Indirectly predicting the maintenance effort of open-source software

Liguo Yu
Journal of Software Maintenance and Evolution: Research and Practice 2006

2007. 1. 31
Taehoon Song
Contents

- Introduction
- Background
- Related work
- Open-source indirect maintenance effort estimation process
- Indirect maintenance effort measures
- Empirical study
  - NASA SEL projects
  - LINUX project
- Conclusion
- Discussion
Introduction (1/3)

- Software maintenance
  - Process of enhancing and optimizing existing software
  - Major types
    - Utilizing software support
    - Source code change
    - Documentation update
  - Activity-based view
    - Corrective maintenance
      - Activity performed on an artifact in order to remove a residual fault while leaving the intended semantics unchanged
    - Perfective / Adaptive maintenance
      - Including enhancive maintenance
      - Activities performed on a software artifact in order to implement changed functionality and/or to change software properties
        - Such as security, performance, usability and so on or to adapt to a new platform
Introduction (2/3)

- Software maintenance (cont’d)
  - Most complicated software process
  - Involves

What are involved?
Introduction (3/3)

- Maintenance effort estimation process (conventional)
  - Only works for strictly managed software

- Research goal
  - Present a process to indirectly predict maintenance effort of open-source software

- Extract maintenance data
- Build and validate maintenance effort model
- Predict maintenance effort

Maintenance effort, number of task, size of product from previous maintenance records

Prediction of future maintenance effort

Maintenance effort is represented as a mathematical model
Background (1/2)

- **Open-source software**
  - Refers to software that is free to use
  - Its source code is fully accessible via the internet

- **Closed-source software**
  - Developed by a single company
  - Source code is kept secret
    - Normally copyrighted or patented
    - Legally protected as intellectual property

What's the difference from the viewpoint of maintenance?
Background (2/2)

- Differences between open-source software and closed-source software
  - Viewpoint of maintenance

Evolves in response to either *user requirements* or to *fixing a reported bug*
## Related work (1/3)

<table>
<thead>
<tr>
<th>Researcher</th>
<th>Article</th>
<th>Area of research</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Belady and Lehman</strong></td>
<td>Statistical computer performance evaluation (1972)</td>
<td>• Present a <strong>model</strong> to <em>estimate the effort required to update</em> an old version and <em>release</em> a new version of the product</td>
</tr>
<tr>
<td><strong>Boehm et al</strong></td>
<td>Cost models for future software life cycle processes:COCOMO2.0 (1995)</td>
<td>• Extend the COCOMO model for development cost to maintenance effort by introducing a <strong>size change factor</strong></td>
</tr>
</tbody>
</table>
| **Jorgensen**          | Experience with the accuracy of software maintenance task effort prediction models (1995) | • Report experiences from the development and use of 11 different software maintenance effort prediction models  
                          |                                                        | • Models were developed using regression analysis, neural networks, pattern recognition ⇒ most accurate predictions: multiple regression analysis and pattern recognition |
| **Shepperd et al**     | Effort estimation using analogy (1996)                                   | • Use **analogy** to estimate staffing needs for software maintenance  
                          |                                                        | • Evaluate with six distinct datasets drawn from a range of different environments                          |
| **Niessink and Vliet** | Predicting maintenance effort with function points (1997)               | • Find **function points are not strongly correlated** with maintenance effort  
                          |                                                        | • Find **size of the component to be changed** has a **much larger impact** on maintenance effort than the size of the change itself |
## Related work (2/3)

<table>
<thead>
<tr>
<th>Researcher</th>
<th>Article</th>
<th>Area of research</th>
</tr>
</thead>
</table>
| Niessink and Vliet  | Two case studies in measuring software maintenance effort (1998)       | • Investigate possible *effort drivers* for software maintenance  
• Use standard statistical techniques, principal component analysis and multiple regression analysis, to analyze the datasets ⇒ conclude that the **existence of a consistently applied process is an important prerequisite** for a successful measurement of maintenance effort |
| Ramil and Lehman    | Metrics of software evolution as effort predictors-a case study (2000) | • Describe models that predict maintenance effort as a **function of a suite of metrics of software evolution** (case study of the evolution of the kernel of a mainframe operating system)  
• Find models based on **coarse granularity measures**, such as ‘subsystem counts’ |
| Polo et al          | Using code metrics to predict maintenance of legacy programs: A case study (2001) | • Based on the study of the **correlation of simple code metrics and maintenance effort**  
• Demonstrate that it is **possible to estimate** the maintenance effort in the **initial stages (maintenance contract)** of a maintenance project |
| Hayes et al         | A metrics-based software maintenance effort model (2004)               | • Derive a model for **estimating adaptive maintenance effort**  
• Find that the **number of lines of code (LOC) changed** and the **number of operators changed** are **strongly correlated with the maintenance effort** |
## Related work (3/3)

<table>
<thead>
<tr>
<th>Researcher</th>
<th>Article</th>
<th>Area of research</th>
</tr>
</thead>
</table>
| **Lucia et al** | • Effort estimation for corrective software maintenance (2002)  
• Assessing effort estimation models for corrective software maintenance through empirical studies (2005) | • Present a corrective maintenance effort estimation model  
• Use multiple linear regression analysis to construct the effort estimation models and validate them against real maintenance project data  
• Find that the performance of the maintenance effort model can be improved if the types of the different maintenance tasks are taken into account |
| **Hayes et al** | A regression analysis of measures of evolving systems (2005) | • Use maintenance effort to build a maintainability predictor model  
→ The model possesses predictive accuracy of up to 83% |

- Maintenance effort estimation
  - Ongoing study for many researchers
  - All performed on closed-source software
  ⇒ No reported studies of the maintenance effort estimation for open-source software
Open-source indirect maintenance effort estimation process

- Indirect maintenance effort estimation process (Six steps)
  - For loosely managed software

1. Identify indirect maintenance effort measure
2. Extract effort data and indirect effort data
3. Validate the correlation between maintenance effort and indirect maintenance effort
4. Extract indirect maintenance effort data
5. Build and validate indirect maintenance effort model
6. Predict indirect maintenance effort

Loosely managed open-source software

- Number of person-hours
- Source code change, etc.
- Related to maintenance effort
- Independent of management of the project

Strictly managed closed-source software

- Data of strong correlation
- Function of other related measures
- Check strong correlation

11/20
Indirect maintenance effort measures

- Software measures
  - Related to maintenance effort
  - Used to indirectly represent maintenance effort

<table>
<thead>
<tr>
<th>Source code change</th>
<th>Lag time</th>
<th>Defect tracking system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consider maintenance as the <em>activities</em> to change the source code</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Module level</td>
<td></td>
<td></td>
</tr>
<tr>
<td>: Count the number of modules <em>added, deleted, and modified</em> in one maintenance task</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• Line level</td>
<td></td>
<td></td>
</tr>
<tr>
<td>: Count the number of LOC <em>added, deleted, and modified</em></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time between *starting* a maintenance task and *closing* the task
- For *corrective* maintenance
  : Include the duration *from* when a *defect is reported* and the task is assigned, *until* the *defect is fixed* and the task is closed
- For *perfective* and *adaptive* maintenance
  : Include the duration *from* when a *maintenance request is submitted* and the task is assigned, *until* the *job is finished* and the task is closed
Empirical study of the NASA SEL projects (1/3)

- Data description
  - NASA SEL dataset was provided by the Data and Analysis Center for software (DACS)
    - Data include over 22,000 records of maintenance tasks
  - Software Engineering Laboratory (SEL)
    : Organization sponsored by the NASA GSFC
    : To investigate the software engineering processes

- Results
  - Collected measures from SEL projects

<table>
<thead>
<tr>
<th>Measure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Effort</td>
<td>Actual effort on one maintenance task measured in person-hours</td>
</tr>
<tr>
<td>Lag_time</td>
<td>The time between starting a maintenance task and closing the task. It is measure in days.</td>
</tr>
<tr>
<td>LOC_change</td>
<td>The number of lines of source code changed in one maintenance task</td>
</tr>
<tr>
<td>Module_change</td>
<td>The number of modules changed in one maintenance task</td>
</tr>
</tbody>
</table>
Empirical study of the NASA SEL projects (2/3)

- Results (cont’d)
  - Three null hypotheses
    - $H_{01}$: No linear relationship between the maintenance effort and the lag time
    - $H_{02}$: No linear relationship between the maintenance effort and the LOC changed
    - $H_{03}$: No linear relationship between the maintenance effort and the number of modules changed

- Correlations between ‘Effort’ and other measures

<table>
<thead>
<tr>
<th>Measures</th>
<th>Lag_time</th>
<th>LOC_change</th>
<th>Module_change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correlation coefficient</td>
<td>0.263</td>
<td>0.561</td>
<td>0.491</td>
</tr>
<tr>
<td>Number of datasets</td>
<td>22,592</td>
<td>980</td>
<td>980</td>
</tr>
<tr>
<td>$p$-value</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
<td>&lt;0.01</td>
</tr>
</tbody>
</table>

Statistically significant
⇒ Positive linear correlation
Empirical study of the NASA SEL projects (3/3)

- Analysis of empirical study
  - Lag time and source code change can also be determined in open-source software
  - Study shows weak correlations between lag time and maintenance effort
    - Need to reconsider whether lag time is a valid indirect maintenance effort measure
  - Problems with using lag time as *indirect maintenance effort*
    - Risk of over reporting maintenance effort
    - Granularity of lag time is usually *one day*
    - Different bugs usually have different severities
    - Lag time data *extracted from* the defect-tracking system
  - Merit of source code change
    - More independent of management of the maintenance process
    - Amount of change is only related to the task itself
    - Not affected by the open or closed property of the software
Empirical study of the LINUX project (1/4)

- **Overview**
  - Linux is one of the *most successful* open-source projects
  - However, *no research* has been done to study its maintenance effort
  - In this experiment,
    - Use *source code change* to indirectly represent maintenance effort
      - **Module** is referred to as ‘.c’ or ‘.h’ file
      - **Size** of the product is counted as thousand lines of code (*KLOC*), including comments
  - **Build two models** to predict the amount of source code change (indirect maintenance effort) to release a new version of LINUX
    - Source-code-based maintenance effort model
  - Available **data repositories** are source code and change log
    - Amount of source code change is counted using the LINUX cross reference tool lxr
    - Change log is used to extract the number of maintenance tasks for each version of LINUX

<table>
<thead>
<tr>
<th>Version</th>
<th>2.4</th>
<th>2.5</th>
<th>2.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subversions</td>
<td>2.4.1-2.4.31</td>
<td>2.5.1-2.5.75</td>
<td>2.6.0-2.6.14</td>
</tr>
<tr>
<td>Number of subversions</td>
<td>31</td>
<td>75</td>
<td>15</td>
</tr>
</tbody>
</table>

Use for model training

Use for model validating
Empirical study of the LINUX project (2/4)

- Source-code-based maintenance effort model
  - Collected measures of each subversion of LINUX

<table>
<thead>
<tr>
<th>Measure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_Line</td>
<td>Indirect maintenance effort on the system measured in the summation of KLOC added, deleted, and modified</td>
</tr>
<tr>
<td>E_Module</td>
<td>Indirect maintenance effort on the system measured in the summation of the number of modules added, deleted, and modified</td>
</tr>
<tr>
<td>NC</td>
<td>Number of corrective maintenance tasks on the system</td>
</tr>
<tr>
<td>NO</td>
<td>Number of other (perfective and adaptive) maintenance tasks on the system</td>
</tr>
<tr>
<td>T_Line</td>
<td>Total size of the system measured in KLOC</td>
</tr>
<tr>
<td>T_Module</td>
<td>Total size of the system measured in number of modules</td>
</tr>
<tr>
<td>K_Line</td>
<td>Kernel size of the system measured in KLOC</td>
</tr>
<tr>
<td>K_Module</td>
<td>Kernel size of the system measured in number of modules</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sel projects</th>
<th>Maint’ activities</th>
<th>Previous research</th>
<th>LINUX property</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Two consecutive subversions of LINUX</td>
<td>Determined according to the differences between the base version and the evolved version</td>
<td>Determined from the source code of the base version</td>
</tr>
<tr>
<td></td>
<td>Determined from the change log of the evolved version</td>
<td>Determined from the change log of the evolved version</td>
<td></td>
</tr>
</tbody>
</table>
Empirical study of the LINUX project (3/4)

- Source-code-based maintenance effort model (cont'd)
  1. \(E_{\text{Line}} = a_1NC + a_2NO\)
  2. \(E_{\text{Module}} = a_1T_{\text{Module}} + a_2K_{\text{Module}} + a_3NC + a_4NO\)

### Descriptive statistics of the collected measures

<table>
<thead>
<tr>
<th>Measure</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Mean</th>
<th>Standard deviation</th>
</tr>
</thead>
<tbody>
<tr>
<td>E(Line) (KLOC)</td>
<td>0.09</td>
<td>18.00</td>
<td>2.002</td>
<td>1.56</td>
</tr>
<tr>
<td>E(Module) (module)</td>
<td>18.00</td>
<td>5380.00</td>
<td>1320.90</td>
<td>1103.42</td>
</tr>
<tr>
<td>NC (task)</td>
<td>32.29</td>
<td>1320.90</td>
<td>121.17</td>
<td>428.68</td>
</tr>
<tr>
<td>NO (task)</td>
<td>58.91</td>
<td>1320.90</td>
<td>121.17</td>
<td>428.68</td>
</tr>
<tr>
<td>T(Line) (KLOC)</td>
<td>6.20</td>
<td>49.00</td>
<td>31.57</td>
<td>4.67</td>
</tr>
<tr>
<td>T(Module) (module)</td>
<td>49.00</td>
<td>6.17</td>
<td>31.57</td>
<td>4.67</td>
</tr>
</tbody>
</table>

- Data are spread widely
- **Statistically significant linear correlation** between
  - ‘E_Line’ and ‘NC’, and ‘NO’
  - ‘E_Module’ and the other four measures (‘T_Module’, ‘K_Module’, ‘NC’, and ‘NO’)

- Based on this observation,
  - **Build two indirect maintenance effort estimation models**
    - To represent the *line level* and *module level* changes

### Correlations between dependent variables and independent variables

<table>
<thead>
<tr>
<th></th>
<th>T_Line</th>
<th>K_Line</th>
<th>T_Module</th>
<th>K_Module</th>
<th>NC</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>E_Line</td>
<td>0.080</td>
<td>0.022</td>
<td>0.092</td>
<td>0.037</td>
<td>0.647</td>
<td>0.594</td>
</tr>
<tr>
<td>E_Module</td>
<td>0.050</td>
<td>0.031</td>
<td>0.423</td>
<td>0.481</td>
<td>0.621</td>
<td>0.530</td>
</tr>
</tbody>
</table>
Empirical study of the LINUX Project (4/4)

- Source-code-based maintenance effort model (cont’d)
  - Linear regression analysis of the two indirect maintenance effort models

<table>
<thead>
<tr>
<th>Model</th>
<th>Dependent variable</th>
<th>Independent variable</th>
<th>(a_i)</th>
<th>(p)-value</th>
<th>(R^2)</th>
<th>Adjusted (R^2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>E_Line</td>
<td>NC</td>
<td>0.93</td>
<td>&lt;0.001</td>
<td>0.510</td>
<td>0.498</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NO</td>
<td>0.005</td>
<td>&lt;0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>E_Module</td>
<td>T_Module K_Module</td>
<td>0.022</td>
<td>&lt;0.001</td>
<td>0.817</td>
<td>0.803</td>
</tr>
<tr>
<td></td>
<td></td>
<td>NC</td>
<td>11.87</td>
<td>&lt;0.001</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>NO</td>
<td>5.24</td>
<td>&lt;0.001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Performances of two models

<table>
<thead>
<tr>
<th></th>
<th>Model (1)</th>
<th>Model (2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(PRED_{25})</td>
<td>0.33</td>
<td>0.53</td>
</tr>
<tr>
<td>(PRED_{50})</td>
<td>0.60</td>
<td>0.87</td>
</tr>
</tbody>
</table>

\(R^2\): Squared multiple correlation coefficient

\(\ast\) Number of modules is more accurate

\(\ast\) \(PRED_x\) = percentage of the prediction with relative error less than or equal to \(x\)
Conclusion & discussion

**Conclusion**
- Propose a process for indirectly predicting the maintenance effort needed for open-source software
  - NASA SEL project
    - Find that linear relations exist between maintenance effort and lag time, and maintenance effort and source code change
  - LINUX project
    - Build and validate two source-code-based indirect maintenance effort models
      - Effort of module level change can represent maintenance effort indirectly

**Discussion**
- Need to study other closed-source and open-source projects
  - To determine more measures
- Need to increase statistically validity
  - Some correlations are not strong
    - This may result in a type I error (mistakenly rejects a null hypothesis)
Considering encapsulations

- **Laws of software evolution**
  - Continuing change
  - Increasing complexity
  - Self regulation
  - Conservation of organisational stability
  - Conservation of familiarity
  - Continuing growth
  - Declining quality
  - Feedback system

- **FEAST Hypothesis**
  - Represents the encapsulation and formulation of the 1970s observation
  - Their reflection in the laws outlined above and process insight developed more recently

- **FEAST project**
  - Rooted in hypotheses
  - *Seeks to investigate* the role and influence of feedback in the *evolution of E-type software systems* and on the *improvement of the software process*

Background

- **Planning**
  - Important factor for successful software maintenance process
  - Effective planning
    - Based on accurate effort estimation of the maintenance task

- **Effort**
  - Important metric to measure the software process
  - Spent on maintenance partly reflects the complexity of the maintenance process

- **Accurate maintenance effort estimation**
  - Based on an accurate maintenance effort model
    - Maintenance effort usually represented as the number of person-hours required to perform a maintenance task
Defect tracking system

- Lag time data
  - Can be extracted from the defect tracking system, CVS, or change log
    - CVS: Concurrent Version System

- Bugzilla (example)
  - Server software designed to help you manage software development
  - "Defect Tracking System" or "Bug-Tracking System"
    - Defect Tracking Systems allow individual or groups of developers to keep track of outstanding bugs in their product effectively
  - Contains
    - The data an error is reported
    - The data the job is assigned
    - The data the job is closed
To test these hypothesis

- Need to calculate the correlations
  - Summarize the strength of the relationship between the two variables X and Y

- Correlation coefficient
  - Pearson’s correlation coefficient
    - For valid, variables X and Y both need to be normally distributed
    - However, it is unlikely that the lag time or source code change X or the maintenance effort Y has normal distribution
  - Spearman’s rank correlation coefficient
    - If the rank correlation coefficient proves to be statistically significant at, say, 0.01 level, we will reject the null hypothesis and accept the alternate hypothesis
Empirical study of the NASA SEL projects

- Results
  - Scatter plots
Cross-Referencing LINUX

Motivation
- Linux Cross-Reference project is the testbed application of a general hypertext cross-referencing tool (Or the other way around)

Techniques
- Index generator is written in Perl and relies heavily on Perl's regular expression facilities
- Algorithm used is very brute force and extremely sloppy

Availability
- Source code for the LXR engine is of course available
- Released under the GNU Copyleft license
Empirical study of the LINUX project

- Results
  - Scatter plots

Model (1)

Model (2)
Threats and limitations (1/2)

- **Statistical validity**
  - Strength of the correlations between dependent variables and independent variables
    - Some correlations are not strong
      - This may result in a type I error (mistakenly rejects a null hypothesis)
      - To reduce this threat, future research should study other independent variables that are strongly correlated to the dependent variables to build the prediction models

<table>
<thead>
<tr>
<th></th>
<th>T_Line</th>
<th>K_Line</th>
<th>T_Module</th>
<th>K_Module</th>
<th>NC</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>E_Line</strong></td>
<td>0.080</td>
<td>0.022</td>
<td>0.092</td>
<td>0.037</td>
<td>0.647</td>
<td>0.594</td>
</tr>
<tr>
<td><strong>E_Module</strong></td>
<td>0.050</td>
<td>0.031</td>
<td><strong>0.423</strong></td>
<td>0.481</td>
<td>0.621</td>
<td>0.530</td>
</tr>
</tbody>
</table>

- **Internal threats**
  - Consider *module-level change*
  - Consider *line-level change*
  - Consider *the number of tasks*
Threats and limitations (2/2)

- **Management differences**
  - Correlations between maintenance effort and the indirect measures determined in closed-source software
    - May not exist in open-source software
  - To reduce this threat
    - Future research should study the management similarities/differences between closed-source software and open-source software

- **How to discard outlier**
  - These measures are not limited to a certain value
  - Cannot simply confirm an outlier just
    - Because the measurement is far from the mean value
  - **If more information** about the project and the supporting data is available
    - Can remove some confirmed inaccurate data (such as those that result in outliers),
    - Performance of the models will be improved