Triangles within Triangles

Introduction

The “Triple Constraint” or “Iron Triangle” is a concept whereby constraints on the system are in opposition to each other. There are several trilemmas in computing including CAP[i], Zooko's Triangle[ii] and the “Project Triangle”, to be considered here. In relation to a software development project, the constraints are usually considered to be *scope*, *cost* and *time*.

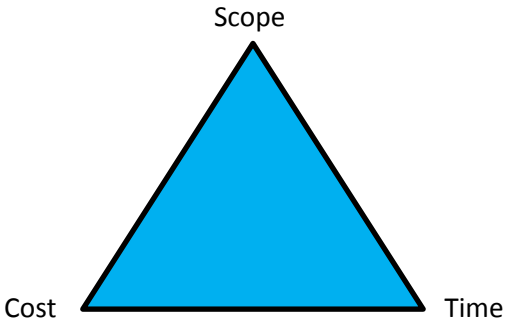

Figure 1: The Iron Triangle

The *scope* can be thought of as the features and functionality of the system, while the *cost* is derived from the resources used, such as human resources and anything else needed to support them e.g. computers and software. *Time* is the schedule to produce the system. Changing one constraint can affect one or both of the other two constraints e.g. to reduce *time* would require either the reduction in *scope*, the increase in resources (*cost*) or both.

Other terms used are “good” to refer to scope, “cheap” to refer to cost and “quick” to refer to time. The phrase “pick any two” is often used to simplify the situation - that is you can have a “good cheap” system, but it will take more time, or you can have a “quick good” system but it will cost more. It follows a “cheap quick” system will not have all the functionality.

Iron versus Elastic

The reason why this model is called the “Iron Triangle” is because these constraints were traditionally fixed at the beginning of the project (using the Waterfall method). This relies on the premise that time can be estimated accurately, but unfortunately this is rarely true. If the time is not correctly estimated, then it can be said the triangle is unbalanced and it is unlikely that the situation will improve as the project progresses. In order to regain the balance, two of the points need to be fixed and then the third can be allowed to change to bring back the balance.

If the team is well formed, then it would follow that the cost of the team can be predicted. It would make sense to pin constraints that are well known and in many circumstances there will be a budget, so this may make the case for fixture. It then comes down to the choices of pinning the scope or the time. If all the features are required, then the time will need to be flexible as the other two constraints will be fixed. If the time is fixed, then scope will need to be flexible. Now, the triangle is no longer an “iron triangle” but rather an elastic one! It is better to have an informed flexible triangle, than an ignorant rigid one.

However, if the project needs to keep the same scope, but needs to deliver earlier then more resources are required, increasing the costs. The case is the same if the scope needs to be increased but is to be delivered in the same time frame. This all seems straight forward, doesn't it?

Hidden Complexity

In relation to software development, it is a well-established principle that just adding more staff doesn't result in an immediate gain in progress, as documented by Fred Brooks in his book "The Mythical Man Month". Brooks noted, "Nine women can't make a baby in one month". The reason is that unlike other fields where for example providing more lorries and drivers results in more goods being delivered quicker, software is intrinsically complex – we shall see why presently. Bringing extra developers on to the team can result in the project slowing down initially. Like a medical doctor, a developer should "first do no harm" and understand the system before changing or adding to it, so that such changes do not break existing behaviour or make the system less stable. Initially, their progress will be slow as they acquaint themselves with the system. Moreover, there will be more communication and coordination and they will need support from the existing team to gain knowledge, understanding and guidance and will therefore make the existing staff less productive. It could take up to 3 months before extra resource has significant impact and during this time, productivity will have dropped.

Software is developed by people and as Tom DeMarco and Tim Lister[iii] found in researching software projects, the main causes of failure were not technical issues but people related issues such as communication and understanding. "Peopleware" (anything that has to do with the role of people in the development or use of computer software and hardware systems) is the reason for this complexity.

The Quality Triangle

To complicate matters further, there is a fourth constraint at the centre of the 3 competing constraints, namely *quality* as shown in figure 2. Manipulating the other constraints has an effect on quality e.g. if all constraints are fixed and the project is behind schedule, then the only way to bring in the project on time is to cut corners with the process. As an example, traditionally in the Waterfall method, the stage that gets cut is the testing phase and thus this can lead to a product that has not been fully tested and may contain many defects. This will be a lower quality product.

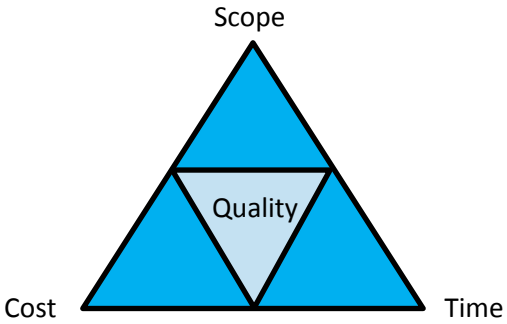

Figure 2: The Quality Constraint

The problem with quality is how to define it! To different stakeholders it means different things. The famous management consultant W. Edwards Deming said:

> *"The problem inherent in attempts to define the quality of a product, almost any product, were stated by the master Walter A. Shewhart. The difficulty in defining quality is to translate future needs of the user into measurable characteristics, so that a product can be designed and turned out to give satisfaction at a price that the user will pay. This is not easy, and as soon as one feels fairly successful in the endeavour, he finds that the needs of the consumer have changed, competitors have moved in, etc."*

Steve McConnell defines software quality as:

> *"The degree to which the software satisfies both stated and implied requirements."*

On a simple level it may be instinctive to think that the result of expending more effort on a product will be an increase in quality – the perception is that hand crafted products are better quality than mass produced ones. Proponents of Deming came up with such a formula for quality (when quality is the focus):

$$Quality = \frac{Results\ of\ Work\ Effort}{Total\ Cost}$$

In reality it is difficult to get an exact measure of quality, rather it is possible to measure different aspects of quality and then by some method of weighting or averaging, to get an overall estimate of quality. For example a simple measure of defect quality could be the proportion of defects found to every thousand lines of source code (KLOC), although more comprehensive assessments have been formed.[iv]

One perspective of quality could be the things that may be appraised as making up the product rather than the product as a whole. This could be the Design, Code, Tests and Documentation artefacts that are produced. In turn these can be further decomposed, so for example the code could be produced using Test Driven Development (TDD), as shown in figure 3. This produces both code and unit tests, demonstrating there is some overlap between decomposed elements.

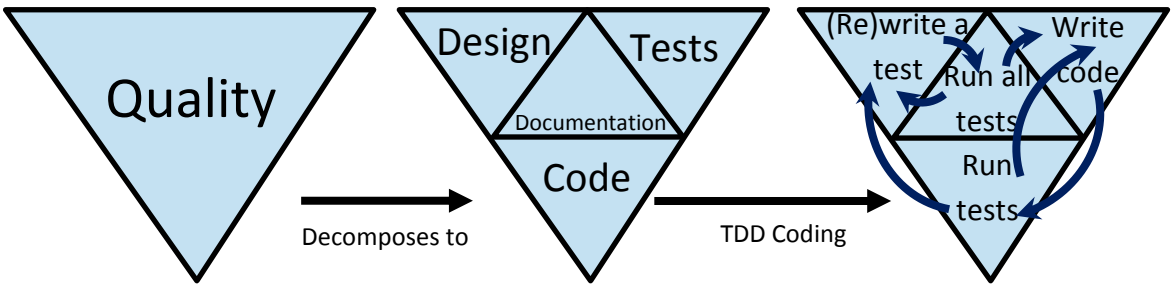

Figure 3: One Quality Decomposition Perspective

Tools can be used to show how much of the code is covered by the tests and a quality standard can be defined accordingly e.g. “at least 70% of code will be covered by unit tests”.

Quality can also be viewed as either functional or structural quality, where functional quality is based upon the functional requirements and structural quality is based upon non-functional requirements. There is another perspective of quality which is the external versus internal quality. External is what the user experiences (behaviour), while the internal is how well the system is architected, designed and coded. These perspectives are shown in figure 4.

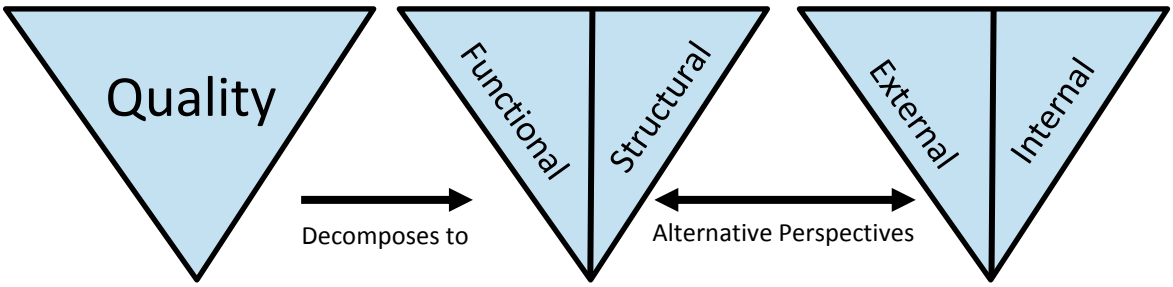

Figure 4: Quality Decomposition Perspectives

The structure, classification and terminology of attributes and metrics applicable to software quality management have been derived from the ISO 9126-3 and the subsequent ISO 25000:2011[v] quality model, also known as SQuaRE. Effectiveness, Efficiency, Satisfaction, Freedom from risk and Context coverage are the 5 attributes listed. The Consortium for IT Software Quality (CISQ) has used this as a base and defined their own 5 major desirable structural characteristics needed for a piece of software to provide business value (quality might be hard to define but value is easier to quantify either directly or indirectly). These characteristics are Reliability, Efficiency, Security, Maintainability and Size.

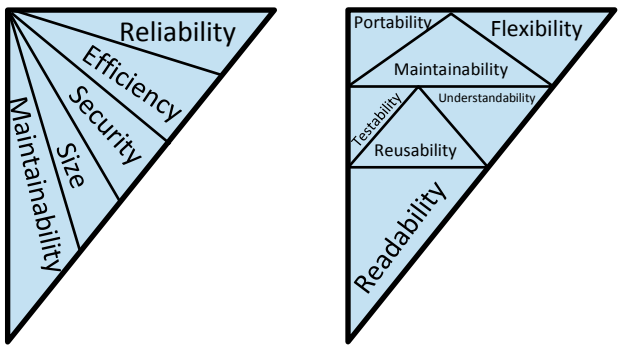

Figure 5: Decomposition of Structural Quality (left) and Internal Quality (right).

Steve McConnell lists 7 attributes for internal quality in Code Complete[vi], these being Flexibility, Maintainability, Portability, Reusability, Readability, Testability and Understandability. The both perspectives are shown in figure 5.

ISO 25000:2011 decomposes some of their attributes further, for example Satisfaction is decomposed into Usefulness, Trust, Pleasure and Comfort. Functional and external quality can also be decomposed. Some of these attributes can work in harmony with each other while others can work in opposition. So, just at this level of decomposition we can see that the picture is quite complex.

To the product owner or user, the external quality is the most important; however to those that need to change and maintain the software, the internal quality is more important. Getting the balance between *perceived* and *actual* quality is a balancing act and requires good stakeholder management.

There is a continual improvement perspective of quality using the Plan Do Check Act cycle (PDCA) or PDSA where S is for Study, emphasising analysis rather than just checking, as shown in figure 6.

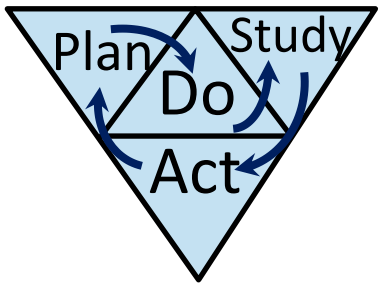

Figure 6: Quality Improvement Perspective

The Cost Triangle

There are several perspectives on cost, from funding (e.g. two stage funding process), to the resources used and to the accountant's view.

One is looking at the resources used to develop the software product. This requires people, computers and software such as development environments and tools etc. People can be permanent staff and also contractors that are brought in for a specific project (their costs may be allocated to a different budget than permanent staff). This is shown in figure 7.

Another perspective of costs is the categorisation into fixed costs (overheads that don't vary greatly with amount of product) and variable costs (vary proportionally with amount of product) and without getting too far into accountancy, further subdivision into capital costs etc.

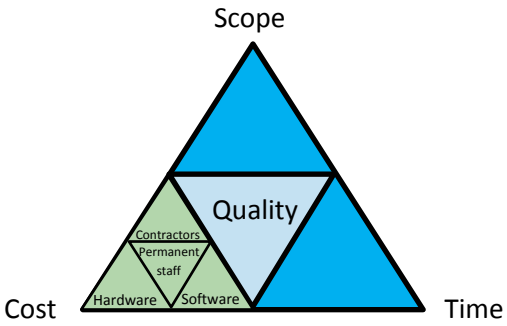

Figure 7: One Cost Perspective

A further perspective is how the budget is spent on different activities of the project. These include managing, administration, process development, requirements development, prototyping, architecture, design, component acquisition, implementation, integration, testing, release and metrics.

The Time Triangle

Early on in the software project there are a lot of unknowns and many aspects may be unclear. The *Cone of Uncertainty*[vii] demonstrates that estimation in the initial stages of the project is subject to a large degree of variation. However, as the project progresses, the uncertainty decreases and estimates have more accuracy.

The time or effort required to complete the project can be calculated in a number of ways. One way is for experienced people to estimate a project based upon previous experience (estimated based on similar projects completed in the past). However, this is a ball park estimate.

Another way is each feature can be estimated in days and hours and the total time for all the features totalled to give the total project time. This is difficult in reality as the estimate needs to encompass all the work required for each feature (design, coding, testing, documenting etc.) which assumes all the requirements are known and have been described to a degree that an a meaningful estimation can be given. There are several problems here. Firstly, who supplies the estimates e.g. do you use 1 designer, 1 programmer, 1 tester etc. or just 1 person. If you use 1 programmer, are they a front end programmer or a back end programmer? Another problem is that unless scope has been fixed, changes or additions to requirements will affect the effort required.

The Wideband Delphi method uses collaboration in a formal manner so consensus is reached using anonymous time estimations. This method is purported to suit government organisations more than private business and is thought to be linked to culture – it is not always possible to talk openly so an anonymous process might help to elicit ideas and consensus.

In the agile methodology, "Planning Poker" is often employed using "Story Points" to estimate the effort required. This is not a direct link to time, but more of an abstract metric using effort or complexity linked to a known base line story. The process of discussing the story elicits both an understanding of what is involved as well as an idea of the "size". The whole development team show their estimation using either cards or a mobile app, often based on a *Fibonacci* series of numbers or a variation on this e.g. 0, 0.5, **1, 2, 3, 5, 8, 13**, 20, 40 and 100 are quite common. The people who give the outlying estimates explain their decisions and another round of estimates are given. This process is repeated until a consensus is reached. Where a task cannot be confidently estimated because of "unknowns" then a *spike* can be used whereby a fixed amount of time is assigned to experiment and evaluate the issues involved. Once the *spike* is completed a better idea of the issues will help in producing a better estimate.

There are also estimation methods based on *Function Points*[viii], *Use Case Points*[ix] and *COCOMO*[x] amongst many others. One other method is the three-point estimation technique. This involves producing three figures based on prior experience or best-guesses:

a = the best-case estimate

m = the most likely estimate

b = the worst-case estimate.

From these a weighted average *E* can be calculated using the following formula:

$$E = \frac{(a + 4m + b)}{6}$$

The standard deviation (*SD*) can also be calculated which will give an indication of the variability or uncertainty in the estimation, using the formula below:

$$SD = \frac{(b - a)}{6}$$

The three-point estimation technique is shown in figure 8.

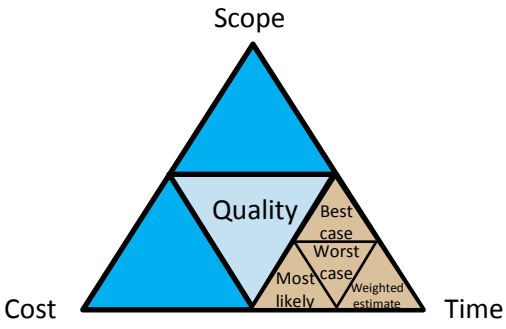

Figure 8: Three-Point Estimation Perspective

Time management of the software project can also be broken down into the following steps:

1. Defining Activities
2. Sequencing Activities
3. Resource Estimating for Activities
4. Duration and Effort Estimation
5. Development of the Schedule
6. Schedule Control

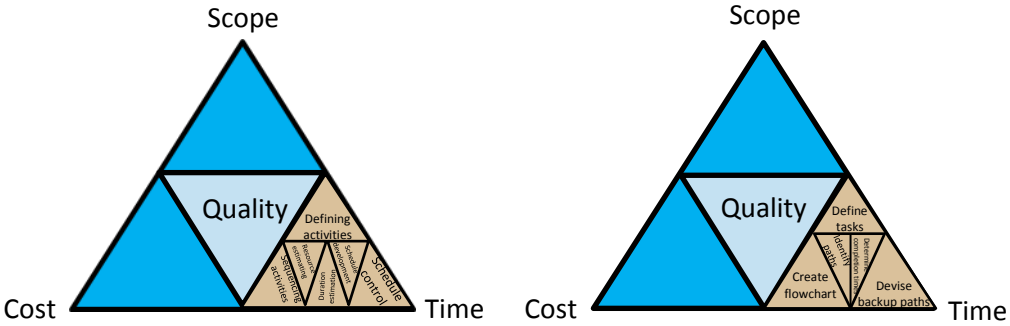

Figure 9: Time Management (left) and Critical Path Management (right)

Another step-by-step project management technique for process planning that defines critical and non-critical tasks with the goal of preventing time-frame problems and process bottlenecks is the critical path method (CPM). The steps are:

1. Define the required tasks and put them down in an ordered (sequenced) list.
2. Create a flowchart or other diagram showing each task in relation to the others.
3. Identify the critical and non-critical relationships (paths) among tasks.
4. Determine the expected completion or execution time for each task.
5. Locate or devise alternatives (backups) for the most critical paths.

These steps are shown in figure 9 together with the time management perspective.

The Scope Triangle

Like quality, a formula has been devised that demonstrates that *scope* is related to *time* and *costs* (resources). Using this formula, if one baker can make 50 loaves in a day, then 2 bakers can make 100 loaves in a day. This could hold true for software development if Fred Brooks' experiences are taken into account.

$$Scope = Time \times Resources$$

Rearranging this formula we can also view in terms of time:

$$Time = \frac{Scope}{Resources}$$

And in terms of resources:

$$Resources = \frac{Scope}{Time}$$

As with the other triangles we have encountered, there are many perspectives to *Scope*. An initial perspective could be requirements priority i.e. which features are to be developed first? One method is called MoSCoW[xi] which is a mnemonic for *Must*, *Should*, *Could* and *Won't* (or *Wish List*). This is shown in figure 10.

Requirements in the Must category are essential – without these there is no point in developing the system! These are required to meet the business needs and must provide a coherent solution.

The Should category has the next priority, but the project's success does not rely upon them.

The Could requirements are developed after the Should category and are developed if they do not affect anything else in the project.

The wish list or Wont's are requirements that have been recorded, but will not be developed for the current version.

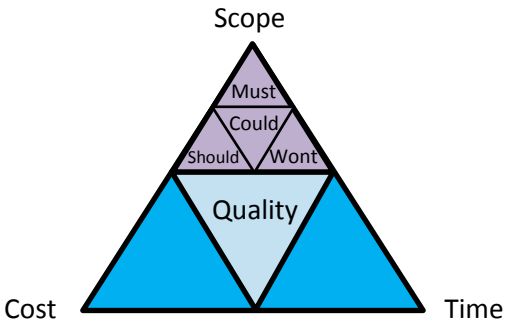

Figure 10: The Scope Constraint

If the cost and time scopes are fixed, then the scope will be flexible and it is likely that the *Could's* will be the first casualties, followed then by the *Should's*. The *Musts* cannot be dropped as they form a coherent core that is required for the project to be useful. If these are dropped then the project fails.

Triangles of Triangles

From this it will be appreciated that the original triangle contains a central triangle and 3 other triangles. Each of these triangles contains triangles which can contain further triangles and some of these triangles can also contain triangles! This can be likened to a Sierpinski triangle[xii] (a fractal based on triangles). Not only can each triangle be decomposed and have an effect on each other, but there can be several perspectives for each triangle, thus in reality forming a tetrahedron or pyramid in 3 dimensions.

You may ask what is the significance of using triangles to represent these ideas? Perhaps an *Influence Diagram*[xiii] or *Systems Thinking*[xiv] may convey these ideas better than a triangle? Ultimately the aim is to find a way of representing a problem and solution that is easily understood.

Steve McConnell has devised a test to evaluate the likelihood of the project's success in his book "Software Project Survival Guide"[xv]. What if a model could be built by combining the points used by the test, selecting the perspectives that are used in a particular project, quantifying the variables (using Fuzzy Logic?) and by use of algorithms a balanced triangle could be produced that shows the scope, costs and time accurately for an acceptable level of quality for a given methodology[xvi]?

On the face of it, this may seem unlikely; however, could this problem be similar to predicting the weather? If good enough models can be produced (like the Metrological Office developed) then even though there are a massive amount of variables, like weather prediction, it may be possible to get a fairly good result at least 70% of the time? Some research has been carried out on modelling the influence of unknown factors in risk analysis using Bayesian Networks[xvii]. This can use a concept called "leaky variables" – factors that cannot easily be quantified but can affect the risk. However, I would be interested in having your thoughts on modelling software development projects and if you know of any specific related research?

[i] http://en.wikipedia.org/wiki/CAP_theorem
[ii] http://en.wikipedia.org/wiki/Zooko%27s_triangle
[iii] Peopleware: Productive Projects and Teams, 1999, Tom DeMarco & Timothy Lister
[iv] https://resources.sei.cmu.edu/asset_files/TechnicalReport/1992_005_001_16088.pdf.
[v] http://www.iso.org/iso/catalogue_detail.htm?csnumber=35733
[vi] Code Complete 2nd Edition, 2004, Steve McConnell.
[vii] http://www.construx.com/Thought_Leadership/Books/The_Cone_of_Uncertainty/
[viii] http://en.wikipedia.org/wiki/Function_point
[ix] http://en.wikipedia.org/wiki/Use_Case_Points
[x] http://en.wikipedia.org/wiki/COCOMO
[xi] http://www.projectsmart.co.uk/moscow-method.php
[xii] http://www.oftenpaper.net/sierpinski.htm
[xiii] http://en.wikipedia.org/wiki/Influence_diagram
[xiv] http://en.wikipedia.org/wiki/Systems_thinking
[xv] Software Project Survival Guide, 1998, Steve McConnell
[xvi] http://www.tutorialspoint.com/management_concepts/project_management_methodologies.htm
[xvii]http://www.researchgate.net/publication/274456154_Modeling_the_Influence_of_Unknown_Factors_in_Risk_Analysis_using_Bayesian_Networks