Measuring Quality: A Cornerstone of Theory in Software Engineering

Howell Jordan  
Lero @ University College Dublin  
Ireland  
howell.jordan@lero.ie

Rem Collier  
University College Dublin  
Ireland  
rem.collier@ucd.ie

Abstract—In any engineering domain, a detailed understanding of what constitutes a 'good' product is vital for the development of theories that are both general and useful. However, software engineering researchers’ understanding of desirable product qualities is not yet fully mature, especially for continuously-evolving software systems. Inspired by two historical examples, this paper calls for a discipline-wide effort to precisely define the attributes and variables of software product quality in a measurable way. We expect this effort will lead to two major contributions. Firstly, the defined attributes and variables should act as units in any general theory of software engineering. Secondly, once instruments to measure these attributes and variables are developed, systematic large-scale empirical studies of software product quality will become much easier, eventually yielding a rich corpus of data which should prove fertile for further theory building.

Keywords—Software quality; Software measurement

I. INTRODUCTION

It seems certain that the scope of any general theory of software engineering must include not only software developers and development methods, but also the software product itself. The task of building a general theory of software engineering then consists of identifying relationships among the attributes and variables - or units - of these objects [1]. This paper aims to spark a discussion about which product attributes and variables should be included in such a theory, and how they should be measured. Since the utility of a theory is determined by the hypotheses that can be derived from it, we begin by asking: what predictions should a universal theory of software engineering be capable of making?

II. PREDICTION AND SOFTWARE EVOLUTION

In mature engineering disciplines, theories from the mathematical and physical sciences are used extensively to ensure that a proposed product will meet the customer’s requirements. Similarly, many software engineers assert that the correctness of programs should be assured through mathematical proof [2], or more generally that “To build something good... you have to predict in the design stage the qualities of the end product” [3]. In this section we will argue that a broad definition of quality is required, if theory is to help developers achieve long-term customer satisfaction. A limitation of the proof-of-correctness view is that a definitive ‘end product’ does not always exist. Many software systems continue to evolve long after deployment, often leading to variants and enhancements far in excess of their original scope. Therefore it is crucial that software is not only correct, but also easy to modify.

Despite advances in requirements elicitation and specification techniques, it is likely that software evolution will become increasingly common for three reasons. Firstly, customers don’t always know what they want. A customer - afflicted by the I’ll Know It When I See It (IKWISI) syndrome - may be unable to provide a system’s requirements until a prototype has been delivered, and this is especially common for graphical user interfaces [4]. Secondly, for any product operated by humans there is no such thing as a perfect design: “the problems of multiple users or changing fashion or new aesthetics will always be lying in wait” [5, ch.3]. Even for software with no human user interfaces, as interconnection between systems grows, updates to keep pace with external changes become routine [4]. Finally, future variant systems, if any, are likely to be accommodated in software where possible, due to the relatively malleable nature of software as an engineering medium. For example, “most car manufacturers now offer engines with different characteristics ... frequently these engines... differ only in the software of the car engine controller” [6]. In summary, a practical unifying theory of software engineering should be capable of predicting all qualities that are important to a long-lived evolving family of software products.

III. DEFINING PRODUCT QUALITY

Software product quality is often modelled as a hierarchy of attributes [7]. While some attributes such as reliability are well defined, there is no overall agreement on the content or structure of this hierarchy; in particular, many attributes relating to the nonfunctional properties of software and its evolution are poorly understood. We argue that a major research effort is required to clearly define them.

From a scientific perspective, it is crucial that such definitions are not predicated on specific software technologies. For example, several authors have proposed object-oriented coupling and cohesion (e.g. [8]) as measures of maintainability, and this view is supported by empirical evidence
in certain contexts (e.g. [9]). However, these definitions are rooted in the object-oriented paradigm, so they cannot be used to compare, for example, object-oriented and functional programming approaches. It follows that any theories built on these definitions cannot predict which method or language would be preferable under given conditions - and these are currently some of the most significant questions in software engineering research [3].

IV. EXAMPLES FROM ENGINEERING HISTORY

The history of software engineering is relatively short, so for illustrative examples of product quality definition problems we must turn to other engineering disciplines.

A. Tacoma Narrows Bridge Collapse, 1940

The first Tacoma Narrows bridge, which opened in July 1940 and collapsed five months later, is a well-known example of design failure due to incomplete theoretical knowledge. The collapse was caused by violent torsional oscillations in the bridge deck, induced by aeroelastic fluttering. The design, while ambitious in its scope and daring in its economy of materials, did not apparently violate any contemporary theories of good bridge building [10, ch.9]. However it is surprising to note that, by 1939, flutter in aircraft wings was theoretically quite well understood, and had been widely reported for at least two decades [11]. The Tacoma Narrows disaster can thus be viewed as a collective failure to understand that aeroelasticity is an important attribute of bridge design quality.

An investigation into the disaster was led by Burt Farquharson of the University of Washington, who began his study of the bridge soon after its opening, and was present at the time of its collapse. Figure 1 shows a still image of the bridge on that day, taken from 16mm Kodachrome video footage which was fortunately recorded by the owners of a nearby camera shop. To the left of the picture, an optical gauge constructed by Farquharson’s team can be seen; this allowed the bridge’s vertical oscillations to be precisely measured and recorded using a film camera positioned on the shore1. These observations and measurements were crucial to the disaster investigation, and ultimately led to the integration of aeroelastic flutter into mainstream bridge engineering theory during the 1950s [12]. In the intervening period, the Bronx-Whitestone bridge, which is of similar design, was strengthened against the symptoms of aeroelastic fluttering by adding extra material [10, ch.9]; however such quick-fix solutions to quality issues are rarely available in software engineering.

It is likely that the first Tacoma Narrows bridge was not the first bridge to be damaged or destroyed by aeroelastic fluttering [11]; and some authors consider that the 1940 collapse might have been prevented if these earlier incidents had been observed, measured, and studied in more detail [10, ch.9]. Collectively, the history of long-span suspension bridge design illustrates that a rich set of observations and measurements may be an essential prerequisite to successful theory building. The lessons learned from Tacoma Narrows also suggest that software engineering may have much to gain from detailed studies of project failure.

B. Langley Field Aircraft Experiments, 1919-1941

Once recognised, the problem of a poorly-understood product quality attribute can apparently be overcome by a concerted research effort.

Before 1920, the manoeuvrability of aircraft could not be predicted at design time, and was mostly a matter of trial and experience. Prototype aircraft sometimes exhibited dangerous handling characteristics, and often these flaws could only be corrected by costly and time-consuming modifications [13, ch.4]. At Langley Field aeronautical laboratory, Virginia, from 1919 to 1923, this problem began to receive serious research attention. A lengthy series of flight tests successfully moved the focus of investigation from qualitative judgements by pilots to quantitative measurements of the control forces required to perform various manoeuvres. Key to this progress was the development at Langley of experimental procedures based on new measuring instruments and data recorders, such as the three-axis accelerometer [14] and synchronizing chronometer [15]. These devices relieved pilots from having to pause during and between manoeuvres to record measurements manually, and thus allowed a far greater quantity of more accurate product data to be collected [16, ch.3].

By the mid 1930s, the growth of commercial air travel and the problem of pilot fatigue over longer journeys led to

1At the instant shown in figure 1, the striped vertical pole and the markers attached to the street lamps behind it are clearly not aligned, due to the extreme twisting motion of the bridge. Clearly, the displacement gauge was not designed to measure torsional oscillations; the bridge’s final mode of collapse apparently came as a surprise even to Professor Farquharson.
renewed interest in flying qualities. Under the leadership of Robert Rowe Gilmith, comprehensive flight tests of at least 18 aircraft and ground experiments to discover the forces exerted by pilots on the controls led to the publication in 1941 of the first full flying qualities specifications. These specifications included a beautifully simple measure of maneuverability - the stick force per g - that is equally applicable to all types of aircraft and is still in use [16, ch.3]. The experiments also yielded the large body of data necessary to build theories of maneuverability [13, p.32], that today allow flying qualities to be accurately predicted from a given aircraft design.

V. Conclusions

Any holistic view of software product quality should include attributes that are relevant to continuously-evolving systems; however many such attributes are currently only poorly understood. Our first historical example, taken from the pioneering era of long-span suspension bridge design, illustrates the desirability of a complete theoretical understanding of product quality in any engineering endeavour. This theoretical understanding might take many decades to arise, unless a concerted effort is made to identify the phenomena of interest and study them by observation and measurement of product instances.

Our second example, taken from the early years of long-range aircraft design, shows how instrument development can enable large-scale product observations, which in turn may yield concise and useful theoretical knowledge. Vincenti identifies in this example seven phases of product quality research: familiarization with problem; identification of variables, concepts, and criteria; development of instruments and techniques for measurement; growth of opinion regarding desirable qualities; scheme for (empirical) research; measurement of qualities for a cross-section of products; and assessment of results to arrive at general conclusions [16, p.102]. While these phases do not represent a strict ordering, in our example the important theoretical results only began to emerge during the empirical phase. Noting the apparent shortage of widely-accepted theories in software engineering [3], we suggest that a lack of interest in measurements and instruments may be holding back empirical research in our discipline. We hope that the removal of this barrier will eventually lead to significant theoretical progress.

Acknowledgments

We thank Klaas-Jan Stol, and the participants of the SEMAT General Theory of Software Engineering 2012 workshop, for many useful comments and reading suggestions. This work was supported, in part, by Science Foundation Ireland grant 10/CE1/1855.

References
