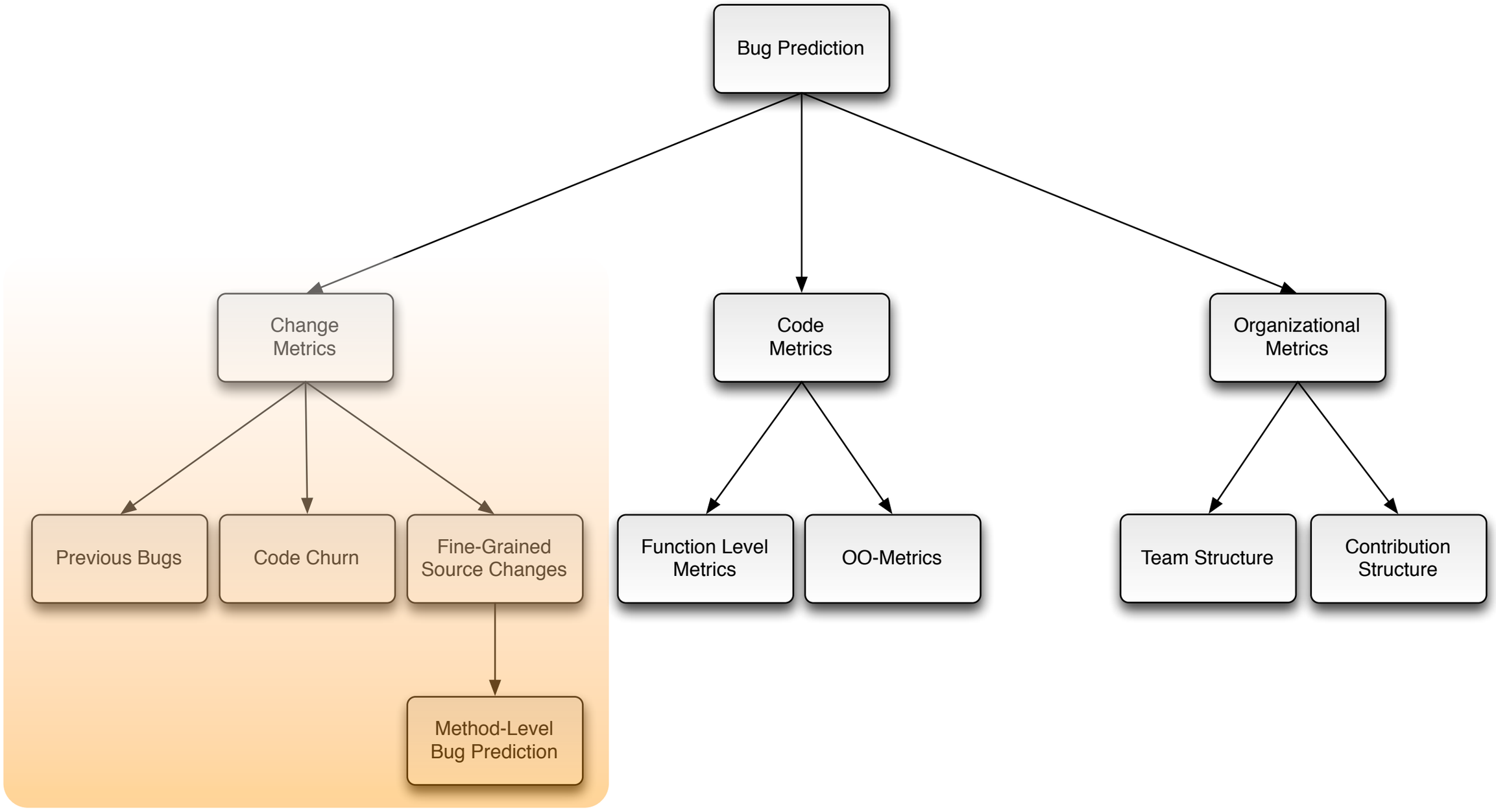
Bug Prediction Setup



Bug Prediction Setup



Bug Prediction Models



Code Changes

Revisions

Commits to version control systems

Coarse-grained

Files are the units of change

Revisions

There is more than just a file revision

```
    private IStructureComparator fStructureComparator;

    public boolean setInput(ITypedElement newInput, boolean force) {
        boolean changed = false;
        if (force || newInput != fInput) {
            removeDocumentRangeUpdaters();
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangedListener);
            fInput= newInput;
            if (fInput == null) {
                if (fStructureComparator instanceof IDisposable) {
                    IDisposable disposable = (IDisposable) fStructureComparator;
                    disposable.dispose();
                }
                fStructureComparator= null;
            } else {
                refresh();
                changed= true;
            }
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangedListener);
        }
        return changed;
    }

    /**
     * Remove any document range updaters that were registered against the document.
     */
    private void removeDocumentRangeUpdaters() {
        if (fStructureComparator instanceof IDocumentRange) {
            IDocument doc = ((IDocumentRange) fStructureComparator).getDocument();
            try {
```

```
    private ITypedElement fInput;
    private IStructureComparator fStructureComparator;

    public boolean setInput(ITypedElement newInput, boolean force) {
        boolean changed = false;
        if (force || newInput != fInput) {
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangedLister
            fInput= newInput;
            if (fInput != null) {
                refresh();
                changed= true;
            } else {
                if (fStructureComparator instanceof IDisposable) {
                    IDisposable disposable = (IDisposable) fStructureComparator;
                    disposable.dispose();
                }
                fStructureComparator= null;
            }
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangedListener)
        }
        return changed;
    }

    public IStructureComparator getStructureComparator() {
        return fStructureComparator;
    }

    public void refresh() {
        IStructureComparator oldComparator = fStructureComparator;
        fStructureComparator= createStructure();
```

Revisions

There is more than just a file revision

```
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        removeDocumentRangeUpdaters();
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangedListener);
        fInput= newInput;
        if (fInput == null) {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fS
                disposable.dispose();
            }
            fStructureComparator= null;
        } else {
            refresh();
            changed= true;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangedListener);
    }
    return changed;
}

/**
 * Remove any document range updaters that were registered against the document.
 */
private void removeDocumentRangeUpdaters() {
    if (fStructureComparator instanceof IDocumentRange) {
        IDocument doc = ((IDocumentRange) fStructureComparator).getDocument();
        try {
```

```
private ITypedElement fInput;
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangedLister
        fInput= newInput;
        if (fInput != null) {
            refresh();
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator= null;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangedListener)
    }
    return changed;
}

public IStructureComparator getStructureComparator() {
    return fStructureComparator;
}

public void refresh() {
    IStructureComparator oldComparator = fStructureComparator;
    fStructureComparator= createStructure();
```

```
removeDocumentRangeUpdaters();
```

Revisions

There is more than just a file revision

```
    private IStructureComparator fStructureComparator;

    public boolean setInput(ITypedElement newInput, boolean force) {
        boolean changed = false;
        if (force || newInput != fInput) {
            removeDocumentRangeUpdaters();
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangedListener);
            fInput= newInput;
            if (fInput == null) {
                if (fStructureComparator instanceof IDisposable) {
                    IDisposable disposable = (IDisposable) fStructureComparator;
                    disposable.dispose();
                }
                fStructureComparator= null;
            } else {
                refresh();
                changed= true;
            }
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangedListener);
        }
        return changed;
    }

    /**
     * Remove any document range updaters that were registered against the document.
     */
    private void removeDocumentRangeUpdaters() {
        if (fStructureComparator instanceof IDocumentRange) {
            IDocument doc = ((IDocumentRange) fStructureComparator).getDocument();
            try {
```

```
    private ITypedElement fInput;
    private IStructureComparator fStructureComparator;

    public boolean setInput(ITypedElement newInput, boolean force) {
        boolean changed = false;
        if (force || newInput != fInput) {
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangedLister
            fInput= newInput;
            if (fInput != null) {
                refresh();
                changed= true;
            } else {
                if (fStructureComparator instanceof IDisposable) {
                    IDisposable disposable = (IDisposable) fStructureComparator;
                    disposable.dispose();
                }
                fStructureComparator= null;
            }
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangedListener)
        }
        return changed;
    }

    public IStructureComparator getStructureComparator() {
        return fStructureComparator;
    }

    public void refresh() {
        IStructureComparator oldComparator = fStructureComparator;
        fStructureComparator= createStructure();
```

Revisions

There is more than just a file revision

```
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        removeDocumentRangeUpdaters();
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangedListener);
        fInput= newInput;
        if (fInput == null) {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator= null;
        } else {
            refresh();
            changed= true;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentC
    }
    return changed;
}

/**
 * Remove any document range updaters that were registered against the document
 */
private void removeDocumentRangeUpdaters() {
    if (fStructureComparator instanceof IDocumentRange) {
        IDocument doc = ((IDocumentRange) fStructureComparator).getDocument();
        try {
```

```
fInput= newInput;
if (fInput == null) {
    if (fStructureComp
        IDisposable di
        disposable.dis
    }
    fStructureComparat
} else {
    refresh();
    changed= true;
}
```

```
private ITypedElement fInput;
private IStructureComparator fStructureComparator;

public boolean setInput(ITypedElement newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangedLister
        fInput= newInput;
        if (fInput != null) {
            refresh();
            changed= true;
        } else {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable) fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator= null;
        }
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangedListener)

urn changed;

IStructureComparator getStructureComparator() {
urn fStructureComparator;

public void refresh() {
    IStructureComparator oldComparator = fStructureComparator;
    fStructureComparator= createStructure();
```

Revisions

There is more than just a file revision

```
    private IStructureComparator fStructureComparator;

    public boolean setInput(ITypedElement newInput, boolean force) {
        boolean changed = false;
        if (force || newInput != fInput) {
            removeDocumentRangeUpdaters();
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangedListener);
            fInput= newInput;
            if (fInput == null) {
                if (fStructureComparator instanceof IDisposable) {
                    IDisposable disposable = (IDisposable) fStructureComparator;
                    disposable.dispose();
                }
                fStructureComparator= null;
            } else {
                refresh();
                changed= true;
            }
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangedListener);
        }
        return changed;
    }

    /**
     * Remove any document range updaters that were registered against the document.
     */
    private void removeDocumentRangeUpdaters() {
        if (fStructureComparator instanceof IDocumentRange) {
            IDocument doc = ((IDocumentRange) fStructureComparator).getDocument();
            try {
```

```
    private ITypedElement fInput;
    private IStructureComparator fStructureComparator;

    public boolean setInput(ITypedElement newInput, boolean force) {
        boolean changed = false;
        if (force || newInput != fInput) {
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangedLister
            fInput= newInput;
            if (fInput != null) {
                refresh();
                changed= true;
            } else {
                if (fStructureComparator instanceof IDisposable) {
                    IDisposable disposable = (IDisposable) fStructureComparator;
                    disposable.dispose();
                }
                fStructureComparator= null;
            }
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangedListener)
        }
        return changed;
    }

    public IStructureComparator getStructureComparator() {
        return fStructureComparator;
    }

    public void refresh() {
        IStructureComparator oldComparator = fStructureComparator;
        fStructureComparator= createStructure();
```

Revisions

There is more than just a file revision

```
    private IStructureComparator fStructureComparator;

    public boolean setInput(ITypedElement newInput, boolean force) {
        boolean changed = false;
        if (force || newInput != fInput) {
            removeDocumentRangeUpdaters();
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangedListener);
            fInput= newInput;
            if (fInput == null) {
                if (fStructureComparator instanceof IDisposable) {
                    IDisposable disposable = (IDisposable) fStructureComparator;
                    disposable.dispose();
                }
                fStructureComparator= null;
            } else {
                refresh();
                changed= true;
            }
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangedListener);
        }
        return changed;
    }

    /**
     * Remove any document range updaters that were registered against the document.
     */
    private void removeDocumentRangeUpdaters() {
        if (fStructureComp
            IDocument doc =
            try {
```

```
    private ITypedElement fInput;
    private IStructureComparator fStructureComparator;

    public boolean setInput(ITypedElement newInput, boolean force) {
        boolean changed = false;
        if (force || newInput != fInput) {
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangedLister
            fInput= newInput;
            if (fInput != null) {
                refresh();
                changed= true;
            } else {
                if (fStructureComparator instanceof IDisposable) {
                    IDisposable disposable = (IDisposable) fStructureComparator;
                    disposable.dispose();
                }
                fStructureComparator= null;
            }
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangedListener)
        }
        return changed;
    }

    public IStructureComparator getStructureComparator() {
        return fStructureComparator;
    }
```

```
private void removeDocumentRangeUpdaters() {
    if (fStructureComparator instanceof IDocumentRange) {
        IDocument doc = ((IDocumentRange) fStructureComparator).getDocument();
        try {
```

Revisions

There is more than just a file revision

```
    private IStructureComparator fStructureComparator;

    public boolean setInput(ITypedElement newInput, boolean force) {
        boolean changed = false;
        if (force || newInput != fInput) {
            removeDocumentRangeUpdaters();
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangedListener);
            fInput= newInput;
            if (fInput == null) {
                if (fStructureComparator instanceof IDisposable) {
                    IDisposable disposable = (IDisposable) fStructureComparator;
                    disposable.dispose();
                }
                fStructureComparator= null;
            } else {
                refresh();
                changed= true;
            }
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangedListener);
        }
        return changed;
    }

    /**
     * Remove any document range updaters that were registered against the document.
     */
    private void removeDocumentRangeUpdaters() {
        if (fStructureComparator instanceof IDocumentRange) {
            IDocument doc = ((IDocumentRange) fStructureComparator).getDocument();
            try {
```

```
    private ITypedElement fInput;
    private IStructureComparator fStructureComparator;

    public boolean setInput(ITypedElement newInput, boolean force) {
        boolean changed = false;
        if (force || newInput != fInput) {
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangedLister
            fInput= newInput;
            if (fInput != null) {
                refresh();
                changed= true;
            } else {
                if (fStructureComparator instanceof IDisposable) {
                    IDisposable disposable = (IDisposable) fStructureComparator;
                    disposable.dispose();
                }
                fStructureComparator= null;
            }
            if (fInput instanceof IContentChangeNotifier)
                ((IContentChangeNotifier)fInput).addContentChangeListener(fContentChangedListener)
        }
        return changed;
    }

    public IStructureComparator getStructureComparator() {
        return fStructureComparator;
    }

    public void refresh() {
        IStructureComparator oldComparator = fStructureComparator;
        fStructureComparator= createStructure();
```

Code Changes

Revisions

Commits to version control systems

Coarse-grained

Files are the units of change

Code Churn

Textual UnixDiff between 2 File Versions

Ignores the structure of code

No change type information

Includes textual changes

Code Churn

Does not reflect the type and the semantics of source code changes

```
/*******************************************************************************
 * Copyright (c) 2000, 2004 IBM Corporation and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 *
 * Contributors:
 *     IBM Corporation - initial API and implementation
 *******************************************************************************/
package org.eclipse.compare.structuremergeviewer;

import org.eclipse.swt.events.DisposeEvent;
import org.eclipse.swt.widgets.*;
import org.eclipse.jface.util.PropertyChangeEvent;

import org.eclipse.compare.*;
import org.eclipse.compare.internal.*;


/**
```

```
/*******************************************************************************
 * Copyright (c) 2000, 2004 IBM Corporation and others.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Common Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/cpl-v10.html
 *
 * Contributors:
 *     IBM Corporation - initial API and implementation
 *******************************************************************************/
package org.eclipse.compare.structuremergeviewer;

import org.eclipse.swt.events.DisposeEvent;
import org.eclipse.swt.widgets.*;
import org.eclipse.jface.util.PropertyChangeEvent;

import org.eclipse.compare.*;
import org.eclipse.compare.internal.*;


/**
```

Code Changes

Revisions

Commits to version control systems

Coarse-grained

Files are the units of change

Code Churn

Textual UnixDiff between 2 File Versions

Ignores the structure of code

No change type information

Includes textual changes

Fine-Grained Changes[1]

Compares 2 versions of the AST of source code

Very fine-grained

Change type information

Captures all changes

Code Changes

Revisions

Commits to version control systems

Coarse-grained

Files are the units of change

Code Churn

Textual UnixDiff between 2 File Versions

Ignores the structure of code

No change type information

Includes textual changes

Fine-Grained Changes[1]

Compares 2 versions of the AST of source code

Very fine-grained

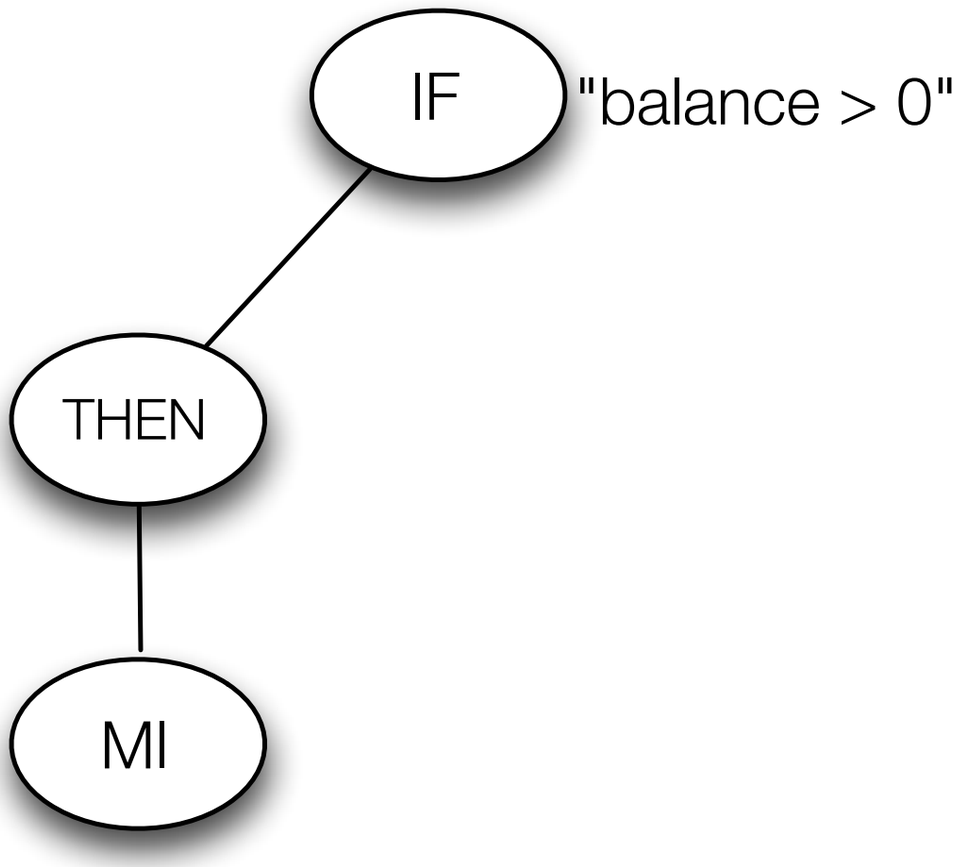
Change type information

Captures all changes

[1][Fluri et al. 2007, TSE]

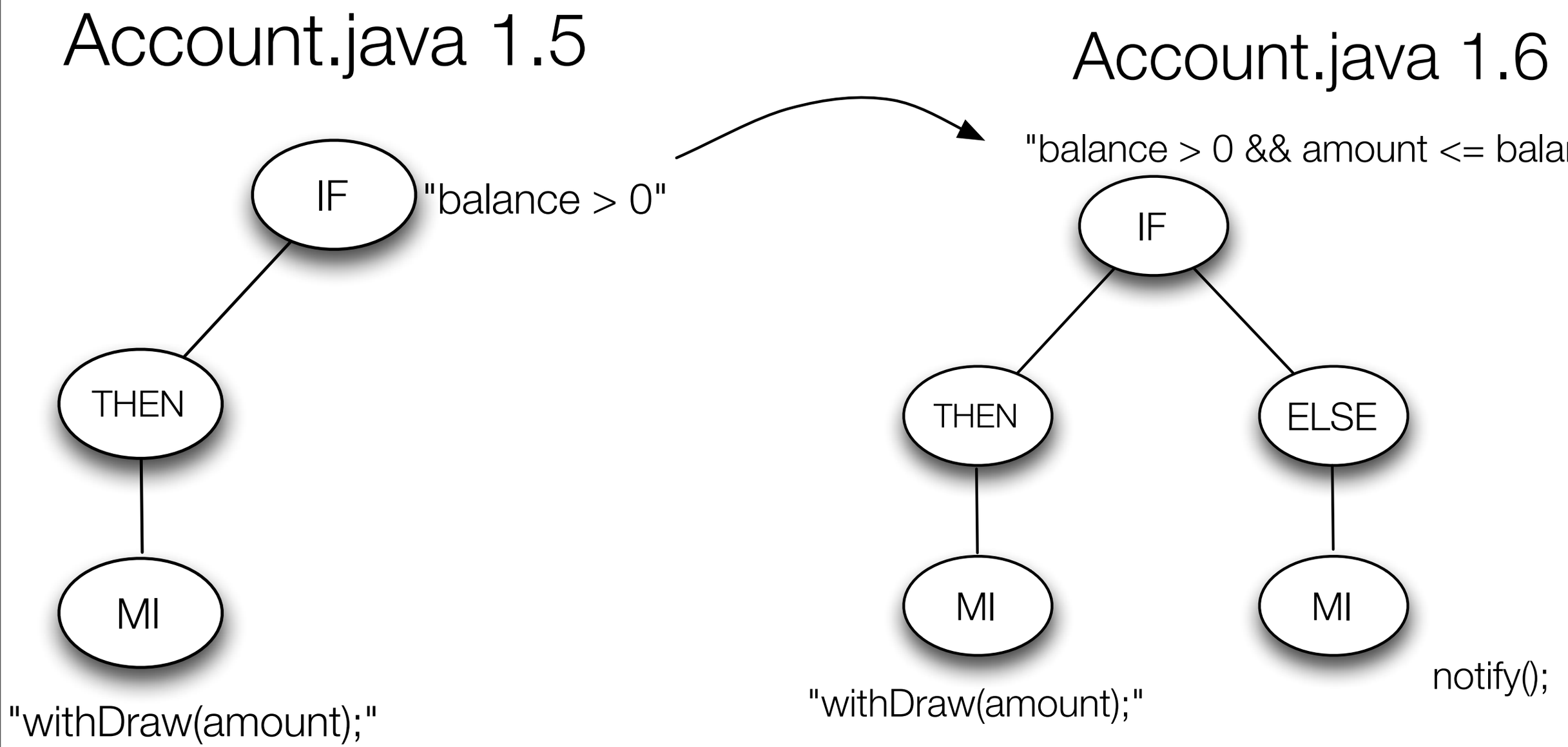
Fine-grained Changes

Account.java 1.5

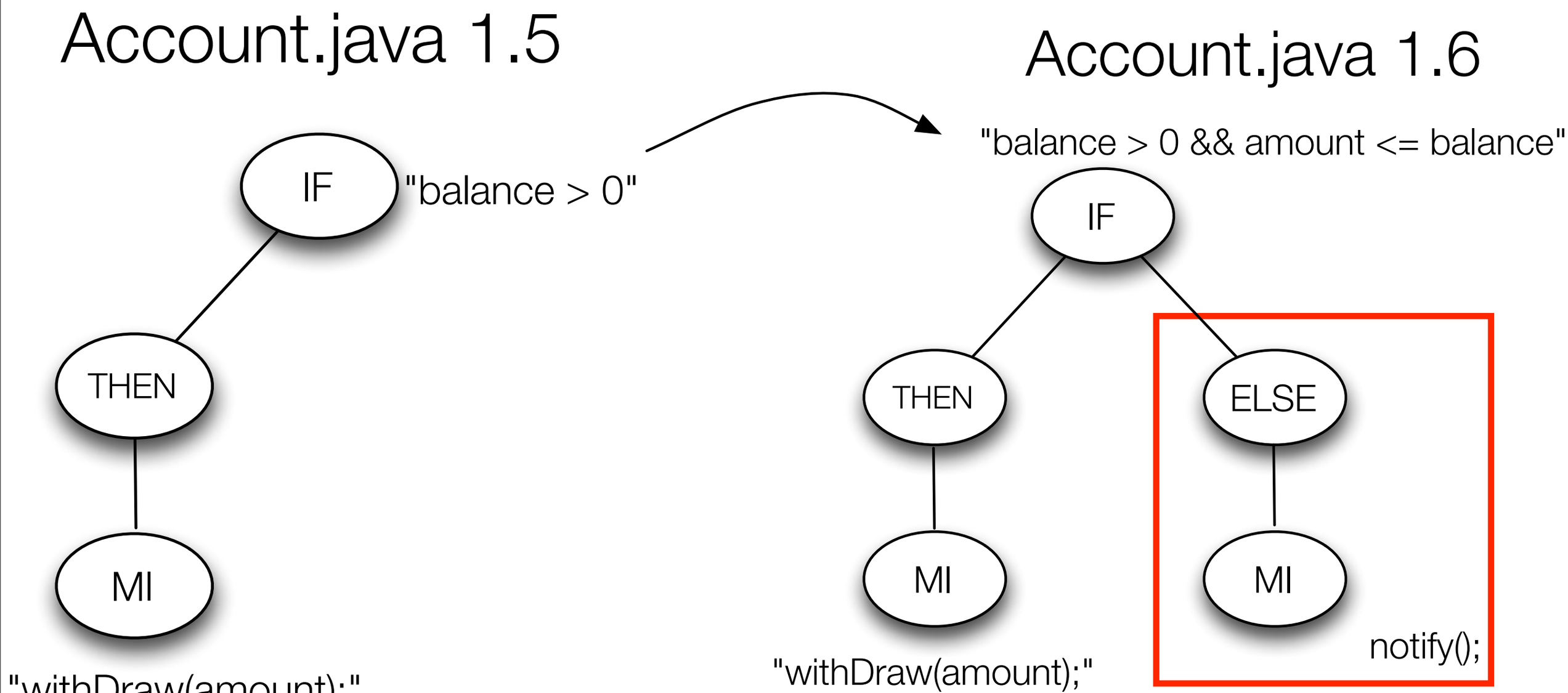


"withDraw(amount);"

Fine-grained Changes

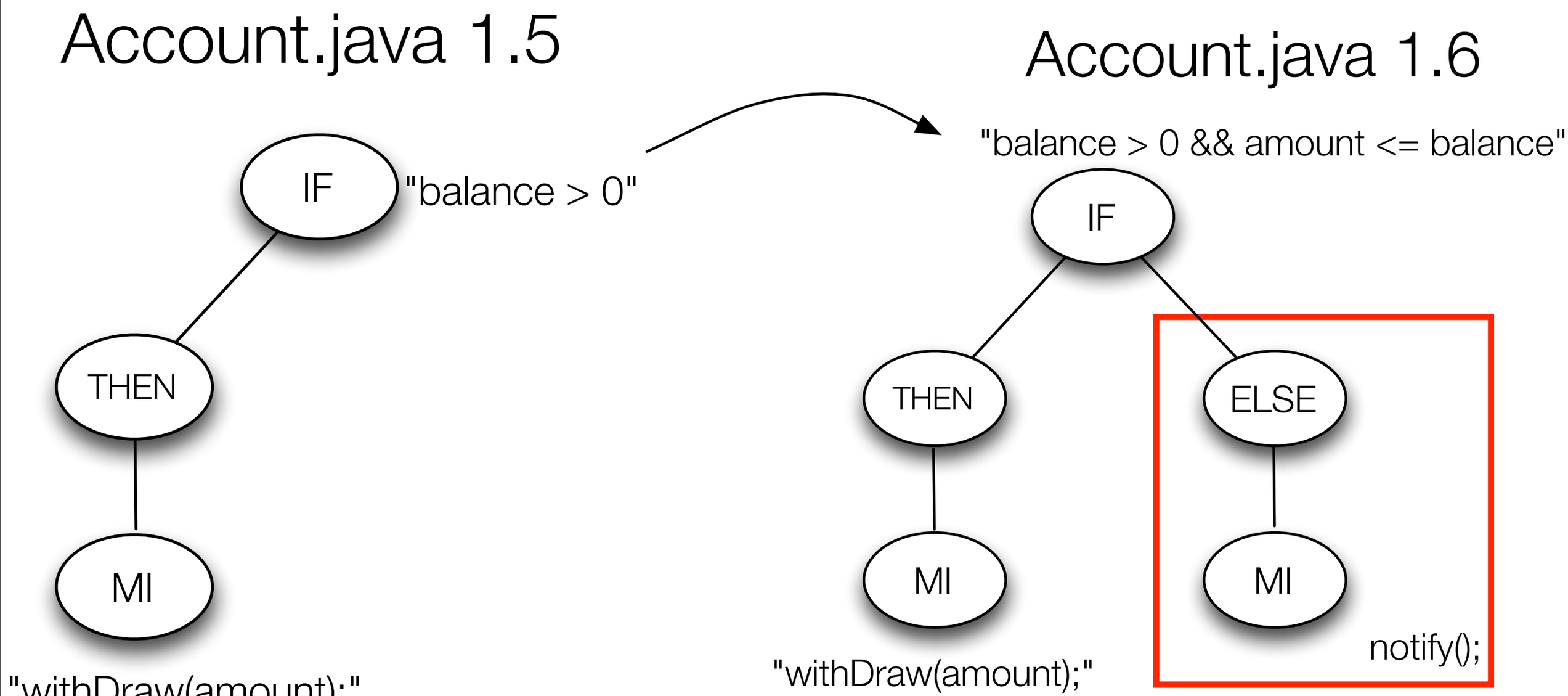


Fine-grained Changes



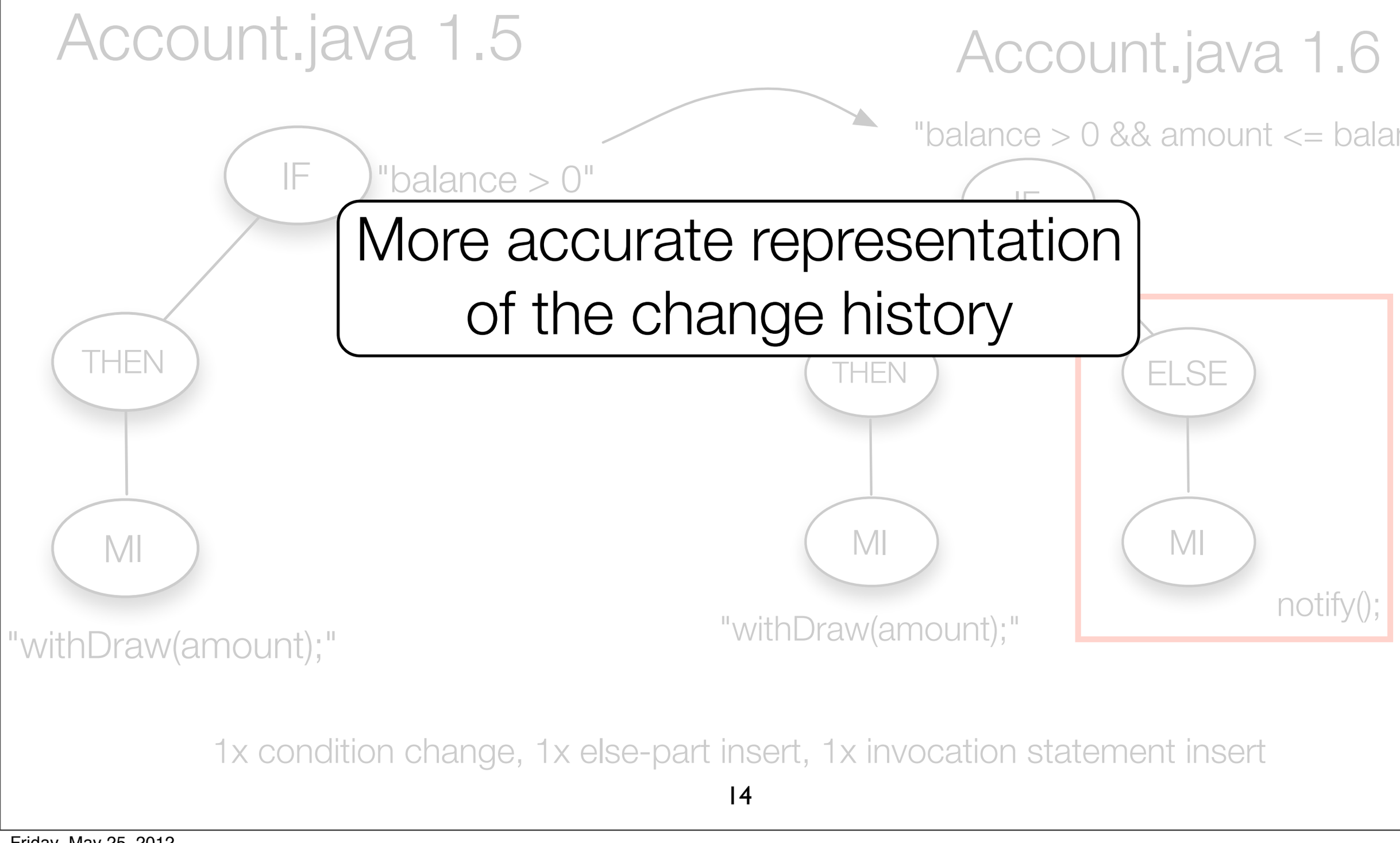
1x condition change, 1x else-part insert, 1x invocation statement insert

Fine-grained Changes

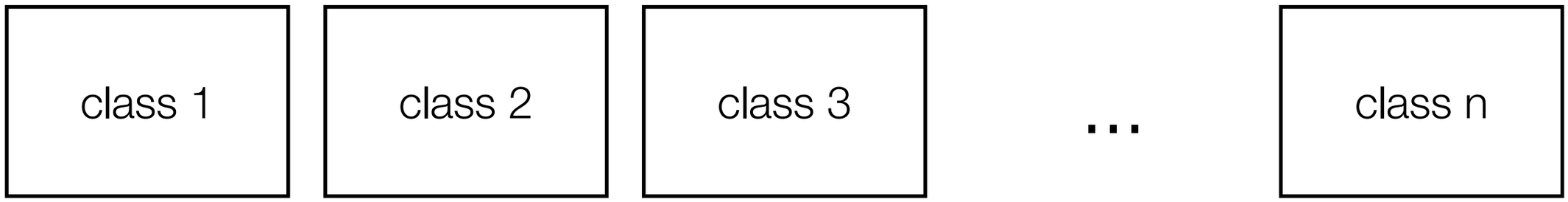


1x condition change, 1x else-part insert, 1x invocation statement insert

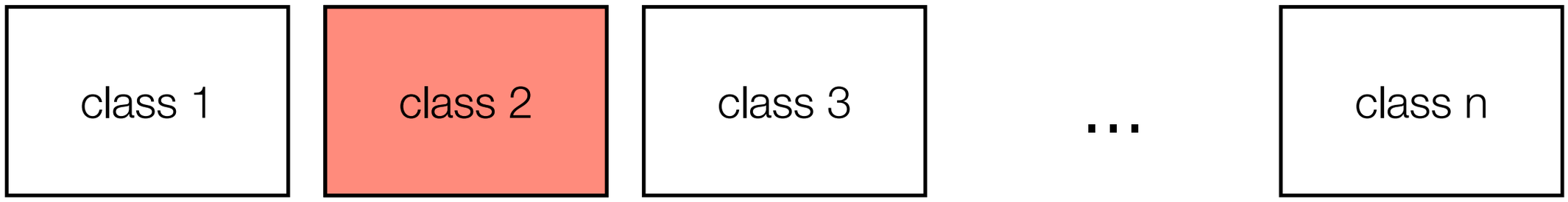
Fine-grained Changes



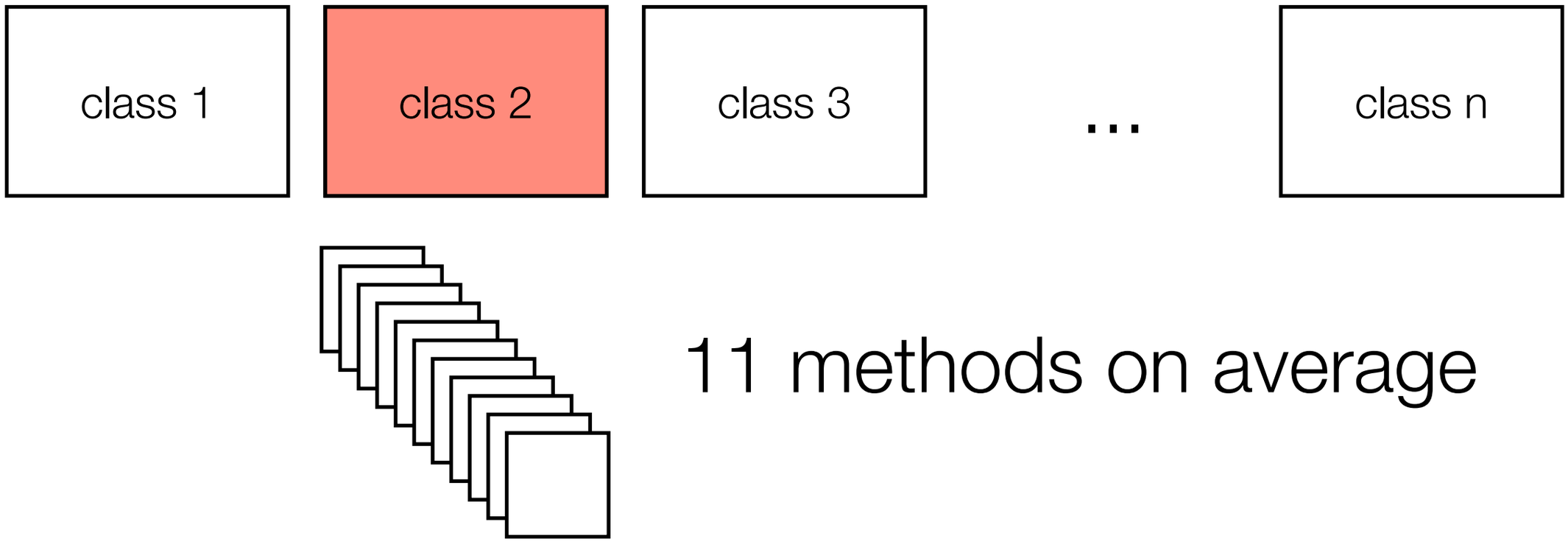
Method-Level Bug Prediction



Method-Level Bug Prediction

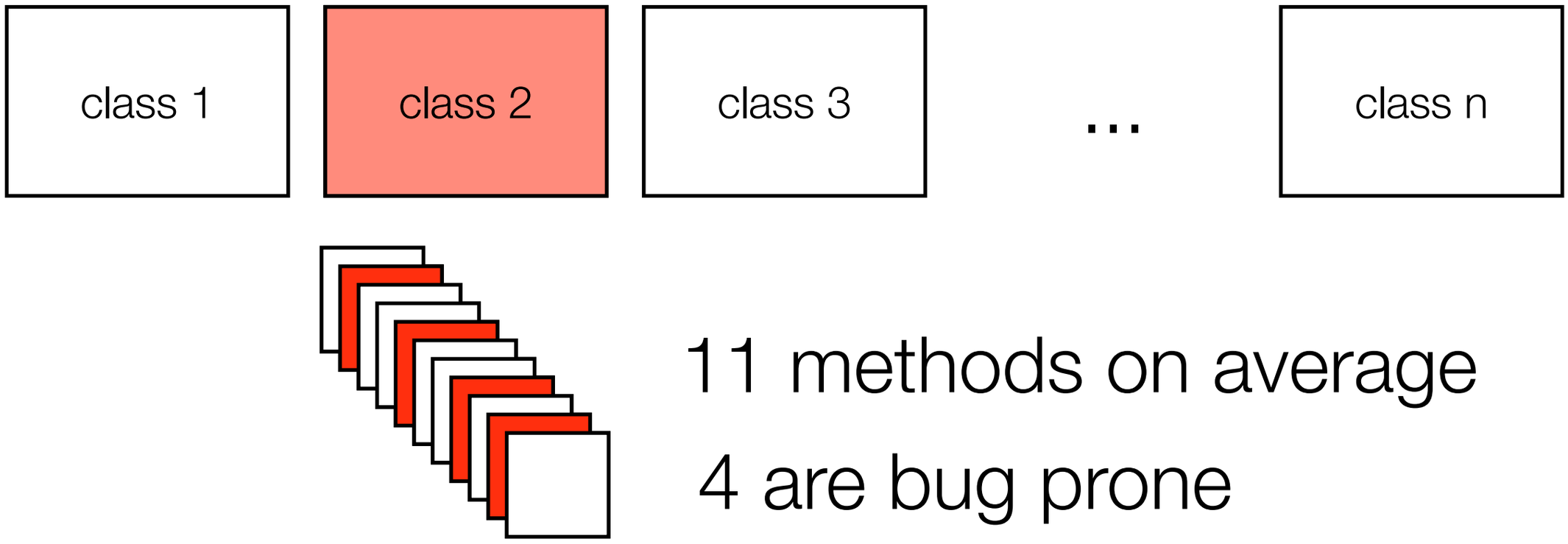


Method-Level Bug Prediction



11 methods on average

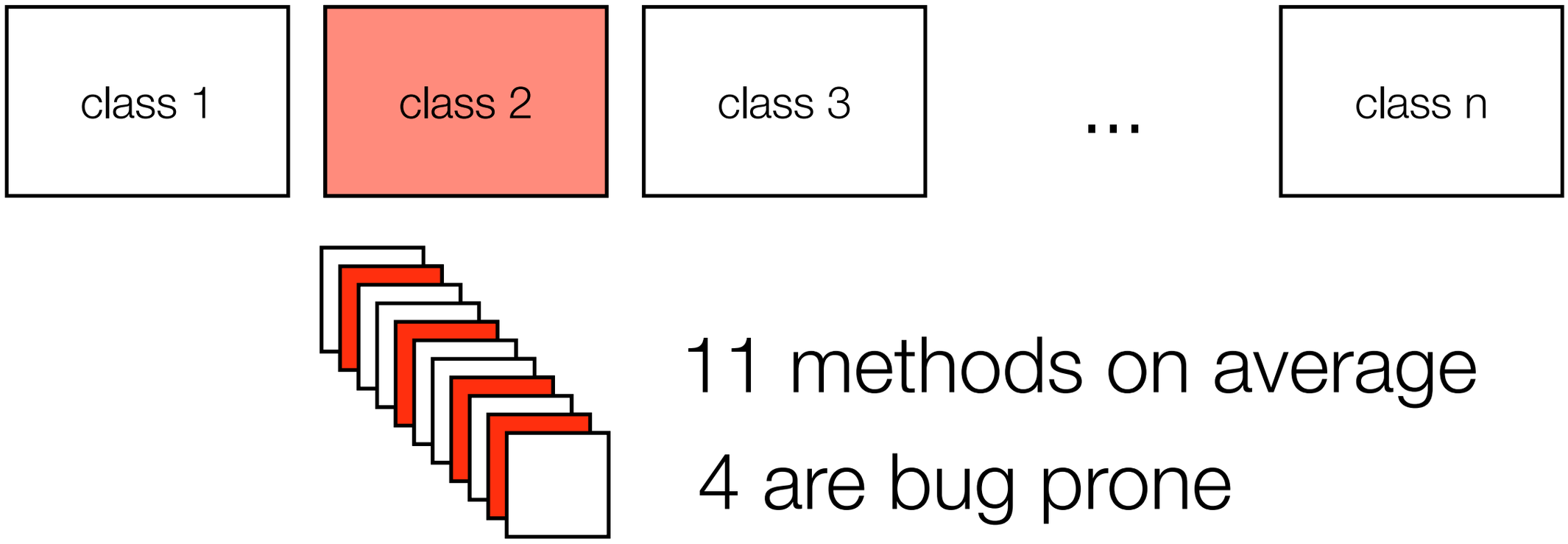
Method-Level Bug Prediction



11 methods on average

4 are bug prone

Method-Level Bug Prediction

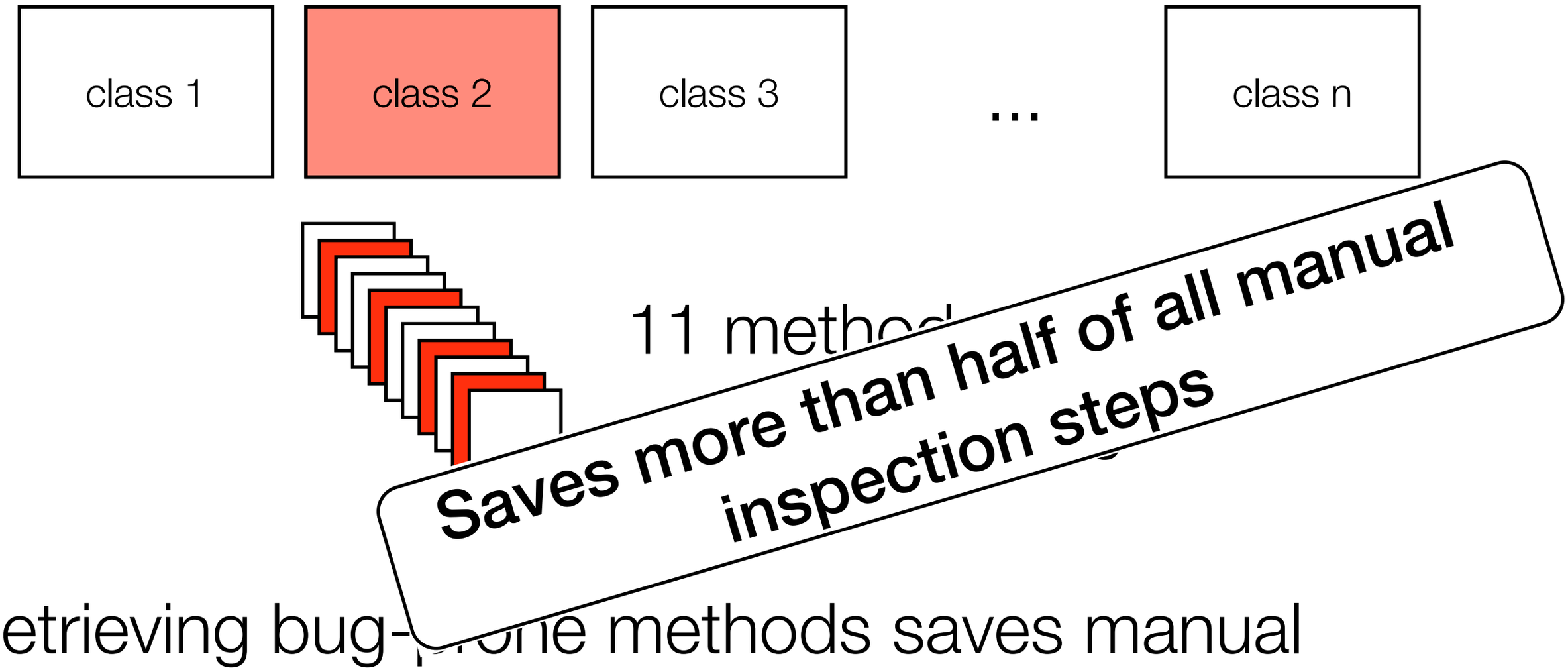


11 methods on average

4 are bug prone

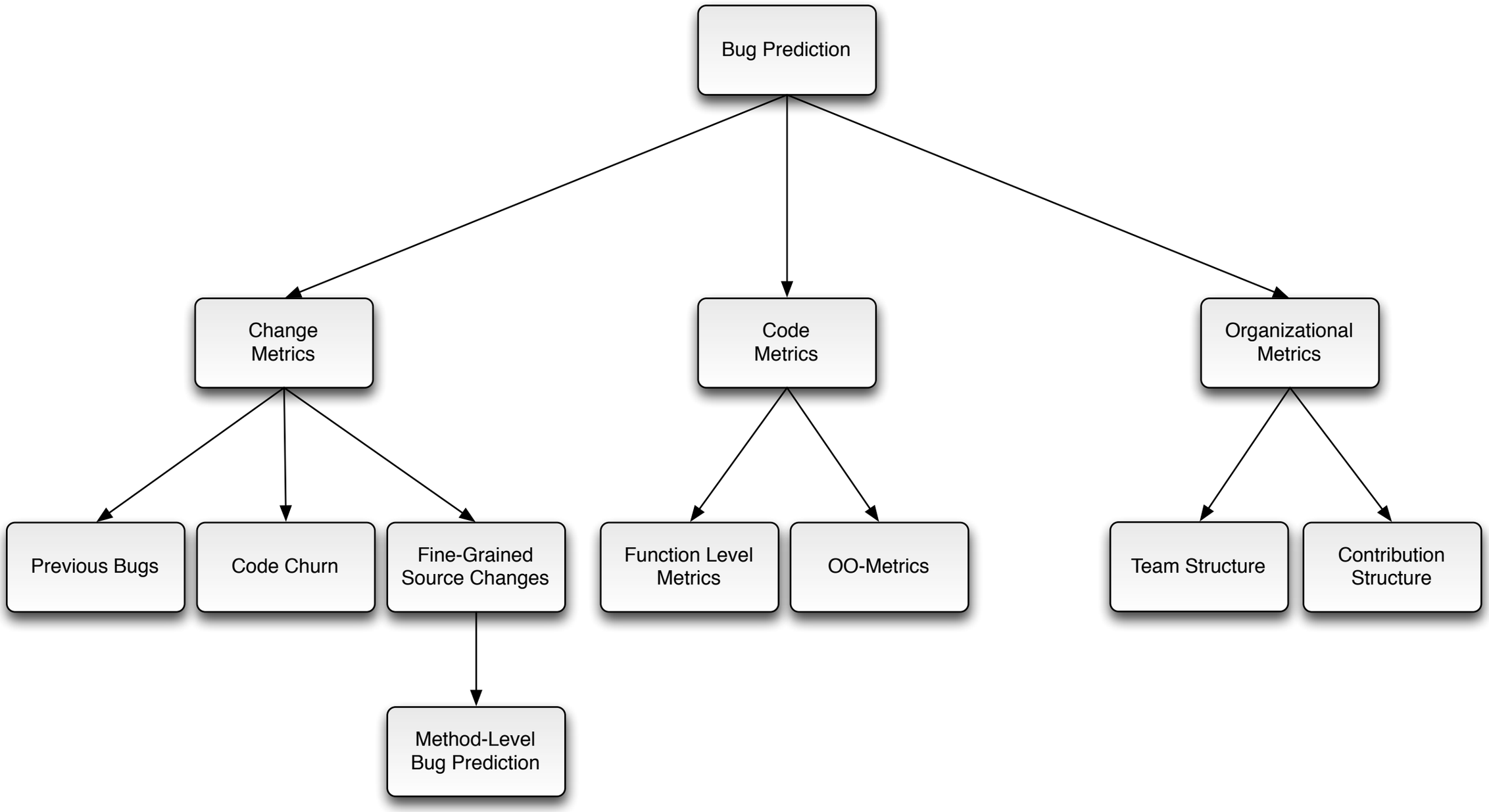
Retrieving bug-prone methods saves manual inspection steps and improves testing effort allocation

Method-Level Bug Prediction

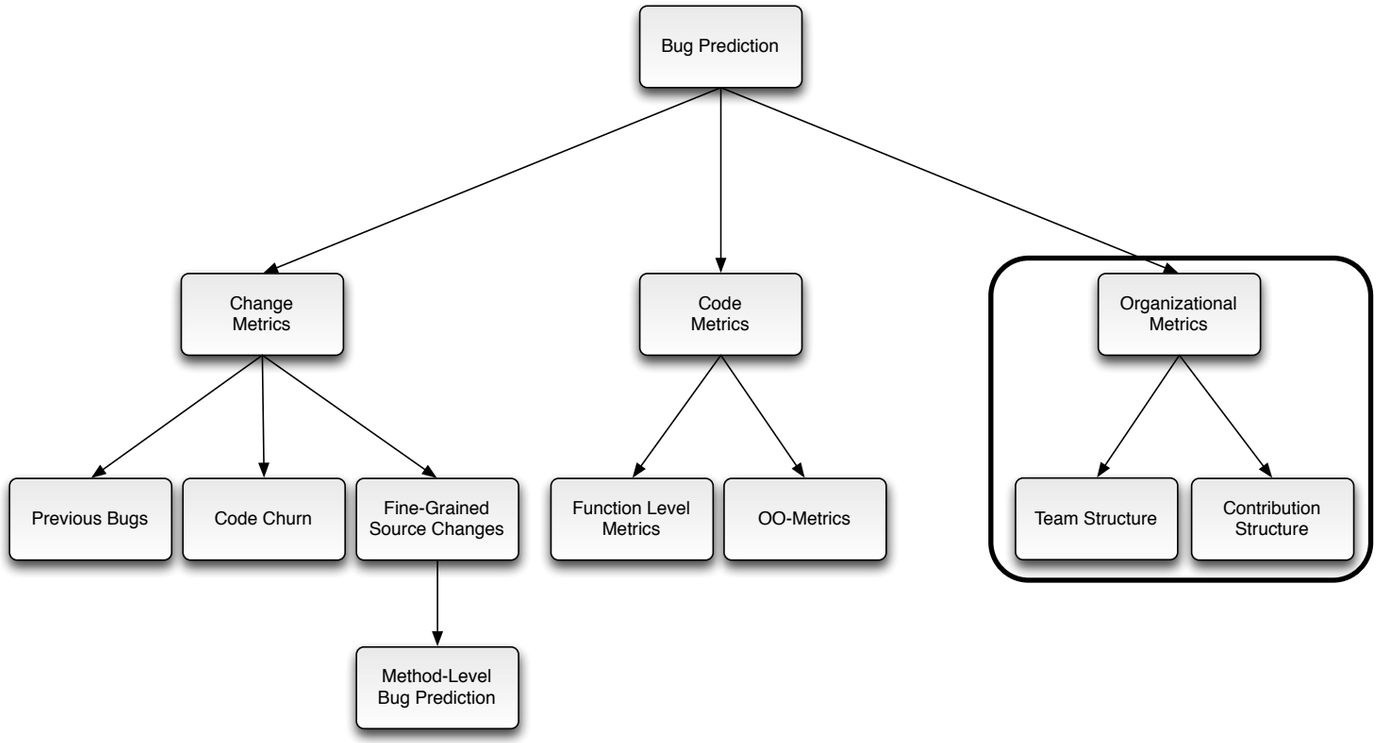


Retrieving bug-prone methods saves manual inspection steps and improves testing effort allocation

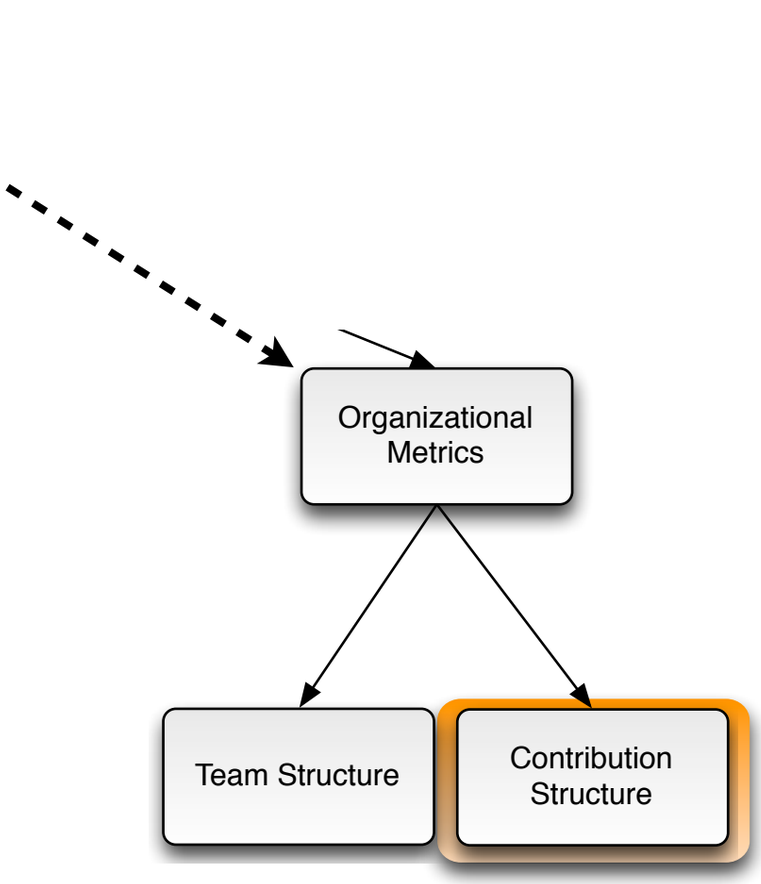
Bug Prediction Models



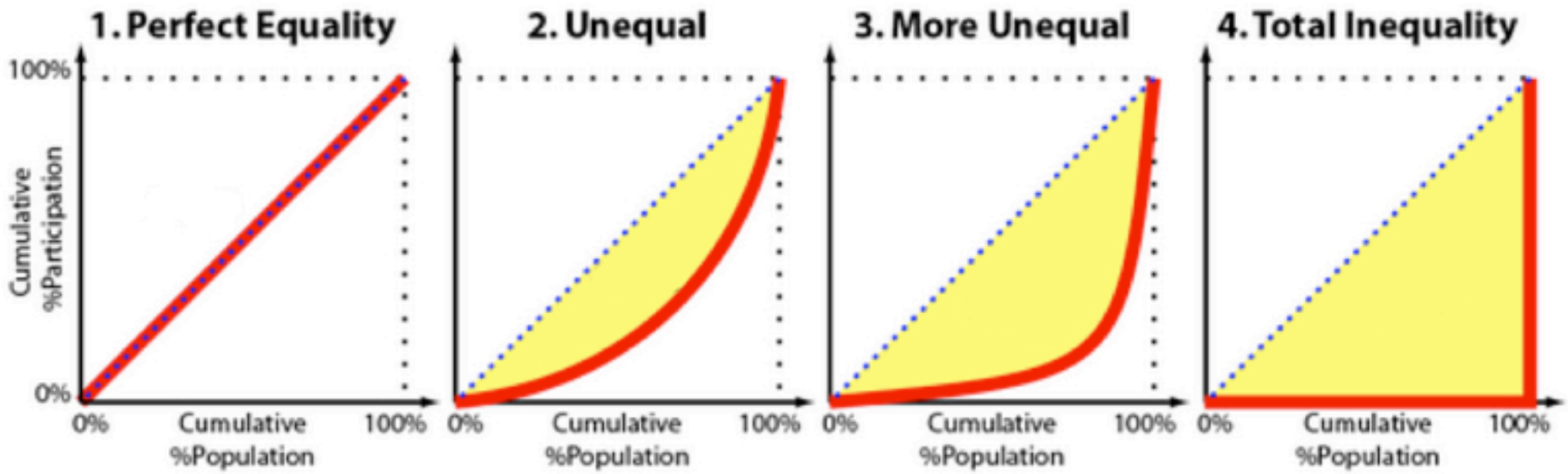
Bug Prediction Models



Using the Gini Coefficient for Bug Prediction

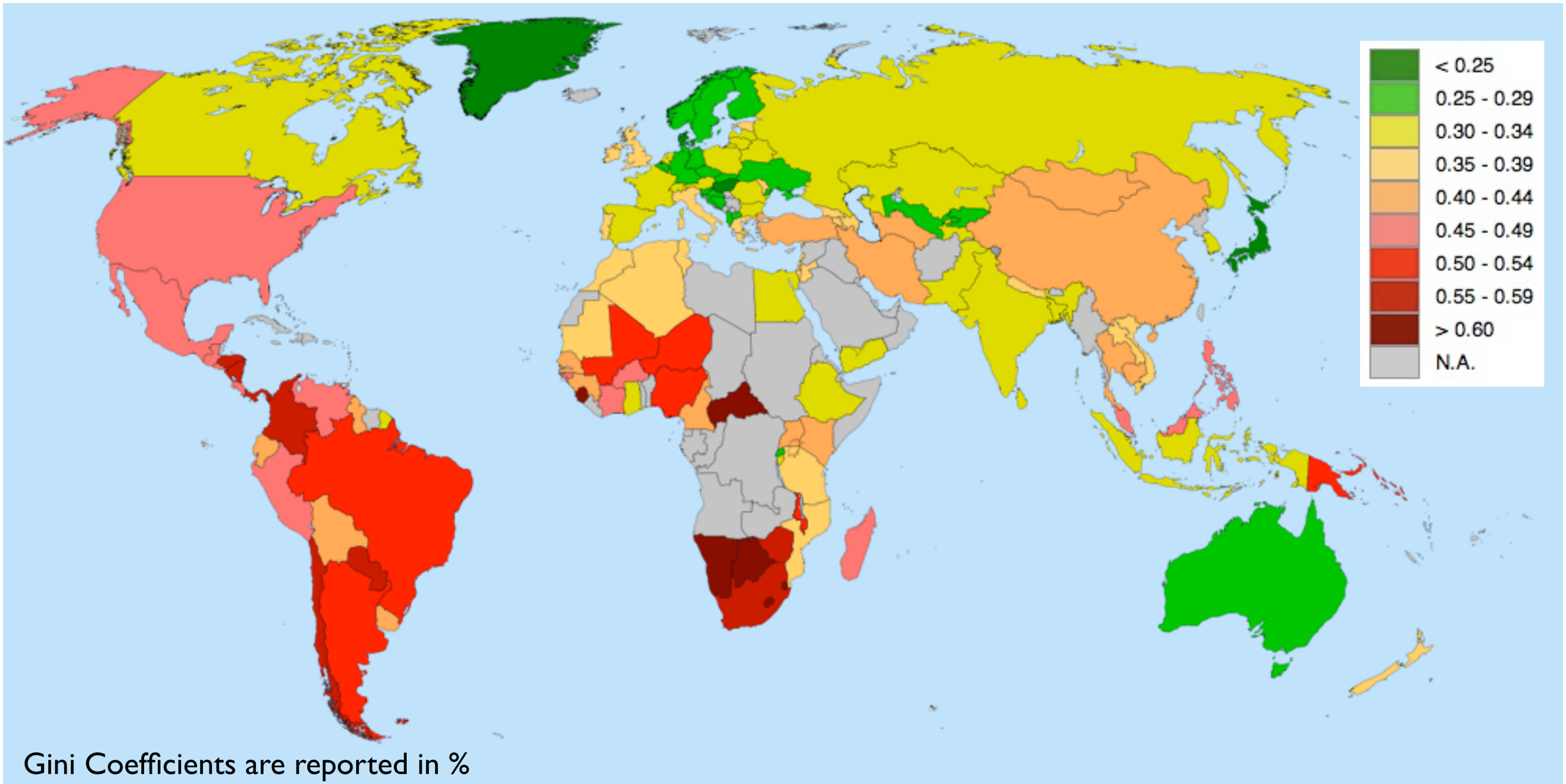


Gini Coefficient



- The Lorenz curve plots the cumulative % of the total participation against the cumulative % of the population
- Gini Coefficient summarizes the curve in a number

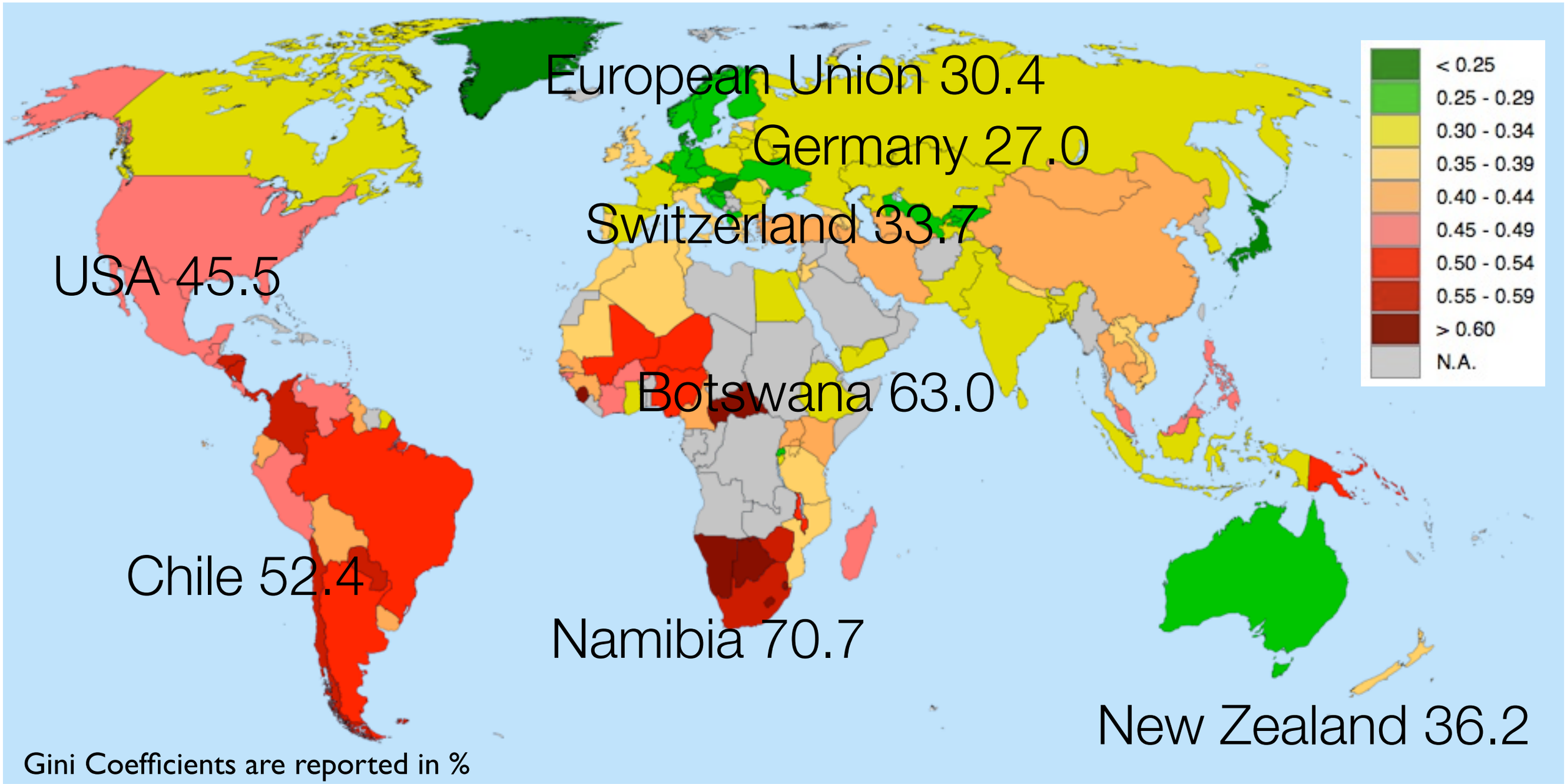
Income Distribution



Gini Coefficients are reported in %

[1]CIA - The World Factbook, **DISTRIBUTION OF FAMILY INCOME - GINI INDEX**, *https://www.cia.gov/library/publications/the-world-factbook/rankorder/2172rank.html*

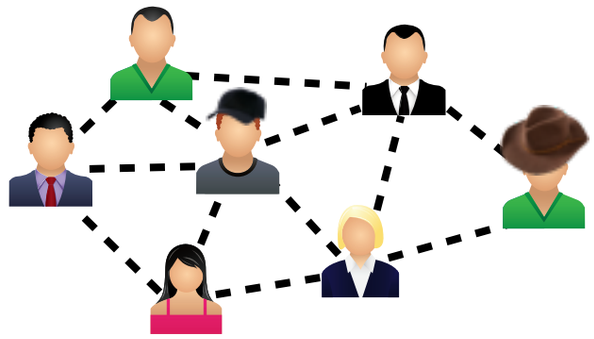
Income Distribution



[1]CIA - The World Factbook, **DISTRIBUTION OF FAMILY INCOME - GINI INDEX**, *https://www.cia.gov/library/publications/the-world-factbook/rankorder/2172rank.html*

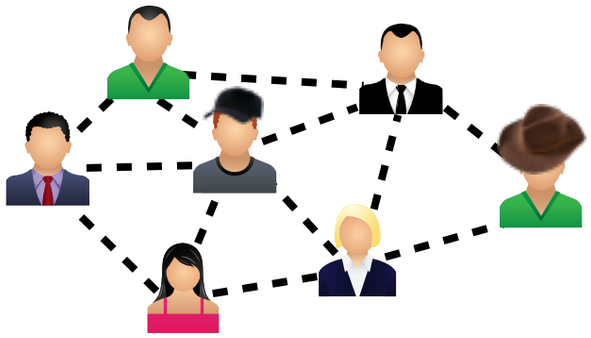
What about Software?

What about Software?



Developers = Population

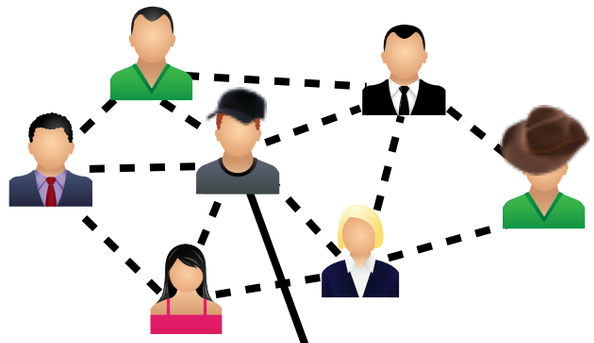
What about Software?



Developers = Population

Files = Assets

What about Software?

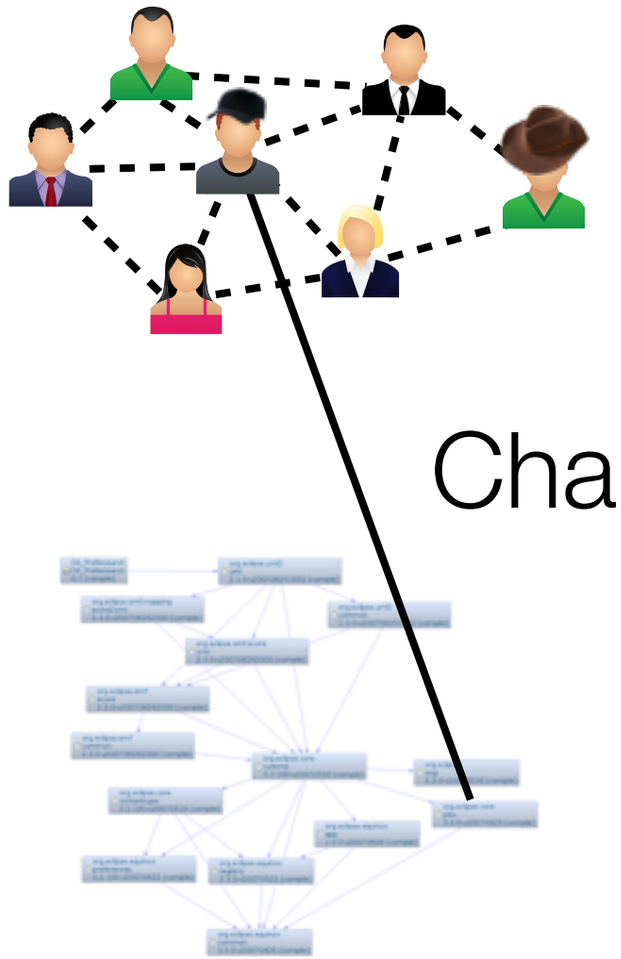


Developers = Population

Changing a file = “being owner”

Files = Assets

What about Software?



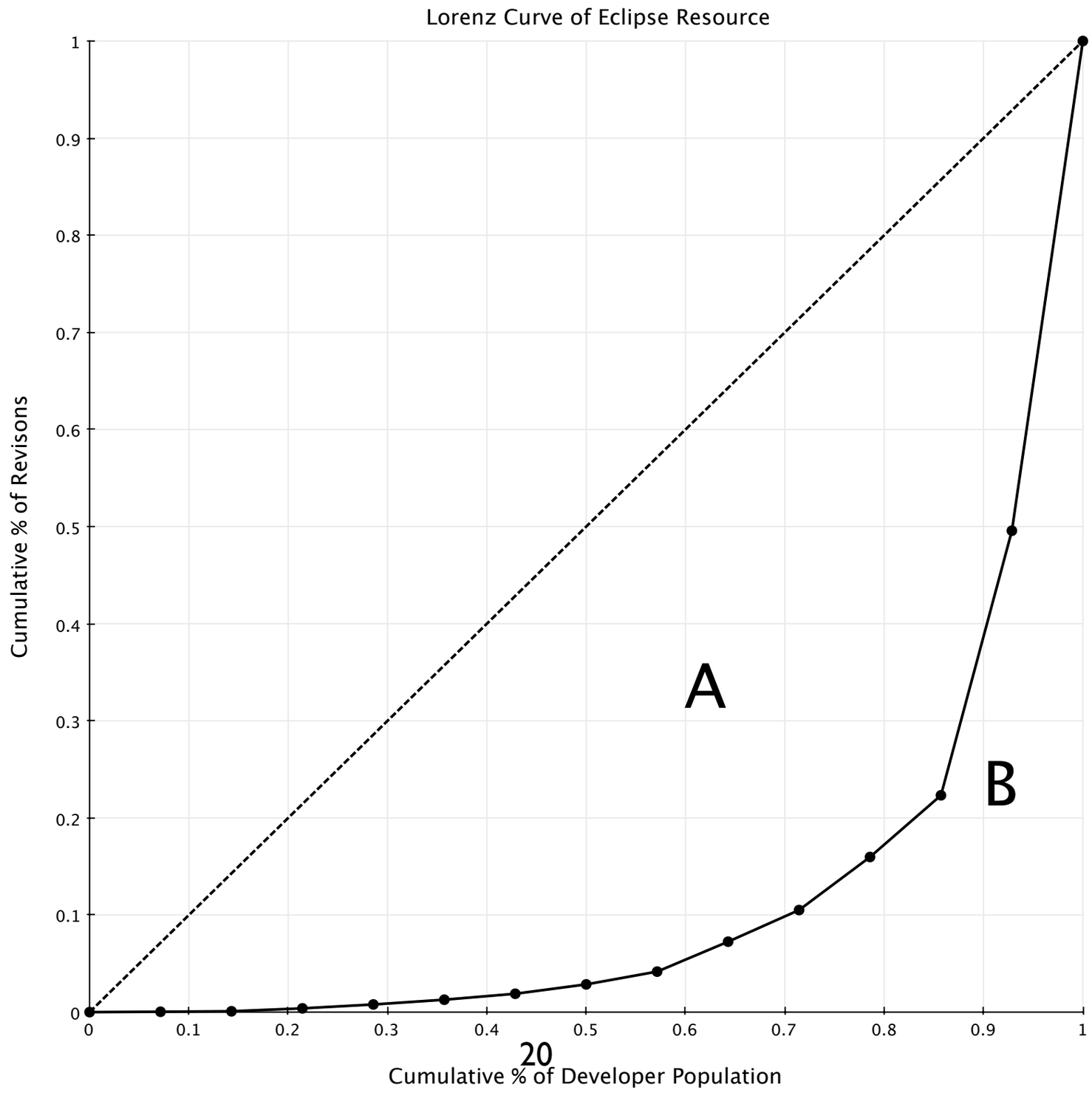
Developers = Population

Changing a file = “being owner”

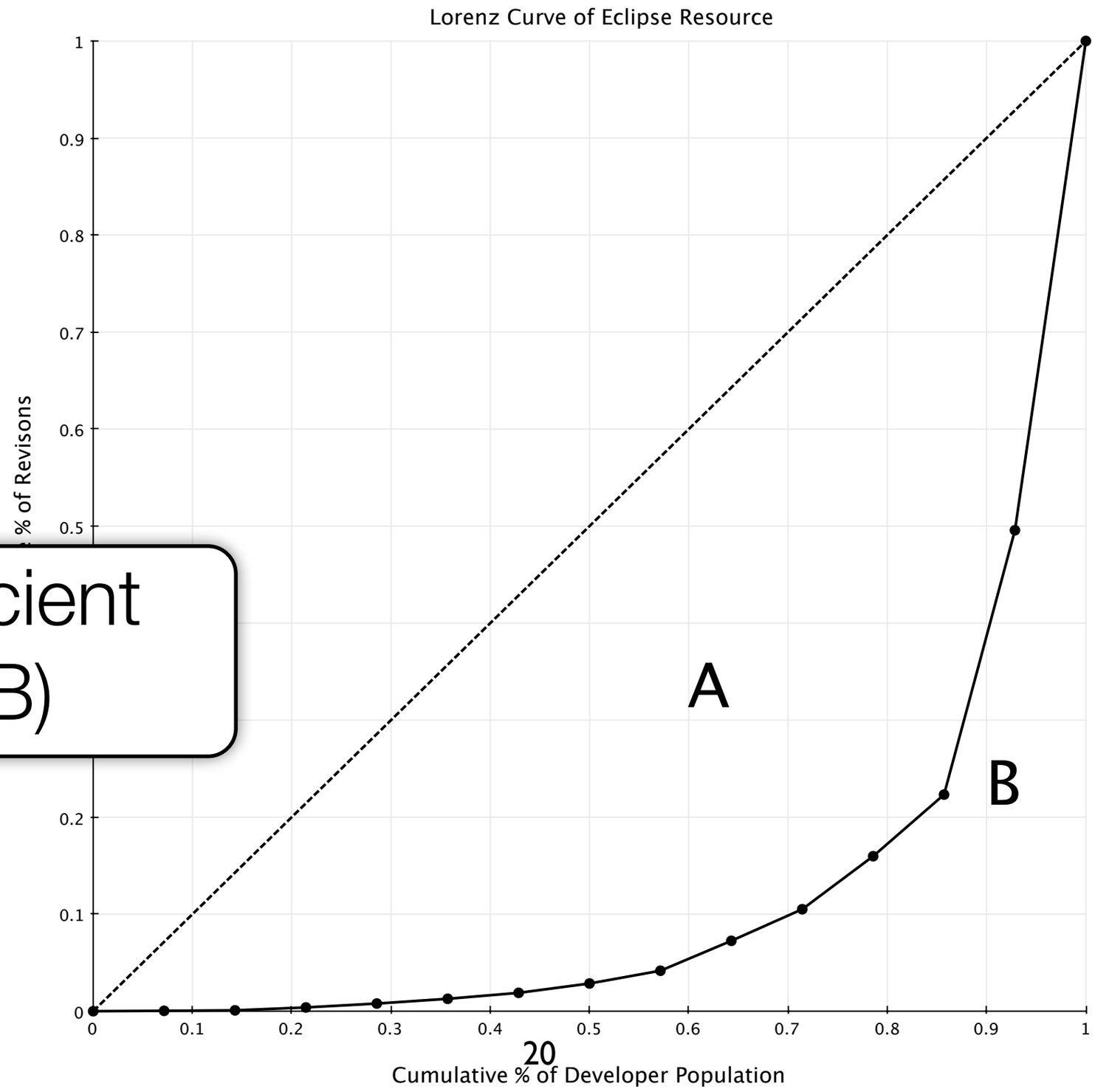
Files = Assets

How are changes of a file distributed among the developers and how does this relate to bugs?

Eclipse Resource



Eclipse Resource



Gini Coefficient
= A / (A + B)

Study

- Eclipse Dataset
- Avg. Gini coefficient is 0.9
- Namibia has a coefficient of 0.7
- Negative Correlation of ~-0.55
- Can be used to identify bug-prone files

Study

- Eclipse Dataset
- Avg. Gini coefficient is 0.9
- Namibia has a coefficient of 0.7
- Negative Correlation of ~-0.55
- Can be used to identify bug-prone files

The more changes of a file are done by a few dedicated developers the less likely it will be bug-prone!

Economic Phenomena

- Economic phenomena of code ownership
- Economies of Scale (Skaleneffekte)
- I’m an expert (in-depth knowledge)
- Profit from knowledge

Economic Phenomena

- Economic phenomena of code ownership
- Economies of Scale (Skaleneffekte)
- I’m an expert (in-depth knowledge)
- Profit from knowledge

Costs to acquire knowledge can be split, e.g., among several releases if you stay with a certain component

Diseconomies of Scale

- Negative of effect of code ownership?
- Loss of direction and co-ordination
- Are we working for the same product?

Another Phenomena

- Economies of Scope (Verbundseffekte)
- Profiting from breadth-knowledge
- Knowledge of different components helps in co-ordinating
- Danger of bottlenecks!

Implications & Conclusions

- How much code ownership & expertise?
- What is your bus number?
- What is better? In-depth- or breadth-knowledge?
- What' is the optimal team size?