Software Sizing units – SSU

Introduction

Software development industry has spawned into a major independent industry in its own right. As software development industry is maturing, standards for the process of software development got stabilized to the extent of offering certification. Software development activity graduated from the domain of creative to the domain of engineering. This resulted in improving the ability to specify more precisely the software requirements. Therefore, customers are increasingly shedding the practice of T&M (Time and Material) mode of paying for the software services and are demanding "fixed price" bids. One specific aspect of this more or less, remains gray and that is the tools for comparing accurately proposals from different vendors.

This issue has been around us for quite some time and a number of software size measures have been proposed and are being used. However, there remains ambiguity and a lack of general acceptance about these measures among experts themselves leave alone the practitioners.

The purpose of this paper is to discuss the attributes of a good measurement unit, drawbacks of existing measures and to propose a new unit – Software Sizing Unit (SSU) for consideration and discussion across the industry.

Attributes of a good measurement unit

A good measurement unit needs to have the following attributes –

1. It should be well defined with no room for vagueness and ambiguity.
2. It should be simple to understand and use – it should not need extensive training, examination and certification to be able to use it
3. It should be tangible – there should be a standard unit available for all and it should be possible to compare with the standard unit
4. There should be no room for qualitative measures requiring discretion and judgment

Most cases there are instruments to measure the attribute. We have instruments to measure distance between two locations, weight measures the mass of a body, temperatures the hotness of a body, density, speed etc. Some attributes - such as heat, force, strength, impact or software size, - do not lend themselves to be measured by an instrument – easily. It took some time for science to invent instruments to measure attributes like heat. But the science of Physics has precise definitions for attributes like heat, electricity while computer science has so far not come up with a precise definition for software size.

This paper attempts to define software size.

Drawbacks of existing measures

The first measure that was accepted by the industry was LOC (Lines of Code). Some issues were left in the gray and they still continue to be so. These are –

1. Definition a line of code – is it logical line (a statement that may span across multiple lines) or is it a physical line of code (line terminated by a carriage return / line feed character)
2. How to treat inline documentation lines should they be counted as lines of code or not? Vendors would obviously want them counted and customers may not.
3. How to treat data declaration statements –declaration of multiple variables in the same statement (this would reduce LOC) and declaring one variable per line of code (this would increase LOC)
4. Programmers have differing programming styles and achieve the same functionality in differing LOC. It is perhaps not practical to define the number of standard LOC for all perceivable functionalities

Suffice it to say, that there are gray areas in this measure and hence not a mature good unit of measure.

Next came the Function Points followed by a few other points, namely, Use Case Points, Object Points, Feature Points, Internet Points and UML Points to the best of my knowledge and it is possible that some more Points of measure are there. There are some incongruent aspects to most of these points.

Then there is COCOMO and parametric models.

All these adjust the size of the software depending on complexity and qualitative attributes !! Complexity and size adjustment is discussed in the below section.

The paradox of Complexity Vis-à-vis Size

"Everything is simpler than you think and at the same time more complex than you imagine." - (Johann Wolfgang von Goethe)

What is complexity? There are no accepted definitions of Complexity either !!

I quote Dr. Sam Vaknin `(http://samvak.tripod.com)`, from his paper "The Complexity of Simplicity" –

- A straight line can be described with three symbols (A, B, and the distance between them) - or with three billion symbols (a subset of the discrete points which make up the line and their inter-relatedness, their function). But whatever the number of symbols we choose to employ, however complex our level of description, it has nothing to do with the straight line or with its "real world" traits. The straight line is not rendered more (or less) complex or orderly by our choice of level of (meta) description and language elements.

This implies that it is up to us to make something complex or simple by the choice of words or symbols.

These are few of the interpretations of the word "Complexity" –

1. We do not fully comprehend something – hence we say that it is "Complex". Nobody has built a "model" (physical or mathematical) for it – so we call it complex.
2. There are unforeseen "hurdles" on the way and we are not sure how many hurdles, and how to get across (there is no map) – so we may say that it is "Complex"
3. The work is "delicate", that is, if we are not careful something may break or fail or cause severe damage – therefore, we label it as "Complex"
4. There may not be room for maneuvering/manipulation, congestion in the workplace – then again, we may label it as "Complex"

Often we label some thing as complex, we do not understand or lack the capability to either understand or do it. For example,

- For a householder, plumbing is a complex activity while it is a simple task for a professional plumber
- For an orthopedic surgeon who can construct a joint in the human body, carpentry (which is far simpler task than operating bones and joints) becomes complex
- For a carpenter plumbing is complex, for a plumber electrification work is complex, for an electrician masonry work is complex
- Extending the above argument, COBOL is complex for a Java programmer, GUI (Graphical user Interface) programming is complex for a CUI (Character User Interface) programmer, Java is complex for a VB Programmer and so on

I am sure that you got the drift by now – I am implying that we find an activity complex when we are not trained in it or are naturally endowed to do it.

It is also a fact that we do not allow a person to do a job in which he is not adept (skilled) at. A novice is never allowed by professional organizations to handle tasks independently unless a senior artisan closely supervises him. Since we do not assign a task to an unskilled person, is it proper to put a measure of complexity to the software development? Surely we don't allow a COBOL programmer to write Java programs and vice-versa !!

Therefore, is it proper to label a component as simple, average (medium) or high in complexity? What we can surely say is that a component is small, medium or large sized components.

Here I come to the next incongruity on this aspect – does complexity come in three convenient steps in the nature?

Complexity is a continuum – it is analog in nature. It cannot be restricted to three levels only. By putting complexity in three levels, people, including experts, can have different perceptions especially in borderline cases.

If I may say so, what is wrong by saying the complexity level is 2.5 or 3.75 or 5.25 and so on?

Another incongruity is adjusting the size owing to the complexity. Consider this convesation between two persons A and B –

A: "How much distance did you walk to day?"
B: "Today I walked on an even surfaced side walk – so it would be three miles"
A: "Why did you mention the quality of surface (complexity) you walked on? What possible bearing it has on the answer to the question I asked?"
B: "See I adjust the distance by the type of surface on which I walk. Yesterday, I walked indoors on my walking machine – thus it would be 3.9 miles. The day before, I walked on a mountain trail, it would be six miles"
A: "Wow – how did you arrive at these figures?"
B: "I walk for 45 minutes. I adjust the distance I walked based on the quality of surface (complexity) I walked on. On an even surfaced sidewalk, the multiplication factor is 1 – indoors on a walking machine, the multiplication factor is 1.3 and on a mountain trail the multiplication factor is 2"

Would you silently accept the explanation given by B? Not likely. You are wont to say that the distance does not change due to the quality of surface (complexity) but the rate of walking (productivity) changes.

Let us consider another scenario of conversation between A and B -

A: "How many steps (on a staircase in a multi storied complex) are there?"
B: "That depends on the climber"
A: "What do you mean?"
B: "I will explain. We adjust the count of the stairs (points) based on the climber (Programmer) capability"

"If the age of the person is between 3 to 5 years, we multiply the count of stairs by a factor 10"
"If the age of the person is between 6 and 12, then we multiply the count of stairs by a factor of 1.2"
"If the age of the person is between 13 and 18, then we multiply the count of stairs by a factor of 0.9"
"If the age of the person is between 19 and 35, then we multiply the count of stairs by a factor of 1.0"
"If the age of the person is between 36 and 45, then we multiply the count of stairs by a factor of 1.35"
If the age of the person is between 46 and 55, then we multiply the count of stairs by a factor of 3

If the age of the person is between 56 and 65, then we multiply the count of stairs by a factor of 5
If the age of the person is between 66 and above, then we multiply the count of stairs by a factor of 20

Would you agree with the explanation given by B?

You are likely to tell him "Look Mister, you should know that the count of stairs does not change – they always remain the same. What changes is the rate-of-climbing (productivity) the stairs. Where did you learn that size changes because of the person's capability?"

Thus you can see that the size remains constant irrespective of the complexity and personnel expertise. What changes is the **rate** (**productivity**) at which the act is accomplished (productivity).

So when complexity increases, productivity decreases – but not the size!

When personnel expertise is low, the productivity is low – but the size remains the same.

In both the cases, the size does not change !! Remember that Mount Everest remains at an altitude of twenty nine thousand feet (approximately or 8,848 meters / 29,028 feet to be precise) irrespective whoever tries to climb it !!

Then why do all present software estimation techniques adjust the size due to complexity or programmer capability or similar factors?

Density – not Complexity

What term perhaps, should we use that increases the size, in my opinion, is "**Density**".

Same volume of liquid with more density is heavier than a liquid of lesser density.

Same volume of solid too, with more density is heavier than a solid of lesser density – a good example is the question we ask youngsters – "which is heavier – a dime made of gold or steel?" to trick them into saying dime made of steel!!

In engineering, drawings come in normally five sizes –

1. A0 – measuring one Square Metre
2. A1 – measuring half Square Metre
3. A2 – measuring one-fourth Square Metre
4. A3 – measuring one-eighth Square Metre
5. A4 – measuring one-sixteenth of Square Metre

You ask any Engineering Draftsman as to how much time it will take to prepare a A0 size drawing – he will tell you that it depends on the "Density" of the lines/pictures that are packed on to the drawing!!

Here we have a parallel for our software development.

A screen, even when it is developed in the same development platform, takes different amount of time depending on the density of controls we place on the screen!!

A report similarly, takes different amount of time based on the density of data packed into it.

Take an example close to computers.

An integrated chip that has more transistors packed into it takes more time to develop than an IC that has lesser number of transistors. The chipmakers use the term "Density" and not complexity to describe their chips. They perhaps, say, that they are now packing a million transistors per square inch as against one hundred thousand earlier.

Thus, a functionality achieved with lesser number of lines code has lesser density that is achieved with more number of lines – this is assuming that the minimum possible LOC are used in both cases.

As can be seen, I think and propose that we use "Density" in place of Complexity when it comes to software estimation.

Definition and explanation of SSU – Software Sizing Unit

I propose **Software Sizing Units (SSU)** – taking a cue from **British Thermal Units (BTU)** ! Just in case it is not known, BTU is a unit of measure used for measuring heat – it is defined as the amount of heat required to heat one pound (lb) of water by one degree Fahrenheit at sea level.

I like to define an SSU as a " function of data elements that enter the system and the process elements that comprise the system".

The terms are explained below.

Data Element – it is an input to the system. It may enter the system thru user input; or read from a master data file/table; or received from another system. It could be a constant or a variable. It is classified into three types, namely, Numeric, Alphanumeric and Control.

Numeric Data Element consists of digits 0 (zero) to 9, one decimal point and a positive or negative sign. Numeric Data Element transforms and is transformed in the system. The whole system revolves around this data element.

Alphanumeric Data Element consists of all humanly readable characters, including space character. The Alphanumeric Data Element simply passes thru the system. When it enters, validation checks are performed to ensure their type and length.

Control Data Element consists of humanly unreadable characters with the exception of space character that are directed to the hardware. Control Data Element triggers some process within the system

System – is a software that fulfils some need and performs a set of defined functions

Enters the system – it is a necessary input given either from the key board by the user, or is retrieved from a master file/table which was prepared thru some other system, or is received over network from another system.

Process Elements – Process Elements act on the Data elements and transform input to desired outputs.

Process Elements are classified into five types, namely, Input Process, Output Process, Counting Process, Computational Process, Decision or Classification Process,

Input Process Elements get Data Elements from external environment into the system – may be thru keyboard, or from a file/table, or from a device like scanner, or from a network.

Output Process Element sends Data Elements from the system to the external environment – may be to a screen, or a report, or to a device like printer or to a network.

Counting Process Element – performs counting within the system – may be count the number of times a process is performed, or may keep track of the records being input, processed or output.

Computational Process Element – performs computations using mathematical equations and produces a result.

Decision Process element – performs classification (decides) based on a set of rules of a data element into a class.

Procedure for Software Size Estimation

1. Count the number of Numeric Data Elements
2. Count the number of Alphanumeric Data Elements
3. Count the number of Control Data elements.
4. Count the number of Input Process Elements
5. Count the number of Output Process Elements

6. Count the number of Counting Process elements
7. Count the number of Computational Process Elements
8. Count the Decision Process Elements

When we say **Estimate -** we imply that it is done before beginning the actual work. It is to get an idea of the cost and time involved in doing something or getting something done. It is "pre" to actual performance. At this stage, most things are unclear. The users, in general, do not or cannot clearly specify their requirements. Architects draw a sketch, a one sheet drawing to freeze the requirements of the user.

In software development, a Systems/Business Analyst assists the user to define requirements based on which a proposal is made and submitted to the customer.

Therefore, the estimate should be based on the requirements agreed with the customer.

But, it is foregone conclusion that users do not define their requirements precisely so as to measure them.

If we approach a stores executive, he is likely to show us his Kardex card and ask for its computerization! It consists of –

i. Material Master Info
ii. Material Receipts Info
iii. Material Issues Info
iv. Balance Info
v. Material Returns
vi. Etc.

This one user requirements is likely to become many requirements –

i. Material Master Maintenance
ii. Material Receipts
iii. Material Issues
iv. Material Returns
v. A number of reports

Therefore, it is necessary to prepare User Requirements at a granularity at which each user Requirement would be one function.
Now the question comes as to what is the level of granularity that is required?

i. The granularity needs to be well understood by the User

ii. The granularity needs to be well understood by the software developer
iii. There should not be any ambiguity or vagueness in the definition

Thus, a User Requirement, when agreed upon by both the User and the Developer,

i. It is one User Identifiable Function
ii. It results in one software artifact (Screen, Report, Stored Procedure, Document etc.) being developed
iii. The size of the software artifact needs to be measurable in an analog manner

The following guidelines are proposed –

Screens – The login screen is the smallest screen and has three inputs (user id, password, forgot-login), two processes (user authentication, pass on control to the next process either password help or next function), and appropriate message. Each input can be considered as a process as it involves data validation procedure. Similarly each message also can be considered as a process as it involves decision-making procedure. Therefore, a login screen can be considered a screen with six processes. This is a screen with a weight of 1.

All other screens would be similarly measured against this standard. For example, there is a screen that –

i. Takes ten inputs
ii. Performs twelve processes
iii. Generates six messages

Thus the screen has 28 processes – thus this screen would have a weight of 4.33 (28 divided by 6).

Thus we have a continuous measurement.

Reports – A table (file) "dump" is considered as the basic report. A "Dump" is one that refers to one table (or file) and produces a columnar report (normally six columns) that fits on an A4 size paper in portrait format, with report totals for two fields. Thus, -

i. It has one table (file) reference – one process
ii. One type of page headings – one process
iii. One report header – one process
iv. One report footer – one process
v. Two running totals – two processes
vi. No groups – each group is one process

Thus as we can see, this Dump Report has six processes. This Dump Report can be given a weight of 1.

All other reports can be sized in contrast with this report. Fro example, a report that –

i. Refers to two tables – two processes
ii. One Report Header – one process

iii. One report footer – one process
iv. Two types of page headings – two processes
v. Two Groups – two processes
vi. Two running totals for report footer – two processes
vii. Two running totals for each group – four processes

Thus this report has 14 processes. Therefore, this report would be assigned a weight of 2.33 (14 divided by 6).