Predicting Faults Using the Complexity of Code Change

International Conference on Software Engineering (2009)
Ahmed E. Hassan

2013-07-09
Yoo Jin Lim
Fault Prediction

- Predict the incidence of faults in code
  - Early warning for developers and managers
  - Affects test efforts, costs, and product quality
- Commonly associated with code complexity
  - Ex) More LOC $\rightarrow$ More complex code
- **Process** metrics outperform **code** metrics
  - Then, consider code change process complexity
Motivation

- To prevent fault occurrences, predict faults.
- To manage negative effects of complexity, measure it.

Goal

- Suggest a predictor of future faults based on code change complexity models
Code change process

- Pattern of source code modifications
- Recorded by source control systems
- Repository has all files’ change history

Modification Record

- Fault Repairing modifications (FR)
- General Maintenance modifications (GM)
- Feature Introduction modifications (FI)
Shannon Entropy

- Measure **entropy (amount of uncertainty)** in a distribution

\[
H_n(P) = - \sum_{k=1}^{n} (p_k \cdot \log_2 p_k), \quad p_k \geq 0 \text{ and } \sum_{k=1}^{n} p_k = 1
\]

\( p_k = \text{probability of occurrence for element } k \)

- Minimal vs. Maximum Entropy
  - Ex) Output distribution of a system with 4 symbols

- **H\(_4\)(P) = 0**
- **H\(_4\)(P) = 2**
Quantifying the code change complexity

- Highly scattered changes
- Highly complex project

Change patterns with high entropy are harder to track!
Basic Code Change Model (1/2)

- Complexity of a change period

\[ H_4(P) = - \sum_{k=1}^{4} (p_k \times \log_2 p_k) = 1.68 \]

- Measurement approach
  - Use the file as unit of code
  - Only use the FI modifications
  - Quantify for several changes within a period

\[ p_{file} = \frac{\text{Change count of a file}}{\text{Total change count}} \]
Basic Code Change Model (2/2)

- Evolution of code change entropy

![Graph showing the evolution of code change entropy over time with fixed number of periods and files.]

- Fixed number of periods
- Fixed number of files
Evolution Periods

- Time based periods
- Modification limit based periods
- Burst based periods
  - Burst = many code modifications followed by none
  - The most general method

# Modifications

Time
Adaptive System Sizing

- Normalized Static Entropy, $H = [0,1]$

\[
H(P) = \frac{1}{\text{Max Entropy for Distribution}} \quad H_n(P) = \frac{1}{\log_2 n} \quad H_n(P)
\]

- $n$ = number of all files
- entropy of distributions of different sizes can be compared

- Adaptive Sizing Entropy, $H'$

- $n'$ = number of *recently* modified files
  - Using time: Set of files modified in the preceding $x$ months
  - Using previous periods: those modified in the preceding $x$ periods
### Complexity of a file

- **History Complexity Period Factor (HCPF)**
  - For a file $j$ during period $i$, a set of files $F_i$ is modified:
    
    $HCPF_i(j) = \begin{cases} 
    c_{ij} \cdot H_i, & j \in F_i \\
    0, & \text{otherwise} 
    \end{cases}$

    - $c_{ij}$ = contribution of entropy $H_i$ to file $j$

    - $HCPF^1$ with $c_{ij} = 1$: All modified files are affected by full entropy.
    - $HCPF^2$ with $c_{ij} = p_j$: The more a file is modified, more it is affected.
    - $HCPF^3$ with $c_{ij} = \frac{1}{|F_i|}$: The more modified files, less each is affected.
Example of HCPF

\[ HCPF_i(j) = \begin{cases} c_{ij} \times H_i, & j \in F_i \\ 0, & \text{otherwise} \end{cases} \]

- Calculate HCPF for file A during this period

<table>
<thead>
<tr>
<th>file</th>
<th>HCPF1(A)</th>
<th>HCPF2(A)</th>
<th>HCPF3(A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1 * 1 = 1</td>
<td>0.25 * 1 = 0.25</td>
<td>(\frac{1}{4} \times 1 = 0.25)</td>
</tr>
<tr>
<td>B</td>
<td>0.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>0.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>0.25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[ H(P) = 1 \]
History Complexity Metric (HCM)

- Simple HCM for a file
  - For a file $j$ over a set of evolution periods $\{a,\ldots,b\}$:
    \[
    HCM_{\{a,\ldots,b\}}(j) = \sum_{i \in \{a,\ldots,b\}} HCPF_i(j)
    \]

- HCM for a subsystem
  - For all files in subsystem $S$
    \[
    HCM_{\{a,\ldots,b\}}(S) = \sum_{j \in S} HCM_{\{a,\ldots,b\}}(j)
    \]
History Complexity Metric (HCM)

- Decay model of HCM
  - Earlier modifications have less contribution to complexity

\[
HCM_{\{a,\ldots,b\}}(j) = \sum_{i \in \{a,\ldots,b\}} e^{\varphi (T_i - \text{Current Time})} \cdot HCPF_i^1(j)
\]

- \(T_i = \) end time of period \(i\)
- \(\varphi = \) decay factor

- Four metrics in total

\[
HCM^{1s}, HCM^{2s}, HCM^{3s}, \text{ and } HCM^{1d}
\]
Case Study (1/4)

- Summary of the studied systems

<table>
<thead>
<tr>
<th>App. Name</th>
<th>Type</th>
<th>Start Date</th>
<th>Subsystem Count</th>
<th>Prog. Lang.</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetBSD</td>
<td>OS</td>
<td>Mar 1993</td>
<td>235</td>
<td>C</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>OS</td>
<td>Jun 1993</td>
<td>152</td>
<td>C</td>
</tr>
<tr>
<td>OpenBSD</td>
<td>OS</td>
<td>Oct 1995</td>
<td>265</td>
<td>C</td>
</tr>
<tr>
<td>Postgres</td>
<td>DBMS</td>
<td>Jul 1996</td>
<td>280</td>
<td>C</td>
</tr>
<tr>
<td>KDE</td>
<td>Windowing System</td>
<td>Apr 1997</td>
<td>108</td>
<td>C++</td>
</tr>
<tr>
<td>KOffice</td>
<td>Productivity Suite</td>
<td>Apr 1998</td>
<td>158</td>
<td>C++</td>
</tr>
</tbody>
</table>

- Study approach
  - Build Statistical Linear Regression (SLR Model)
    - To predict faults in subsystems during the 4th and 5th years
  - Measure & compare the error between models
    - Modifications vs. Faults vs. Entropy (4)
  - Determine statistical significance of the difference in error

ⓒ KAIST SE LAB 2013
Case Study (2/4)

- **Linear Regression Models**
  - \( y = \beta_0 + \beta_1 x \)
  - \( y \) = number of faults in a subsystem (FR modifications)
  - \( x \) = specific metrics for each subsystem

<table>
<thead>
<tr>
<th>SLR Model</th>
<th>Value of x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model m</td>
<td>Number of modifications</td>
</tr>
<tr>
<td>Model f</td>
<td>Number of prior faults</td>
</tr>
<tr>
<td>Model HCM1s</td>
<td>HCM_{1s} value</td>
</tr>
<tr>
<td>Model HCM2s</td>
<td>HCM_{2s} value</td>
</tr>
<tr>
<td>Model HCM3s</td>
<td>HCM_{3s} value</td>
</tr>
<tr>
<td>Model HCM1d</td>
<td>HCM_{1d} value</td>
</tr>
</tbody>
</table>

- The SLR Model **HCM^{1d}** has the **best fit** for all.
Case Study (3/4)

- **Prediction Error for the SLR Models**

  \[ \hat{y}_i = \beta_0 + \beta_1 x_i \]

  - \( \hat{y}_i \) = Number of predicted faults in the subsystem in 4th and 5th years
  - Absolute prediction error: \( e_i = |\hat{y}_i - y_i| \)
  - Total prediction error:
    \[ E = \sum_{i=1}^{n} e_i^2 \]

- **Statistical Significance of Differences b/t Models**

  - Paired t-test is used
  - P-value < 0.05: we can with high probability reject \( H_0: \)
    \[ H_0: \mu(e_A, i) - e_B, i = 0 \]
Statistical significance of differences in error

- Significant results (P-value < 0.05) highlighted

- $E_m \geq E_f \geq E_{HCM}$ in most cases
  - Predictors based on faults are better than modifications, and complexity models are better than faults or modifications.
Related Work

- Barry et al. (2003)
  - Studied a retail software’s code modification records
  - Identified evolution patterns in the software system

- Adb-El-Hafiz (2001)
  - Quantified the complexity of the source code to measure entropy.

- This paper’s model uses code modification records to quantify the complexity of the code change process.
Conclusion

- Verified the conjectures:
  - Complex code change process negatively affects the software system.
  - The more complex changes to a file, the higher the chance the file will contain faults.

Contributions

- Computed the complexity of the code change process
  - instead of just that of the source code
- Presented a better predictor of future faults
  - Help managers plan ahead and be ready for future
Discussion

❖ Limitations
  - Did not consider unrepaired faults
  - Generalization limited to large open source systems
  - Results do not show a causality relation

❖ Future work
  - Study commercial systems
  - Build a richer and detailed metric for complexity
Thank You.
Code Change Process (cont’d)

- Modification Types
  - Fault Repairing modifications (FR)
    - Fixing a fault
    - Not used in calculating the complexity of the change process
    - Used to count fault in case study for validation
  - General Maintenance modifications (GM)
    - Bookkeeping modifications such as copyright update
    - Not used in analysis
  - Feature Introduction modifications (FI)
    - Adding or enhancing features
    - Used to calculating the code change process complexity
SLR example for $x = \text{number of modifications}$

- $\beta_0$ and $\beta_1$ estimated using fault data from 2\textsuperscript{nd} and 3\textsuperscript{rd} years

### Modification Record

- Date: ---
- Name: ---
- Line #s: ---
- Lines of code: ---
- Message: FR

### Automatic Lexical Analysis

- $y = \beta_0 + \beta_1 x$
- $R^2$ value

### Modification Count

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Modification Count</th>
<th>Actual Fault Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>#2</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>#n</td>
<td>7</td>
<td>10</td>
</tr>
</tbody>
</table>

### $R^2$ values for different apps

<table>
<thead>
<tr>
<th>App</th>
<th>$R^2_{f}$</th>
<th>$R^2_{m}$</th>
<th>$R^2_{1s}$</th>
<th>$R^2_{2s}$</th>
<th>$R^2_{3s}$</th>
<th>$R^2_{1d}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetBSD</td>
<td>0.57</td>
<td>0.55</td>
<td>0.54</td>
<td>0.53</td>
<td>0.61</td>
<td>0.71</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>0.65</td>
<td>0.48</td>
<td>0.57</td>
<td>0.58</td>
<td>0.59</td>
<td>0.65</td>
</tr>
<tr>
<td>OpenBSD</td>
<td>0.45</td>
<td>0.44</td>
<td>0.54</td>
<td>0.55</td>
<td>0.54</td>
<td>0.57</td>
</tr>
<tr>
<td>Postgres</td>
<td>0.57</td>
<td>0.36</td>
<td>0.49</td>
<td>0.51</td>
<td>0.60</td>
<td>0.61</td>
</tr>
<tr>
<td>KDE</td>
<td>0.31</td>
<td>0.26</td>
<td>0.28</td>
<td>0.29</td>
<td>0.36</td>
<td>0.57</td>
</tr>
<tr>
<td>KOffice</td>
<td>0.30</td>
<td>0.27</td>
<td>0.33</td>
<td>0.33</td>
<td>0.27</td>
<td>0.41</td>
</tr>
</tbody>
</table>
Prediction error example

- Predict faults in the application in the 4th and 5th years

\[ \hat{y} = \beta_0 + \beta_1 x_i \]

<table>
<thead>
<tr>
<th>Subsystem #1</th>
<th>Subsystem #2</th>
<th>...</th>
<th>Subsystem #n</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modification Count</td>
<td>5</td>
<td>20</td>
<td>...</td>
</tr>
<tr>
<td>Actual Fault Count</td>
<td>5</td>
<td>19</td>
<td>...</td>
</tr>
<tr>
<td>Predicted Fault Count</td>
<td>6</td>
<td>13</td>
<td>...</td>
</tr>
</tbody>
</table>

\[ E = \sum_{i=1}^{n} e_i^2 \]
Modification vs. Faults

Prior faults should be used to predict faults instead of prior modifications.

<table>
<thead>
<tr>
<th>App</th>
<th>$E_m - E_f$ (%)</th>
<th>$P(H_0 \text{ holds})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetBSD</td>
<td>+11.7 (+04%)</td>
<td>0.67</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>+71.2 (+48%)</td>
<td>0.00</td>
</tr>
<tr>
<td>OpenBSD</td>
<td>+03.7 (+02%)</td>
<td>0.84</td>
</tr>
<tr>
<td>Postgres</td>
<td>+47.2 (+49%)</td>
<td>0.02</td>
</tr>
<tr>
<td>KDE</td>
<td>+26.3 (+07%)</td>
<td>0.32</td>
</tr>
<tr>
<td>KOffice</td>
<td>+26.3 (+04%)</td>
<td>0.51</td>
</tr>
</tbody>
</table>
### Modifications vs. Entropy

<table>
<thead>
<tr>
<th>App</th>
<th>( E_{HCM3a} - E_m ) (%)</th>
<th>( P(H_0 \text{ holds}) )</th>
<th>( E_{HCM1d} - E_m ) (%)</th>
<th>( P(H_0 \text{ holds}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetBSD</td>
<td>-39.8 (-14%)</td>
<td>0.03</td>
<td>-106.5 (-36%)</td>
<td>0.00</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>-47.4 (-22%)</td>
<td>0.02</td>
<td>-72.0 (-33%)</td>
<td>0.00</td>
</tr>
<tr>
<td>OpenBSD</td>
<td>-40.4 (-18%)</td>
<td>0.01</td>
<td>-53.8 (-23%)</td>
<td>0.00</td>
</tr>
<tr>
<td>Postgres</td>
<td>-52.7 (-37%)</td>
<td>0.04</td>
<td>-56.9 (-40%)</td>
<td>0.03</td>
</tr>
<tr>
<td>KDE</td>
<td>-52.1 (-13%)</td>
<td>0.01</td>
<td>-165.2 (-42%)</td>
<td>0.00</td>
</tr>
<tr>
<td>KOOffice</td>
<td>+03.3 (+01%)</td>
<td>0.83</td>
<td>-69.9 (-18%)</td>
<td>0.01</td>
</tr>
</tbody>
</table>

- Both HCM based models are likely to outperform prior modifications.
Faults vs. Entropy

<table>
<thead>
<tr>
<th>App</th>
<th>$E_{HCM3s} - E_f$ (%)</th>
<th>$P(H_0 \text{ holds})$</th>
<th>$E_{HCM1d} - E_f$ (%)</th>
<th>$P(H_0 \text{ holds})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>NetBSD</td>
<td>-28.14 (-10%)</td>
<td>0.26</td>
<td>-94.84 (-34%)</td>
<td>0.00</td>
</tr>
<tr>
<td>FreeBSD</td>
<td>+23.81 (+16%)</td>
<td>0.30</td>
<td>-00.79 (-01%)</td>
<td>0.97</td>
</tr>
<tr>
<td>OpenBSD</td>
<td>-36.59 (-16%)</td>
<td>0.02</td>
<td>-50.05 (-22%)</td>
<td>0.01</td>
</tr>
<tr>
<td>Postgres</td>
<td>-05.53 (-06%)</td>
<td>0.71</td>
<td>-09.71 (-10%)</td>
<td>0.55</td>
</tr>
<tr>
<td>KDE</td>
<td>-25.72 (-07%)</td>
<td>0.32</td>
<td>-138.87 (-38%)</td>
<td>0.01</td>
</tr>
<tr>
<td>KOffice</td>
<td>+19.20 (+05%)</td>
<td>0.34</td>
<td>-54.07 (-15%)</td>
<td>0.04</td>
</tr>
</tbody>
</table>

- Models based on entropy are as good as (or even better) predictors of faults in comparison to prior faults for most studied software systems.