Exploratory Study of a UML Metric for Fault Prediction

Erika Camargo
Advisor: Prof. Ochimizu Koichiro
ICSE, 2010
1. Motivation

A basic software development process

- Analysis
- Design
- Implementation
- Testing

The greater the number of defects found, the greater amount of resources needed to fix them back in the process.
2. Research Question

*How to make earlier detections of low-quality elements within the life-cycle of the software?*
3. Approach

**STEP 1 Approximation**
- CODE Design Complexity Metrics

**STEP 2 Prediction**
- Logistic Regression
- Code-based Prediction Model

**Challenges**
Make a code-based prediction model works properly with:
- UML metrics (approximation of the CODE metrics)
- different projects or packages (different to those used for the construction of the model)

**Solution**
Data Normalization (Scaling to Unit Variance) and elimination of Outliers

May 3th, 2010
1. Approximation of Design Complexity metrics using UML

- Average Relative Error of **0.68**
- Correlation Coefficients: (BNS) 0.7, (CRS) 0.57 and (ECS) 0.86

**Code RFC**

- Num. of Methods of a given class +
- Num. of methods of other classes directly called by any of the methods of the given class

**UML RFC**

- All different Messages Received +
- Messages Sent to different objects (but not those which only return a value)

(*) systems developed by students of JAIST, described in: Gomaa Hassan, Designing Concurrent, Distributed, and Real-Time Applications with UML, Addison Wesley-Object Technology Series Editors, July 2000.
1. Approximation of Design Complexity metrics using UML

- Average Relative Error of 0.68
- Correlation Coefficients: (BNS) 0.7, (CRS) 0.57 and (ECS) 0.86

**Code RFC**

- Number of methods of a given class + Number of methods of other classes directly called by any of the methods of the given class

---

**BNS**: Banking system *
**CRS**: Cruise control system *
**ECS**: E-commerce system *

(*) systems developed by students of JAIST, described in: Gomaa Hassan, Designing Concurrent, Distributed, and Real-Time Applications with UML, Addison Wesley-Object Technology Series Editors, July 2000.
Boxplots for RFC measures

BNS: Banking system *
CRS: Cruise control system *
ECS: E-commerce system *
MYL: Mylyn system **

(**) Eclipse Project
(*) systems developed by students of JAIST, described in: Gomaa Hassan, Designing Concurrent, Distributed, and Real-Time Applications with UML, Addison Wesley-Object Technology Series Editors, July 2000.
AFTER ELIMINATION OF OUTLIERS AND DATA NORMALIZATION

BNS: Banking system *
CRS: Cruise control system *
ECS: E-commerce system *
MYL: Mylyn system **

Number of Correct Predictions improved up to 32%

Raw Code RFC VS Normalized Code RFC VS Normalized UML RFC
BNS: 80% (50% LF, 100% MF) → 80% (100% LF, 67% MF) → 80% (100% LF, 67% MF)
CRS: 71% (100% LF, 0% MF) → 57% (80% LF, 0% MF) → 64% (80% LF, 25% MF)
ECS: 67% (100% LF, 0% MF) → 33% (50% LF, 0% MF) → 67% (75% LF, 50% MF)

LF (Least Faulty Classes), MF (Most Faulty Classes)

(**) Eclipse Project
(* systems developed by students of JAIST, described in: Gomaa Hassan, Designing Concurrent, Distributed, and Real-Time Applications with UML, Addison Wesley-Object Technology Series Editors, July 2000.)
5. Summary of contributions and Upcoming work

Contributions

• A close approximation of a code metric using UML collaboration diagrams
• A procedure that improves prediction results across projects/packages and that enabled our UML metric to predict faulty code

Upcoming work

• Study of other UML metrics and their relationship with fault-proneness
• Formal Definition of metrics for UML collaboration diagrams
• Target “Traceability” within the process development to identify possible errors, faults or simply progress.
Thank you!
Questions?
erika.camargo@jaist.ac.jp
**STEP 1. Approximation of Design Complexity**

**Complexity metrics using UML**

**RFC Approximation**

```java
class A{
    B b;
    void dosth() {
        int flag;
        flag = b.request_sth_toB();
    }
    int do_sthelse() {
        int status;
        status = d.request_sth_toD();
        return(status);
    }
}
```

**UML RFC (UML Collaboration Diagrams)** =

All different **Messages Received** +

**Messages Sent** to different objects (but not those which only return a value)

**Code RFC** =

Num. of Methods of a given class +
Num. of methods of other classes
directly called by any of the methods of
the given class.

**RFC (A) = 2 + 2 = 4**
Can a LR model built with these kind of metrics work efficiently with different software projects?

P (y=1)

LEAST FAULTY

PREDICTION MODEL BUILT WITH CRS

MOST FAULTY

PREDICTION MODEL BUILT WITH PACKAGE MYL-13

X = Design Complexity Metric

May 3th, 2010

JAIST
STEP 2. Predicting using UML Metrics (Normalization)

Approach

1. Elimination of Outliers
2. Data Normalization using *Linear Scaling to Unit Variance*

\[ x' = \frac{x - \mu}{\frac{3\sigma}{2}} + 1 \]

- \( x' \) is the new normalized value [0..1]
- \( x \) raw value
- \( \mu \) media
- \( \sigma \) standard deviation

May 3th, 2010

JAIST
STEP 2. Predicting using UML Metrics 
(Case Study)

1. **Preprocessing** the datasets of the 13 different packages of the MYLYN plus the 6 datasets of UML and CODE metrics of the small-size SW projects (*Normalization and Outliers Elimination*)

2. **Construction** of 13 univariate LR models (one per MYLYN package), where:
   - Dependent Variable is the code CK-RFC
   - Independent Variable is classified as
     - *Most Faulty* if Num. Faults>=2Q
     - *Least Faulty* Otherwise

3. Chose the best model and **Test** it with:
   - the other *12 packages* and
   - datasets of *UML* and *CODE* measures of the ECS, BNS and CRS projects
## Predicting using UML Metrics

### Case Study Results

<table>
<thead>
<tr>
<th>Project</th>
<th>Correctness</th>
<th>Specificity</th>
<th>Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RFC</td>
<td>NRFC</td>
<td>RFC</td>
</tr>
<tr>
<td>MYL-P01</td>
<td>63%</td>
<td>75%</td>
<td>59%</td>
</tr>
<tr>
<td>MYL-P02</td>
<td>63%</td>
<td>75%</td>
<td>44%</td>
</tr>
<tr>
<td>MYL-P03</td>
<td>72%</td>
<td>72%</td>
<td>78%</td>
</tr>
<tr>
<td>MYL-P04</td>
<td>51%</td>
<td>67%</td>
<td>52%</td>
</tr>
<tr>
<td>MYL-P05</td>
<td>48%</td>
<td>61%</td>
<td>52%</td>
</tr>
<tr>
<td>MYL-P06</td>
<td>61%</td>
<td>85%</td>
<td>56%</td>
</tr>
<tr>
<td>MYL-P07</td>
<td>62%</td>
<td>80%</td>
<td>59%</td>
</tr>
<tr>
<td>MYL-P08</td>
<td>60%</td>
<td>80%</td>
<td>50%</td>
</tr>
<tr>
<td>MYL-P09</td>
<td>75%</td>
<td>81%</td>
<td>74%</td>
</tr>
<tr>
<td>MYL-P10</td>
<td>76%</td>
<td>90%</td>
<td>76%</td>
</tr>
<tr>
<td>MYL-P11</td>
<td>74%</td>
<td>79%</td>
<td>72%</td>
</tr>
<tr>
<td>MYL-P12</td>
<td>87.5%</td>
<td>87.5%</td>
<td>91.7%</td>
</tr>
<tr>
<td>MYL-P13</td>
<td>56%</td>
<td><strong>88%</strong></td>
<td>50%</td>
</tr>
</tbody>
</table>

% of total classes (MOST FAULTY + LEAST FAULTY) correctly detected

% of LEAST FAULTY classes correctly detected

% of MOST FAULTY classes correctly detected

May 3th, 2010  JAIST
## STEP 2. Predicting using UML Metrics

### (Case Study Results)

<table>
<thead>
<tr>
<th>Project</th>
<th>Correctness</th>
<th>Specificity</th>
<th>Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RFC NRFC</td>
<td>RFC NRFC</td>
<td>RFC NRFC</td>
</tr>
<tr>
<td>ECS</td>
<td>80% 80%</td>
<td>50% 100%</td>
<td>100% 67%</td>
</tr>
<tr>
<td>CRS</td>
<td>71% 57%</td>
<td>100% 80%</td>
<td>0% 0%</td>
</tr>
<tr>
<td>BNS</td>
<td>67% 33%</td>
<td>100% 50%</td>
<td>0% 0%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project</th>
<th>Correctness</th>
<th>Specificity</th>
<th>Sensitivity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NRFC UML-NRFC</td>
<td>NRFC UML-NRFC</td>
<td>NRFC UML-NURFC</td>
</tr>
<tr>
<td>ECS</td>
<td>80% 80%</td>
<td>100% 100%</td>
<td>67% 67%</td>
</tr>
<tr>
<td>CRS</td>
<td>57% 64%</td>
<td>80% 80%</td>
<td>0% 25%</td>
</tr>
<tr>
<td>BNS</td>
<td>33% 67%</td>
<td>50% 75%</td>
<td>0% 50%</td>
</tr>
</tbody>
</table>
Outlier:

- A **boxplot** mainly represents the **IQR** (interquartile range) which is the difference between the 75th and 25th percentiles.

- These values are the edges of the box and are called **upper hinge** and **lower hinge**.

- Data points **1.5 times** the **IQR** above the upper hinge or below the lower hinge are considered potential **outliers**.