A framework for development of a predictive model for software process quality

H. Younessi, D.D. Grant

School of Computer Science and Software Engineering,
Swinburne University of Technology, Hawthorn, Victoria, Australia 3122

Abstract

We have commenced an ambitious project which aims to develop a formal and predictive model of software process quality.

Central to our perspective is the notion that software product quality is dependent on process quality, which is in turn dependent on the organisational context, on the methodological approach used to develop software, and on the technology utilised. Accordingly, a predictive model for process quality is of significance in constructing a process to be used to develop software with specified product quality requirements.

In this paper we present a framework for the development of the predictive model for process quality. We view the concept of the utilisation of a software ‘system’, and therefore the issue of determination of its quality, as a ‘Human Activity System’, whose boundaries are in fact context dependent. Accordingly, we take a ‘systems’ approach to the construction of the framework for model development, based on general systems theory and the Soft Systems methodology.

In proposing a framework for the conduct of our research we have followed recent advice of Adrion and Glass in respecting the need for following accepted scientific or engineering methods. In extending this to consider the human dimension, we have embraced the orientation of Hirschheim, which is inspired by recent developments in the social sciences.

1 Introduction

1.1 Context

In recent years, the adoption of sound software engineering practice has resulted in improvements in the ability to deliver software that meets quality requirements. Despite this, the steady reports of software failures continue to be publicised [22].
In an important recent article, Glass [12] has drawn attention to the failure of software engineering researchers to provide solid evidence to support their continuing proclamation of the state of ‘software crisis’. Our local personal experience suggests that indeed there are many significant software projects that still go off the rails, for many different reasons. On the other hand, we also know of many successful software projects. Whilst we may no longer suffer a ‘crisis’, much remains to be done in establishing an expectation that quality software will be produced as a matter of course.

In [7] Davis provides a glimpse of his forthcoming book (“201 Principles of Software Engineering”) by describing his top 30; of these the first two are “Make Quality Number 1” and “High-Quality Software is Possible”. Agreeing with this, we are motivated not by the need to resolve a crisis, but rather by the need to sustain a continuous improvement in the quality of software.

1.2 The Enigma of Quality
What makes writing quality software so problematic?
To begin with, the concept of quality itself is enigmatic. Davis [7] acknowledges that the notion of quality in relation to a software product often means different things to the different actors involved with the development and use of the product, and that these notions may be contradictory. In other words quality is not absolute and depends on the purpose or viewpoint for which it is being assessed.

As an example, in order to determine the quality of a bottle of wine, it is first necessary to determine the purpose for which the bottle is being purchased. These could include:
- To enjoy with a meal
  Here it is necessary to know the type of meal being served, the time of day for the meal, the season and the ambient temperature, amongst other considerations. Once these are known, a number of criteria will be used to decide on the wine to be consumed.
- To take to a party
  What one needs to know here is whether one is likely to be the only person drinking the wine or will one be sharing? If one is to share, who will be there? What will their taste in wine be? .....  
- To impress a guest or a host
  Here one needs to know about the recipient’s level of knowledge of wine, their current financial situation, their socio-economic background, their taste in wine.

Having determined the purpose and the issues that need to be considered relating to that purpose, only then will one be in a position to use specific criteria to choose a particular bottle. In order to enjoy the wine with a meal, assuming a good knowledge of wine, the criteria might be: region, style, bouquet, colour, acidity and price; otherwise the choice might be made on the basis of producer, attractiveness of label and price. A wine to be taken to a party is probably selected using similar criteria, yet price and style may probably play a more significant role. If the purpose is to impress, criteria to
use will be price, rarity of vintage, and possibly packaging. Having decided on the criteria of assessment, what constitutes a good price or an appropriately rare vintage or good colour? Some decisions are easily made. For example it is a matter of a simple numerical comparison to chose a bottle with a smaller price, but such determination is hard when measurements are absent, say in the case of bouquet. Even if measurable, such assessments are highly personal and culturally based. For example a price of $50.00 for a bottle may be low for one person and unreachably high for another. Determinations of this nature, especially those that do not fall into a universally accepted scale, are based on the assessor’s past history (social status, place of birth, education, financial status, and a myriad of other influences).

Accordingly, we recognise that determination of the quality of an artefact is heavily culturally based.

It should be noted that in our example the choice of a bottle of wine is often not solely made on the measure of one criterion. Assessment usually means weighing up merits and levels of a number of criteria that are important for assessing the quality for a given purpose. It is also conceded that despite all of the above, there are wines that are just “better” than others, seemingly irrespective of purpose.

Firstly, a “better” wine, it seems, is one that rates highly by the majority of assessors on the majority of criteria. Note however that being a “better” artefact does not mean that it is often “the artefact of choice”. If you can not afford to drink the wine, you will never know if it is a quality wine. Without experiencing the artefact, discussion of quality is meaningless.

Secondly, the rule of majority or authority often creates an aura of “desirability”, or “popularity” around a certain product, irrespective of its internal properties as assessed by the person experiencing it. We can identify wines which we acknowledge as being of high quality (because of our respect for experienced wine judges), yet which we dislike.

The above observations indicate to us that quality is determined through assessing the appropriateness of a product for fulfilling a given purpose, and is determined by:

i. determining the purpose for assessment;
ii. determining the premises for determination;
iii. determining the criteria for assessment;
iv. measurement against such criteria;
v. weighing up measurements against each other in view of the premises.

This view of quality is in accord with the popular definition of quality as “fitness for purpose”. It recognises that the selection of purposes, viewpoints and criteria, and the process of determination of quality are multi-faceted.

1.3 Implications for Software Quality
Software is an artefact that is subject to quality determination. Curtis identifies seven reasons (neither orthogonal nor exhaustive) why organisations develop software and information technology [5]. These are:
Software Quality Management

the current system cannot cope;
- cost savings;
- the provision of better internal information for decision making;
- the provision of competitive customer service;
- the opportunities provided by new technology;
- high technology image;
- change in legislation or regulations.

Any one of these reasons, or any combination, might be the context that may be used to determine the “purpose” with respect to which quality is to be assessed.

As noted by Davis [7], the “purpose” of a system may be viewed differently by different individuals in an organisation. For example the “purpose” of a particular MIS may be, according to the CEO of the firm, to provide a strategic alliance between the firm and the manufacturer of the system or the firm and its customers. To an administrative officer, the purpose of the MIS may be to assist her through her daily administration. In a scenario like this it is quite possible that the CEO may consider the MIS as meeting its purpose, yet the administrative officer, forced to battle with a poorly designed user interface, may not share this opinion.

The enigmatic nature of quality has contributed to considerable confusion in the investigation of software quality in order to find ways of improving it.

Research towards software quality improvement has traditionally been focused on functionality (ie, meeting stated requirements) and the premise or viewpoint has been that of the “construct” of software. This issue is however often interwoven into those of politics, user acceptance, utility, cost of purchase, equity and strategic ownership, to name but a few. These are all legitimate perspectives from which to view software quality, yet they cover a much broader spectrum than we wish to handle. We still have some way to go to ensure our ability to build functional and reliable software, and until this takes place, other considerations and perspectives tend to be viewed as secondary. Accordingly, whilst recognising the other potential facets through which software quality can be viewed, we limit ourselves essentially to the concept of “functionality”, the definition of which is itself open to interpretation, and which needs to be particularised in context as part of the quality assessment exercise.

Current hurdles to determining the quality of software include the following.

I. It is very difficult - if not impossible - to meet an undefined target

As mentioned above, software quality is very subjective in nature. Despite many years of research in the field, there is still no universally agreed definition of software quality, let alone a measurement system whose validity and suitability has been agreed. This makes defining and communicating a particular level of quality exceedingly challenging, resulting in what is ultimately a futile exercise of “shooting in the dark.”

II. Views differ as to what represents quality

This again is a consequence of the multiplicity of perceptions of the various
stakeholders as to what is a quality product. These perceptions are impacted not only by technology, economics and negotiating position, but also by culture, philosophy and the psychology of the viewholders. Determining what represents quality therefore becomes akin to “feeling around in the dark”.

III. Products and processes are not clearly related in a theoretical context.

Neither researchers nor practitioners have yet established a direct, explicit and well understood relationship between the characteristics of the development process to be employed and its capability of yielding a product of a given quality. In this sense, software development resembles “stabbing in the dark.” (This is despite the existence of the SEI CMM [19] which provides mechanisms to assess the maturity and capability of software processes. The model is essentially reductionist in nature and is a mapping of many aspects of organisational and process characteristics into a linear 5 point scale, and is fundamentally comparative in nature. The CMM also does not explicitly consider the assessment of product quality from diverse human perspectives.)

1.4 Research Aims
The central aim of this research is to develop a formal and defensible model for the prediction of the fitness of a software process in its ability to produce quality products.

We offer an explicit attempt to address all three issues cited in the previous section in order to provide software process and software engineering researchers with a means to design and/or evaluate software development processes with a high degree of formality, enabling meaningful comparisons.

We aim to provide practitioners with means to:
- evaluate (measure) the relative quality of existing software development processes in producing high quality software;
- predict the level of suitability of an existing software development process to yield a pre-defined level of final product quality;
- predict the level of suitability of a new software development process to yield a pre-defined level of final product quality;
- assist in the design of new software development processes suitable for generating products of a required quality.

The outcomes of this project should assist in further highlighting the relationship between a process and product(s) generated by that process and the impact of one on the other. This can assist research in areas where processes are under study in terms of their ability to yield quality results. Examples include software process modelling [3], and business process re-engineering [6,16].

2 Software Product Quality

In any manufacturing process, it is on the quality of the product that the company’s external reputation is going to rest. In the context of software
quality, it is imperative to focus initially on software product quality. Software is largely a commercial product, and as such subject to market orientation [20]. Market orientation necessitates a customer view of product quality. The quality of a software product, at least from a commercial viewpoint, is largely assessed through explicit measurement of a set of attributes, which must be external and customer-oriented.

2.1 External Quality Attributes
External attributes are those which can be measured in relation to the context or the environment in which the product persists [8]. For example, modularity (internal) and reliability (external) are both attributes of a software product. However, the average customer is likely to be far more interested in the reliability of software they will receive than whether proper abstract data types have been developed to increase the modularity of the product.

External product attributes of importance relate to:
- **Functionality** (How closely does the product meet the requirements as stated or implied in the requirements document?)
- **Reliability** (What is the likelihood of failure free operation?)
- **Useability** (What is the extent to which the product is convenient and practical to use?)
- **Maintainability** (What is the level of ease with which software can be corrected, adapted and/or enhanced to fit an altered set of requirements?)

It should be noted that a number of attributes including reuseability, efficiency and cost effectiveness, conceivably might be added to the list above. We have not included them for reasons enumerated below.

*Reuseability* measures the degree to which components of a software product can be used in constructing other software of similar utility. This is largely not an external product attribute but an internal one. (It is also in most cases of greater concern to the developer than to the customer.)

*Efficiency* measures the degree to which software efficiently uses the hardware resources available to it. Efficiency can be argued to be absorbed into the concept of functionality, as usually the resource utilisation requirements of a software system feature prominently in a requirements document.

*Cost Effectiveness* may be considered to have (at least) two distinct meanings when considered as a software quality attribute. Firstly, it may represent a measure of the level of efficiency and economy by which available resources were or are being utilised in the process of creating a software product. This is clearly a process attribute and for the moment lies outside the scope of our discussion. Secondly, it may represent the extent to which the price paid for a software product is recouped through effective and productive utilisation of the product in the organisation. In this context, the effectiveness becomes a function of other external attributes of software quality such as reliability and functionality, and also of how intelligently the customer has put the product to use. The cost part becomes a function of the cost effectiveness of the process, and also the marketing decisions made as to the level of profit to
be made by the software vendor and in any case, in our opinion, is not a direct and external product quality attribute.

(Yet, the question of what quality is a function of cost and therefore cost effectiveness is pervasive. Our response to this question reviews the content of our opening section in that the issue of quality is multi-dimensional and therefore with respect to certain product “purposes” or “perspectives” of quality, cost may be a criterion and often is one. For example when assessing the effectiveness of a process of building an artefact such as software, cost effectiveness is likely to be paramount. Yet when assessing the quality of the “construct” of software - that is an assessment of how a software product is put together - cost is not a consideration. In other words a product is either put together well or not, irrespective of how much money was spent on putting it together. A product of a given quality of construction is as fit for the purpose it is to be put to irrespective of whether it cost $X to build it or $10X. This however does not mean that other quality assessment perspectives will be unaffected by an increase in cost of ownership. One that may be particularly impacted is the “level of strategic utility” of the software, measured through a number of criteria including the return on investment of owning and operating that software product.)

To conclude the discussion on external attributes, it is conceded that on occasion there are some internal attributes, to an extent identifiable and measurable and utilised by “informed” customers which may be predictive of important external attributes of software which are generally harder to measure statically. Reusability, or modularity are good examples of such internal attributes.

2.2 Customer Oriented Quality Assessment
Ultimately responsible product marketing dictates customer orientation. (This is not meant to imply, however, that customers can not be educated in what they can expect, or converted to concur with possibly more appropriate or realistic alternatives set forth by a vendor, or for that matter that the views, requirements, restrictions and abilities of a vendor are not also important.) Accordingly, product attributes assessed to determine the quality of a product must be meaningful to, and in line with, the quality aspirations of the customer. For example from the external attributes mentioned above, reliability is a good example of a customer oriented attribute, whereas reusability is a developer oriented one. As such, whilst customers are critically conscious of the reliability of software, they are likely to be largely unconcerned with the degree of the reusability of the components making up the source code. (We concede an exception in the case of management, as customers of internally developed software, being interested in the economics of a company’s MIS Department.)

Using the criteria above, we have investigated and classified a variety of product attributes [8,11] as follows:
A. Important customer oriented external product attributes:
   - Functionality (including efficiency)
B. Important developer oriented external product attributes:
   - Maintainability (see below)
   - Reuseability

As per our earlier discussion, we recognise that “informed” customers may, on occasion, identify and establish links between developer oriented quality attributes and customer oriented ones. In such occasions the former attributes also become important to the customer. A good example of this is Maintainability.

A primary task of this research is the development of a measurement for product quality in terms of its external attributes, including the ones above. This will be a topic upon which we will expand in section 5.

3 Processes and Process Quality

As noted earlier, we base our research on the perspective that the quality of a process (in our case a software development process) and the quality of the product it yields are directly related. For example the reliability and maintainability of software is likely to be directly related to the robustness of the process employed in its development and to the degree to which it can generate readable, modular code. In this context, once adequate measures of external product attributes are determined, it is our goal to link these to “process attributes” that define the types of process that ensure the delivery of the stated level of product quality. To do this, we also need to investigate processes and their characteristics.

3.1 Process: a Working Definition
Identification of a fully comprehensive set of influences is germane to a successful predictive model for software quality. In discussing the relationship between the quality of a product and the process that yields such a product, we postulated that the external product quality attributes can be deemed as functions of “process attributes”. This in turn implies that a software process must have a methodological dimension and a technological aspect [14]. Furthermore, there is also an undeniable organisational dimension [19] which pervades any actual instantiation of a specified software process. Therefore for our purposes, we define a software process as: “the collection of technologies and methods utilised within a particular context by an organisation in order to produce a software product.”

This definition differs from those, prevalent in the 1970s and 1980s, that equate a software process and a development methodology in that they imply that the mere adoption of a software development methodology is sufficient for ensuring increased software quality [25]. Such a definition fails to explain why a given formal methodology succeeds in one situation or in one organisation
Similar arguments also hold with respect to definitions that only consider the organisational context (e.g. only the extent of the organisation’s capability maturity [24]) or the technological aspects in isolation.

Whilst we believe the stated definition is a workable one, we do recognise that the dimensions cited may not be 1) exhaustive, or 2) orthogonal.

In fact, most contemporary definitions for process at least implicitly recognise the existence of all the three dimensions mentioned in our definition.

3.2 A Process as a Human Activity System
We assert that software development has a significant human dimension. This implies that a software process can be classified as a Human Activity System (HAS) [4,22], which further strengthens the argument that the study of the issue of software quality - built on the concepts of product attributes, the multiplicity of their perceptions, and also that of processes - has a significant human dimension.

The study of the issue of software quality from a human activity perspective, however critical, must not be made in isolation [22] and to the exclusion of empirical and experimental approaches. This is so, especially if the ultimate aim is the development of formal and metricated models of process fitness prediction that will be useful in practice.

The next two sections of this paper explore the characteristics of an appropriate research framework that we propose to follow.

4 Characteristics of an Appropriate Research Framework

4.1 Scientific/Engineering Research and Interpretivist Research
In [2] Adrion proposes a taxonomy of possible research models for software engineering. According to that taxonomy, it has been proposed by Glass [12] that extant research in software engineering largely follows the analytical method, condemnation characterised as advocacy research.

Most work done in the area of software quality and its relationship with software products and processes has been done from a software engineering standpoint [19]. This body of research has been largely quantitative in nature with heavy reliance on experimentation and case studies [13,16]. The best of this research can be accepted as following Adrion’s scientific method, which requires validation of hypotheses and repetition of results. In the absence of these, we have only advocacy.

Another popular form of software engineering research is one in which new technologies, methodologies and tools are created. This type of research is largely developmental and as such is akin to research done in the traditional engineering fields such as electrical engineering [1]. In Adrion’s taxonomy, such research follows the engineering method where it takes account of existing technologies, methodologies and tools, and seeks measurable and
measured improvements. Without the anchor in current practice and the measurement of improvement, we have only advocacy.

Nonetheless, the scientific and engineering approaches are vital for our purpose as they - amongst other benefits - can assist in forming the basis for the formal metrics we are to develop.

Yet in the study of software quality, organisational and human aspects are also paramount as indicated in the previous section. It has been shown [4] that scientific/engineering or “hard science” research approaches are often of limited utility - or are at least unnecessarily restrictive - when the study concerns organisations, and involves human interaction, and contains social, economic, cognitive, political, cultural, and ethical dimensions. The taxonomy of Adrion is inadequate to encompass these dimensions. We must turn to recent work in Information Systems research to extend our horizons.

In his taxonomy of research methods for information systems, Galliers [9] classifies IS research methods into two classes according to the table below:

<table>
<thead>
<tr>
<th>Research Categories and Approaches</th>
<th>Scientific</th>
<th>Interpretivist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lab. Experiments</td>
<td>Subjective/Argumentative</td>
<td></td>
</tr>
<tr>
<td>Field Experiments</td>
<td>Reviews</td>
<td></td>
</tr>
<tr>
<td>Surveys</td>
<td>Action Research</td>
<td></td>
</tr>
<tr>
<td>Case Studies</td>
<td>Descriptive</td>
<td></td>
</tr>
<tr>
<td>Theorm Proof</td>
<td>Futures Research</td>
<td></td>
</tr>
<tr>
<td>Forecasting</td>
<td>Role/Game Playing</td>
<td></td>
</tr>
</tbody>
</table>

Table 1. A Taxonomy of Research Approaches

For the purposes of the remainder of this article, we include the full taxonomy of Adrion under the general term “scientific”, in order to provide the major distinction with interpretivist “systemic” approaches.

Checkland strongly argues [4] that the interpretivist systemic approaches such as action research, are more suitable for study of human activity systems.

4.2 The Need for a Multi-paradigm Approach
Both scientific and interpretivist “systemic” approaches are essential, in different contexts and instances, in the study of software quality.

The two approaches, unfortunately, have been distinct for a long time, with distinct underpinning’s for either approach (physical sciences, mathematics and reductionism for the former and a tradition of subjective, descriptive and interpretive approaches typical of the social sciences for the latter) [10].

Recent developments in information systems have widened this scope and fields of study such as this one, business process re-engineering, process design and Human Computer Interaction (HCI) have necessitated the merging of these
two previously distinct modes of research. (see Figure 1 below).

![Figure 1: A Merging of Approaches](image)

A multi-paradigm view is necessary in the investigation of software quality issues in order to avoid the rather naive view of product and process quality that disregards models and issues of social, cultural and human computer interaction needs - to mention but a few. It is stressed that such a rich expanse of relevant pre-existing research from sociology, anthropology, management and psychology, especially social and cognitive psychology [10] can be critical in increasing the effectiveness of research in this field.

5 A Framework for Development of our Model

5.1 Use of a Systems Paradigm Framework
The activity of seeking to construct a predictive model of process quality itself is a human activity and as such can be appropriately investigated through a systemic approach. In order to conduct this investigation, we have employed the Soft Systems Methodology (SSM), a highly successful systemic methodology well suited for investigative work and planning purposes [4].

Having applied the methodology to the domain in question, the “conceptual Model” presented in Figure 2 has emerged as a framework of action. This top level “conceptual model” along with many lower level models of each of the top level activities (omitted here), defines a framework of purposeful action which is designed to guide the investigation towards its intended logical and defensible results.

In the next section, to conclude the paper we will briefly examine each of the activities arising from the conceptual model.

5.2 A Brief Description of Steps to be Taken.

5.2.1 Define product quality The aim of this step is to identify product attributes that seem to be universally accepted as important in defining the quality profile of a product.
454 Software Quality Management

As reported in an earlier section, this work has been largely done through surveying current literature with particular emphasis on software quality on the one hand and product quality on the other. Through this process, we have hypothesised that: customers assess product quality primarily through the external attributes of functionality, reliability and useability whilst developers assess product quality principally through the external attributes of maintainability and reuseability. We aim to examine these hypotheses through a survey of a cross section of customers, developers and researchers. The results of this survey will influence our perception of how product quality is assessed.

5.2.2 Measure product quality Once a stable set of product quality attributes has been agreed, we investigate those internal attributes and characteristics that contribute to the composition of the external attributes of our agreement. Based on the composition of these attributes, we will attempt to produce a numerical measure of each primary attribute to yield a measure of product quality.

5.2.3 Develop hypotheses, determine a framework of process attribute identification, and identify process attributes that define a developmental
process This work follows the inspiration of Hirschheim et al [18] and is largely based on the contention rooted in recent works in sociology and social philosophy particularly the Social Action Theory of Habermas [15] that any process of change can be identified by four distinct “orientations” described as Instrumental, Strategic, Communicative and Discursive.

Secondly, as indicated before, we suggest that a generic model of a software development process is composed of three “domains”, those of Technology, Methodology and Organisation [21]. In other words, developers approach the activity of software development through a mixture of four different orientations and influence, utilise and are influenced by three domains. We propose to use these two dimensions of “orientation” and “domain”, to define a matrix - initially proposed by Hirschhiem [18] - that can act as a framework for classification and/or determination of the principles, strategies and methods of software development and also for reporting of software development outcomes.

5.2.4 Examine these hypotheses and propose, formulate and validate relationships In this largely empirical step we will attempt to validate and formalise the relationships asserted in the previous step. This will yield a number of individual relationships between process and product.

5.2.5 Consolidate and unify The next step is to bring together the diverse array of relationships found in the previous steps and to formulate a unified and predictive model of software process quality from the perspective of its fitness to yield a product of given quality.

5.2.6 Test and validate The proposed consolidated model needs to be validated through empirical assessment and case studies.

References

456 Software Quality Management


