Estimating the Effective Size of Auto-Generated Code in a Large Software Project

Pam McDonald
THAAD Project Office

Dan Strickland
Dynetics, INC.

Charles Wildman
THAAD Project Office

DISTRIBUTION A: Approved for public release; distribution unlimited.
Overview

- Background
- Terminology
- Nature of Testing
- Inputs
- Testing vs. Non-Testing Effort
- Synthetic Estimate Cases
- Additional Areas of Interest
Background

- Massive amounts of software can be generated by code generation languages or tools
- Graphical User Interfaces (GUIs) or high-order languages can generate large amounts of code saving time and effort
- Auto-generated code counts can be over-inflated making inaccurate counts for size, effort, productivity, and defect density
- Parametric models like COCOMO II and SEER-SEM don’t have the ability to account for auto-generated code sizes
- GOAL: Find a solution that correctly estimates size/effort of Auto-Generated Code
• **generating** – related to the development, input effort used to auto-generate code

• **resultant** – related to the code that is output by the auto-generation

• **synthetic** – the normalized estimate of effort that takes into account the effort in non-testing activities (generating language) and testing activities (resultant language)
Determine the type of testing to be performed on the code

- No testing on resultant code
  - Product constrained by time-to-market issues
  - Testing to be performed on generating language only
  - Uses generating estimate as estimate for size in cost model
  - No calculation of synthetic effort

- Testing on resultant code required
  - Product is mission critical with potential loss of life risks
  - Non-testing activities are performed in the generating language
  - Testing activities are performed in the resultant language
  - Synthetic effort must be calculated for size input to cost models
**Inputs**

- Generating Language (e.g. Visual Basic, IDL, PowerBuilder)
- Resultant Language (e.g. Ada95, C++, FORTRAN)
- Generating Estimate (in SLOC or a measure convertible to SLOC)
- Resultant Estimate (in SLOC or a measure convertible to SLOC)
- Function Point Estimate (one estimate works for generating and resultant languages)
Testing vs. Non-Testing Efforts – COCOMO II Phases

- Requirements, Preliminary Design and Detailed Design are non-test related; Integration Testing is test related
- Code and Unit Testing contains activities in both areas
Testing vs. Non-Testing Efforts - Code and Unit Testing Phase

- Code and Unit Test broken up into default activities
- Requirements, Design, and Programming are non-test activities
- Test Plans and V & V are test activities
- Project Office, CM/QA, and Manuals are non-specified - split evenly between both efforts
Testing vs. Non-Testing Effort - Synthetic SLOC

COCOMO II Effort Distribution

- Normalized effort values for non-test and test efforts
- 67.2% of development size counted in synthetic size
- 32.8% of translated size counted in synthetic size
- Example: 10KSLOC translated to 50KSLOC
  - 10,000 (0.672) + 50,000 (0.328) = 23,120 SLOC
- Synthetic SLOC Rule of Thumb: 66.7% non-test, 33.3% test
### Using the Programming Languages Table

<table>
<thead>
<tr>
<th>Language</th>
<th>Average Statements per FP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ada95</td>
<td>49</td>
</tr>
<tr>
<td>C</td>
<td>128</td>
</tr>
<tr>
<td>C++</td>
<td>53</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>ZLISP</td>
<td>64</td>
</tr>
</tbody>
</table>

- Programming Languages Table (Capers Jones) allows for a rough order of magnitude conversion between SLOC and Function Points
- The table allows for taxation of language based on a common denominator (Function Points)
Synthetic Estimate – Case 1

- Inputs: Generating Language Known, Resultant Language Known, Generating Estimate in SLOC
- Formula:
  - Synthetic SLOC = (66.7%) * (Generating SLOC) + (33.3%) * (Resultant Language PLT Value / Generating Language PLT Value) * (Generating SLOC)
- Example:
  - 5000 SLOC of IDL will be used to auto-generate a software item in Ada95. What size input should be used?
  - Synthetic SLOC = (66.7%) * (5,000) + (33.3%) * (49/20) * (5000)
  - Synthetic SLOC = 3335 + 4080
  - Synthetic SLOC = 7415
Synthetic Estimate – Case 2

- Inputs: Generating Language Known, Resultant Language Known, Resultant Estimate in SLOC
- Formula:
  - Synthetic SLOC = (66.7%) * (Generating Language PLT / Resultant Language PLT) * (Resultant SLOC) + (33.3%) * (Resultant SLOC)
- Example:
  - 50000 SLOC of C will be auto-generated using PowerBuilder. What size input should be used?
  - Synthetic SLOC = (66.7%) * (16/128) * (50000) + (33.3%) * (50000)
  - Synthetic SLOC = 4169 + 16650
  - Synthetic SLOC = 20819
Synthetic Estimate – Case 3

- Inputs: Generating Language Known, Resultant Language Known, Estimate in Function Points

- Formula:
  - Synthetic SLOC = (66.7%) * (Generating Language PLT * Function Point Estimate) + (33.3%) * (Resultant Language PLT * Function Point Estimate)

- Example:
  - A 4GL auto-generation language will be used to produce a software item in C++. There are 450 function points estimated. What size input should be used?
  - Synthetic SLOC = (66.7%) * (20) * (450) + (33.3%) * (53) * (450)
  - Synthetic SLOC = 6003 + 7943
  - Synthetic SLOC = 13946
Synthetic Estimate – Case 4

- Inputs: Generating Estimate in SLOC, Resultant Estimate in SLOC
- Formula:
  - Synthetic SLOC = (66.7%) * (Generating SLOC) + (33.3%) * (Resultant SLOC)
- Example:
  - An estimated 3500 statements in IDL auto-generate 50000 Ada95 SLOC. What size input should be used?
  - Synthetic SLOC = (66.7%) * (3500) + (33.3%) * (50000)
  - Synthetic SLOC = 2335 + 16650
  - Synthetic SLOC = 18985
Adapted Code

- Adapted code is pre-existing code modified to be reused with new development.
- Modification percentage is made up of percent re-design, percent re-coding, and percent re-testing/integration factors.
- If the adapted code is in the same language as the generating language, more attention must be paid to the percent re-design and percent re-coding factors.
- If the adapted code is in the same language as the resultant language, more attention must be paid to the percent re-test and integration factor.
- Adapted SLOC is added to Synthetic SLOC and any new, non-auto generated SLOC for use as the size input when reuse exists.
- \[ \text{Size} = \text{Adapted SLOC} + \text{Synthetic SLOC} + \text{New [Non-AGC]} \]
• COCOMO II Maintenance Model:
  – Major size input is Total Delivered SLOC (TSLOC)
  – Major modifier is Annual Change Traffic Percentage
• When there is auto-generated code, the TSLOC is at least as large as
  the resultant SLOC, potentially inflating the estimate
• Annual Change Traffic is driven by changes to code in both
  generating and resultant languages
• Changes to generating language mean complete regeneration and
  retesting of the code, inflating the change traffic
• Changes to resultant language could be lengthy and time-consuming
  given the size-to-effort ratio and potential unfamiliarity with the
  code
• Continued research is needed in this topic
Conclusion

• The use of code generators is becoming more prevalent in large software projects, but parametric models don’t have the ability to properly model the effort of auto-generation

• Using generating and resultant languages, specific inputs, and testing vs. non-testing effort percentages allows for the calculation of a Synthetic size to be used for effort calculations

• Further research is needed into how auto-generated code affects maintenance

*Using the methods described here, auto-generated code effort can be accounted for in parametric estimates*