The Triangular Life Cycle Model

Phil Robinson, lonsdale@iinet.net.au
Lonsdale Systems, www.lonsdalesystems.com

Abstract

Everyone knows that the waterfall life cycle model suffers from a number of problems but in spite of this, it continues to be the most widely used life cycle model. This paper argues that many of these problems stem from project management best practices that are inappropriately applied in the waterfall model.

A different life cycle approach is proposed that emphasises the product life cycle rather than the project life cycle, quality management priorities rather than project management priorities and views of quality rather than views of the project schedule.

A quality management tool based on different views of quality is used to identify the gaps that inevitably exist between a user's needs, the requirements specification and the product that is delivered. This is followed by a brief discussion of how these gap can be closed.

The paper concludes by pointing out that the Triangular Life Cycle Model can peacefully coexist with project management best practices but will provide some balance to the dominant project priorities of staying on schedule and within budget.

The waterfall life cycle model

The waterfall life cycle model is the best known and most widely used life cycle model. A recent survey found that more than a third of organisations still base their software development life cycle on the waterfall model [Lap08].

The introduction of the waterfall life cycle model is frequently attributed to Winston Royce [Roy70]. Interestingly, "waterfall life cycle model" is never mentioned in the Royce's article. In fact, he seems to argue for a more iterative approach to software development!

So it would seem that the waterfall lifecycle model is rather like an "urban myth" [Wik08] everyone claims to know its source but it does not stand up to a close examination of the facts.

One possibility source of the term "waterfall" is the typographic layout of the diagrams in Royce's paper, which seem to suggest a waterfall.

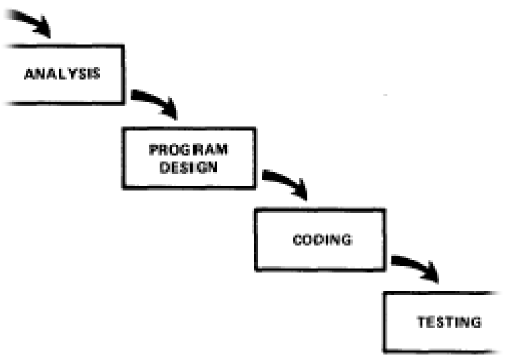

Figure 1: The original waterfall?

Another possibility is that the waterfall life cycle model is simply another way of describing project management practices.

Project priorities

The definitive guide to project management best practice is the PMBOK standard [PMBOK00]. PMBOK identifies 44 best practice project management processes, which are organised into nine knowledge areas. The knowledge areas cover topics such as the management of human resources, communication, risk and procurement in a project context.

But for most project managers the four areas that are uppermost in their minds are time, cost, scope and quality.

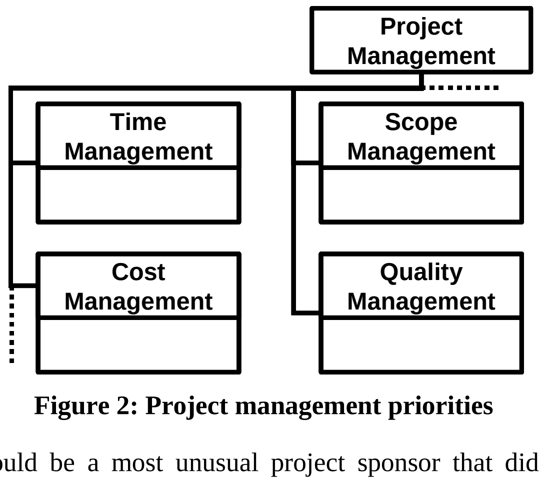

Figure 2: Project management priorities

It would be a most unusual project sponsor that did not prioritise these areas into the following order:

1. time;
2. cost;
3. scope; and
4. quality.

The project life cycle

PMBOK stresses the temporary nature of projects and project teams:

A project is a temporary undertaking to create a unique product or service.

The project team, as a working unit seldom outlives the project - a team created for the sole purpose of performing the project will perform that project and then the team is disbanded and the team members reassigned when the project ends.

It also highlights the differences between projects and operations:

The purpose of a project is to attain its objective and then terminate

The objective of an ongoing operation is to sustain the business

PMBOK defines a project life cycle model that classifies the activities of a project into a number of phases:

The project life cycle defines the phases that connect the beginning of a project to its end.

The completion and approval of one or more deliverables characterises a project phase.

Phases are generally sequential and are usually defined by some form of technical information transfer or technical component handoff.

The phases include an initial phase, a number of intermediate phases and a final phase. One of the objectives of the final phase is the closure of the project:

… includes the processes used to formally terminate all activities of a project …

…hand off the completed product to others…

Software projects

For many categories of project, a life cycle that comes to a final end makes sense. It is true; that once a building project is finished there is little more to do other than move the new occupants into the building. If the project has remained on schedule and within the budget, the project manager will most likely receive well-deserved praise. The project team will either be disbanded or move on to new projects.

While a "temporary undertaking" may be suitable for producing the products of many industries, software products are frequently refined and enhanced throughout their lifetime. It is quite common for teams of developers to work on a single software product for years (or possibly decades).

Many software products play a crucial role in "sustaining the business" either as products in their own right, embedded in hardware products or by supporting business processes. This means that the "final phase" is only reached when a software product is eventually retired.

Nowhere is the influence of the project life cycle more detrimental than in its insistence on the "sequential" organisation of project phases. For software projects, this is normally interpreted as the need to organise project activity around deliverables such as requirements specifications, designs, test plans and program code. The deliverables must be approved and handed off before the next phase can commence.

This means that requirements specification must be complete before design activities can commence, the product design must be complete before coding can commence and coding must be complete before testing can commence.

It is the last of these prerequisites that leads to yet another shortcoming of the waterfall life cycle – testing comes at the end of a project when it is also most expensive to correct any errors found in the product.

This shortcoming is often magnified by slippage in the project schedule. Slippage is more critical towards the end of a project. This means that for many projects, software testing is conducted in an atmosphere of intense pressure to "get it done" as quickly as possible.

It is common to cut short the time allocated to testing in order to achieve project deadlines. Frequently project managers can be heard complaining that, "The project was going well until it got held up in testing".

Poor planning of test activities can magnify the problem. Testing is often represented in the project plan by a single activity called "testing" when it should in fact be shown as two quite different activities:

- a testing activity with the goal of identify failures; and
- a repair activity with the goal of removing product defects that cause the failures.

With this in mind, it would probably be more appropriate for project managers to complain that, "The project was going well until it got held up in repair"!

One of the objectives a project manager must achieve before the closure of the project is acceptance of the product – this often leads to pressure on the stakeholders to accept the product irrespective of whether it properly meets their needs.

The well-known CHAOS report [Sta95] was one of the first surveys of software project failures. Its widely quoted findings include the mind-boggling facts that:

- 31.1% of projects were cancelled before they ever get completed;
- 52.7% of projects cost over 189% of their original estimates (most of this cost overrun is surely attributed to rework);
- at the time of the study American companies and government agencies spent $81 billion for cancelled software projects; and
- $59 billion for software projects that were completed, but will exceed their original time estimates.

It is easy to blame project managers and their project teams for these disastrous failures but it is important to remember that often it is the project sponsor in conjunction with the stakeholders that:

- dictate the schedule;
- allocate the budget; and
- define (and nearly always expand) the scope.

It is not suspiring that Ed Yourdon describes such projects that are set up to fail from the very beginning as "death march projects" [Yor97]. Yourdon defines a "death march project" as one that is allocated a schedule or budget that is less than 50% of a rational estimate. It is frightening to compare the CHAOS findings, that more that half of the projects surveyed exceeded their original estimates by nearly 200%, with Yourdon's definition of a "death march project". The comparison suggests that more than half of all software projects are in fact "death march projects"!

The Triangular Life Cycle Model

Many of the problems described above are well known. Over the years, refinements to the waterfall model have been proposed and alternative life cycle models suggested.

While there is always a lot of interest in improving on the waterfall model, many organisations are found lacking when it comes to actually implementing improvements (Lap08). It is possible that one of the reasons for this could be that many of the alternative approaches are based on elaborate concepts and sometimes accompanied by an all-embracing ideology. This can make them difficult for more outcomes-focussed project managers to accept.

With this firmly in mind; the Triangular Life Cycle Model (TLCM) starts from the highly successful consultant's premise that everyone understands a triangle! Reflecting the three sides of a triangle, it is based on three fundamental principles:

- emphasise the product life cycle rather than the project life cycle;
- emphasise quality management priorities rather than project management priorities; and
- emphasise views of quality rather than views of the project schedule.

Product life cycle

The product life cycle commences with the needs, wants and expectations of its users. These are captured as the product requirements on which the development of the product is based.

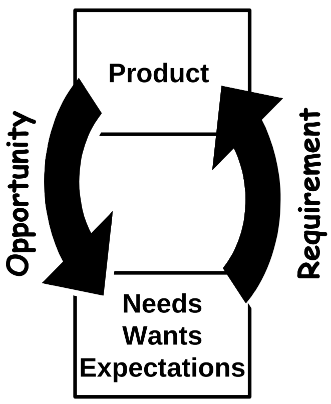

Figure 3: The product life cycle

Once it is put into operation, the users will identify opportunities for the product's enhancement and refinement. These opportunities lead to a revised set of user needs, wants and expectations. These in turn lead to a new set of requirements and ultimately a new version of the product.

For software products, this cycle of revised needs leading to product upgrades can go on for decades with the same core team responsible for ongoing development of the product over this time. In fact, it can be argued that the product life cycle has more in common with the on-going nature of a process rather than the "temporary" nature of a project.

Frustrated by the inconsistencies between the project and product life cycles, some software developers have turned to the Japanese concept of "wabi-sabi" in search of a better model for software development. Wabi-sabi is an aesthetic principle based on the acceptance of transience – "nothing lasts, nothing is finished, nothing is perfect" [Pow04].

Another area of difference between the product and project life cycles is the measure of success. For the product life cycle, success is measured by how well the final product meets it user's needs, in other words by quality and scope. In contrast, success for the project life cycle emphasises time and cost.

Quality management priorities

The quality management order of priorities is the inverse of those for project management:

1. quality;
2. scope;
3. cost; and
4. time.

The reason for quality's place at the top of the list is self-evident. Scope is the second item because "a product's ability to satisfy its user's needs" is a fundamental measure of quality and also the product life cycle measure of success. Cost appears before time because the "cost of quality" is a well-defined concept [AS 2561] that measures both the cost of poor quality and the cost of achieving good quality.

However the placement of time at the bottom of the list does not mean that it is unimportant but rather that from a quality perspective, it is less important than the other priorities.

Views of quality

It is not surprising that Gant charts are the universal tool for planning and monitoring projects. Gant chats use horizontal bars to represent time, which is the first project priority. To emphasise quality's position as the first priority in the TLCM, a similar universal tool is required. David Garvin's views of quality [Gar84] provide an excellent starting point for developing such a tool:

Garvin identifies five views of quality:

- **Transcendental** – this view of quality associates quality with "innate excellence" that is "absolute and universally recognizable". This view is useful for marketing products or establishing brands but because of its subjective nature, not so useful for quality improvement.

- **User** – this view of quality focuses on the ability of a product to satisfy the needs of its users.
- **Manufacturer** – this view associates quality with "conformance to (engineering and manufacturing) requirements". It focuses on how well a product conforms to its specification.
- **Product** – this view of quality associates quality with product characteristics that can be measured using "a precise and measurable variable". It focuses on measurable attributes of a product[1].
- **Value** – this view of quality measures quality "in terms of costs and prices…". A quality product is one that provides performance at an acceptable price or conformance at an acceptable cost.

Three of these views have been selected as the basis for the quality tool:

- the user's view which is represented by the user's **needs**;
- the manufacturer's (software developer) view which is represented by the requirements **specification**; and
- the **product** view.

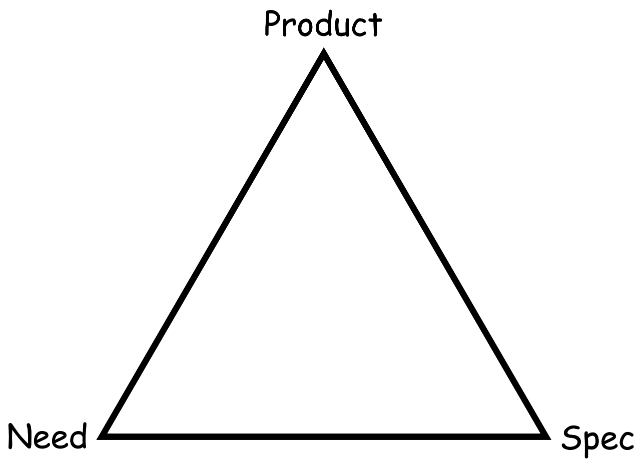

Figure 4: The quality triangle

The value view of quality is implied by the user's needs, which include the price they are prepared to pay for the product and the manufacturers view, which includes the cost of developing the product. The transcendental view of quality is probably best left to the marketing department.

Except in the case of an imaginary "perfect" product, it is unlikely that the stakeholder needs, the specification and the final product will all be in perfect alignment. This will lead to discrepancies or "gaps" between the user, manufacturer and product views of quality.

The gaps between the three views of quality can be represented by a triangle with one of the views placed at each corner of the triangle.

[1] See ISO 9126-1:2001 product quality standard that describes measurable attributes of software products.

- The need-specification gap represents how well the specification describes the user's needs.
- The specification-product gap represents how well the product conforms to its specification.
- The product-need gap represents how well the final product satisfies the user's needs.

The length of the sides represents the magnitude of the gap between the views. The length of any side of a triangle always depends on the length of the other two sides. This means that the magnitude of the product-need gap experienced by the users of the product will always depend on the magnitude of the need-specification and specification-product gaps. In other words, there are two different scenarios that can result in a product ultimately not meeting the needs of its user:

- a poor understanding of the user's needs which results in a need-specification gap; or
- a not following the specification which results in a specification-product gap.

Six Sigma is a widely used business improvement strategy that describes these scenarios using two metaphors – "the voice of the customer" to and the "voice of the process" [Geo04].

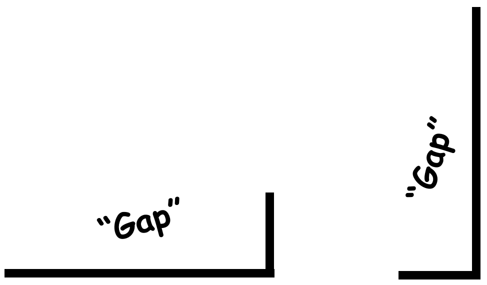

Figure 5: Gaps between the views of quality

Users of software products often have difficulty articulating their needs and providing feedback on requirements specifications. The result is often numerous changes to the software product when the users see it for the first time and realise that it is not what they require.

Barry Boehm has described this as the, "I'll Know It When I See It" (IKIWISI) phenomenon [Boe99].

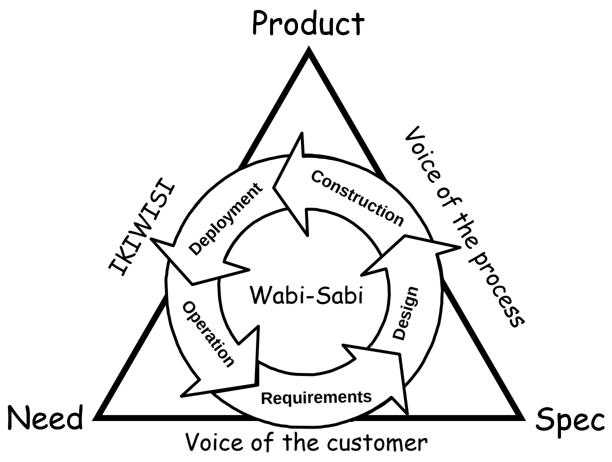

Figure 6: The Triangular Life Cycle Model

Although time appears as the last priority in the TLCM, it is obviously still a critical factor. Time is added to the quality triangle by superimposing a number of sequential life cycle stages onto the triangle. To align properly with the views of quality and the gaps between them, the life cycle stages are arranged into a circle – this also reflects the cyclic nature of the product life cycle.

The Requirements stage of the life cycle contribute to the need-specification gap, while Design and Construction stages contribute to the specification-product gap. The magnitude of the product-need gap is determined during the Deployment stage and is experienced by the users during the Operation stage.

The role of verification and validation

The terms "verification" and "validation" can be confusing. They are frequently used inconsistently. For example, PMBOK's definition of verification is as follows:

Scope verification is the process of obtaining the stakeholder's formal acceptance of the completed project scope and associated deliverables.

Scope verification differs from quality control in that scope verification is primarily concerned with the acceptance of the deliverables…

In contrast, Barry Boehm [Boe79] describes verification and validation as:

…verification involves the comparison between the requirements baseline and the successive refinements descending from it – the product design, detailed design, code, data base, and documentation – in order to keep these refinements consistent with the requirements baseline.

…validation identifies problems which must be resolved by a change of the requirements specification.

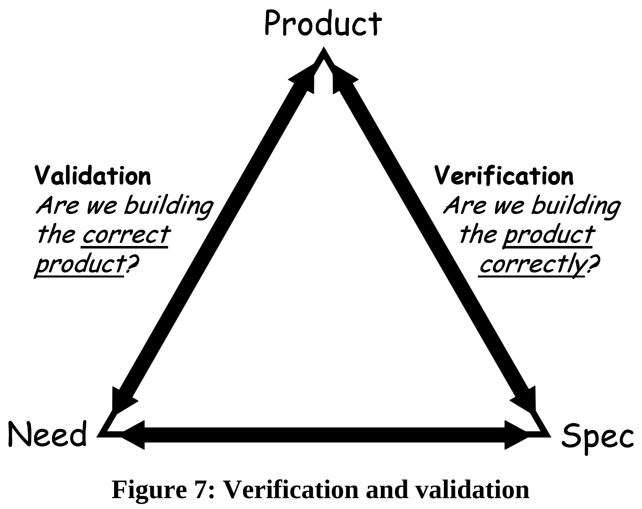
Product
Validation
Are we building the correct product?
Verification
Are we building the product correctly?
Need
Spec

Figure 7: Verification and validation

Since changes to a project's scope would require changes to the requirements specification, the PMBOK view of verification would in fact be regarded as validation according to Barry Boehm's definition!

The TLCM provides an opportunity to clarify the role of verification and validation.

Validation is represented on the triangle in two places:

between the user's need and the specification; and

between the product and the user's need.

In both cases, validation answers the question – are we building/have we built the **correct** product?

Verification is shown on the remaining side of the triangle between the specification and the product. It answers the question – are we building the product **correctly**?

Closing the gaps with verification

Verification is a technique for closing the specification-product gap during the Design and Construction stages of the life cycle. It achieves this by identifying discrepancies between the product and the specification. The discrepancies can then be corrected before construction of the product is completed.

As well as the final product, there are many interim work products that need to be developed during the life cycle. Many of these work products are documents such as architectural designs, detailed designs or test plans. Interim work products such as these can be verified against the work products from which they are derived. For example, a test plan could be verified against a detailed design document, an architectural design document as well as the requirements specification.

Testing is one of the techniques that can be used for verification. Testing is defined as:

…the process of exercising software to verify that it satisfies specified requirements; and to detect errors [Glo08].

Traditionally there are three different levels of testing performed during the Construction phase of the life cycle:

Component testing – the testing of individual software components. (Glo08)

Integration testing – testing performed to expose faults in the interfaces and in the interaction between integrated components. (Glo08)

System testing – the process of testing an integrated system to verify that it meets specified requirements. (Glo08)

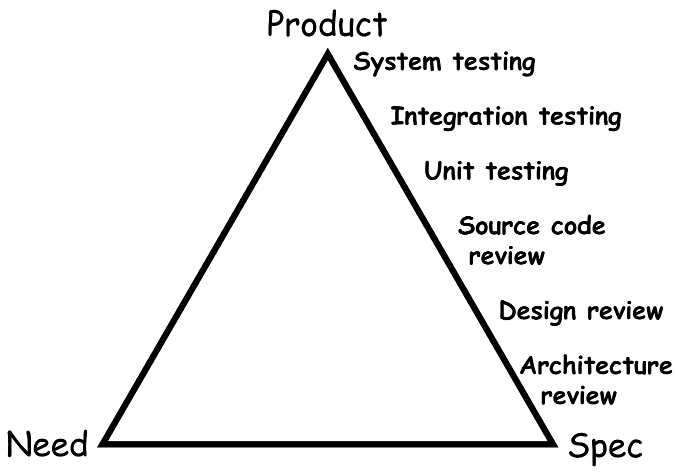

Figure 8: Closing the gaps with verification

All levels of testing can be used for verification, even though the definition for system testing is the only one that explicitly mentions verification. For example, integration testing can be used to verify the product against the system architecture and component testing can be used to verify components against detailed designs.

There are numerous work products that cannot be tested because they cannot be "exercised" (executed). For example, it is not possible to exercise documents, models or source code.

Reviews are a means of verifying work products that cannot be exercised. The IEEE standard for software reviews [IEEE1028] describes four types of review that can be used for verification:

Technical reviews – a systematic evaluation of a software product by a team of qualified personnel that examines the suitability of the software product for its intended use and identifies discrepancies from specifications and standards.

Inspections – a visual examination of a software product to detect and identify software anomalies, including errors and deviations from standards and specifications.

Walk-throughs – a static analysis technique in which a designer or programmer leads members of the development team and other interested parties through a software product, and the participants ask questions and make comments about possible errors, violation of development standards, and other problems.

Audits – an independent examination of a software product, software process, or set of software processes to assess compliance with specifications, standards, contractual agreements, or other criteria.

Closing the gaps with validation

Validation appears twice in the TLCM. Requirements validation takes place during the Requirements stage of the life cycle and is a technique for closing the need-specification gap. It achieves this by ensuring that the specification accurately describes the user's needs, wants and expectations.

Product validation takes place during the Deployment and Operation stages of the life cycle but it can only be used to measure the magnitude of the product-need gap. At these late stages of the life cycle, backtracking and rework will be required to actually close the gap. Product validation determines how well the completed product satisfies the user's needs.

Requirements validation

There are many different techniques that can be used for requirements validation. Four of the more popular techniques are:

- workshops;
- modelling;
- prototypes; and
- stakeholder reviews.

Workshops are good technique for ensuring stakeholder participation and resolving conflicting requirements. Workshops must have clear objectives and will require an experienced workshop facilitator who is responsible for ensuring that the workshop achieves its objectives.

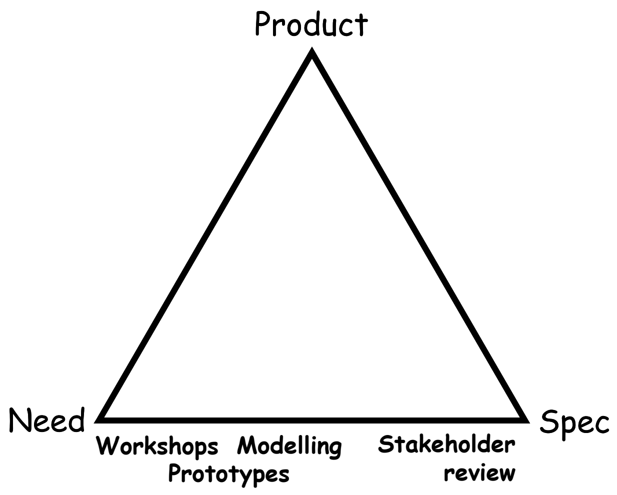

Figure 9: Closing the gaps with requirements validation

Workshops are superior to interviews as a means of gathering information because they provide an opportunity to resolve conflicting and inconsistent requirements. Often, workshop participants are able to describe a more coherent set of requirements by working as a team. However, the success of a workshop depends to a great extent, on the skills of the facilitator.

Natural language is inherently ambiguous. This makes it a poor choice for the precise description of requirements. In contrast, diagrams and models have the ability to describe requirements with less ambiguity. Diagrams and models are often more compact, easier to change and better at enforcing consistency than natural language. Modelling standards such as the UML [UML07] have further enhanced the clarity of diagrams and models.

The IKIWISI phenomenon means that users frequently have problems articulating their needs and reviewing formal requirements specifications.

Prototypes are a way to address the IKIWIS phenomenon. A prototype is a working model of the final product that can be demonstrated to (or possibly used by) stakeholders. Stakeholder feedback on the prototype can be incorporated into the final specification.

Stakeholder reviews are a type of technical review that includes participation by the stakeholders. They provide an opportunity for the stakeholders to provide feedback on the specification and ultimately confirm that it will serve as a reasonable basis for the development of the product.

Product validation

Testing is the most common technique used for product validation. While there can be many types of validation testing, acceptance testing is the type most commonly encountered.

Acceptance testing – formal testing conducted to enable a user, customer, or other authorized entity to determine whether to accept a system or component (Glo08).

Because acceptance testing can only measure the magnitude of the product-need gap, it is best viewed as an important life cycle milestone rather than as a technique for closing the gaps of the quality triangle.

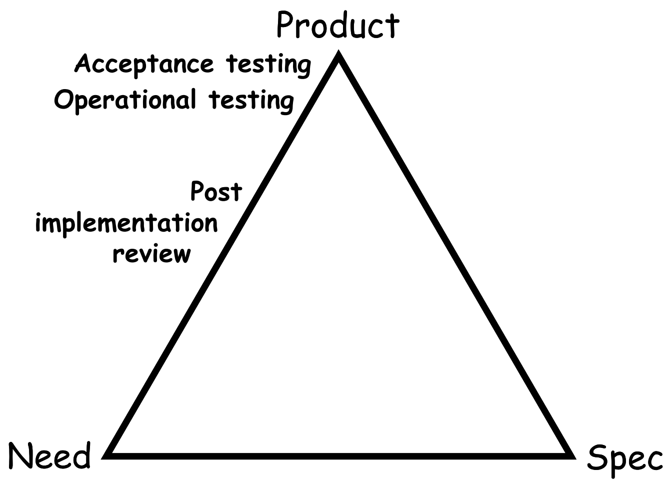

Figure 10: Measuring the final gap with product validation

In addition to acceptance testing which validates the product from the user's point of view, operational testing is sometimes performed to validate the product in its operational environment.

> *Operational testing – testing conducted to evaluate a system or component in its operational environment (Glo08).*

It is a widely held belief that reviews are inherently a verification technique. However, this is not the case. For example, it is sometimes appropriate to use a walk-through as a technique for validating a simple enhancement to a product or a defect repair.

Another use of reviews as a validation technique is to conduct a post implementation review after a product has been in operation for some time. A post implementation review validates the product in its operational environment and ensures that the product continues to meet the user's needs.

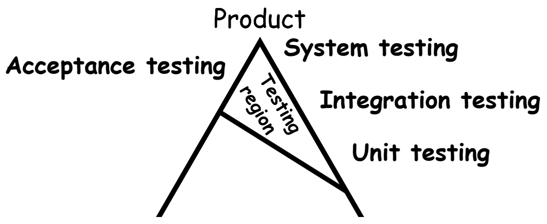

Figure 11: The testing region

Arranging the different types of testing around the quality triangle provides some insight into the somewhat limited role of testing as a verification and validation technique. As can be seen, the "testing region" encompasses only a relatively small area of the triangle and thus has a limited role in closing the gaps.

Closing the gaps with configuration management

Configuration management is concerned with the correct assembly of a product from its component parts. It is a management practice designed to ensure that the correct version of a component is used for each "build" of the product and that changes to the product and its components can be controlled, traced and tracked over time. [Ber97].

Configuration management can be used as a technique to close the specification-product gap during the design and construction phases of the life cycle. It achieves this by formally identifying different versions of a product and its components and by controlling changes to the product, its specification, its components and other interim work products.

A product may be assembled incorrectly as a result of selecting the wrong components or the wrong version of a component. Different versions of a product and its components will exist at different points in time. In addition, variants of a product may be created to meet the needs of different users and operational environments.

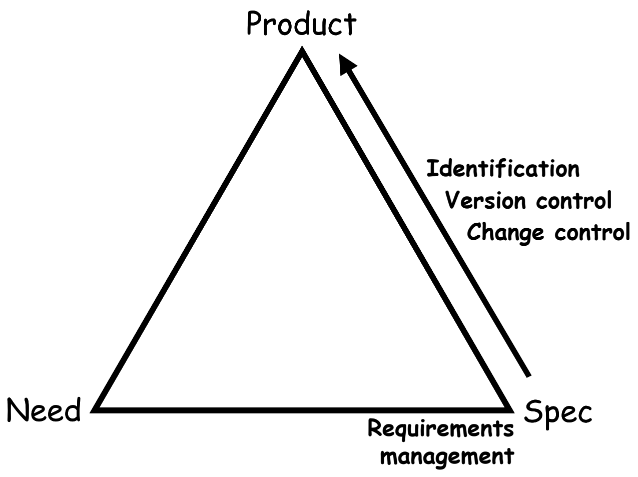

Figure 12: Closing the gaps with configuration management

A product that is assembled from the incorrect components is unlikely to conform to its specification. This effectively leads to an increase in the magnitude of the specification-product gap. Positive identification of components coupled with version control helps to ensure the correct assembly of a product and will close the gap between the specification and the product.

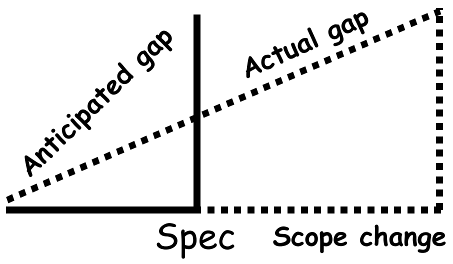

Figure 13: The effect of change on the specification-product gap

Change can have a subtle effect on the magnitude of the specification-product gap. Changing stakeholder needs after development has commenced will increase the magnitude of the specification-product gap but the increase may not be reflected in the specification. This means that the developers are often not aware of the increased gap.

The increased gap is often not discovered until acceptance testing performed during the Deployment stage. The solution to this problem is to ensure that the understanding of stakeholder needs continues to be updated during the Design and Construction stages of the life cycle.

Changes to requirements together with problems and inconsistencies identified during the design and construction stages will nearly always require changes to other interim work products. These changes together with the resulting changes to the product and its components need to be formally controlled. The ability to trace requirements to interim work products, allows the impact of proposed changes to be analysed before they are approved and implemented.

Closing the gaps with defect prevention

Activities performed during the Requirements, Design and Construction stages of the life cycle "inject" defects into a product, its specification, its components and other interim work products. The role of verification and validation is to identify these defects so that they can be removed.

In addition to removing individual defects, it is possible to identify "classes" of defects by applying error analysis – this involves collecting and analysing data for a large number of individual defects [Pen93]. Bug taxonomies provide a good example of some generic classes of defect [Bei90].

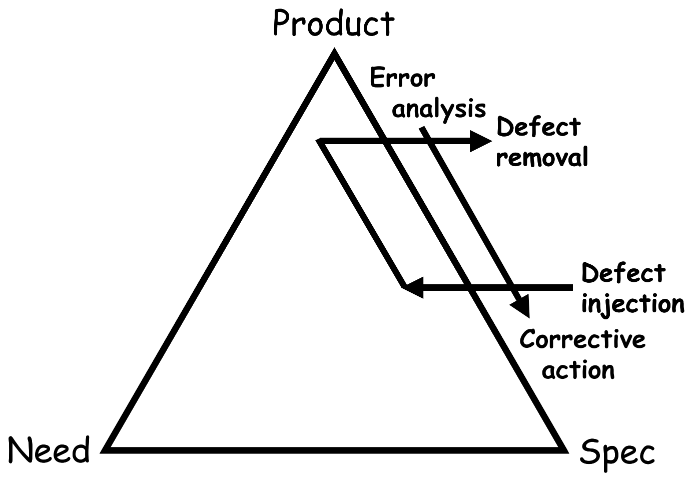

Figure 14: Closing the gaps with defect prevention

Defect classes can be used to predict the types of defect that will most likely be injected in the future and to take some form of corrective action to prevent this from occurring. Determining the underlying or "root" cause of a class of defects often helps to identify the most appropriate corrective action. Classes of defect can also be used to improve verification and validation activities by providing guidance on the most likely types of defect that will be found during testing and reviews.

Corrective actions might involve creating, revising or enforcing the use of standards, policies, procedures, checklists and other guidelines. In other cases it might involve changing activities performed, providing training for staff, reallocation of people or resources, or improving the effectiveness of life cycle audit activities.

Defect prevention can be used as a technique to close the specification-product gap during the Requirements, Design and Construction phases of the life cycle. It achieves this by preventing the magnitude of the gap from growing as a result of defects.

Closing the gaps with rework

Removing defects from a product, its specification, its components and other interim work products will normally require working backwards through life cycle to correct earlier errors and mistakes. Many activities that have already been performed will need to be performed again and many components and work products that have previously been completed will need to be modified.

The need to backtrack and revisit earlier life cycle activities is often referred to as "rework". Rework leads to additional development costs because activities are performed more than once. However, rework adds no value to the product as it simply corrects earlier errors.

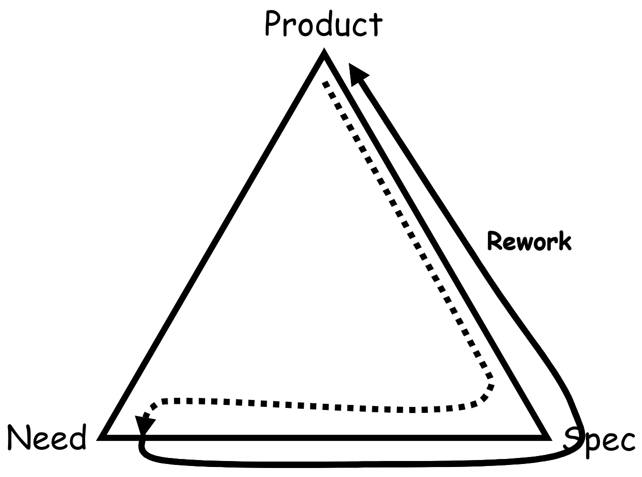

Figure 15: Closing the gaps with rework

Rework is an error prone activity that often injects many new defects into a product. These new defects will lead to more rework in order to remove them. The result is that rework frequently becomes a vicious circle that leads to large schedule and budget overruns.

Rework is probably the technique most widely used to close the need-specification gap during the Requirements phase of the life cycle and the specification-product gap during the Design and Construction phases of the life cycle. This is in spite of the fact that it is the least effective technique.

Closing the gaps with iteration

Iteration involves performing life cycle activities more than once. Although this may sound similar to rework, iteration is quite different. Rework consists of unplanned activities required to remove defects.

Iteration on the other hand, involves the successive refinement of a product, its specification, its components or other interim work products by repeating the stages of the life cycle.

It is important that each iteration is a planned with clear objectives, outcomes and deliverables in mind [Boe88]. The traditional waterfall life cycle milestones based on the approval and hand off of deliverables are not suitable for planning iterative projects. For this reason, many iterative life cycles are based on the following generic set off milestones (Boe99):

- Definition of the "Life Cycle Objectives" (LCO) in the form of the most important requirements together with their priority.
- Definition of the "Life Cycle Architecture" (LCA) in the form of an executable architecture that will support the most important requirements.
- Delivery of an "Initial Operational Capability" (IOC) that will allow the users to perform the first acceptance test.

Iteration also provides an opportunity for additional validation in the form of an iteration review. The findings of the iteration review serve as a major input to the planning of the next iteration.

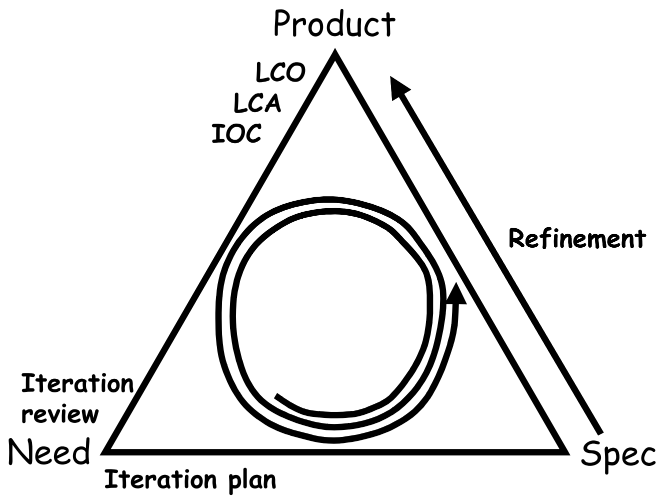

Figure 16: Closing the gaps with iteration

Iteration can be used as a technique to close all three gaps. It achieves this by repeating the life cycle stages thus providing multiple opportunities to close the gaps.

Closing the gaps with process improvement

The most common objective of process improvement is to improve the quality of life cycle activities and their outputs. It can also be used to achieve other objectives such as improving productivity or reducing costs. However, cost reductions are often only achieved as a by-product of improving quality. The reason for this is the manner in which quality contributes to the overall cost of a product [AS 2561].

Quality related costs have two components:

- the cost of poor quality primarily resulting from rework but may also including the cost of product support, product updates, complaint handling, concessions to disgruntled customers and loss of sales; and
- the cost of performing activities intended to close the gaps such as verification, validation, configuration management, defect prevention and additional activities associated with iteration.

The cost of poor quality is represented on the triangle by the product-need gap while the cost of closing the gaps is represented by the need-specification and specification-product gaps.

Spending money on closing the need-specification and specification-product gaps will result in a reduction in the magnitude of the product-need gap and a corresponding improvement in quality. If the increased spending on activities designed to close the gaps leads to a equal reduction in the cost of poor quality, then the improvement in quality has been achieved at no additional cost [Cro79].

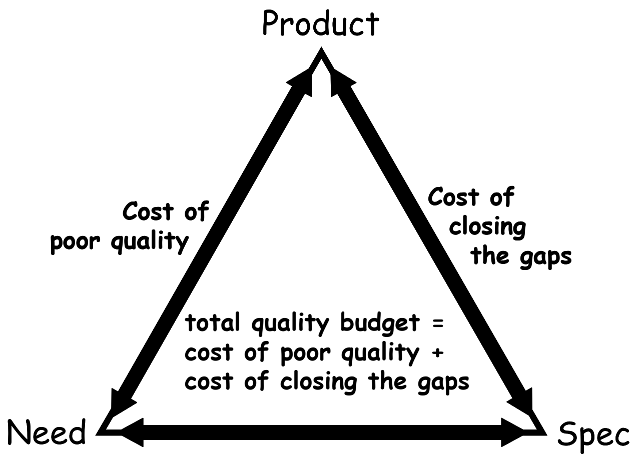
Product
Cost of poor quality
Cost of closing the gaps
total quality budget = cost of poor quality + cost of closing the gaps
Need
Spec

Figure 17: The cost of quality

Because the improvement of software development processes normally starts from quite a poor level of quality, it is not difficult to achieve a reduction in the cost of poor quality that is greater than the amount that has been invested in closing the gaps.

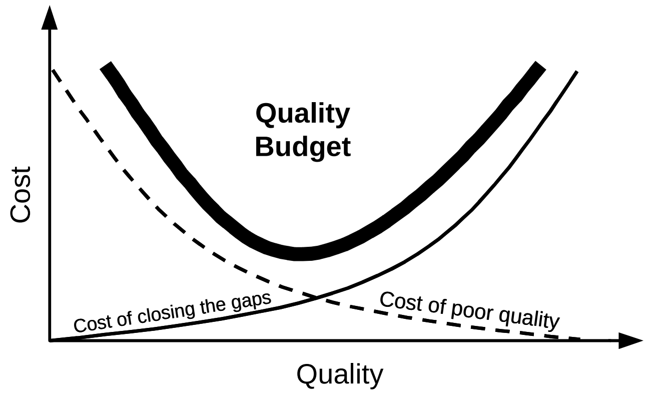

Figure 18: Plotting the total cost of quality

The right hand side of the graph shown in Figure 18 represents the traditional value view of quality (Gar84) that is based on how much a customer is willing to pay for quality. However, the left hand side of the graph represents the counter intuitive proposition that it is necessary to spend less in order to achieve better quality!

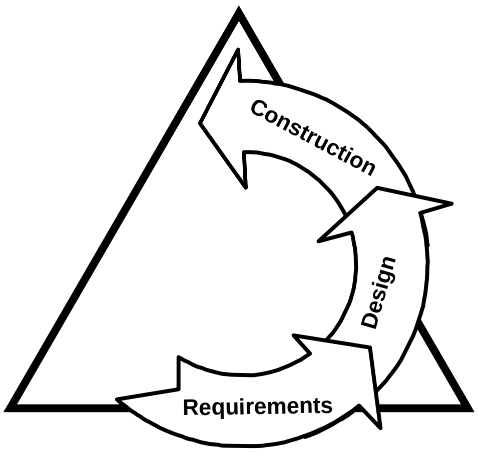

Figure 19: Closing the gaps with process improvement

Process improvement can be used as a technique to close the need-specification gap during the requirements phase of the life cycle and the specification-product gap during the design and construction phases of the life cycle.

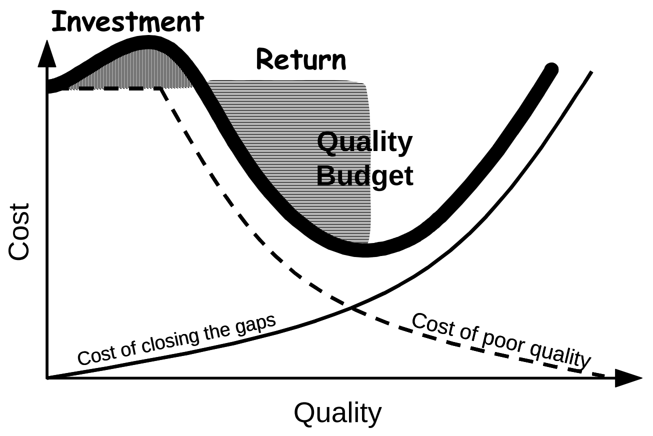

Figure 20: Investing in process improvement

However, there will always be a time delay between spending more on closing the gaps and a corresponding reduction in the cost of poor quality. This means that process improvement should be viewed as an investment proposition that will provide a return on the investment (ROI) at some point in the future.

“Triangular” maturity models

The shape of a triangle is determined by the relative length of its sides. It is interesting to compare the shapes of triangles that reflect different project scenarios.

The ideal triangle has a small need-specification gap and a small specification-product gap. The result is a product that meets most of the user's needs as represented by the small product-need gap.

In fact, a triangle representing a perfect product that met all of the user's needs would not be a triangle! This is because as the gaps shrink, the triangle becomes a single point with all three views of quality perfectly aligned.

The Communicate triangle represents a situation in which the need-specification gap is large but is corrected by the developers during the Design and Construction stages of the life cycle. This is usually achieved by communicating frequently with the stakeholders - hence the name of the triangle. This triangle can deliver products that meet the stakeholder's needs but will normally involve more rework than the ideal triangle.

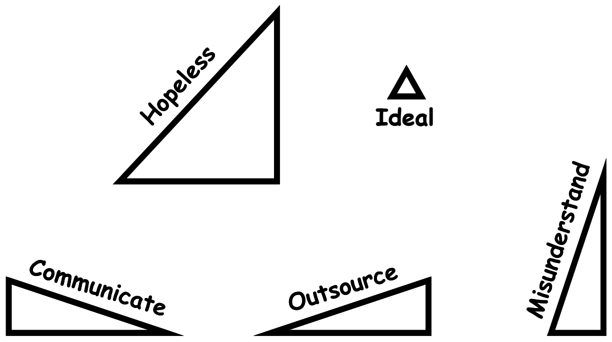

Figure 21: Triangular maturity models

The Outsource triangle has a large needs-specification gap but a small specification-product gap. The result is a product that fails to meet many of the user’s needs. This triangle often occurs when the Design and Construction stages of the life cycle are outsourced to a third party who delivers a product that conforms closely to its specification but the specification does not refect the true stakeholder's needs.

The Misunderstand triangle has a small needs-specification gap but a large specification-product gap. The result is a product that fails to meet a many of the user's needs. This triangle can occur for two reasons:

- the specification is very complex and difficult for the developers to understand; or
- the developers do not follow the specification.

The Hopeless triangle is, well simply hopeless! Large need-specification and specification-product gaps result in a product that manages to satisfy very few of the user’s needs.

A question of balance

Project management best practices such as those described by PMBOK are intended to have relevance to a wide variety of projects undertaken in many different industries. By aiming for universal relevance these practices often miss some of the subtleties of software development.

The TLCM is intended to fill this gap. However, is not intended as a alternative to project management practices but rather a way to supplement and enhance them with a software engineering perspective. The practices described in PMBOK can and should be applied to projects based on TLCM. It is hoped that the TLCM’s priorities of quality, scope, cost and time can provide a useful counterweight to the more dominant time, cost, scope and quality priorities of project management.

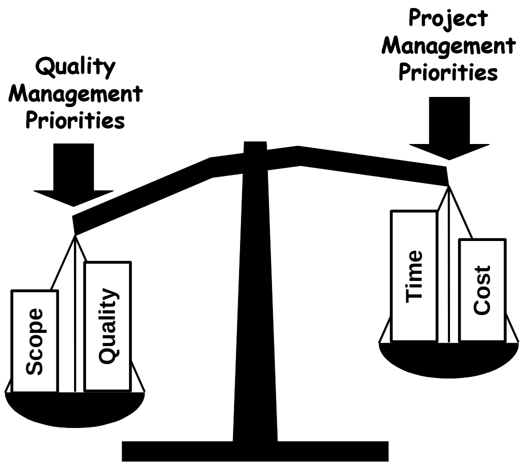

Figure 22: Balancing quality and project management priorities

[Lap08] Laplante, P. A., Neill, Colin J., “The Demise of the Waterfall Model and Other Urban Myths”, *Game Development*, Vol 1, No 10, Feb 2004.

[Roy70] Royce, W., "Managing the Development of Large Software Systems", *Proceedings of IEEE WESCON*, 1970.

[Wik08] *Urban Myth*, http://en.wikipedia.org/wiki/Urban_myth. Retrieved on 31st Jul 2008.

[PMBOK00] *A Guide to the Project Management Body of Knowledge (PMBOK)*, The Project Management Institute, 2000 (also available as IEEE Std 1490-2003).

[Sta95] *The Standish Group Report: Chaos*. http://net.educause.edu/ir/library/pdf/NCP08083B.pdf, Retrieved 7th August, 2008.

[Yor97] Yourdon, E., *Death March: The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects*, Prentice Hall, 1997.

[Pow04] Powell, Richard R., *Wabi Sabi Simple,* Adams Media, 2004.

[AS 2561] *AS 2561-1982: Guide to the determination and use of quality costs*, Standards Australia, 1982

[Gar84] Garviv, D., "What Does 'Product Quality' Really Mean?", *Sloan Management Review*, Fall 1984, pp25-45.

[Geo04] George, Michael L., et al. *The Lean Six Sigma Pocket Toolbook: A Quick Reference Guide to 100 Tools for Improving Quality and Speed*, McGraw-Hill, 2004.

[Boe99] Boehm, B. 1999. “Escaping the software tar pit: model clashes and how to avoid them”, *SIGSOFT Software Engineering Notes 24*, Jan. 1999.

[Boe79] Boehm, B., “Guidelines for Verifying and Validating Software Requirements and Design Specifications”, *IEEE Software Volume 1, Issue 1*, 1984.

[Glo08] *Glossary of Software Testing Terms*, http://www.testingstandards.co.uk/glossary.htm, Retrieved on 1st Aug 2008

[IEEE1028] IEEE Std 1028-1997 IEEE Standard for Software Reviews.

[UML07] *OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2*, Object management Group (OMG), 2007.

[Ber97] Bersoff, E. H., "Elements of Software, Configuration Management," in *Software Engineering,* M.Dorfman and R. H. Thayer, Eds., IEEE Computer Society Press, 1997.

[Pen93] Peng, W. W. and Wallace, D. R., *Software Error Analysis,* NIST Special Publication 500-209, 1993.

[Bei90] Beizer, B., *Software Testing Techniques*, Van Nostrand Reinhold, New York, 1990.

[Boe88] Boehm, B. “A spiral model of software development and enhancement”, *SIGSOFT Software Engineering*, 1986.

[Cro79] Cosby, P., *Quality is Free*, New American Library, 1979.