A dilemma that numerous software engineering organizations face is to determine whether quality, and quality programmes should become part of the organizational strategy. Almost more important, how should the implementation of quality programmes in software engineering organizations be approached?

STRATEGIC QUALITY - A SOFTWARE ENGINEERING APPROACH

By Roelof van Staden, Rand Afrikaans University

Organizations realize that the quality of a service or product could make the difference between existing or closing down, hence an organized approach to quality is needed – a quality strategy. The software engineering industry faces it's own specific challenges, one of these being the quality of software systems.

The author has found numerous resources for identifying and measuring software quality, but has noticed a general lack of practical guidance in implementing quality approaches as part of the organizational strategy. This article investigates the creation of a process framework for continuous improvement and measurement needed to align the strategic and quality goals of the software organization.

1. The Need for Software Quality

A general view of quality is the totality of features and characteristics of a product or service to satisfy specified or implied needs. Juran also states "fitness for purpose", and other authors mention "conformance to requirements" as elements of software quality [6], [13].

These statements imply that a link exists between software quality and user needs; i.e. the more closely the software matches the user requirements (needs), the higher the quality of the software. Conformance to requirements supports the concept that

quality should be built-in during the design phases, because software quality is generally determined by the earlier phases of the software development process [4].

Gillies [2] offers some insights into general software quality issues. He states that:

- Software quality can have many different meanings in different situations.
- Software quality is multidimensional with many contributing factors being difficult to summarize in a simple, quantitative way.
- Software quality is subject to constraints such as cost and resources.
- Software quality criteria are generally dependent on other quality criteria and often interact with each other, causing conflicts.

In reality, software quality is more than just conformance to user requirements, it is about compromises and trade-offs to create software systems that provide tangible benefits to the organizations that utilize these systems. Quality function deployment charts (QFD) are among the tools that can be applied to decide on critical quality factors in the search for an optimum compromise.

It is interesting to note that people generally spend about 50% of their time to write a program, and then the other 50% of their time trying to find the errors. In some programs that have been tested, more than 10 faults per 1000 lines are observed, which equates to 10 000 faults for a program that has 1 million lines of code [4].

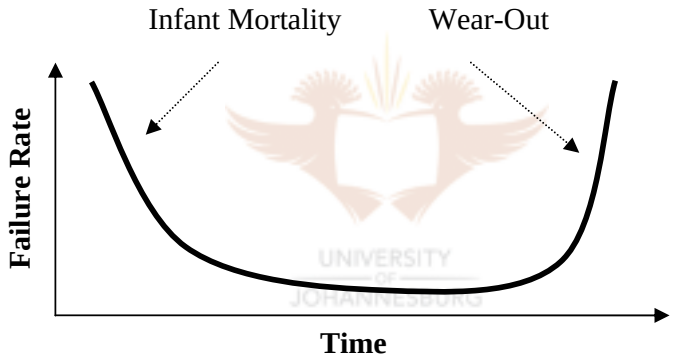
According to the Software QA/Test Resource Center the main reason that is being given for this unfortunate set of events is that software is generally not tested very well [3]. Other conditions that are cited include [3], [13], [16]:

- The complexity of modern software packages.
- The pressure to get the software product to market.
- The software industry does not accept any liability for errors.
- Poor work methodologies.
- Miscommunication or even no communication at all.

- Egos, because people like to believe that they are capable of many things.
- Poorly documented code.
- Software development tools that are flawed and contain errors.

1.1 SOFTWARE RELIABILITY

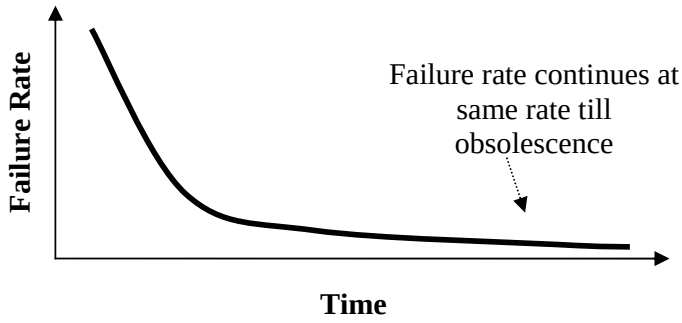
One of the software myths is that "software does not wear out", which is factually true, but practically incorrect. Software does deteriorate with age, and this is one of the reasons why software reliability has become an important topic in the past two decades. Software deterioration is due to the fact that the environment surrounding the software application changes (hardware upgrades, software upgrades, data corruption, etc), causing the software to crash or experience failures.



Source: Pressman [13], Figure 1.2

Figure 1: Hardware Failure Curve

The "bathtub" curve, shown in Figure 1 depicts hardware failures, indicating that hardware exhibits high failure rates early in its life (design or manufacturing defects) [11], [13], [14]. As the defects are corrected, the failure rate drops to a steady-state level for some time. As time passes the hardware will start to experience rising failure rates. This is simply the graphical depiction of hardware that is starting to wear out [11], [14].

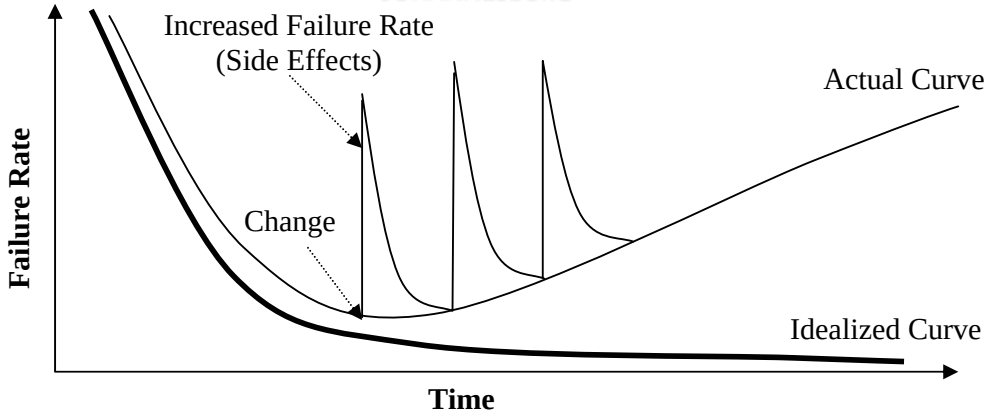


Source: Pressman [13], Figure 1.4

Figure 2: Idealized Software Failure Curve

Software, on the other hand, is not susceptible to the environmental factors that affect hardware. Hence, in theory the failure curve for software should look something as shown in Figure 2. Defects such as incorrect logic that are not discovered will lead to high failure rates early in the life of software. However, in the ideal world these will be discovered and fixed, which leads to the flattening of the failure curve for software [13].

In the real world, software will undergo maintenance (change) which will lead to new defects being introduced, causing the failure rate to increase (spike), not decrease, as shown in Figure 3.



Source: Pressman [13], Figure 1.3

Figure 3: Actual Software Failure Curve

In order for the curve to return to the original steady-state idealized curve, more changes are requested, causing the curve to increase and spike again. This leads to an increase in the minimum failure rate, which depicts the deterioration of the software due to change [13]. This is a highly simplified model of software failures, but it does

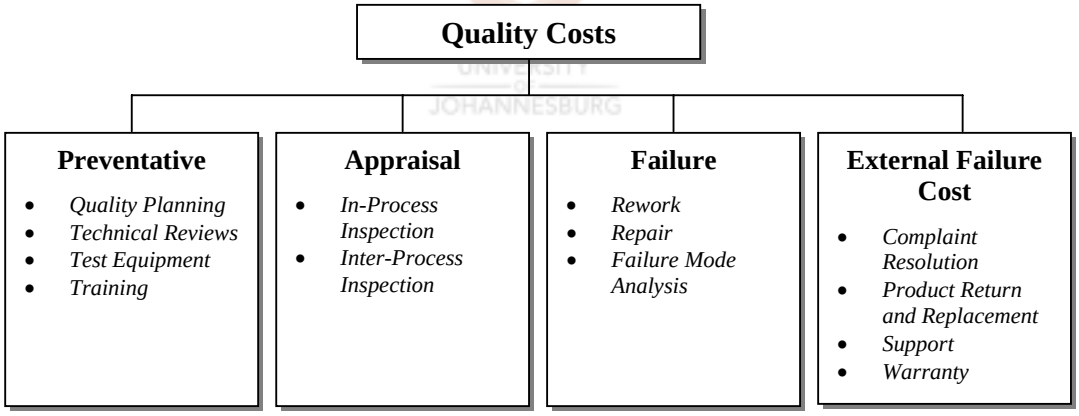
illustrate one important point – software does not wear out…but the software can deteriorate.

An important aspect of wear illustrates one of the major differences between software and hardware. If and when a hardware component wears out, it is replaced by a spare part. Software does not have "spare parts". Any error in software indicates an error in the design, code or the process through which the design was implemented in machine code. Hence software maintenance involves more complex processes than the simple replacement of a hardware item.

1.2 IMPACT OF CHANGES TO SOFTWARE

Changing software requirements, or even the actual code of the software will result in changes to the cost of developing the software.

Quality costs include all the costs incurred in the pursuit of quality or in performing quality related activities [1]. Cost of quality studies are conducted to know the current cost of quality and to find out the opportunities for reducing the costs of quality and to provide a basis for comparison [1], [9].



Source: Kit [9]

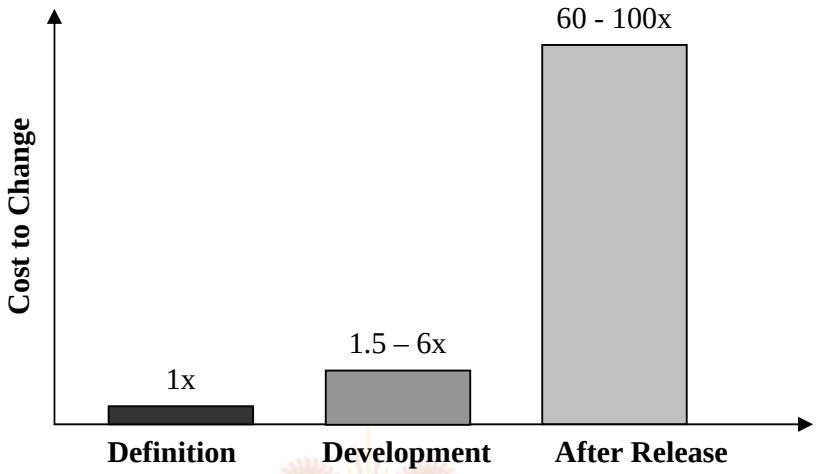
Figure 4: Quality Costs

Figure 4 depicts the four general groups of quality costs; preventative, appraisal, failure, and external failure costs. These are general quality costs, and are typically used in cost of quality studies.

A lack of quality generally results in increased costs. A Price Waterhouse study [4] found that problem-solving accounted for more than 50% of the total effort in the

software industry. The problem solving included problem resolution during testing, changing functionality to meet misunderstood requirements and the maintenance on production systems.

Software requirements often change, and the impact of the changes to the software is not always felt until late in the software development lifecycle as shown in Figure 5.



Source: Pressman [13], Figure 1.5

Figure 5: The Impact of Change on Software

If serious attention is given early in the development cycle to ensure that the project definition is clear, changes can be accommodated relatively easily. The customer will be able to review the requirements and will be able to recommend modifications with relatively little cost impact. If the changes are requested during the software design phase, then the costs escalate rapidly. The reasons are that resources are now committed and have already started working on the project, and a design framework is in place that will have to be changed. Once the project has reached the implementation stage (coding, testing), any change will lead to severe impacts on cost [13].

Figure 5 also indicates the costs associated with changes requested once the software application has been placed into production. The costs are estimated at an optimistic factor of between 60 and 100 times the cost of changes requested at the time of project definition. This can lead to an *exponential* cost increase [13].

2. Defining the Organizational Strategy

Strategic planning is a management function that is aimed at the improvement of the organizational functions, products and services [10], [12], [17].

In order to produce a world-class quality strategy to enhance, strengthen, and improve the organization's competitive advantage, Kinlaw [8] states that:

> *"…there are many tools that people need in order to undertake improvement projects in a careful and systematic way. They need problem-solving tools, such as flow-charts and cause-and-effect diagrams. They need to know to create run charts, histograms and Pareto charts. They need statistical tools to control their systems. They need to know to plan and collect data and how to use check sheets."*

Together with all the tools and techniques a clear vision or overview of the process of improvement and measurement is needed. Software teams need an improvement and measurement model, and they require guidance in using such a model [7]. A strategic framework is used to establish such an improvement and measurement model for software engineering organizations.

In order to get everyone in the organization to change their behavior, to learn and to find ways to improve the way the organization functions, it is necessary to *develop a strategy, and to implement the strategy* [10], [15]. It needs to be understood that *a strategy's main aim is to improve the accomplishments and profitability of the organization in the short- and long-term on a sustainable basis*. An organizational strategy for a software engineering organization can be defined as a plan or tactics for improving current processes and methods.

The business concept of the organization needs to be understood in order to decide how the organizational strategy will be defined. As suggested in Figure 6 the nature of a business is defined by the targeted market preferences, the product offers, and the key activities.

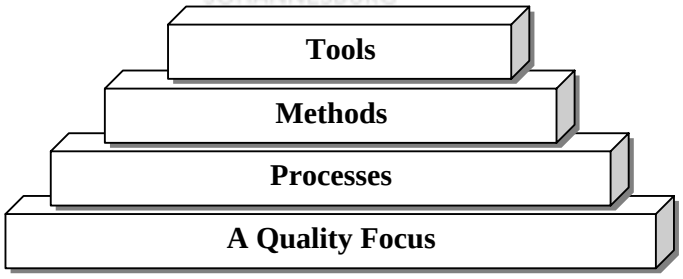
Source: Sanchez [15], Figure 6.2

Figure 6: The Three Elements of the Business Concept

Key activities are those activities that an organization should perform well in order to create and deliver product offers to the targeted markets and customers, i.e. software systems. The key activities in any organization's business concept become top priority concerns of management in designing core processes in an organization's strategy [15].

The key activities of a software engineering organization covers all facets of software engineering. Software engineering principles covers issues such as customer satisfaction, on-time product delivery, measurement and metrics, and mature processes.

Software engineering can be considered a layered technology as shown in Figure 7:



Source: Pressman [13], Figure 2.1

Figure 7: Software Engineering Layers

Figure 7 illustrates that the organizational approach to engineering (including software engineering) should rest on a commitment to quality. Any quality philosophy leads to continuous process improvements, and this ultimately leads to mature approaches to software engineering [13]. This illustrates the point that quality is a critical success factor of the software engineering process. The strategic management

of the organization needs to give serious attention to issues of quality as it affects the nature of the software engineering business.

3. Strategic Software Quality Framework

Management is responsible for the definition of responsibilities, the quality policy, and the quality strategy for maintaining or improving the quality of the organization's products or services [17]. The software quality policy should relate to their commitment and positive belief of the quality philosophies, principles and practices.

The quality policy is the first substantial visible evidence that management is serious about what the organization wants to achieve with quality. The quality policy is a guideline of what is to be done, not how, and should be guiding in nature. It is also extremely important that such a quality policy be applied to the entire organization.

The strategy that needs to be undertaken when dealing with software quality depends on the state of the software engineering processes in the organization.

Level	Description	Definition
1	Initial	*The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort.*
2	Repeatable	*Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications.*
3	Defined	*The software process for both management and engineering activities is documented, standardized, and integrated into an organization-wide software process. All projects use a documented and approved version of the organization's process for developing and maintaining software. This level includes all characteristics defined for level 2.*
4	Managed	*Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled using detailed measures. This level includes all characteristics defined for level 3.*
5	Optimizing	*Continuous process improvement is enabled by quantitative feedback from the process and from testing innovative ideas and technologies. This level includes all characteristics defined for level 4.*

Source: Created by the Author from Pressman [13]

Table 1: SEI Process Maturity Levels

The SEI process maturity levels are shown in Table 1. Very few organizations are at level 5, and even if the organization is at a very high level or maturity, this indicates that continuous improvements are constantly taking place – exactly the kind of mentality needed to keep on improving the quality levels [6]. Table 1 can be considered both a measurement and improvement tool, i.e. it shows at what level of maturity the software engineering organization is, and what processes and functions are needed to reach a higher level of maturity.

Table 2 shows the key performance areas that are required for the organization to attain a specific level on the SEI CMM grading system. The key performance areas can be used as a guideline to assist the organization in reaching a higher level on the CMM process maturity scale [6].

Level	Definition	Key Performance Area
2	Repeatable	*Software configuration management* *Software quality assurance* *Software subcontract management* *Software project tracking and oversight* *Software project planning* *Requirements management*
3	Defined	*Peer reviews* *Intergroup coordination* *Software product engineering* *Integrated software management* *Training program* *Organization process definition* *Organization process focus*
4	Managed	*Software quality management* *Quantitative process management*
5	Optimizing	*Process change management* *Technology change management* *Defect prevention*

Source: Created by the Author from Pressman [13]

Table 2: Process Maturity Key Performance Areas

The key practices are policies, procedures and activities that must occur before a key process area has been fully instituted. The quality department is generally responsible for the definition of the software quality policies, and the planning and implementation of the software quality procedures and activities. Management commitment is essential for the successful implementation of the quality plan.

Once the quality policy has been defined, it will serve as the integrating factor that quantifies the guiding principles of the mission statement into quality objectives. The policy may be scrutinized by both internal and external parties. This means that the organization will have to deliver what it promises or risk affecting its survival.

The quality policy should be developed in conjunction with all employees within the organization. The policy illustrates the serious intention of top management, and as such should be written down and distributed to all employees.

The following factors need to be considered in developing a quality policy [5]:

- Who, what and where are the customers?
- What products/services do they require, and when?
- What are the competitors' intentions and what does their quality policy indicate?
- What is the focus of the quality mission?
- Who should be involved in developing the quality policy and who is going to lead its formation?
- Should the supplier(s) be involved?

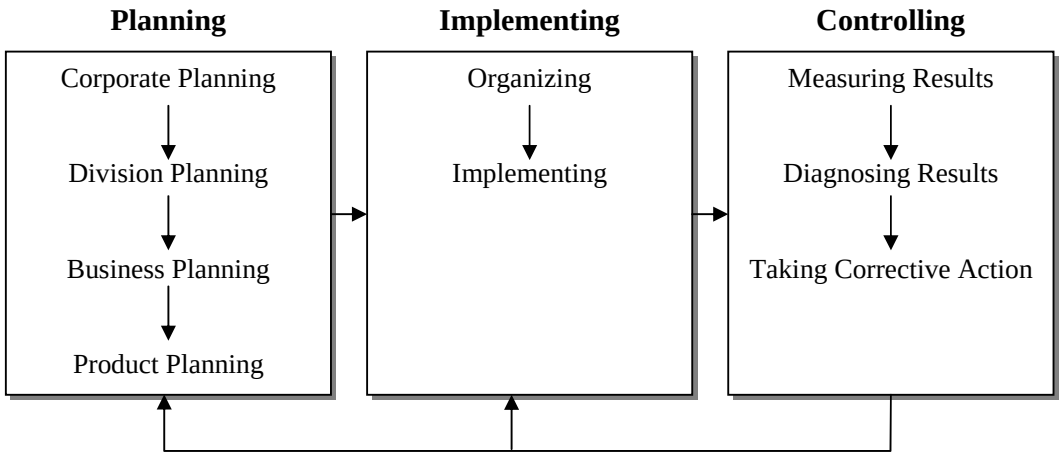
The overall approach is to develop a policy that can be accepted organization wide. The development of the organizational quality policy should not preclude the formulation of localized quality policies at departmental level. The sub-policies at departmental level should be ancillary to the quality policy. The formulation of sub-policies should be encouraged as it leads to further involvement of staff at operations level.

3.1 PLANNING, IMPLEMENTING, CONTROLLING THE SOFTWARE QUALITY STRATEGY

Kotler [10] outlines a method for achieving the goals of a new strategy based on a feedback loop, with the planning phase being the first major milestone in the process. The second milestone is the implementation phase, followed by a control and

measurement section of the process. At each of the planning and implementation stages, feedback should be given, resulting in an improved process [10].

The basic process is shown in Figure 8:



Source: Kotler [10], Figure 4.1

Figure 8: Strategic Planning Process

This methodology can also be applied when implementing a quality strategy for a software engineering organization.

Kinlaw [8] states that:

> *"Improving work processes is the only way to achieve substantive and long-term effects in the quality of services and products."*

With this in mind, the following steps shown in Table 3 can be used as the outline for a quality improvement process.

Step	Detail	TQM Tools
1	Understand the opportunity or problem	*Models* *Brainstorming* *Nominal group technique* *Surveys* *Flow charts* *Cause- and effect diagrams* *Pareto charts* *Histograms* *Run charts* *Scatter diagrams* *Control charts*
2	Define the specific improvement target	*Flow charts* *Cause- and effect Diagrams* *Pareto harts* *Histograms* *Scatter diagrams*
3	Define strategies to reach the target	*Models* *Brainstorming* *Nominal group technique* *Cause- and effect diagrams*
4	Design the data links	*Surveys* *Observations* *Devices (mechanical, electronic, pneumatic, optical)* *Interviews*
5	Design the response process to use data from the data links	*Pareto charts* *Histograms* *Control charts*
6	Determine how the project will be managed	*Project management sheets* *Milestone charts*

Source: Kinlaw [8], Figure 3-1

Table 3: Project Steps and Tools

The tools used for the attainment of each goal is also shown in Table 3. This table is not meant as a definitive guide to quality improvement, but can be used as a guideline to achieve the quality targets of the software engineering organization. The quality goals should be [15]:

- Determinable.
- Actionable.
- Measurable.

- Specific.

Implementation, which is the actual doing part, is different from the final stage in the quality framework – controlling (measuring, diagnosing, taking corrective action). If sufficient planning has not taken place before the implementation phase, then the work carried out could be wasted.

Quality plans require continuous monitoring in order to determine their effectiveness. This means that for all levels of the quality plan there should be monitoring systems. It is a bottom-up process, where the development of the quality plan is a top-down process. One of the crucial elements of the quality plan is the generation and recording of quality data. The quality data can be generated using quality tools, to provide statistical as well as real measures of quality performance [6].

Monitoring and evaluating quality data should also be used to perform strategic and operational quality plan evaluations. These measures can check the viability and effectiveness of the designed quality plan against the measured outcomes. Using such monitoring systems should ensure that organizational issues of the quality plan can be addressed. Quality audits also need to occur at this stage, i.e. independent and formal reviews of quality related performance issues of the quality plan.

3.2 STRATEGIC QUALITY PROCESS FRAMEWORK

The preceding sections describe the individual steps that culminate in the creation of the author's strategic quality process framework. The basic outline of the process framework is shown in Figure 9.

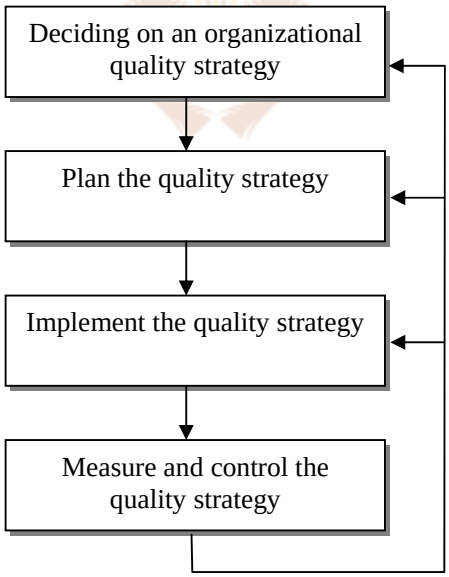
The first step in the strategic quality process framework is the establishment of an organizational policy regarding the quality of the software products and services of the software engineering organization. Deciding on a quality strategy is the most important step in the process, as all the preceding steps rely on the guidance provided by the strategy.

The second step in the strategic quality process framework is to plan for the quality strategy. The planning activities should be characterized by project management activities such as brainstorming, the use of Pareto charts, Ishikawa diagrams, models, PERT charts, etc.

The third step following the planning activities is the implementation stage of the strategic quality process framework. Activities such as requirements analysis, software design and development should take place during the third stage conforming to the policies and procedures that were decided upon during the planning stage.

The fourth and final step in the process framework measures and controls the activities that take place during the entire process. Statistical methods and software testing activities should aim to ensure that the software quality lives up to the promise of the quality strategy.

The measurement and control activities of the process framework should provide ongoing feedback to all other activities as shown in Figure 9. The aim of the quality process framework is to create an organization strategy for the software engineering organization that will lead to lasting gains in the quality of the software engineering process, activities, services and products.

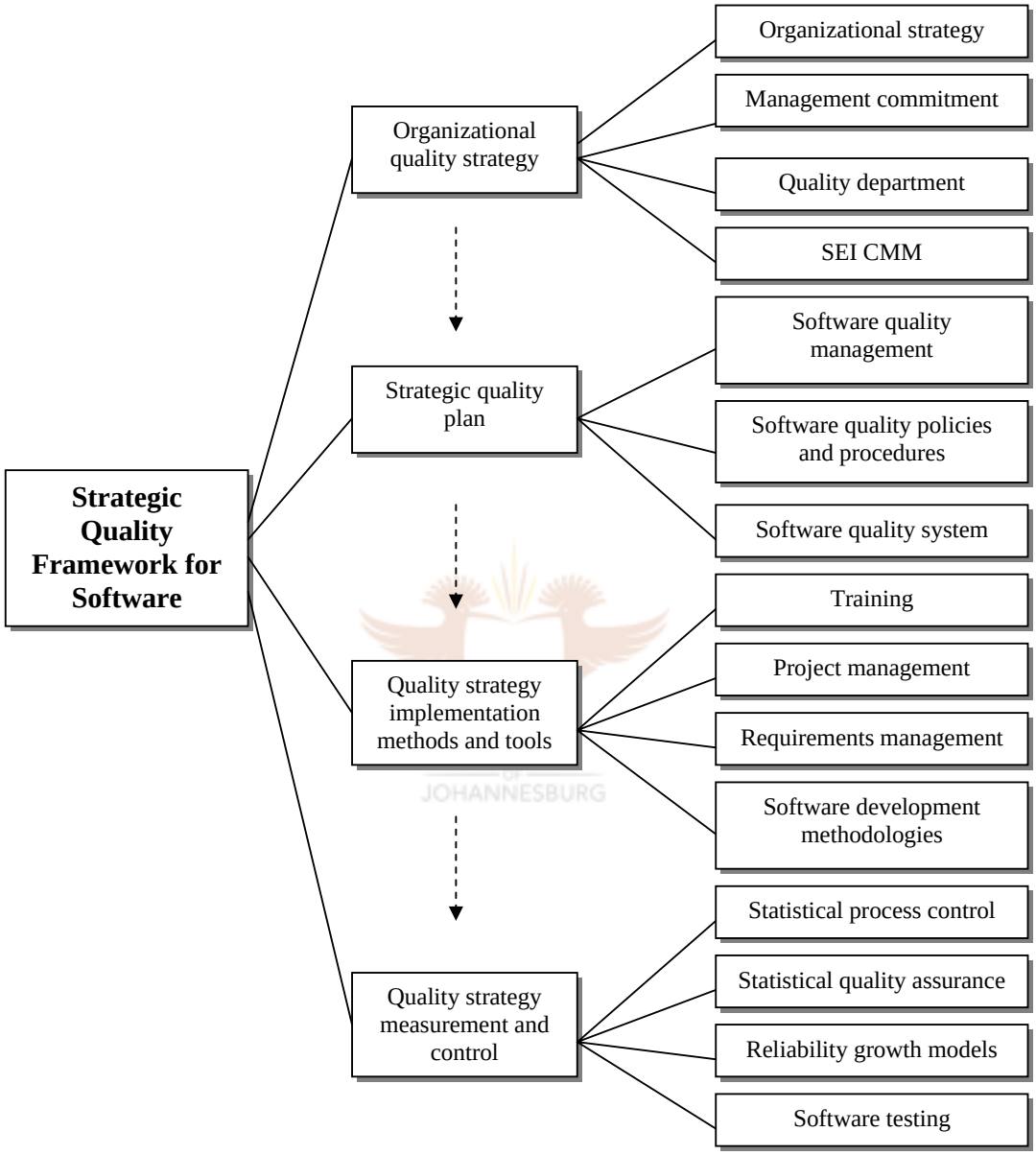


Source: Created by the Author

Figure 9: Strategic Quality Process Framework

A more detailed representation of the strategic quality framework is shown in Figure 10. It illustrates some of the functions, processes and methods that would typically be associated with the intermediate steps of the quality process, i.e. measurement and control would make use of statistical methods to provide measurement data to

management. The framework should be seen as a dynamic framework, adapting to the needs of the organization.



Source: Created by the Author

Figure 10: Basic Strategic Quality Framework

The framework aims to align the organizational strategy with the software engineering strategy as it relates to quality of the software systems. The strategic quality framework is based upon a feedback process to ensure that corrective measures can be

taken to keep on improving all aspects of the software engineering process to provide software that is of a better quality.

4. Conclusion

Quality assurance is an essential activity for software engineering organizations that produce software products and systems. Software developers and consultants generally agree that high-quality software is an important goal [9]. In order to produce software systems of a high quality, a strategic quality framework as created by the author is essential to coordinate all the activities necessary in the development of the software system.

The strategic software quality framework is a guideline for the improvement of the quality of software engineering products and services. The framework should be adapted to suit the organization where it is applied, adding or even removing quality functions and processes that will not add value to the overall objectives of the organization. The framework features a measurement and control feedback loop, and the process of examining quality data should be performed on a regular basis to ensure that the framework is indeed improving the quality of products and services.

The topics of quality and strategy and the links between the two offer a wealth of research opportunities. The effect of organizational strategies on quality efforts should be investigated further, in an attempt to measure and define the benefits that are gained from adopting such practices. The software engineering industry is a fast paced environment, and moving towards an approach of improved quality is exciting and challenging. This may just lead to a change in the way organizations operate in the future.

5. References

1. Freeman-Bell, Gail and James Balkwill, *Management in Engineering*, 2nd ed., Pearson Education, 1996.
2. Gillies, Alan C., *Software Quality: Theory and Management*, Chapman & Hall, 1992.
3. Hower, R., *Software QA and Testing Resource Center*, QA/Test Resource Center, 2003, <http://www.softwareqatest.com/qatfaq1.html>. (23 May 2003)
4. Ince, Darrel, *Software Quality and Reliability: Tools and Methods*, UNICOM, 1991.
5. James, Paul, *Total Quality Management: An Introductory Text*, Prentice-Hall, 1996.
6. Kan, Stephen H., *Metrics and Models in Software Quality Engineering*, Addison-Wesley, 1995.
7. Kaplan, Robert S. and David P. Norton, *The Balanced Scorecard Measures that Drive Performance*, Harvard Business Review 70(1) pp. 71-79, 1992.
8. Kinlaw, Dennis C., *Continuous Improvement and Measurement for Total Quality: A Team-Based Approach*, Pfeiffer and Company, 1992.

9. Kit, Edward, *Software Testing in the Real World*, McGraw-Hill, 1998.
10. Kotler, Philip, *Marketing Management*, 11th ed., Prentice Hall, 2003.
11. O'Connor, Patrick D.T., *Practical Reliability Engineering*, 4th ed., John Wiley & Sons, 2002.
12. Porter, Leslie J. and Steve Tanner, *Assessing Business Excellence: A Guide to Self-Assessment*, Butterworth-Heinemann, 1996.
13. Pressman, Roger S., *Software Engineering: A Practitioner's Approach*, 4th ed., McGraw-Hill, 1997.
14. Ramakumar, Ramachandra, *Engineering Reliability*, Prentice Hall, 1993.
15. Sanchez, Ron, and Aimé Heene, *The New Strategic Management: Organization, Competition and Competence*, John Wiley & Sons, 2004.
16. Smith, David J., *Achieving Quality Software*, 3rd ed., Chapman & Hall, 1995.
17. Stevenson, William J., *Operations Management*, 7th ed., McGraw-Hill, 2002.