Bug Prediction

SW-Wartung & Evolution

Emanuel Giger
Software has Bugs!
Software has Bugs!
Software has Bugs!
Software has Bugs!
Software has Bugs!
Software has Bugs!

Bugs! Bugs! Bugs! Bugs! Bugs!
First case of a bug

Anecdotal story from 1947 related to the Mark II computer.
“...then that 'Bugs' - as such little faults and difficulties are called - show themselves...”

Thomas Edison

“I haven’t failed. I’ve just found 10,000 ways that won’t work.”

Noise in communication infrastructure
Why are bugs in our software?

The Path of a Bug

Code contains a **defect**

```java
if (a <= b) {
    a.foo(); // ......
}
```

**Mistake**

**System failure** may result

**Error (Infection)** may occur

A problem has been detected and Windows has been shut down to prevent damage to your computer.

The problem seems to be caused by the following file: Spmdcon.sys

Page Fault In Nonpaged Area

If this is the first time you’ve seen this stop error screen, restart your computer. If this screen appears again, follow these steps:

Check to make sure any new hardware or software is properly installed. If this is a new installation, ask your hardware or software manufacturer for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware or software. Disable BIOS memory options such as caching or shadowing. If you need to use Safe Mode to remove or disable components, restart your computer, press F8 to select Advanced Startup Options, and then select Safe Mode.
Trace a failure back to identify its *root causes*

Go the *path backwards*: Failure - Error - Defect - Mistake

Find causes & fix the defect: *Debugging*
Stages of Debugging

• Locate cause
• Find a solution to fix it
• Implement to solution
• Execute tests to verify the correctness of the fix
Bug Facts

• “Software Errors Cost U.S. Economy $59.5 Billion Annually”\(^1\)

• ~36% of the IT-Budget is spend on bug fixing\(^1\)

• Massive power blackout in North-East US: *Race Condition*

• Therac-25 Medical Accelerator: *Race Condition*

• Ariane 5 Explosion: *Erroneous floating point conversion*
Quality control: Find defects as early as possible. Prevent defects from being shipped to their productive environment.
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!
Defect Prediction

Identify those components of your system that are most critical with respect to defects

Build forecast (prediction) models to identify bug-prone parts in advance
Defect Prediction

Combines methods & techniques of data mining, machine learning, statistics
Defect Prediction

Input Data → Machine Learning Algorithm → Knowledge, Forecast-Model, ...

- Decision Trees
- Support Vector Machines
- Neural Network
- Bayesian Network
- ...
Crime Fighting, Richmond, VA

- 2005, Massive amount of crime data
- Data mining to connect various data sources
- Input: Crime reports, weather, traffic, sports events and paydays for large employers
- Analyzed 3 times per day
- Output: Forecast where crime was most likely to occur, crime pikes, crime patterns
- Deploy police forces efficiently in advance
Problem: Garbage In - Garbage Out

Defect Prediction Research:

What is the best input to build the most efficient defect prediction models?
Defect Prediction Research:
How can we *minimize* the amount of required *input data* but still get *accurate* prediction models?
Defect Prediction Research:
How can we turn prediction models into *actionable tools* for practitioners?
Bug Prediction Models

**Bug Prediction**

- **Change Metrics**
  - Previous Bugs
  - Code Churn
  - Fine-Grained Source Changes

- **Code Metrics**
  - Function Level Metrics
  - OO-Metrics

- **Organizational Metrics**
  - Team Structure
  - Contribution Structure
Bug Prediction Models

- Change Metrics
  - Previous Bugs
  - Code Churn
  - Fine-Grained Source Changes
  - Method-Level Bug Prediction

- Code Metrics
  - Function Level Metrics
  - OO-Metrics

- Organizational Metrics
  - Team Structure
  - Contribution Structure
Code Metrics

Directly calculated on the code itself

Different metrics to measure various aspects of the size and complexity

Larger and more complex modules are harder to understand and change
Code Metrics

Directly calculated on the code itself

Different metrics to measure various aspects of the size and complexity

Larger and more complex modules are harder to understand and change
Code Metrics

Directly calculated on the code itself

Different metrics to measure various aspects of the size and complexity

Larger and more complex modules are harder to understand and change
Code Metrics

Directly calculated on the code itself

Different metrics to measure various aspects of the size and complexity

Larger and more complex modules are harder to understand and change
Code Metrics

Directly calculated on the code itself

Different metrics to measure various aspects of the size and complexity

Larger and more complex modules are harder to understand and change

- McCabe
- Dependency
- Inheritance
- Lines of Code
- McCabe
Bug Prediction Setup

Eclipse
Bug Prediction Setup

Eclipse → Code Metrics & Bug Data
Bug Prediction Setup

1. Eclipse
2. Code Metrics & Bug Data
3. Random Forest
Bug Prediction Setup

Eclipse → Code Metrics & Bug Data

Code Metrics & Bug Data → Random Forest

Random Forest → X-Validation
Bug Prediction Setup

Eclipse → Code Metrics & Bug Data

Bug-Prone
Not Bug-Prone

Random Forest
X-Validation
Size and complexity are indicators of defects
Change Metrics

- Process Metrics
- Reflect the development activities
- Basic assumptions: *The modules with many defects in the past will most likely be defect-prone in the future as well.*
  
  *Modules that change often have inherently a higher chance to be affected by defects.*
Revisions

Commits to version control systems

Coarse-grained

Files are the units of change
Revisions

There is more than just a file revision
There is more than just a file revision

```java
private IStructureComparator fStructureComparator;

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

```java
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(fContentChangeListener);
        }
        fInput = newInput;
        if (fInput == null) {
            refresh();
            fStructureComparator = null;
        } else {
            if (fStructureComparator instanceof IDisposable) {
                IDisposable disposable = (IDisposable)fStructureComparator;
                disposable.dispose();
            }
            fStructureComparator = null;
        }
    }
    return changed;
}
```
Revisions

There is more than just a file revision
Revisions

There is more than just a file revision

```java
private IStructureComparator fStructureComparator;
public boolean setInput(IElementType newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        removeDocumentRangeUpdaters();
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangeListener);
        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(fContentChangeListener);
    } else {
        refresh();
        changed = true;
    }
    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 {
            IRangeUpdater rangeUpdater = doc.getRangeUpdater(fRangeUpdater);
        } finally {
            fRangeUpdater = null;
        }
    }
}

private ITypeElement fInput;
private IStructureComparator fStructureComparator;
public boolean setInput(IElementType newInput, boolean force) {
    boolean changed = false;
    if (force || newInput != fInput) {
        removeDocumentRangeUpdaters();
        if (fInput instanceof IContentChangeNotifier)
            ((IContentChangeNotifier)fInput).removeContentChangeListener(fContentChangeListener);
        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(fContentChangeListener);
    } else {
        refresh();
        changed = true;
    }
    return changed;
}

public void refresh() {
    IStructureComparator oldComparator = fStructureComparator;
    fStructureComparator = createStructure();
    if (oldComparator != fStructureComparator)
        refresh();
}
```
There is more than just a file revision
Revisions

There is more than just a file revision.

```java
private IStructureComparator fStructureComparator;

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

private void removeDocumentRangeUpdaters() {
    try {
        if (fStructureComparator instanceof IDocumentRange) {
            IDocument doc = ((IDocumentRange)fStructureComparator).getDocument();
            doc.removeDocumentRangeUpdater(fDocumentRangeUpdater);
        }
    } finally {
        fDocumentRangeUpdater = null;
    }
}
```
Revisions

There is more than just a file revision
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
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
- Compares 2 versions of the AST of source code
- Very fine-grained
- Change type information
- Captures all changes

1. Fine-Grained Changes

28
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

- Compares 2 versions of the AST of source code
- Very fine-grained
- Change type information
- Captures all changes

[Fluri et al. 2007, TSE]
IF "balance > 0"

THEN

MI "withDraw(amount);"
Fine-grained Changes

Account.java 1.5

IF
"balance > 0"

THEN
MI
"withDraw(amount);"

Account.java 1.6

IF
"balance > 0 && amount <= balance"

THEN
MI
"withDraw(amount);"

ELSE
MI
notify();
Fine-grained Changes

Account.java 1.5

THEN

IF "balance > 0"

MI

"withDraw(amount);"

Account.java 1.6

THEN

IF "balance > 0 && amount <= balance"

MI

"withDraw(amount);"

ELSE

MI

notify();

1x condition change, 1x else-part insert, 1x invocation statement insert
Fine-grained Changes

Account.java 1.5

IF "balance > 0"

THEN MI

"withDraw(amount);"

Account.java 1.6

IF "balance > 0 && amount <= balance"

THEN MI

"withDraw(amount);"

ELSE MI

 MI

 notify();

1x condition change, 1x else-part insert, 1x invocation statement insert
Fine-grained Changes

Account.java 1.5

Account.java 1.6

More accurate representation of the change history
Method-Level Bug Prediction

class 1  class 2  class 3  ...  class n
Method-Level Bug Prediction

class 1

class 2

class 3

...

class n
Method-Level Bug Prediction

class 1  class 2  class 3  ...  class n

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

11 methods on average

class 1

class 2

class 3

... 

class n

4 are bug prone

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

Saves more than half of all manual inspection steps.
Bug Prediction Models

Bug Prediction

Change Metrics
- Previous Bugs
- Code Churn
- Fine-Grained Source Changes
  - Method-Level Bug Prediction

Code Metrics
- Function Level Metrics
- OO-Metrics

Organizational Metrics
- Team Structure
- Contribution Structure
Using the Gini Coefficient for Bug Prediction
Basic Assumption: Organizational structure and regulations influence the quality of a software system.
The Lorenz curve plots the cumulative % of the total participation against the cumulative % of the population.

Gini Coefficient summarizes the curve in a number.
Gini Coefficients are reported in %

1CIA - The World Factbook, DISTRIBUTION OF FAMILY INCOME - GINI INDEX, 
Income Distribution

Gini Coefficients are reported in %

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

Lorenz Curve of Eclipse Resource

Cumulative % of Developer Population

Cumulative % of Revisions

A

B

37
Gini Coefficient

\[ = \frac{A}{A + B} \]
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?
There are many excellent approaches that reliably locate defects.

Deepens our understanding how certain properties of software are (statistically) related to defects.

X-project defect prediction is an open issue.

Much of it is pure number crunching, i.e., correlation ≠ causality.

Assess practical relevance of defect prediction approaches.
Cross-Project Defect Prediction

- Use a prediction model to predict defect in other software projects

- Study with open source systems (e.g., Eclipse, Tomcat) and MS product (e.g., Win-Kernel, Direct X, IE)

- Results: Only limited success

- Another example of how difficult it is in SE to find generally valid models

**ABSTRACT**

Prediction of software defects works well within projects as long as there is a sufficient amount of data available to train any models. However, this is rarely the case for new software projects and for many companies. So far, only a few have studies focused on transferring prediction models from one project to another. In this paper, we study cross-project defect prediction models on a large scale. For 12 real-world applications, we ran 622 cross-project predictions. Our results indicate that cross-project prediction is a serious challenge, i.e., simply using models from projects in the same domain or with the same process does not lead to accurate predictions. To help software engineers choose models wisely, we identified factors that do influence the success of cross-project predictions. We also derived decision trees that can provide early estimates for precision, recall, and accuracy before a prediction is attempted.

**Categories and Subject Descriptors.** D.2.9 [Software Engineering]: Metrics—Performance measures. Process metrics. Product metrics. D.2.9 [Software Engineering]: Management—Software quality assurance (SQA)

**General Terms.** Management, Measurement, Reliability.

1. INTRODUCTION

Defect prediction works well if models are trained with a sufficiently large amount of data and applied to a single software project [26]. In practice, however, training data is often not available, either because a company is too small or it is the first release of a product, for which no past data exists. Making automated predictions is impossible in these situations. In effort estimation when no or little data is available, engineers often use data from other projects or companies [16]. Ideally the same scenario would be possible for defect prediction as well and engineers would take a model from another project to successfully predict defects in their own project; we call this cross-project defect prediction. However, there has been only little evidence that defect prediction works across projects [32]—in this paper, we will systematically investigate when cross-project defect prediction does work.

The specific questions that we address are:

1. To what extent can we use cross-project data to predict post-release defects for a software system?
2. What kinds of software systems are good cross-project predictors—projects of the same domain, or with the same process, or with similar code structure, or of the same company?

Considering that within companies, the process is often similar or even the same, we seek conclusions about which characteristics facilitate cross-project predictions better—is it the same domain or the same process?

To test our hypotheses we conducted a large scale experiment on several versions of open source systems from Apache Tomcat, Apache Derby, Eclipse, Firefox as well as seven commercial systems from Microsoft, namely Direct-X, IIS, Printing, Windows Clustering, Windows File system, SQL Server 2005 and Windows Kernel. For each system we collected code measures, domain and process metrics, and defects and built a defect prediction model based on logistic regression. Next we ran 622 cross-projects experiments and recorded the outcome of the predictions, which we then correlated with similarities between the projects. To describe similarities we used 40 characteristics: code metrics, ranging from churn [23] (i.e., added, deleted, and changed lines) to complexity; domain metrics ranging from operational domain, same company, etc; process metrics spanning distributed development, the use of static analysis tools, etc. Finally, we analyzed the effect of the various characteristics on prediction quality with decision trees.

1.1 Contributions

The main contributions of our paper are threefold:

1. Evidence that it is not obvious which cross-prediction models work. Using projects in the same domain does not help build accurate prediction models. Process, code data and domain need to be quantified, understood and evaluated before predictions are built and used.
2. An approach to highlight significant predictors and the factors that aid building cross-project predictors, validated in a study of 12 commercial and open source projects.
3. A list of factors that software engineers should evaluate before selecting the projects that they use to build cross-project predictors.
• There are many excellent approaches that reliably locate defects

• Deepens our understanding how certain properties of software are (statistically) related to defects

• Cross-project prediction is an open issue

• Much of it is pure number crunching, i.e., correlation != causality

• Assessment of the practical relevance of defect prediction approaches