Paradoxes of Software Estimation

Murali Chemuturi
Chemuturi Consultants

Wikepedia defines Paradox as "A **paradox** (in Greek - "aside belief") is an apparently true statement or group of statements that leads to a contradiction or a situation which defies intuition. Typically, either the statements in question do not really imply the contradiction, the puzzling result is not really a contradiction, or the premises themselves are not all really true or cannot all be true together. The recognition of ambiguities, equivocations, and unstated assumptions underlying known paradoxes has led to significant advances in science, philosophy and mathematics."

The software development has spawned into an independent industry with organizations offering exclusively software development service. As it is, perhaps, in the nascent stages, the process of asking for service, offering a service and pricing are somewhat haphazard. Software development falls into the category of **Services** industry as opposed to **Product** industry – that is a service is offered and not a product. Many parallels can be drawn with similar service industries. The major difference between **software service** industry and other service industries is that software is much more highly priced and complex.. Where there is complexity and money, academics step in – research is conducted – jargon is developed – concepts are proposed and a new branch of science or engineering comes to life.

Initially there are many paradoxes. This paper discusses some of those paradoxes – why not – all? Reason is simple – there is no comprehensive document on the total paradoxes and the work is still going on.

Paradox of "Why" of Estimation?

Why do we carry out software estimation?

1. For pricing of software development / maintenance contracts
2. Resource estimation
3. Delivery Commitments

When we estimate for resource estimation or delivery commitments, we can always estimate for each of the activities – say – for coding, for code-walk-thru, for testing and so on to arrive at the resource requirements for each of these activities. By summing up the individual requirements, we can arrive at the resource requirements for the project.

The software estimation becomes contentious, when we estimate for software pricing. We need to arrive at an estimate that is understood by the client's purchaser, who is to be assumed **not** to be a software developer and is expected to be unfamiliar with software estimation. Typically, the scenario is like this –

1. There are multiple bidders

2. The capability levels of the bidders vary – in some cases – they vary vastly
3. There will be techno-commercial negotiations – which are mostly commercial in nature
4. The technical question is something like "How did you arrive at this price?"
5. The expected answer is to be in non-technical terms and is expected to be facilitate comparison with other bids

This is the scenario – that poses the issue, as the negotiators want one universal norm that can be applied across platforms, across organizations, across technologies and across board.

This is the crux of software estimation concerns.

Paradox of software size

There is ample literature on sizing software. They try to size software the way distance is sized (measured) or weight is measured. They want to have a unit of measure that is acceptable to all. Result is, we have many measures of software size – Lines of Code, Function Points, Use Case Points, Object Points, Feature Points, Internet Points, Test Points, to my knowledge and there may be more. True, there are multiple measures for distance – mile and Kilometer – for weight – Pound and Kilogram – but you can convert Pounds into Kilograms and Miles into Kilometers! There is no formula that says one Use Case Point is equal to 1.2 Function Points or something like that!

Now every one agrees that some things are not amenable to measurement – such as beauty and love. Each person is beautiful in one's own way and every one loves the other in one's own way. We don't attempt to measure them in beauty points or love points – do we?

There are many examples in the industry too. We do not measure a car – do we have car points to say that BMW car has 25 car points and Toyota car has 15 car points – therefore, BMW is superior by 10 car points? How does one compare different cars? We don't have a measure for cars so that comparison can be made.

We don't attempt to measure software products too. What is the size of SQL Server or Oracle? Both are multi-user RDBMS' – while buying, do we ask their sizes to get a fair comparison?

We do not also have a fair measure for computer hardware too. How do we compare a AS/400 with RS/6000 – are there any computer-points to measure their sizes?

Is there any measure for buildings? A ten thousand square foot building – one a gym, another a theatre, another a home – would they all have the same measure? Do we have a size measure to compare them?

Lastly, let us take catering service. The same menu to be served at the same place gets vastly different pricing – it depends on the caterer and other specifications. Can we ask them the size of their meals – say in meal-points?

The fact of the matter is that every thing is not amenable to measurement. To be able to measure, the flowing criteria should be satisfied –

1. It (that is being measured) must be homogenous
2. It needs to be physical and tangible
3. It should be monolithic – not an assembly of multiple parts (physical or meta-physical)
4. It should not have any **qualitative** features
5. The measure also must be physical and tangible
6. When there are multiple units of measure, it must be possible to have conversion factor to convert one measurement into other units of measure

The how are the others mentioned above measured – they are sized up using the list of features, qualitatively described.

Paradox of Software Productivity

Productivity is roughly defined as "X units of Output per one Unit of Time.

The definition of standard time (productivity) goes thus –

"Standard time is the unit of time taken to accomplish a unit of defined work carried out by a qualified worker after adjustment using a given method in specified working conditions at a pace that can be maintained day after day without any physical harmful effects"

This definition is specified by the AIIE (American Institution of Industrial Engineers).

Thus, in manufacturing industry, productivity cannot be stated in a stand-alone mode – it has to be accompanied by specification of –

1. Defined Unit of Work
2. Work environment
3. Working methods
4. Tools & technologies used
5. Qualified worker, after acclimatization

It is needless to say that productivity varies from organization to organization even in well-established measures of productivity.

We have universally accepted measures of time – Person Hour, person Day, Person Month and Person Year. We are yet to have a universally accepted unit of Measure for software output.

We can have software productivity as –

1. Lines of Code per PD (one Person Day)
2. Function Points per PD
3. Use Case Points per PD
4. Object Points per PD
5. Etc.

In manufacturing or traditional service industry, productivity is measured for one activity at a time – for example –

1. Turning activity
2. Milling activity
3. Brick laying
4. Waiting tables – restaurants
5. Soldering
6. Etc.

Productivity measurement of inspection activity and functional testing are measured only in mass / batch production industries but not attempted in Job-Order (tailor made to customer specs) industry. Productivity measurement of design activity and repair (bug fixing) activity are not attempted as they are considered to contain creativity component in the work.

We have not replicated this manufacturing industry model in software development industry.

We have not defined what software productivity is – these are he questions that crop up normally –

1. Does productivity mean – first Coding only or Coding, Code Walk Thru, Independent Unit Testing and Debugging are also included?
2. Does Productivity include systems Analysis and Design work too?
3. What about the inclusion of Project Management overhead?

In most cases, that I had witnessed, software productivity is specified for the entire development life cycle – without tacitly agreeing to what constitutes “development life cycle”.

In manufacturing industry, productivity is specified for an activity and overall throughput is called as “Capacity”. Capacity of a plant, or an organizations would take into account

all operations, all departments, all activities and specifies one figure – say '300 cars per day" or "1 million tones per year" and so on.

Does this sound familiar; it ought to as we hear frequently phrases like "50 LOC of VB code per person per day" or "two days per screen"!!

We seem to be confusing "Capacity" with "Productivity".

Software industry has not so far engaged an Industrial Engineer to study and come up with possible measures of software productivity.

Incidentally, software development industry is bereft of staff unions and resulting negotiations thereof. Perhaps, that is the reason why no attempts have been made by the industry to carry out scientific studies in the field of software productivity.

Thus, while there are concerns and issues, there are also solutions, if we do not look for one single measure or productivity for the entire workflow of software development. What needs to be accomplished is to define taxonomy of software productivity and publish an industry standard. This facilitates further work.

The paradox of offering fixed bids

Many services offer fixed bids – nothing peculiar about it.

If we take the case of an architect, he offers a fixed bid after receiving complete building specs.

Made-to-order industry offers a fixed bid only after receiving complete specs and the quote would include a high level design drawing too.

A caterer would not offer a fixed bid until he receives the menu and the number of guests.

A builder offers a fixed bid, with an escalation clause, after he receives building plans. In construction industry, mostly, unit rates are offered against a detailed tender document that gives great detail of each of the items. The total cost of the building depends on the actual quantity of the several components of the building.

Software is more like construction Industry!!

1. It is difficult for the end-users to visualize the final deliverable from the design documents (drawings)
2. Users continuously ask for changes
3. There are a lot of qualitative features
4. It is very difficult to ascertain the quality of the end product just thru inspection – destructive testing damages the product and renders it unusable
5. The variety of available components is huge with huge difference in their quality

6. The customer more often than not feels that he is being over-charged
7. Acceptance testing is often in hours or days for that is built in months or years
8. End user is more often than not feels that a better deliverable was possible in the price paid or a better vendor should have been chosen

Paradox of Actual Vs. Estimated

Estimation data in other areas comes not from the people who do the job but from work-study engineers (Industrial Engineers) who specialize in work-measurement.

In software industry, estimation data comes from programmers or project managers – which is derived from actual historical data.

Why doesn't the other industry use historical data for forming estimation data? Because the actual amount of time taken for a piece of work varies and depends on –

1. The skill level of the person doing the work – it varies from Super Skill, Good Skill, Average Skill, Fair Skill and Poor Skill
2. The level of effort put in be the person – it varies from super Effort, Good Effort, Average Effort, Fair Effort, and Poor Effort
3. The motivation level of the person
4. The environment in which the work is carried out
5. The methods of working
6. Clarity of instructions
7. Some more like this

We can bring in some uniformity in factors 3 to 7 – but factors of skill and effort varies even in the same organization.

To my mind the best example for the paradox of actual times can be best described by an analogy to Olympic marathon –

1. The distance to be run is know
2. The actual times in earlier marathons were known
3. All the participants are well trained for the event
4. The marathon conditions are well controlled – no unexpected changes

Still the participants do not take the same amount of time to complete the race!! When even the best conditions still produce variations in actuals, then how can a software project with its myriad uncertainties meet the estimated times?

Industry (other than software development industry) – manufacturing, mining etc - follows the concept of **a fair day's work for a fair day's pay** – therefore, the estimation is based on the average effort put in by a person of average skill in specified conditions. The industry had carried out work-study in their respective organizations and came up with standard times for most of the activity that takes place. Industry also has developed number of techniques for effort estimation including time study, micro-motion analysis,

analytical estimation and synthesis etc. Every one's work is measured. If a person happens to be more skilled and puts in more effort, he gets more money by way of incentive.

The actual time differs from the estimated time because of variance in skill of the person and the effort put in by the person. This is recognized in the industry and they never change the estimates just because the actual time has a variance with the estimated time. Estimates and the norms for estimation are changed only when there is a change in -

1. Working Environment
2. Tools
3. Methods
4. Work itself

Software industry has never undertaken work-study for software development – taking umbrage under the label that software development is a creative activity. It is not the whole truth – the creative component is obviously in the software design but not in coding. The concept of a fair days' work for a fair day's pay is also never heard in software development industry – may be because the software engineers are paid better than others.

The norms for estimation are drawn from earlier projects. These are continuously updated taking cue from the completed projects. This leads to the following scenarios –

1. **The Eager-Beaver PM Scenario** –
    a. An estimate is given to the Eager-Beaver (EB) PM
    b. In order to please the boss, EB beats the estimate.
    c. Now, the project postmortem concludes that the estimate is an under-estimate and estimation norms are tightened (instead of rewarding the EB)
    d. Next estimate is made with the new norms
    e. This iteration the EB is frustrated that his effort was not rewarded, he either delays the project or resigns and leaves
2. **The Smart-Alec PM Scenario** –
    a. An estimate is given to the Smart-Alec (SA) PM
    b. SA – weighs the situation and delays the project until a point of Penalty-Avoidance
    c. Now, the project postmortem concludes that the estimate is an over-estimate and estimation norms are loosened (instead of punishing the SA)
    d. Next estimate is made with the new norms
    e. This iteration the SA knows that he was successful in delaying, follows the same
    f. The Project office keeps on loosening the estimation norms until Marketing complains of high quotes
3. **The Pragmatic-PM Scenario** –
    a. An estimate is given to the Pragmatic-PM (PP)
    b. PP plans his work such that he meets the estimate

c. Now, the project postmortem concludes that the estimate is a right-estimate and estimation norms are retained
d. It would never be known whether or not the estimation norms are right in the first place

In all the three scenarios, it is not clear that any of the estimates in any iteration are the right ones!! The validation of estimates thru comparison with actuals, as can be seen above, does not produce right norms.

Paradox of Uncertainty

Uncertainty is inherent to any human activity – the exceptions are very few. That is why, the Murphy's Law – if any thing can go wrong, it will – has come into vogue and is accepted.

The following are some of the uncertainties that a project face –

1. Project success – in technical terms
2. Project success – in timeline terms
3. Project Success – in cost / profit / budget terms
4. Project success – in quality / reliability terms
5. Project success – in repeat business terms

The following are some of the uncertainties we face while planning a project –

1. Completeness of requirements
2. Stability of requirements
3. Soundness of design
4. Reliability of development platform reliability
5. Team attrition
6. Attrition of key personnel at the client
7. Uncertain productivity
8. Un-specified / under-specified customer expectations

The following are some of the uncertainties we face while estimating effort and duration (schedule) –

1. Uncertain productivity
2. Lack of well-accepted size definition
3. Team skill-level uncertainties
4. Team effort-level uncertainties
5. Paradox of "average"

How is average defined?

1. Simple average

2. Weighted average – simple or complex
3. Moving average
4. Statistical mean
5. Statistical mode
6. Statistical median

How do we compute probability of success (or risk of failure)? Which of the probability distributions to use – Normal, Binomial, Poisson, Beta, Gamma or t-distribution? The general practice is to use Normal or Beta distributions. But there are concerns about their suitability.

Conclusions

The purpose of the above discussion is to bring to surface all the paradoxes that we face in the software development industry. It is also to focus efforts to resolving these paradoxes and bring about industry standards.

Now the time is ripe – there are enough well-understood software size measures. There are also adequate number of metrics being generated and are available to researchers to analyze.

>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
About the Author:

Murali Chemuturi is a Fellow of Indian Institution of Industrial Engineering and a Senior Member of Computer society of India. He is a veteran of software development industry and is presently leading Chemuturi Consultants, which provides consultancy in software process quality and training. He can be reached at murali@chemuturi.com