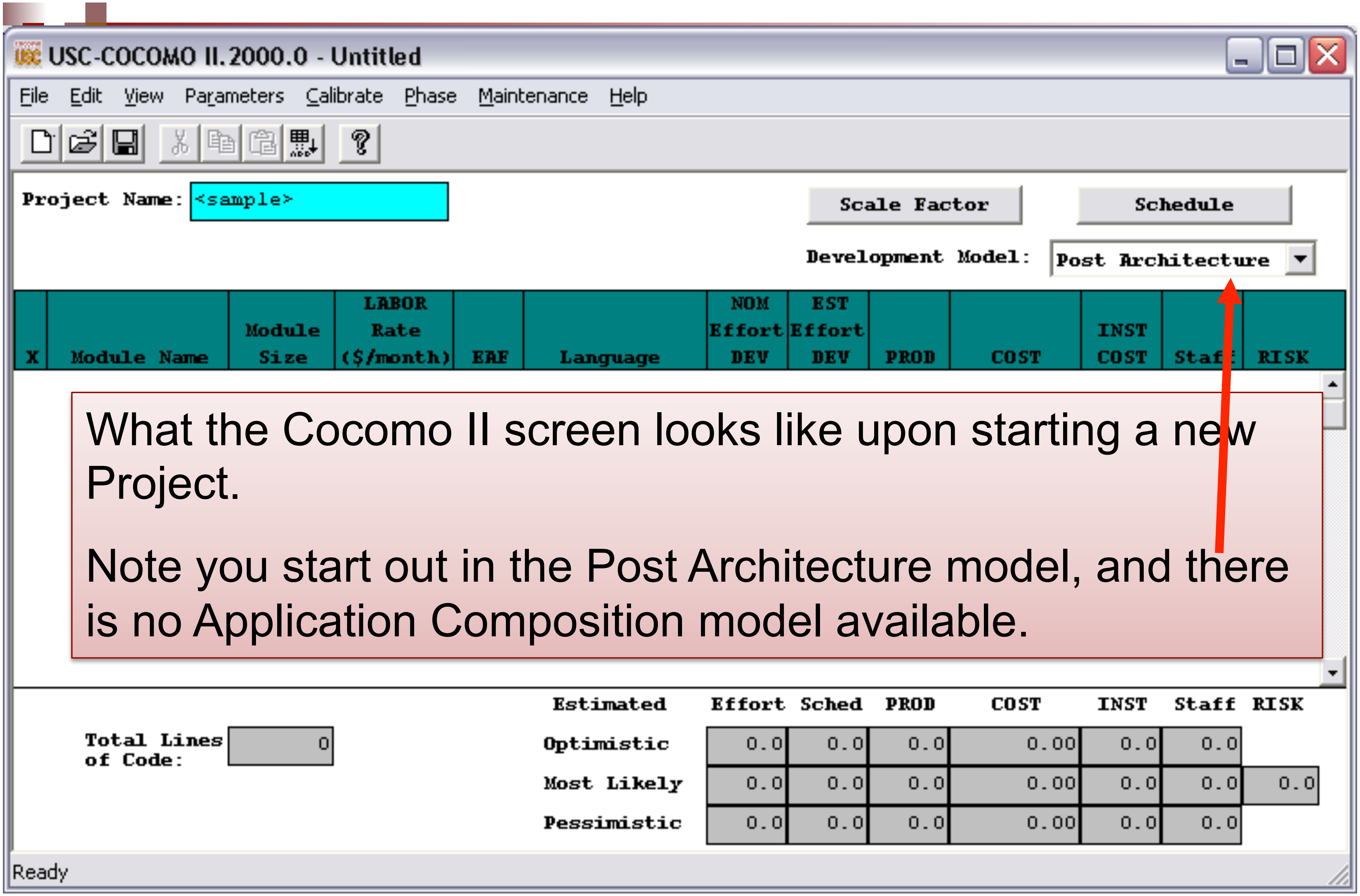


USC-COCOMO II.2000.0 - Untitled

Project Name: <sample>

Development Model: Post Architecture

X	Module Name	Module Size	LABOR Rate ($/month)	EAF	Language	NOM Effort DEV	EST Effort DEV	PROD	COST	INST COST	Staff	RISK

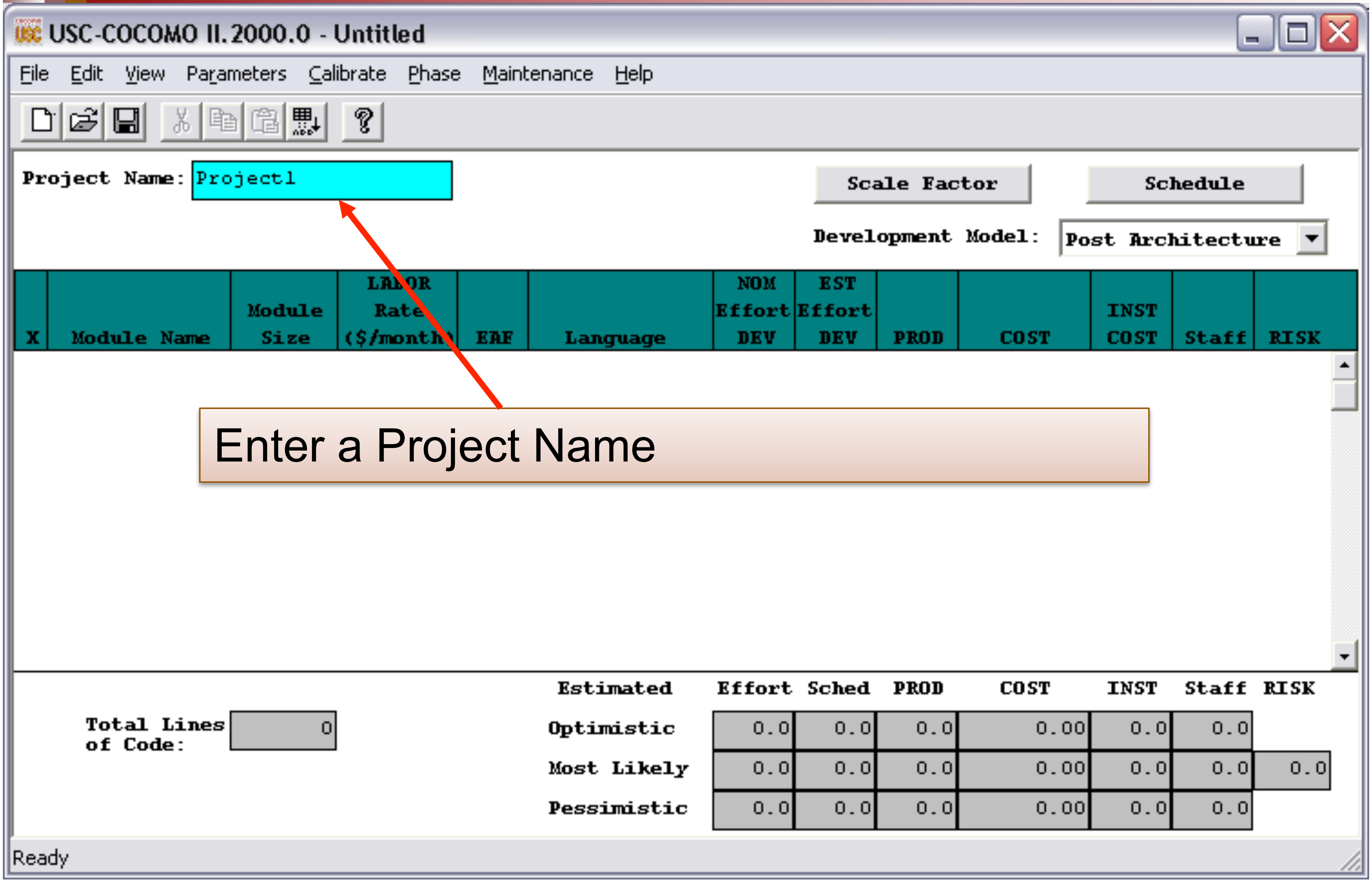
What the Cocomo II screen looks like upon starting a new Project.

Note you start out in the Post Architecture model, and there is no Application Composition model available.

Total Lines of Code: 0

Estimated	Effort	Sched	PROD	COST	INST	Staff	RISK
Optimistic	0.0	0.0	0.0	0.00	0.0	0.0	
Most Likely	0.0	0.0	0.0	0.00	0.0	0.0	0.0
Pessimistic	0.0	0.0	0.0	0.00	0.0	0.0	

Ready



Project Name: Project1

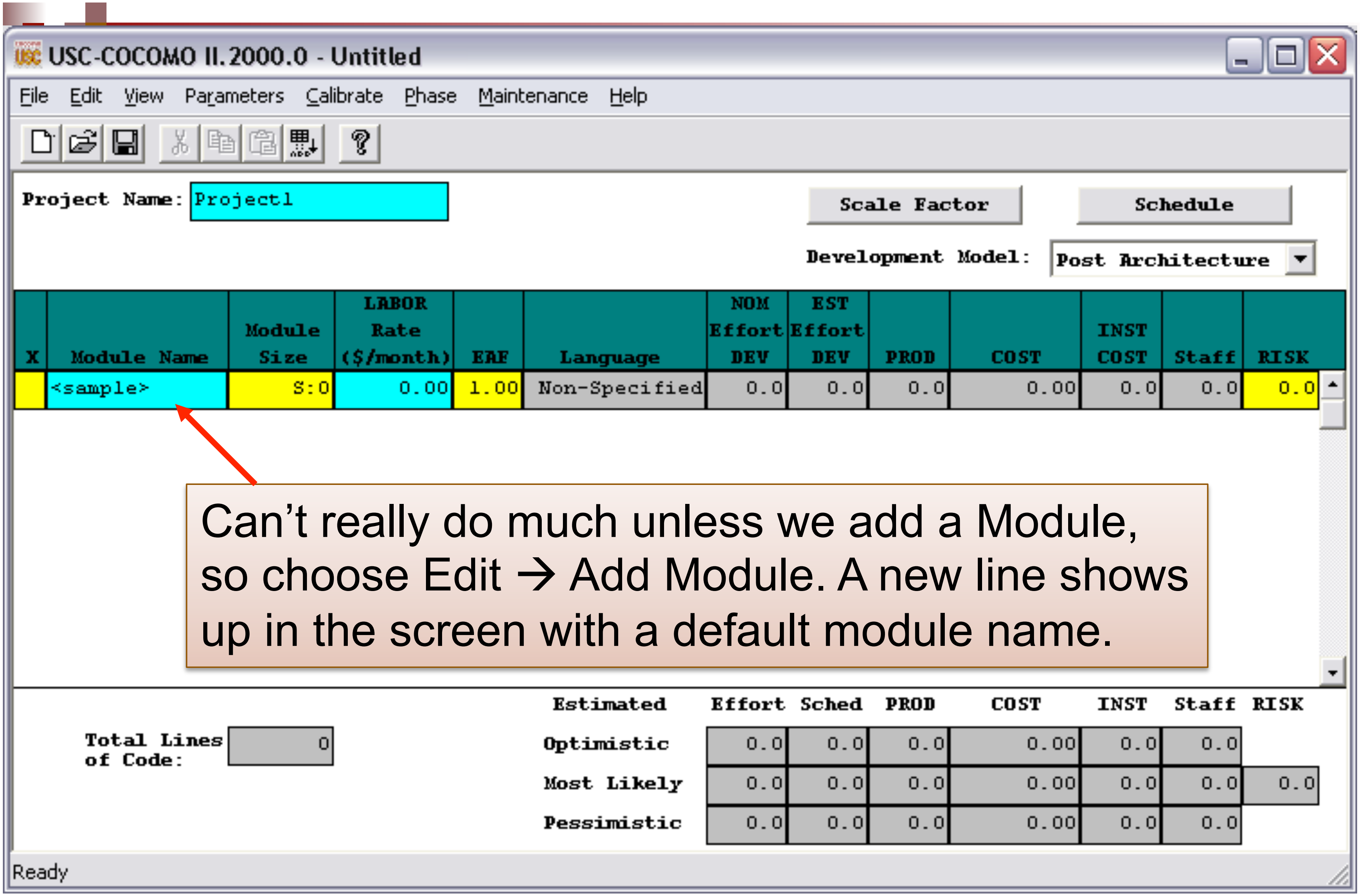
Development Model: Post Architecture

X	Module Name	Module Size	LABOR Rate ($/month)	EAF	Language	NOM Effort DEV	EST Effort DEV	PROD	COST	INST COST	Staff	RISK

Total Lines of Code: 0

Estimated	Effort	Sched	PROD	COST	INST	Staff	RISK
Optimistic	0.0	0.0	0.0	0.00	0.0	0.0	
Most Likely	0.0	0.0	0.0	0.00	0.0	0.0	0.0
Pessimistic	0.0	0.0	0.0	0.00	0.0	0.0	

Enter a Project Name



Project Name: Project1

Development Model: Post Architecture

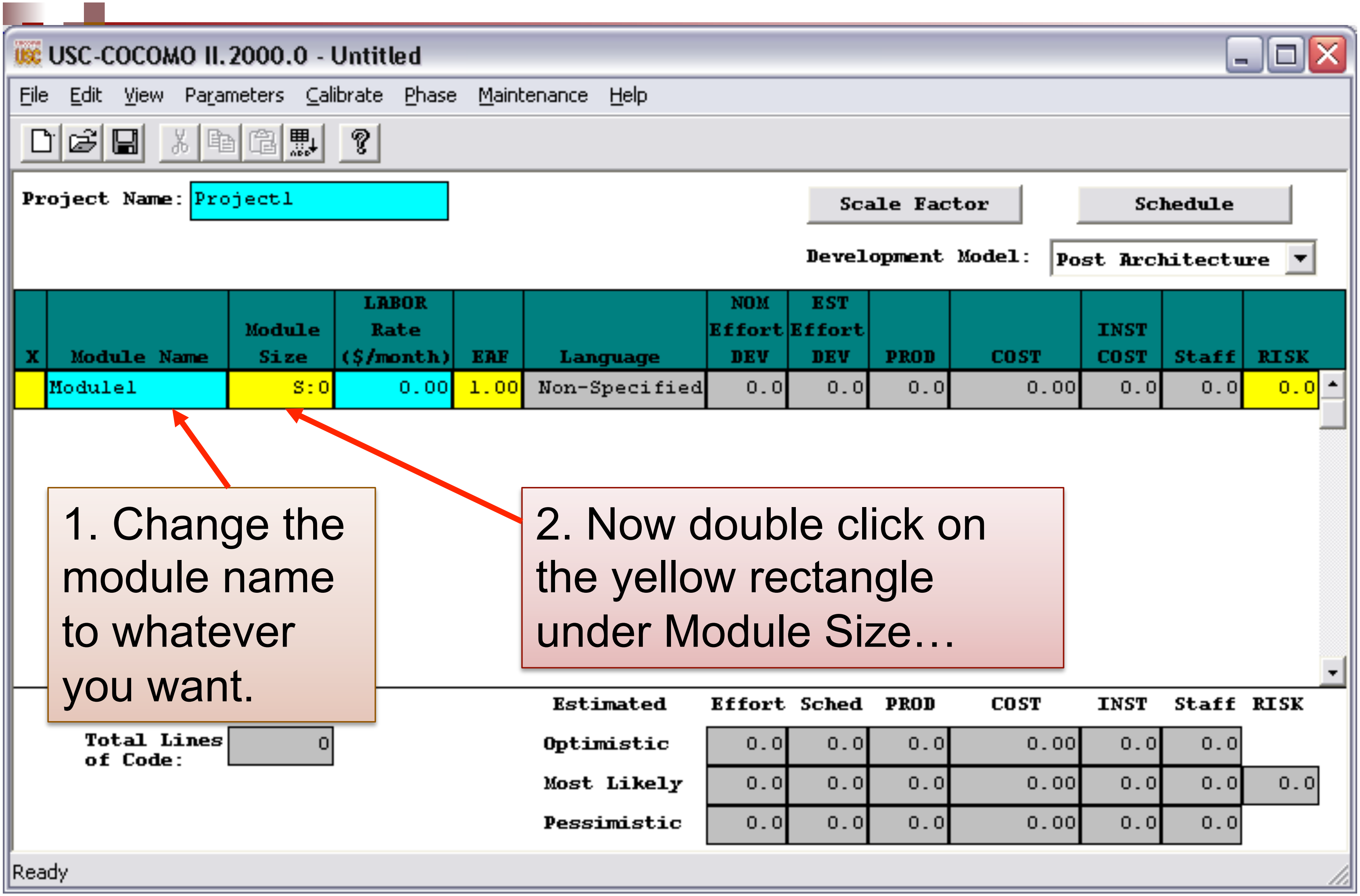
X	Module Name	Module Size	LABOR Rate ($/month)	EAF	Language	NOM Effort DEV	EST Effort DEV	PROD	COST	INST COST	Staff	RISK
	<sample>	S:0	0.00	1.00	Non-Specified	0.0	0.0	0.0	0.00	0.0	0.0	0.0

Total Lines of Code: 0

Estimated	Effort	Sched	PROD	COST	INST	Staff	RISK
Optimistic	0.0	0.0	0.0	0.00	0.0	0.0	
Most Likely	0.0	0.0	0.0	0.00	0.0	0.0	0.0
Pessimistic	0.0	0.0	0.0	0.00	0.0	0.0	

Ready

Can’t really do much unless we add a Module, so choose Edit → Add Module. A new line shows up in the screen with a default module name.


USC-COCOMO II.2000.0 - Untitled
File Edit View Parameters Calibrate Phase Maintenance Help
Scale Factor
Schedule
Ready

Project Name: Project1

Development Model: Post Architecture

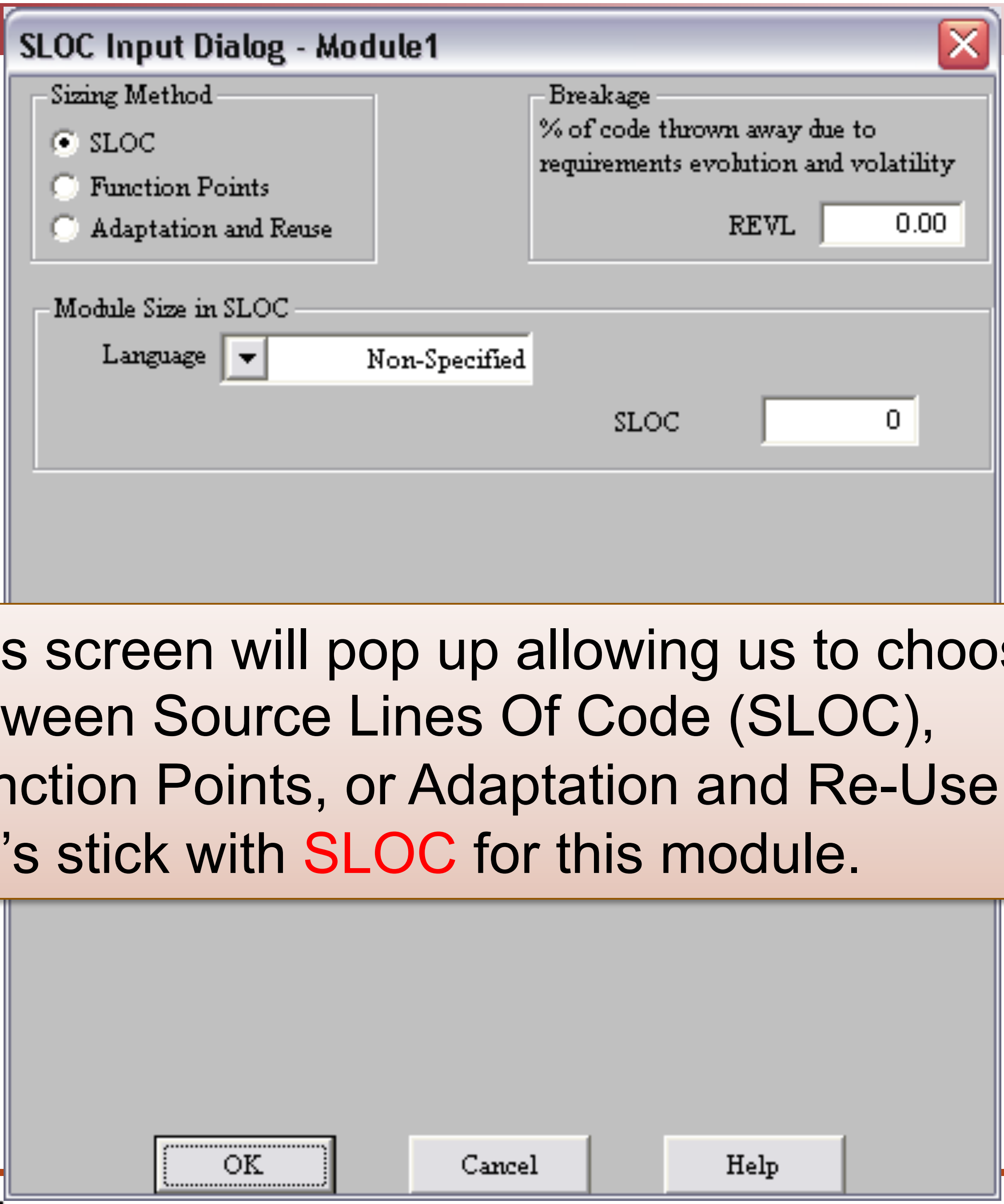
X	Module Name	Module Size	LABOR Rate ($/month)	EAF	Language	NOM Effort DEV	EST Effort DEV	PROD	COST	INST COST	Staff	RISK
	Module1	S:0	0.00	1.00	Non-Specified	0.0	0.0	0.0	0.00	0.0	0.0	0.0

Total Lines of Code: 0

Estimated	Effort	Sched	PROD	COST	INST	Staff	RISK
Optimistic	0.0	0.0	0.0	0.00	0.0	0.0	
Most Likely	0.0	0.0	0.0	0.00	0.0	0.0	0.0
Pessimistic	0.0	0.0	0.0	0.00	0.0	0.0	

1. Change the module name to whatever you want.

2. Now double click on the yellow rectangle under Module Size…



SLOC Input Dialog - Module1

Sizing Method

● SLOC

○ Function Points

○ Adaptation and Reuse

Breakage

% of code thrown away due to requirements evolution and volatility

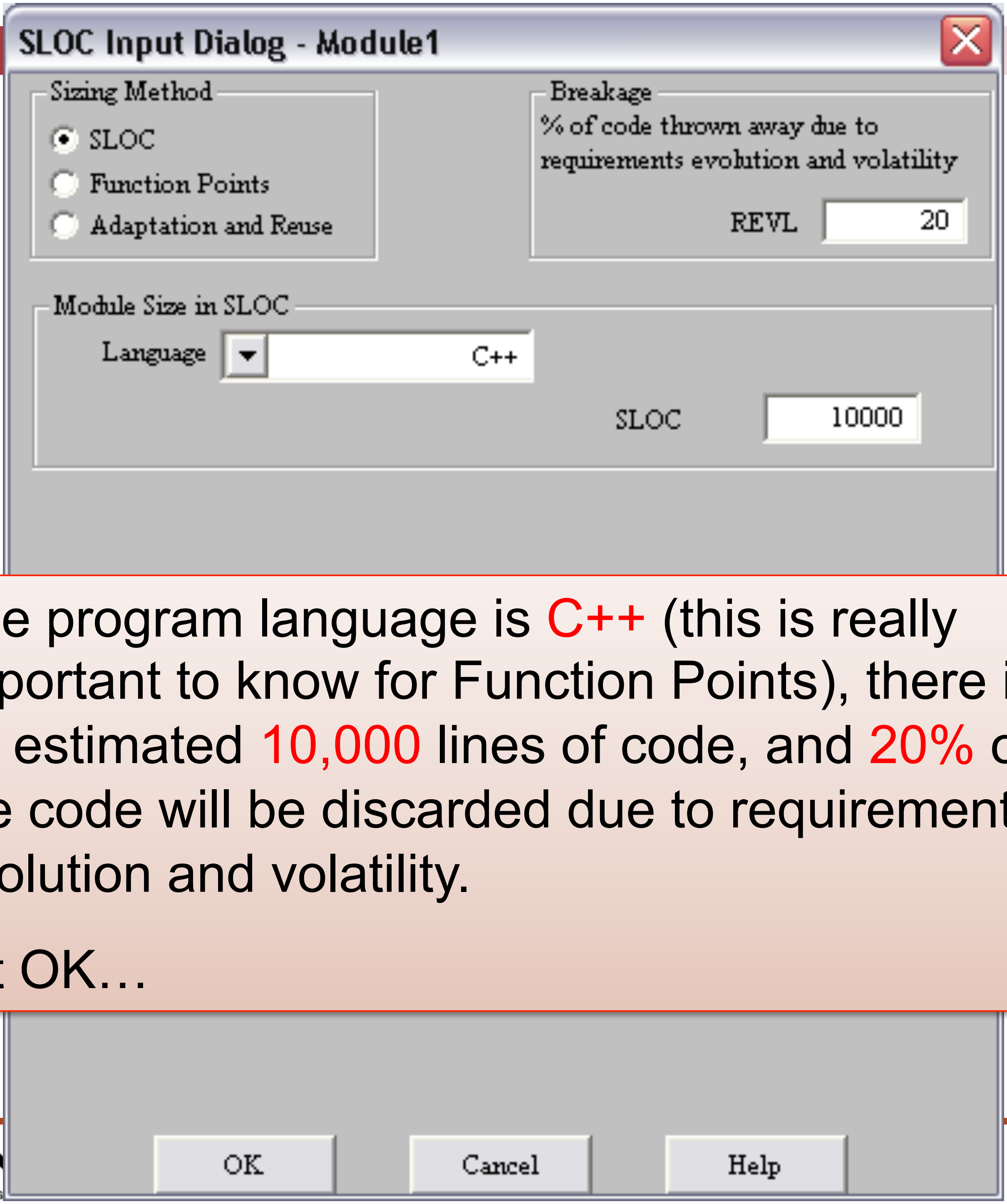
REVL 0.00

Module Size in SLOC

Language Non-Specified

SLOC 0

This screen will pop up allowing us to choose between Source Lines Of Code (SLOC), Function Points, or Adaptation and Re-Use. Let's stick with SLOC for this module.



SLOC Input Dialog - Module1

Sizing Method

● SLOC

○ Function Points

○ Adaptation and Reuse

Breakage

% of code thrown away due to requirements evolution and volatility

REVL: 20

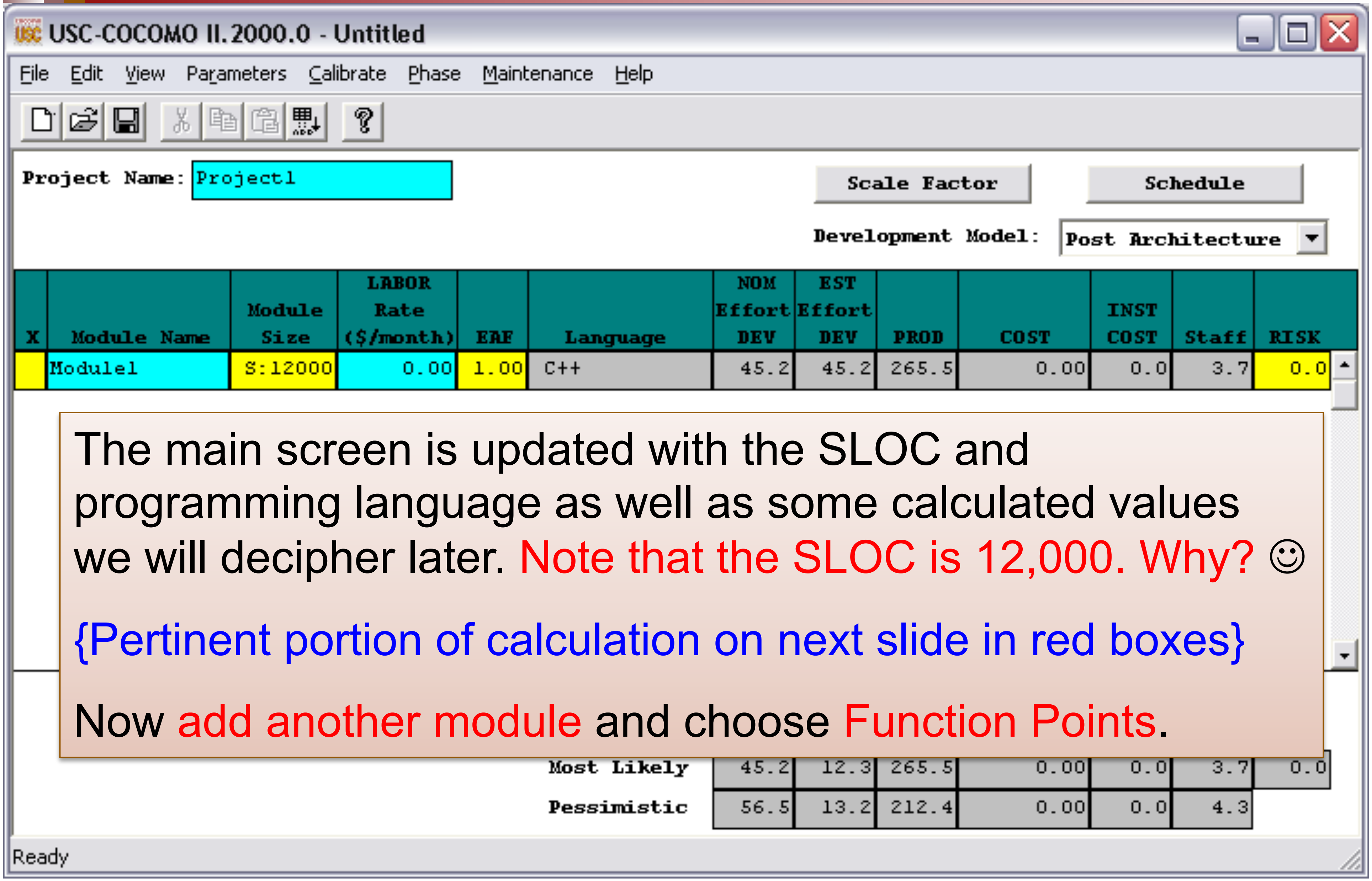
Module Size in SLOC

Language: C++

SLOC: 10000

The program language is C++ (this is really important to know for Function Points), there is an estimated 10,000 lines of code, and 20% of the code will be discarded due to requirements evolution and volatility.

Hit OK…



Project Name: Project1

Development Model: Post Architecture

X	Module Name	Module Size	LABOR Rate ($/month)	EAF	Language	NOM Effort DEV	EST Effort DEV	PROD	COST	INST COST	Staff	RISK
	Module1	S:12000	0.00	1.00	C++	45.2	45.2	265.5	0.00	0.0	3.7	0.0

Most Likely	45.2	12.3	265.5	0.00	0.0	3.7	0.0
Pessimistic	56.5	13.2	212.4	0.00	0.0	4.3	

Ready

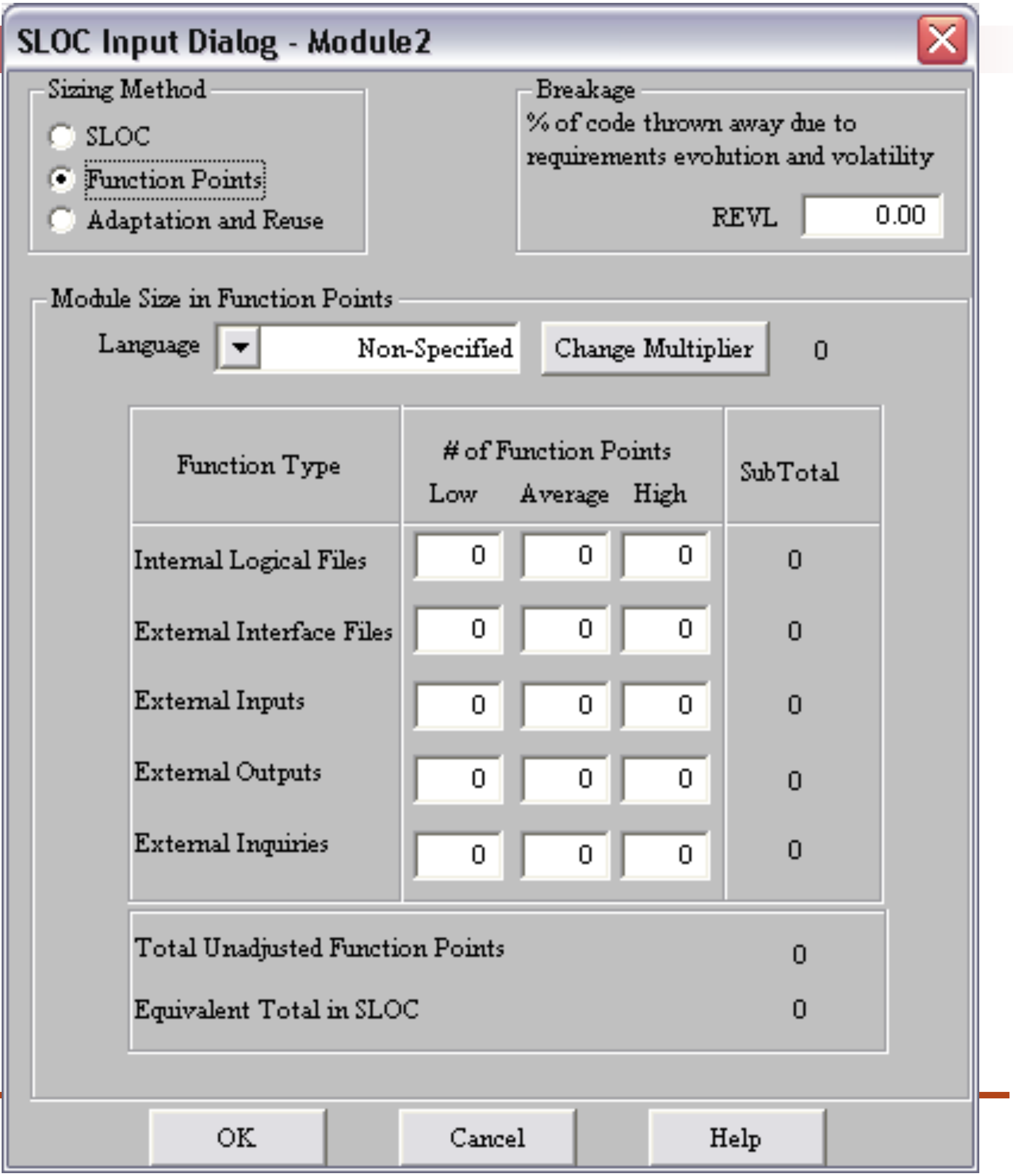
The main screen is updated with the SLOC and programming language as well as some calculated values we will decipher later. Note that the SLOC is 12,000. Why? ☺

{Pertinent portion of calculation on next slide in red boxes}

Now add another module and choose Function Points.

This is the default screen for Function Points.

Let's look deeper at the Function Type descriptions…



SLOC Input Dialog - Module2

Sizing Method

○ SLOC

● Function Points

○ Adaptation and Reuse

Breakage

% of code thrown away due to requirements evolution and volatility

REVL 0.00

Module Size in Function Points

Language: Non-Specified

Change Multiplier 0

Function Type	# of Function Points Low	Average	High	SubTotal
Internal Logical Files	0	0	0	0
External Interface Files	0	0	0	0
External Inputs	0	0	0	0
External Outputs	0	0	0	0
External Inquiries	0	0	0	0

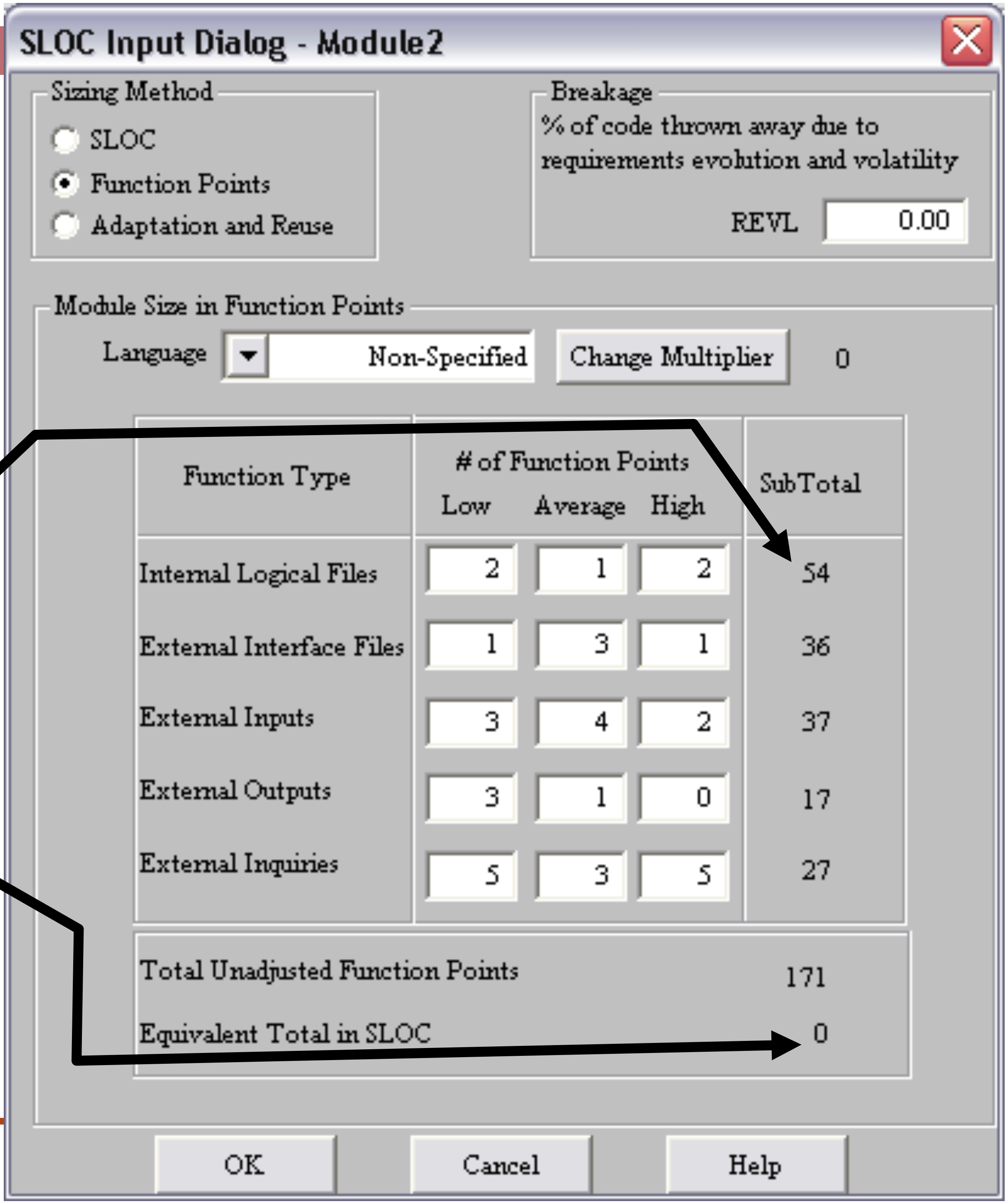
Total Unadjusted Function Points 0

Equivalent Total in SLOC 0

So let's go back into this screen and add some entries in the grid.

Notice, there are some kind of subtotals per line, but the Equivalent SLOC = 0.

Let's change the Language and see what happens.



SLOC Input Dialog - Module2

Sizing Method

○ SLOC

● Function Points

○ Adaptation and Reuse

Breakage

% of code thrown away due to requirements evolution and volatility

REVL: 0.00

Module Size in Function Points

Language: Non-Specified

0

Function Type	# of Function Points Low	Average	High	SubTotal
Internal Logical Files	2	1	2	54
External Interface Files	1	3	1	36
External Inputs	3	4	2	37
External Outputs	3	1	0	17
External Inquiries	5	3	5	27

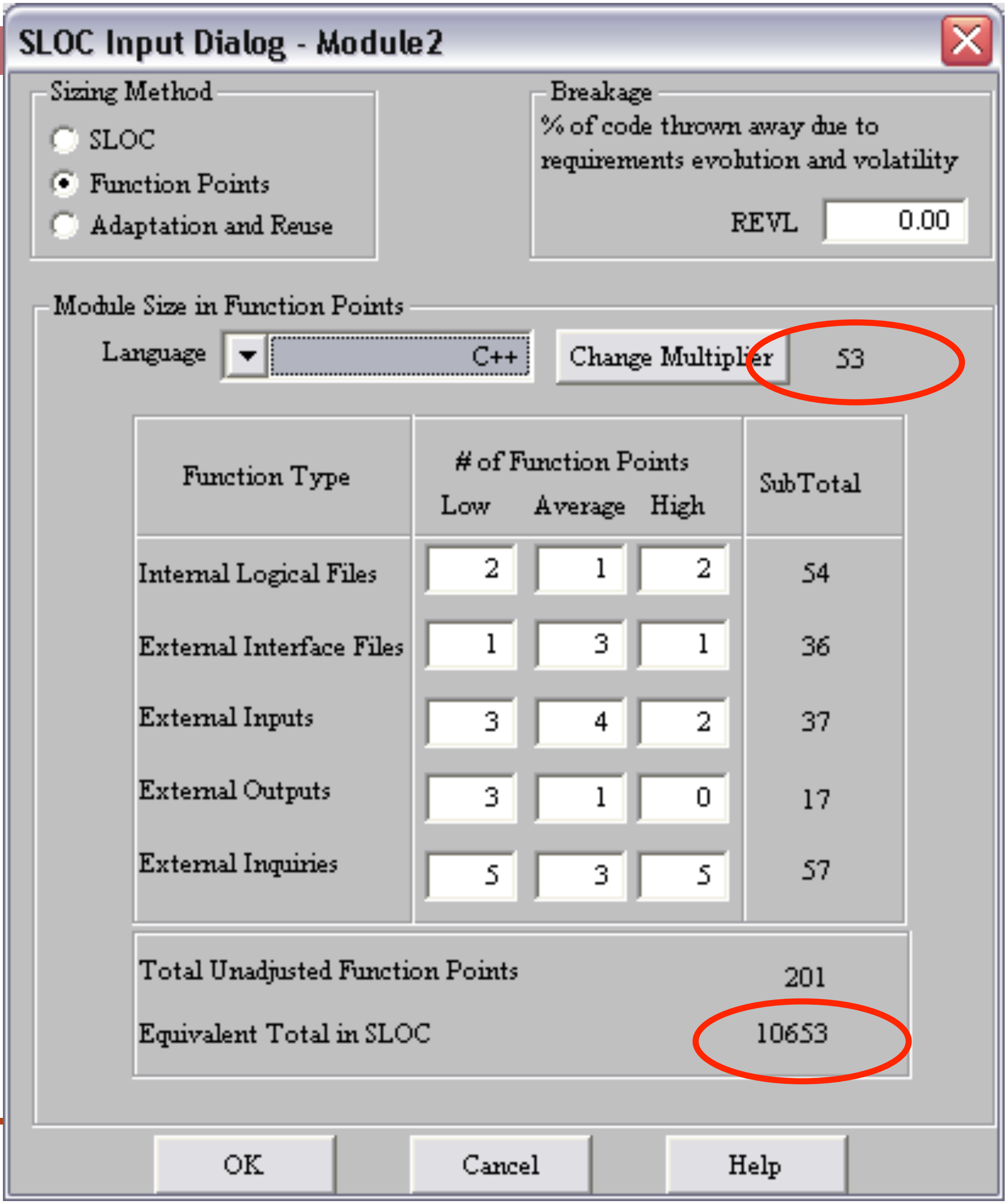
Total Unadjusted Function Points: 171

Equivalent Total in SLOC: 0

By changing the language to C++, we now have an Equivalent Total in SLOC.

Also, we can see a value next to the Change Multiplier button.

Let's change the language to Machine Code! ☺



SLOC Input Dialog - Module2

Sizing Method

○ SLOC

● Function Points

○ Adaptation and Reuse

Breakage

% of code thrown away due to requirements evolution and volatility

REVL 0.00

Module Size in Function Points

Language: C++

Change Multiplier 53

Function Type	# of Function Points Low	Average	High	SubTotal
Internal Logical Files	2	1	2	54
External Interface Files	1	3	1	36
External Inputs	3	4	2	37
External Outputs	3	1	0	17
External Inquiries	5	3	5	57

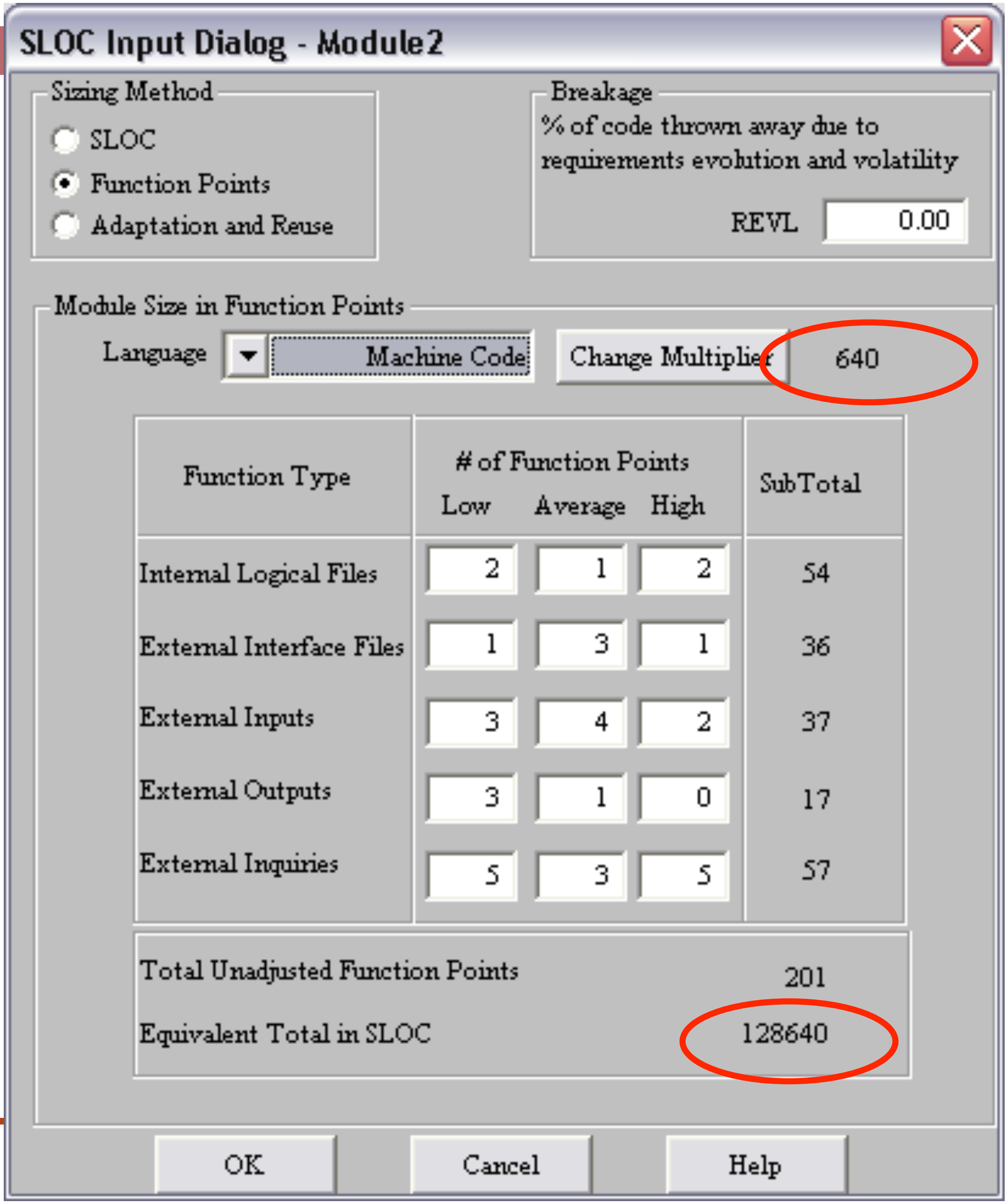
Total Unadjusted Function Points 201

Equivalent Total in SLOC 10653

Quite a difference jumping from 10,653 SLOC to 128,640 SLOC.

Note the multiplier changed from 53 to 640.

Change the language once more to 5th Generation.



SLOC Input Dialog - Module2

Sizing Method

○ SLOC

● Function Points

○ Adaptation and Reuse

Breakage

% of code thrown away due to requirements evolution and volatility

REVL 0.00

Module Size in Function Points

Language: Machine Code

Change Multiplier: 640

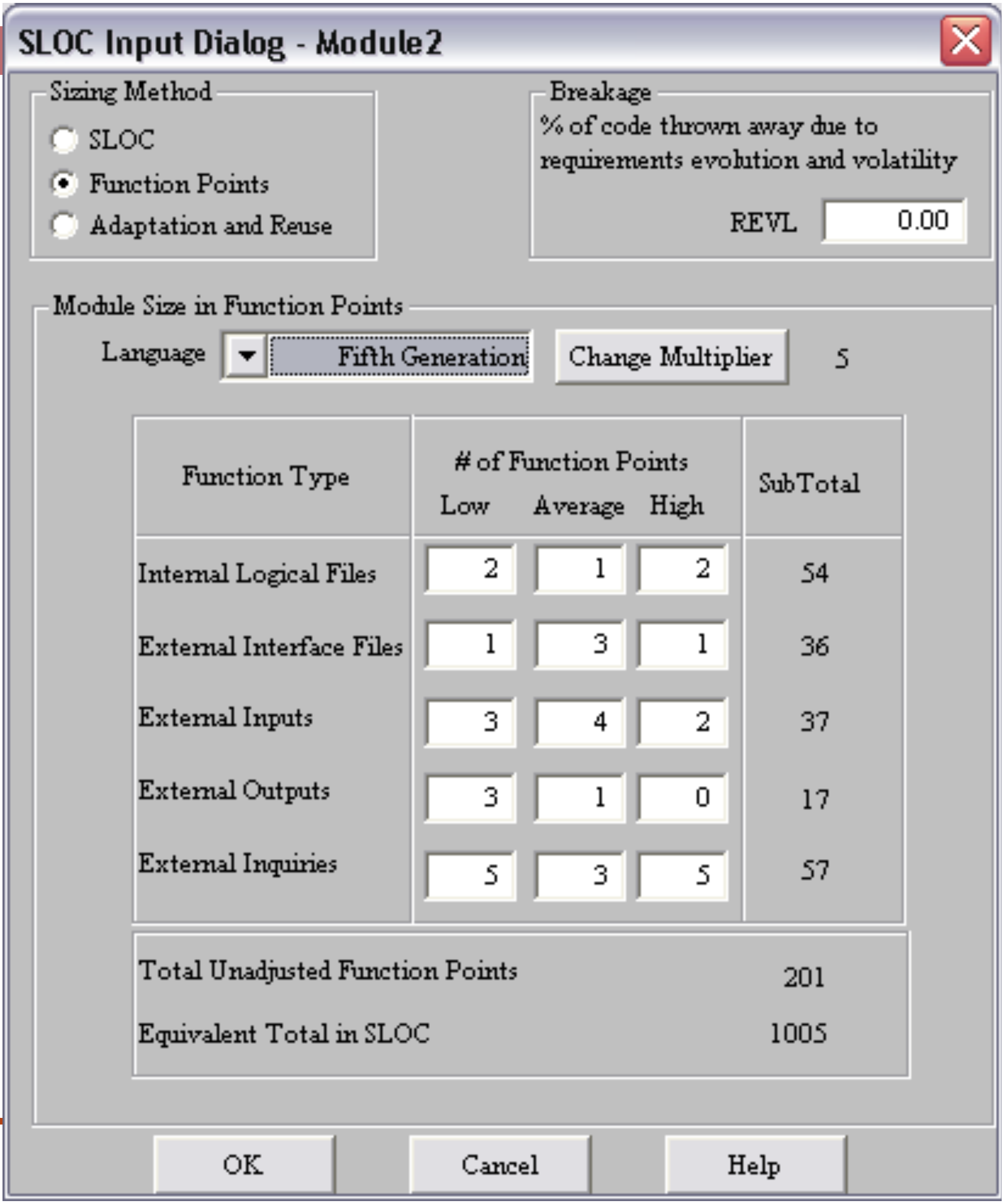
Function Type	# of Function Points			SubTotal
	Low	Average	High	
Internal Logical Files	2	1	2	54
External Interface Files	1	3	1	36
External Inputs	3	4	2	37
External Outputs	3	1	0	17
External Inquiries	5	3	5	57

Total Unadjusted Function Points 201

Equivalent Total in SLOC 128640

So using a 5th generation level language would cut our code base by a factor of 285 times according to COCOMO II's default estimation (not calibrated for your environment, not taking into account other factors).

Change the language to C++ and change REVL to 20%…



SLOC Input Dialog - Module2

Sizing Method

○ SLOC
● Function Points
○ Adaptation and Reuse

Breakage

% of code thrown away due to requirements evolution and volatility

REVL 0.00

Module Size in Function Points

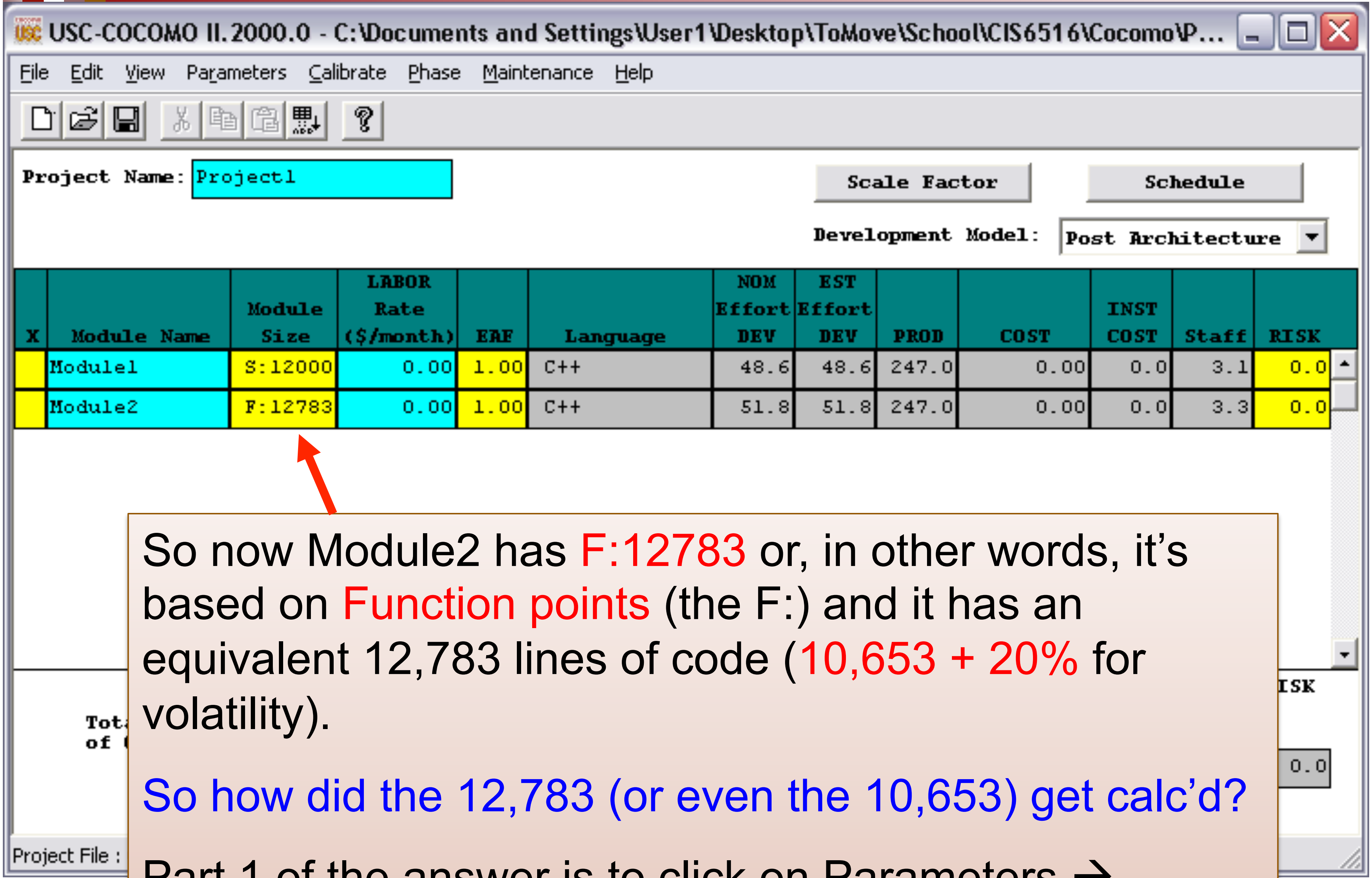
Language: Fifth Generation

Change Multiplier 5

Function Type	# of Function Points Low	 Average	 High	SubTotal
Internal Logical Files	2	1	2	54
External Interface Files	1	3	1	36
External Inputs	3	4	2	37
External Outputs	3	1	0	17
External Inquiries	5	3	5	57

Total Unadjusted Function Points 201

Equivalent Total in SLOC 1005


USC-COCOMO II.2000.0 - C:\Documents and Settings\User1\Desktop\ToMove\School\CIS6516\Cocomo\P...
File
Edit
View
Parameters
Calibrate
Phase
Maintenance
Help
Scale Factor
Schedule

Project Name: Project1

Development Model: Post Architecture

X	Module Name	Module Size	LABOR Rate ($/month)	EAF	Language	NOM Effort DEV	EST Effort DEV	PROD	COST	INST COST	Staff	RISK
	Module1	S:12000	0.00	1.00	C++	48.6	48.6	247.0	0.00	0.0	3.1	0.0
	Module2	F:12783	0.00	1.00	C++	51.8	51.8	247.0	0.00	0.0	3.3	0.0

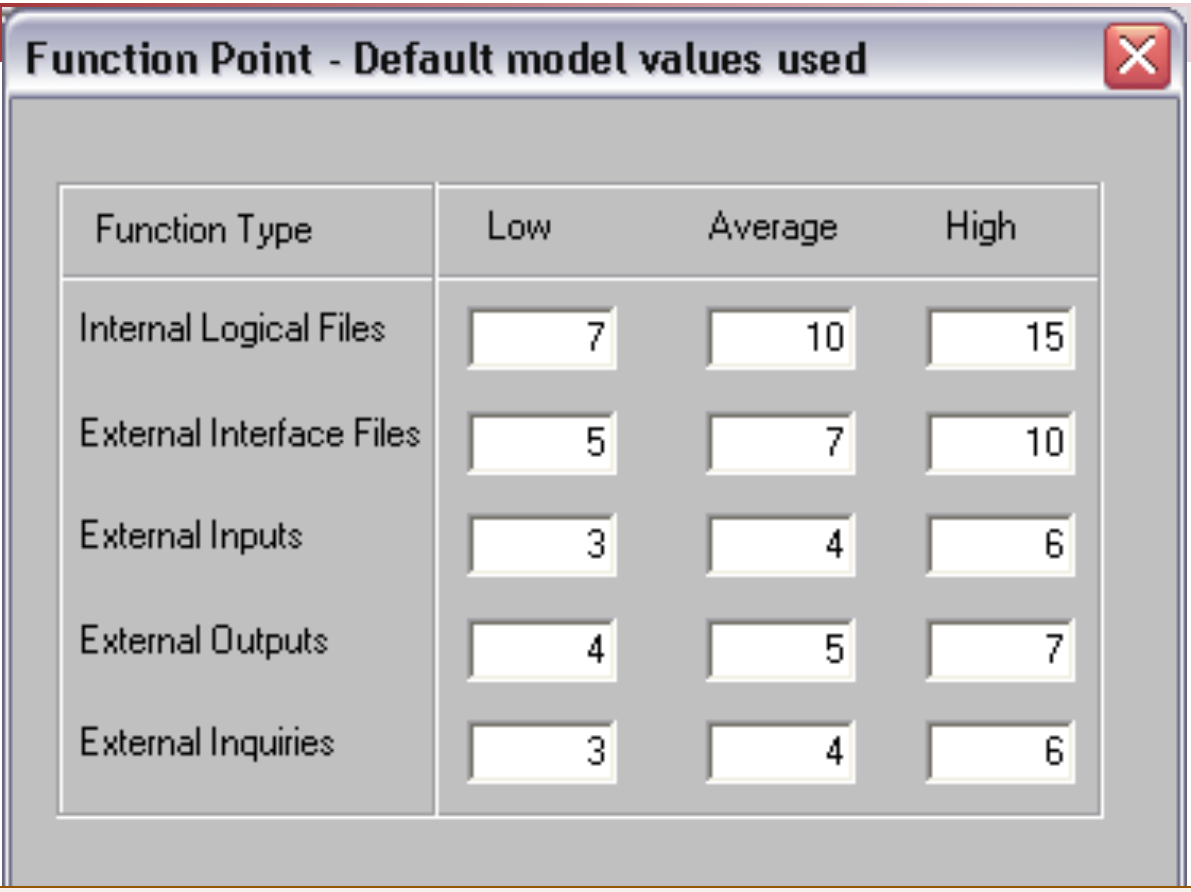
Project File :

So now Module2 has F:12783 or, in other words, it’s based on Function points (the F:) and it has an equivalent 12,783 lines of code (10,653 + 20% for volatility).

So how did the 12,783 (or even the 10,653) get calc’d?

Part 1 of the answer is to click on Parameters → Function Points. You will see the following screen…

Q7



Function Point - Default model values used

Function Type	Low	Average	High
Internal Logical Files	7	10	15
External Interface Files	5	7	10
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6

These are the default values used as weighting factors against the entries you put in. So if you entered 2,3,4 when enter in Function Point information for the first row, the end result would be 2*7 + 3*10 + 4*15. This is then multiplied by The Change Multiplier…