# **PONDER 09 : SIZE ESTIMATION**

##### **Due Saturday at 5:00 PM MST**

A key aspect of size estimation is using the resulting data to make an estimation of how long it will take to write code fulfilling an SI. How long does it take for you to write code? In order to get to the bottom of this, we will perform a collection of experiments to gather data on our code writing speed.

We will first estimate the effective size of the code to be written using the process described in the reading. Next we will measure our design, code writing, and testing efficiency by making three SI changes to a program designed to play Tic-Tac-Toe.

**There are 4 sections. Everyone pick one section and pick one part of the conclusion to do.**

## **Three SIs-Tiffany**

**I added roughly 120 lines of code and it took me about 1.5 hours to complete the modifications to all three SIs. I’ll upload the code later.**

We will compute the effective size and then measure the actual size of three changes to an existing, working program. Start with the code provided at:

/home/cs416/week09/tic-tac-toe.cpp

This program loads a game of Tic-Tac-Toe, displays the state of the board on the screen, and writes the board to a new file. These three SIs are to be done in sequence: each SI is to build upon the work of the previous SI. The three SIs are to: 1) replace literals with macros, 2) write a function determining if someone has won the game, and 3) make the game work with three dimensional boards.

### SI 1: Macros

In the source-code for Tic-Tac-Toe, there are several functions assuming that the pieces in the Tic-Tac-Toe game are either 'X', 'O', or a '.' (displayed as a space to the user). Please modify this code so there are #define macros rather than the hard-coded 'X', 'O', and '.'.

Additionally, create a #define for the size of the board rather than using the literal integer 3. Everything should work if this is changed to a 2 for a 2x2 board or a 5 for a 5x5 board. Recall that the code needs to be tested before it counts as completed.

**Took 15 minutes. I added 4 lines of code and modified other lines.**

### SI 2: Winning

Modify the code from SI 1 to indicate who won the game. In this file, you will find a function called didWin() that is currently stubbed out. Write the necessary code to determine if X did in fact win this game. This occurs if either the X player or the O player has a 3-in-a-row or a diagonal. Note that the size of the board does not have to be 3x3. (See the change from Experiment 1). If the board is 4x4, then the user must get 4-in-a-row or one of the diagonals.

**Took 30 minutes. I added 6 lines of code. This function isn’t working how it’s suppose to. I’m not sure how it’s suppose to work. The instructions aren’t clear enough.**

### SI 3: Three Dimensions

Start with the code from SI 2. Modify the game so it can read, write, display, and determine whether a given player has won for the 3D version of the game. An example of the output is:

**This took 45 minutes. I added 114 lines of code. It ain't pretty but it works :) ;**

## **Estimate Effective Size**

This is done in five steps:

1. First, estimate the effective size of the three changes using the SLOC method described in the first part of the reading. Consider each change to be an SI. In other words, you are to calculate *Snew*, *Smod*, and *Sreused*. For each of these values, compute *Smin*, *Sml*, and*Smax*. From these, you can compute *Smean* from equation 6-1. Please make sure you fully explain where each number comes from; this justification is a major part of your grade for this item.  
   Snew = 48   
   Smod = 50  
   Sreused = 800  
   The first SI seems simple so the numbers wouldnt be that drastic.  
   Smin =3, Sml = 5, Smax = 10. I chose these numbers because I feel it could be done in 3 lines, but is most likely done in 5 and at most, done in 10. Using the equation, we compute Smean to equal (3+20+10)/6 = 5.5  
     
   The second seems a bit more complicated.   
   Smin = 5, Sml = 12, Smax = 17. While I think this could be complicated, I think it could be done in 5. If I understand correctly, we just display who won and so on. The max this would take I think is 17 LOCs. This makes Smean = (5+48+17)6 = 11.67
2. Second, compute *Fdes*, *Fimp*, and *Ftest*. Be careful! These are not well defined in the reading (section 6.4.1). If, for example, you were computing *Fdes*, for SI 2, you would ask yourself "how much easier would it be to design this code if I were to start from scratch verses to start from the existing code." If the existing code were to provide no advantage, the answer would be *Fdes* = 1.0. On the other hand, if the existing code requires zero re-design, then it would be *Fdes* = 0.0.  
   Fdes = .3 I think that there will have to be some change to the code to include the desired changes.  
   Fimp = .2 The implementing of the code I dont think would be too difficult if designed properly.  
   Ftest = .4 Testing is always a hard part. It can take some time to make sure the software is working properly.
3. Once we have these *F* values, we can compute *SAF*.
4. Next, compute *Seffective*.  
   Se = 48 + 800 (.4(.3) + .25(.2) + .35(.4)) = 592
5. The final step is to estimate how long it will take to complete this task. Assume a senior Computer Science or Software Engineering student can write 100,000 lines of code a year (or 50 SLOC an hour). The equation is *T = Seffective / 50).  
   11.84 = time*

## **Experiment**

To determine the speed in which we design, write code, and test, we will make three modifications to an existing program representing the game of Tic-Tac-Toe. You are to carefully record the amount of code you write or modified, as well as how long it took for you to accomplish each activity. Each experiment will be divided into three stages: design, implement, and test. This conforms to the textbook's definition of a SLOC:

SLOC is a measure of the software size. A simple, concise definition of SLOC is any software statement that must be **designed**, **documented**, and **tested.**

In the case of this experiment, the documented requirement will be met with comments.

### Design

The design phase begins when the requirements are read and ends when implementation begins. This design phase may consist of several activities: carefully reading the requirements, familiarizing yourself with the code, sketching out a solution using pseudocode or other design tools, or even just thinking through the problem. Do not leave this stage until you are sure you know how to implement the requirements. That being said, you may have to change your design as unexpected things come up in the implementation phase.

Please make sure you take careful note of how long it takes for you to complete the design phase of each of the three SIs.

### Implementation

The implementation phase includes actually writing the software so the requirements are met. All the code writing, compiling, and preliminary testing are included in the implementation phase. When finished, you should have a working solution meeting the requirements that works in the most common scenarios. Adding comments is part of the implementation phase.

As with the design phase, you need to carefully record how much time was spent in the implementation phase. When you are done, you will need to record how many lines of code were added according to the definition of a SLOC provided in the text. Also, record how many lines of code were deleted or modified. The tool diff may come in handy here.

### Test

The testing phase involves manually testing the software, writing tools and test cases to test the software, and fixing bugs encountered during testing. When finished, you should be relatively confident that all the bugs have been found. You may wish to refer to the white-box and black-box testing strategies we learned about earlier in the semester.

As with the design and implementation phase, please take careful note of how much time was spent on the testing phase. For the case of this experiment, do not add code changes to your SLOC counts from the implementation phase.

## **Conclusions-Everyone pick one**

When finished with the calculations (the first part) and the experimental data (the second part), it is necessary to see how the two sets of values compare. In other words, how accurately did the calculations predict how you actually write code? We will look at three aspects: the size estimations, the time estimations, and the programmer efficiency.

### **Size**

For each of the SIs, compare the computed *Seffective* with the actual changes you made to the code. Please note any discrepancies. Based on the models presented in the reading and on your experimental observations, describe the accuracy of the models and what could be done to improve them.

### **Time**

For each of the three experiments, record the *Tdes*, the *Timp*, and the *Ttest*. Calculate the ratio of time spent in each of these activities. How do these calculations match up with the ratio presented in the reading (from section 6.4.1)?

### **Efficiency-Tiffany**

Compute the amount of time taken to modify a single line of code in SI 1 by dividing the amount of time spent writing code by the number of lines of code. Do the same thing for the amount of time it takes to write a single line of code in SI 2 and for SI 3. How do these measurements compare with the numbers in the text (approximately 1.2 SLOC / hour)? Is there a significant difference in your *Ck* values for the three SIs?