Preparation for 1st Exam

Eduardo Figueiredo

http://www.dcc.ufmg.br/~figueiredo

ese.dcc@gmail.com

7 May 2018

Lecture 01 – Course Overview

- Course presentation
- Assessment method
- Agenda

Lecture 02 – Paper

Failure is a Four-Letter Word

– A Parody in Empirical Research –

Andreas Zeller*
Saarland University
Saarbrücken, Germany
zeller@cs.uni-saarland.de

Thomas Zimmermann
Microsoft Research
Washington, USA
tzimmer@microsoft.com

Christian Bird
Microsoft Research
Washington, USA
cbird@microsoft.com

ABSTRACT

Background: The past years have seen a surge of techniques predicting failure-prone locations based on more or less complex metrics. Few of these metrics are *actionable*, though.

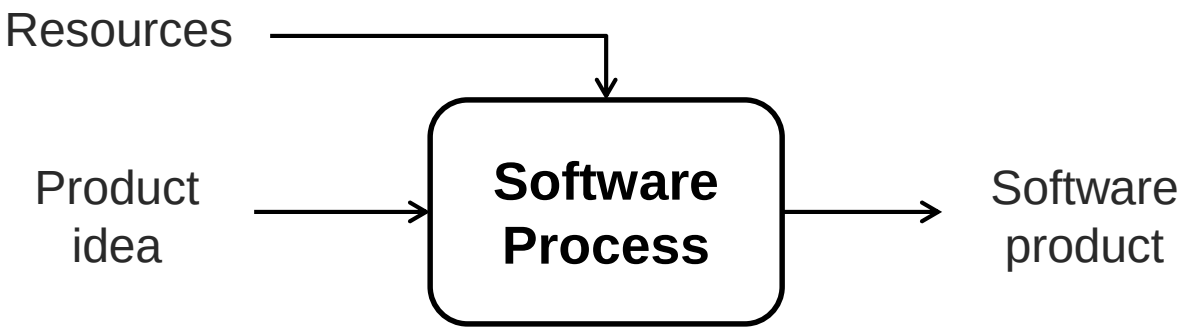
Aims: This paper explores a simple, easy-to-implement method to predict and avoid failures in software systems. The IROP method links *elementary source code features* to known software failures in a lightweight, easy-to-implement fashion.

Method: We sampled the Eclipse data set mapping defects to files in three Eclipse releases. We used logistic regression to associate programmer actions with defects, tested the predictive power of the resulting classifier in terms of precision and recall, and isolated the most defect-prone actions. We also collected initial feedback on possible remedies.

number of developers associated with a file. As elaborate as these approaches may be, they all share the same problem which we call *the cost of consequence*: If I know that a module is failure-prone because it frequently changes, should I stop changing it? If I know failures are related to complexity, should I rewrite it from scratch? Any of these measures induces a new risk – a risk which may be greater than the one originally addressed.

In this paper, we take a different approach. We predict failures from the most basic actions programmers undertake, focusing on the actions that introduce defects *as they are being made* – literally at the moment the source code is typed in. Our recommendations are *immediately actionable:* A simple visual representation associates actions with the likelihood of introducing defects – warning programmers before they might hit the wrong key. Our

Lecture 02 – SE Introduction



- The complexity of software process means that it is hard to optimize it
  - Companies need to continuous try to improve their software processes

Research Methods

- To perform scientific research in software engineering, we have to
  - Understand the methods available
  - Understand their limitations
  - Understand when they can be applied
- There are four main research methods in software engineering
  - **Analytical, Scientific, Engineering, and Empirical**

Bibliography

- C. Wohlin et al. **Experimentation in Software Engineering**, Springer. 2012.
  - Chapter 1 - Introduction

Lecture 03 – Types of Studies

- We can identify two main types of research for empirical studies
  - Exploratory Research
  - Explanatory Research

- The two types are complementary rather than competitive

Lecture 03 – Empirical Strategies

- The main empirical strategies are
  - Survey
  - Case Study
  - Experiment / Quasi-Experiment
- Empirical strategies are not orthogonal
  - Some studies may be viewed as a combination of strategies

Survey

- In Software Engineering, a survey is answered by a sample of developers
- Survey is often performed in retrospect
  - Data is analyzed to derive conclusions
  - Collected information can support both quantitative and qualitative analyses
- Be careful: surveys with many questions may be tedious

Case Study

- Case Study is an empirical strategy that draws on multiple sources of evidences
  - It relies on one instance (or small set of instances) within its real-life context
- It normally aims at tracking a specific attribute or at establishing relationships between attributes

Experiment

- Experiment is an empirical strategy that manipulates one factor (or variable) of the studied setting
- Different treatments are applied to the variable (or variables)
  - Other variables are kept constant
- Experiments are mostly done in a laboratory
  - They require a high level of control

Comparative Table

	Survey	**Case Study**	**Experiment**
Design Type	Fixed	Flexible	Both
Qualitative / Quantitative	Both	Both	Quantitative
Execution Control	No	No	Yes
Control of Measure	No	Yes	Yes
Costs	Low	Medium	High
Replication	High	Low	High

Replication and Ethics

- Replication helps to increase confidence
  - Close replications
  - Differentiated replications
- Key ethical principles
  - Subjects must give informed consent
  - Researchers must maintain confidentiality
  - The study should have value over risks

Bibliography

- C. Wohlin et al. **Experimentation in Software Engineering**, Springer. 2012.
  - Chapter 2 - Empirical Strategies

Lecture 04 – Scale Types

- Scales belonging to the same scale type share the same properties

- The most common scale types are
  - Nominal
  - Ordinal
  - Interval
  - Ratio

Classification of Measures

- Point of view
  - Objective
  - Subjective
- Collection type
  - Direct
  - Indirect

Traditional Metrics

- Fan-in / Fan-out
- Length of Code
- Cyclomatic Complexity
- Vocabulary Size
- Depth of Conditional Nesting

POO Metrics

- Chidamber-Kemerer (CK) Suite
  - Depth of Inheritance Tree (DIT)
  - Number of Children (NOC)
  - Coupling between Object Classes (CBO)
  - Lack of Cohesion of Methods (LCOM)
  - Weighted Methods per Class (WMC)
- Number of Methods Overridden (NMO)

Bibliography

- C. Wohlin et al. **Experimentation in Software Engineering**, Springer. 2012.
  - Chapter 3 – Measurement

- Ian Sommerville. **Software Engineering**, 9th Edtion. Pearson Education, 2010.
  - Section 24.4 Software Measurement and Metrics

Lecture 05 – Refactoring

- Refactoring is a change made to the internal structure of software without changing its observable behavior
  - Its goal is to make the software easier to understand and cheaper to modify
- When is refactoring needed?
  - “If it stinks, change it.”

Grandma Beck

List of Bad Smells

Refused Bequest	**Large Class**	**Long Method**	**Comments**
Divergent Change	**Shotgun Surgery**	**Feature Envy**	**Long Parameter List**
Primitive Obsession	**Switch Statements**	**Lazy Class**	Speculative Generality
Temporary Field	Message Chains	Middle Man	Inappropriate Intimacy
Duplicated Code	Data Clumps	Data Class	Incomplete Library Class
Parallel Inheritance Hierarchies		Alternative Classes with Different Interfaces	

Bibliography

- Martin Fowler. **Refactoring: Improving the Design of Existing Code**. Addison-Wesley Professional, 1st edition, 1999.

Lecture 06 – Paper Discussion

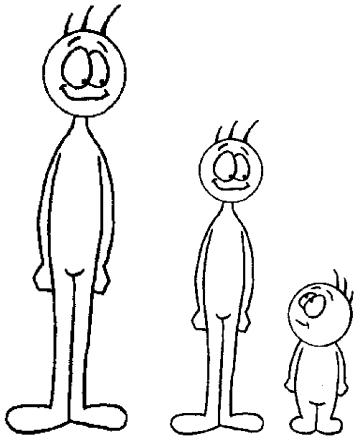
- Paper: Grounded Theory in Software Engineering Research: A Critical Review and Guidelines

Lecture 07 – Thresholds

- We have two major sources for defining threshold values?
  - **Statistical information**: based on statistical measurements, such as mean, median, and standard deviation
  - **Generally accepted semantics**: based on widely accepted knowledge. For example, threshold 3 (three) for number of meals a person consumes per day

Defined Thresholds

- Lower margin
  - AVG - STDEV
- Higher margin
  - AVG + STDEV
- Very high margin
  - (AVG + STDEV) * 1.5
  - A value is considered very high if it is 50% higher than the threshold for a high value



Detection Strategies

- A detection strategy is a composed logical condition, based on metrics, by which code fragments with specific properties are detected
- It relies on two mechanisms
  - Filtering
  - Composition

Detection Strategies

- God Class
- God Method
- Feature Envy
- Shotgun Surgery
- Refused Bequest

Bad Smell Detection Tools

- JDeodorant
- inFusion
- Stench Blossom
- PMD

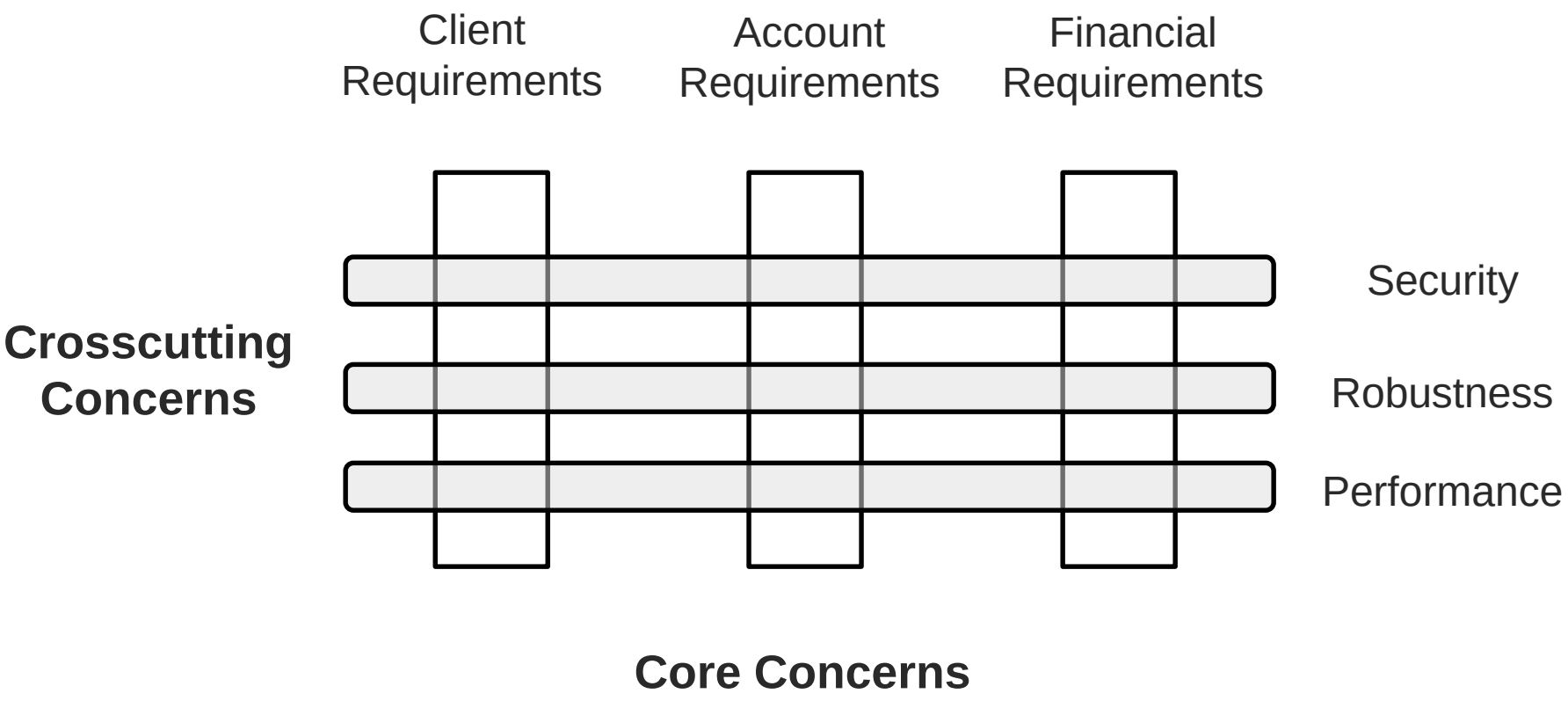
Bibliography

- M. Lanza e R. Marinescu. **Object-Oriented Metrics in Practice**. Springer, 2006.
  - Section 2.1 Metrics and Thresholds
  - Section 4.1 Detection Strategies
  - Section 5.3 God Class
  - Section 5.4 Feature Envy
  - Section 5.6 Brain Method
  - Section 6.5 Shotgun Surgery
  - Section 7.3 Refused Parent Bequest

Lecture 08 – Concerns

- Concerns implemented by several modules of the system and mixed up with other concerns
- Two typical problems
  - Scattering
  - Tangling
- It is hard to change, remove or evolve a crosscutting concern

Representation



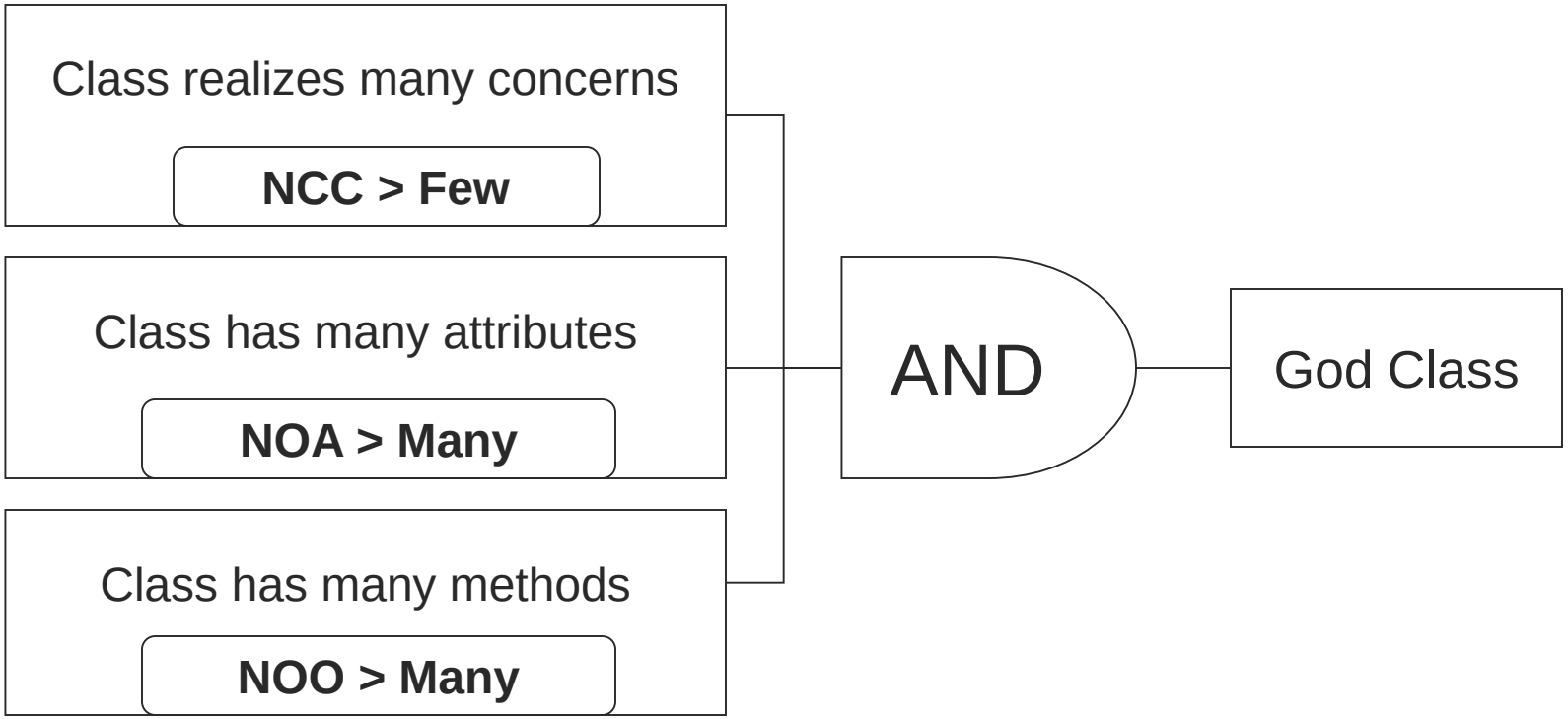
Concern Metrics

- Concern Diffusion over Components (CDC)
- Concern Diffusion over Operations (CDO)
- Concern Diffusion over Lines of Code (CDLOC)
- Number of Concerns per Component (NCC)
- Lines of Concern Code (LOCC)

Detecting Bad Smells

- God Class
- God Method
- Divergent Change
- Shotgun Surgery
- Feature Envy

Strategy for God Class



Bibliography

- Ian Sommerville. **Software Engineering**, 9th Edition. Addison-Wesley, 2010.
  - Section 21.1 Separation of Concerns
- E. Figueiredo et al. **Applying and Evaluating Concern-Sensitive Design Heuristics**. Journal of Systems and Software, pp. 227 - 243, 2012.
- J. Padilha et al. **On the Effectiveness of Concern Metrics to Detect Code Smells: An Empirical Study**. Int'l Conf. on Advanced Information Systems Engineering (CAiSE), 2014.

Lecture 09 – Exercise

- Detecção de Anomalias com Métricas de Software
  - Consent Declaration
  - Background Form
  - Experimental Tasks

Lecture 10 – Papers

- Paper 1: Identifying Design Problems in the Source Code: A Grounded Theory.
  - Presented by Igor
- Paper 2:When and Why Your Code Starts to Smell Bad
  - Presented by Bruno

Lecture 11 – Group Work

- Planning of Group Work