Tool Support for Traceable Product Evolution

Bagert, D.J.; Barbacci, M.; Budgen, D.; Lethbridge, T.C.; Suryn, W.; van Vliet, H.

published in
Proceedings Tenth International Workshop Software Technology and Engineering Practice (STEP 2002)
2003

document version
Publisher's PDF, also known as Version of record

Thoughts on Software Engineering Knowledge, and how to Organize it

Donald J. Bagert
Rose-Hulman Inst. of Techn.
Terre Haute, Indiana
USA

Mario Barbacci
Software Engineering Institute
Pittsburgh
USA

David Budgen
Keele University
Staffordshire
UK

Timothy C. Lethbridge
University of Ottawa
Ottawa
Canada

Witold Suryn
École de Techn. Supérieure
Montreal
Canada

Hans van Vliet
Vrije Universiteit
Amsterdam
The Netherlands

Abstract

SWEBOK describes what knowledge a software engineer who has a Bachelor's degree and four years of experience should have. SEEK describes the knowledge to be taught in an undergraduate program in software engineering. Although different in scope and purpose, there are many similarities between the two, and after all, even experienced developers need an education, don't they? A full-day workshop on the alignment between SWEBOK and SEEK, held at STEP 2002, revealed a number of issues that received either a scant or a scattered treatment in either or both documents. These issues include: software architecture, software measurement, and software quality. In addition, topics of debate were whether or not user interface design should be considered part of software design, or rather deserves its own, separate treatment; and whether maintenance/evolution merits a separate discussion, or should rather be seen as the default mode of operation in software development. This paper elaborates the discussions of this workshop.

1 Introduction

Software engineering is a young discipline. According to [Wang and Patel, 2000], it is in a transition from the art age to the engineering age. The main characteristics of a discipline in the engineering age are:

- adoption of work processes
- established processes
- reinforced standards
- stable professional practices
- defined best practices
- well developed theories and foundations
- proven methods and technologies

The current efforts of SWEBOK and SEEK perfectly fit this transitional stage of software engineering. The Guide to the Software Engineering Body of Knowledge (SWEBOK) is a result of the Software Engineering Coordinating Committee, a Joint IEEE Computer Society - ACM committee[1]. It reflects a widely agreed-upon view on what a software engineer who has a Bachelor's degree and four years of experience should know. The Software Engineering Education Knowledge (SEEK) can be seen as the education counterpart of SWEBOK, aimed at providing guidance for undergraduate curricula.

A full-day workshop on the alignment between SWEBOK (the trial version published

[1]ACM withdrew after being involved in the original development of SWEBOK

Proceedings of the 10th International Workshop on Software Technology and Engineering Practice (STEP'02)
0-7695-1878-8/02 $17.00 © 2002 IEEE

in [Abran *et al.*, 2001]) and SEEK (the first draft, [Sobel, 2002]) was held at STEP 2002. As a result of this workshop, a number of smaller issues were discussed that merit the attention of the bodies responsible for SWEBOK and SEEK. In addition to these, a number of issues were identified that received either a scant or a scattered treatment in either or both of SWEBOK and SEEK:

- Software Architecture. Software architecture has become one of the central topics in software engineering. In early publications, such as [Shaw, 1988], software architecture was by and large synonymous with global design. This is also the view reflected in SWEBOK and SEEK, with their emphasis on (design and architectural) patterns. In a broader view, architecture involves making tradeoffs between quality concerns of different stakeholders. As such, it becomes a balancing act reconciling the collective set of functional and quality requirements of all stakeholders with a (global) design that meets those requirements. This broader view is quickly becoming the received view.

- Software Measurement. One aspect of a maturing engineering field is that we can start measuring its products and processes. Both SWEBOK and SEEK pay attention to measurement at a number of places. However, proper measurement within software engineering merits a discussion of its own, i.e. a separate knowledge area: from basic notions of what a valid measure is to how to set up a proper measurement program.

- Software Quality. Quality more and more often becomes a critical attribute of a software product since its absence results in dissatisfied users, loss of money, and may even cost lives. An increasing business-related recognition of the importance of software quality makes the software engineering "center of gravity" shift from *creating an engineering solution* toward *satisfying the stakeholder*. Development organizations confronted with such an approach are, in general, not prepared to deal with it since their engineers too often are not adequately educated. This reality makes the role of the SEEK initiative critical.

- User Interface Design. SWEBOK lists human-computer interaction (HCI) as a "related discipline" of software engineering. It furthermore states that user interface design deals with specifying the external view of the system. SEEK integrates HCI topics as part of software engineering within various knowledge areas. These two views are at opposite extremes and both have ardent supporters. Keeping HCI as a separate discipline recognizes that HCI has grown up as a distinct field and has its own experts that software engineers can call on; however, this does not sufficiently reflect the fact that for many software systems, human use is a decisive factor for product quality, and that, for many an end-user, the user interface *is* the system. From this perspective, fully incorporating HCI into SEEK seems justified. Unfortunately it is necessary to pick a subset of HCI to consider 'essential' in an SE curriculum, and SEEK can be criticised for not picking the 'right' subset, in any given person's eyes.

- Maintenance or Evolution. Both SWEBOK and SEEK discuss development and maintenance as separate topics. Like in all text books on software engineering, maintenance gets scant attention. That is, 'evolution' is only allocated 10 hours out of more than 500 in SEEK, and the discussion of maintenance in SWEBOK is limited to a separate section on the subject. But greenfield software development is an exception, as we all know. Almost all software development is really evolution. Rather than seeing maintenance as an unfortunate phase following development, it is interesting to consider development as a special, quite rare instance of maintenance.

Each of these issues is further elaborated upon below.

The reader should note that although SWEBOK is in a trial phase where it is being deliberately kept stable for a few years, SEEK is, during Spring 2003, undergoing rapid evolution. The details of SEEK mentioned in this paper should therefore be considered only in their historical context, and may have

Proceedings of the 10th International Workshop on Software Technology and Engineering Practice (STEP'02)
0-7695-1878-8/02 $17.00 © 2002 IEEE

changed. In fact, some of the criticisms discussed in this paper might cause changes. To minimize confusion to the reader, we have given references to SEEK using the second draft, dated December 6, 2002.

1.1 The Mapmaker Metaphor

Mapmakers distort reality when projecting the spherical earth on a flat piece of paper. They usually choose a point of projection above their own country, which is then depicted a bit larger than it really is. Conversely, countries further removed from their home country get depicted a bit smaller than they really are.

If software engineering is the earth, and software engineering topics are the countries, then we, the authors of this paper, are mapmakers. We tend to see our own specialty somewhat bigger and more important than many of the other specialties within software engineering. And of course we are right. All of us are right.

2 Software Architecture

Software Architecture is taking a central role in software development and this is reflected in the strong links between SWEBOK and SEEK architectural design topics listed under the Software Design knowledge areas.

For example the SEEK architectural design topics include:

> DES.ar.1 Architectural styles (e.g. pipe-and-filter, layered, transaction-centered, peer-to-peer, publish-subscribe, event-based, client-server, etc.)
>
> DES.ar.2 Architectural tradeoffs between various attributes
>
> DES.ar.3 Hardware issues in software architecture
>
> DES.ar.4 Requirements traceability in architecture
>
> DES.ar.5 Domain-specific architectures and product-lines

Note that architecture is also found in the following areas:

> REQ.fd.6 Interaction of requirements and architecture
>
> REQ.ma.4 Structure modeling (e.g. architectural, ...)
>
> DES.con.8 Architectural styles, patterns, reuse
>
> DES.nst.1 Architectural structure viewpoints and representations
>
> DES.nst.5 Design support tools (e.g. architectural, ...)
>
> EVO.pro.2 Relationship between evolving entities (e.g.architecture, ...)

The SWEBOK software structure and architecture topics include:

> Architectural Structures and Viewpoints
>
> Architectural Styles (macro-architectural patterns)
>
> Design Patterns (micro-architectural patterns)
>
> Families of Programs and Frameworks

These links focus on architectural representations and design notations and methods. In reality, software architecture decisions involve functional and quality attribute requirements. Although this is becoming accepted by the software engineering community, the SWEBOK and SEEK lag behind.

The SWEBOK knowledge area on Software Design describes the software design process in two steps, "architectural design describes how the system is decomposed and organized into components (the software architecture) whereas detailed design describes the specific behavior of these components." The SWEBOK recognizes that one of the key issues when designing software systems are "really quality concerns that must be addressed by all systems, for example performance." The SWEBOK has a separate knowledge area dedicated to Software Quality, definitions, measures, and analysis and evaluation tools[2].

[2]The Software Engineering Tools and Methods KA includes Software Quality Tools, consisting of Inspection and Static analysis. This is a narrow use of the term 'software quality tools'. Many of the other areas in the Tools and Methods KA mention tools that are (also) used to improve or evaluate quality attributes.

Proceedings of the 10th International Workshop on Software Technology and Engineering Practice (STEP'02)
0-7695-1878-8/02 $17.00 © 2002 IEEE

The SWEBOK knowledge area on Software Quality focuses on planning and executing Software Quality Assurance (SQA) and Verification and Validation (V&V) processes. Although it recognizes the need for evaluating the attributes and the sensitivity of their values when the product changes, it does not provide more guidance than "There is no definitive rule for how the decisions are made, but the software engineer should be able to present quality alternatives and their costs."

In large software systems, the achievement of quality attributes is dependent not only upon code-level practices (e.g., language choice, algorithms, data structures), but also upon the software architecture. Quality attributes can interact or conflict – improving one attribute might worsen one or more of the others – it is necessary to tradeoff among multiple software quality attributes. It is better to do this when the software architecture is specified, before the system is developed.

Unfortunately, there are no metrics or methods for evaluation applicable to all attributes. Different communities use different models and parameters for evaluation of attributes where the models are not necessarily mathematical formulas and can be based on expert opinions on how to evaluate a quality attribute. Some design choices might affect multiple quality attributes. For example, selection of a communication protocol between servers on a network might have an impact on performance (e.g., a simple protocol could improve performance) and security (e.g., a simple protocol could allow an intruder to monitor traffic and obtain valuable information). A design decision that affects multiple attributes provides an opportunity to tradeoff between the attributes (e.g., decreasing performance to improve security in the above example). Making tradeoffs might be necessary to satisfy multiple system requirements.

When the software architecture is specified, designers need to determine the extent to which features of the software architecture influence quality attributes, whether multiple quality attribute requirements can be satisfied simultaneously, and whether techniques used for one attribute support or conflict with those of another attribute.

The SWEBOK and the SEEK need to emphasize the importance of quality attributes and their impact on software architectures. The field is growing and there are recent books that might be used as starting points [Bass *et al.*, 1998, Clements *et al.*, 2002].

3 Software Measurement

As observed earlier, if Software Engineering is to be regarded as a field that is approaching maturity, then an important demonstration of this maturity is through the sound use of measurement to provide the *evidence* that is necessary to give confidence in both its processes and its products. So this section examines the topic of software measurement in terms of the following three aspects:

- the wider context of measurement in Software Engineering;
- its role in the SEEK and the SWEBOK;
- how either the SEEK or the SWEBOK might address it in future versions.

3.1 Measurement in the wider context

Measurement itself is an activity that generally comes down to a process of *counting*, performed according to a particular set of *counting rules*. For engineering in general, the context is the physical world, and hence the reasons for making measurements are largely those laid down by Lord Kelvin (1824–1907), as follows:

> *"When you can measure what you are speaking about and express it in numbers, you know something about it, but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind."*

Indeed, perhaps because ideas about measurement, and about the formalisation of the associated measurement processes tended to be pioneered by physicists, early work in Software Engineering, such as that of Halstead [Halstead, 1977] also tended to be based upon the 'classical' model as summarised by Lord Kelvin. However, ideas about the roles and forms of measurement that might be most appropriate for use in Software Engineering have subsequently evolved further, and a more interdisciplinary approach is now much more

Proceedings of the 10th International Workshop on Software Technology and Engineering Practice (STEP'02)
0-7695-1878-8/02 $17.00 © 2002 IEEE

widespread, while still seeking sound definitions of the rules for measurement, as exemplified in [Fenton and Pfleeger, 1997]. An example of this wider view is the widespread use of ordinal scales, based upon the recognition that direct measurement of relevant properties is often impractical. This is in marked contrast to physics, which almost always makes use of ratio scales for its measurements.

It is difficult (and probably unrealistic) to separate the idea of software measurement from the ways that is it used and from the reasons for making use of it. In particular, as already observed, the motivation for much of our use of measurement is that of producing *evidence*. This evidence may be needed to justify resource allocation, confirm hypotheses, or for any other like purpose. However, any interpretation of that evidence does in turn require an understanding of both the measurement processes that may be been employed, and also of the way that empirical practices can be employed in the software domain, even if use of the latter is not yet 'standard' [Tichy *et al.*, 1995]. Indeed, most empirical practices have likewise gravitated into Software Engineering from a variety of other domains, and so carry with them various assumptions about measurement [Zelkowitz and Wallace, 1998, Kitchenham *et al.*, 2002].

So, while there is a growing recognition of the need for the development of evidence-based Software Engineering practices, the framework necessary for widespread use, including measurement theory and measurement practices, are still relatively immature, and still too rarely taught to our students. In examining how the SWEBOK and the SEEK address these issues, we therefore need to recognise that Software Engineering is still gravitating (slowly) from being a discipline that is based largely upon the practices of assertion and advocacy, to one that is more soundly based upon experimental evidence. That said, this process is one that the SEEK and the SWEBOK are particularly well placed to accelerate!

3.2 Measurement in the SEEK and the SWEBOK

Both the SEEK and the SWEBOK do give significant attention to software measurement. However, both are much more centred upon its *use* than upon the actual measurement forms themselves.

If we begin with the SEEK, then this does have one topic that considers measurement in its proper context of underpinning practices (FND.ef.1 — "Empirical methods and experimental techniques", in the unit "Engineering foundations for software"). In addition, there are references to the use of measurement practices in many other chapters (DES, VAV, PRO, QUA and MGT).

However, given that "Engineering Foundations for Software" as a whole is allocated 25 hours (from a total of 250 hours for "Fundamentals"), and that the topic is only rated as a 'Bloom category' of *comprehension*, we may conclude that the SEEK does not give measurement itself a particularly high priority. In view of its use in later chapters, this may be considered as inadequate.

Going on to the SWEBOK, this takes a slightly curious approach to this topic. Here the topic of measurement is largely addressed in the chapter on "Software Engineering Management". While the coverage there is good, and we would agree that both management and measurement do indeed have a "close relationship", it does seem a rather compartmentalised view of its role. Once again too, since as a topic there are many references to its use in other chapters, including it in this particular chapter would seem to be a matter of organisational convenience rather than one based upon a recognition of the more general role of measurement.

3.3 Addressing measurement in future versions of the SEEK and SWEBOK

As explained above, measurement is not easily separated from its use, and hence it is quite appropriate that issues relating to measurement should appear widely throughout both the SEEK and the SWEBOK.

In the case of the SEEK, as the ideas of measurement are already implicitly included in several of the sections within the "Fundamentals" chapter, as well as included in many others, there are arguments in favour of consolidating this under a separate sub-heading and giving it a 'Bloom level' of *application*. Without this latter modification, it is unlikely that students will gain the understanding needed to appreciate its roles elsewhere in the curriculum

Proceedings of the 10th International Workshop on Software Technology and Engineering Practice (STEP'02)
0-7695-1878-8/02 $17.00 © 2002 IEEE

(and also its limitations), as well as being less likely to be able to make effective use of measurement.

For the SWEBOK the arguments are more finely balanced. On the one hand, a good case could now be made for developing a chapter on "Evidence-Based Practices", that would encompass measurement, empirical practices, and their employment in a Software Engineering context. On the other hand, as indicated at the beginning of this section, this is also an area where the practices and techniques are only now beginning to be codified within a Software Engineering framework. However, to offset this, its recognition and inclusion in the SWEBOK would also be likely to encourage and assist this process. Overall therefore, we would suggest that serious consideration should be given to including such a chapter in subsequent revisions of the SWEBOK.

4 Software Quality (Engineering)

Quality more and more often tends to become a critical attribute of a software product as its absence results in dissatisfied users, loss of money, and may even cost lives. At the same time the definition, or scope, of the domain of software quality has gradually evolved from a somewhat technical perspective to a perspective that also embraces human aspects such as usability and satisfaction [Suryn *et al.*, 2002, Abran *et al.*, 2003].

An increasing business-related recognition of the importance of software quality has also made the software engineering "center of gravity" shift from *creating an engineering solution* toward *satisfying the stakeholders* [Bevan and Bogomolni, 2000]. Such a shift very clearly reflects the trend within the community of stakeholders who more and more often say: "I do not want to know about bits and bytes. I want a solution that satisfies my needs". The critical word here is "*satisfaction*" for it covers both the functional and the quality perception of the software solution being used.

Development organizations confronted with such an approach are, in general, not prepared to deal with it, even if their engineers are adequately educated [Fox and Rakes, 1997]. Moreover, if the education is there, it is too often acquired through experience instead of a regular educational process. A response to the question "*why?*" is rather easy to give: with few exceptions[3], the software engineering curricula being offered do not emphasize the importance of teaching software quality engineering.

In the light of the above considerations the role of SEEK turns from *very important* to *critical*. With its mission, very well-defined outcomes, and its ambition to become an international referential source for modern software engineering curricula, this initiative has a unique opportunity to change the landscape of the software engineering domain. Despite being relatively young, SEEK presents a surprisingly mature vision of what the real software engineering professional should know and how he should be prepared to follow the evolution of the domain. The Knowledge Areas are well analyzed, sometimes down to tiny details, in order to give the reader (user) first-hand guidance on what knowledge is *core*, what is *desirable* and what is *optional*. And Software Quality is there, but it is not Software Quality Engineering. Not yet.

Treating Software Quality as a separate Knowledge Area (KA) considerably differentiates SEEK from some contemporary curricula, where quality may be "glued" to another knowledge area not always well related, or is even entirely forgotten. This distinction makes SEEK take the first important step toward engineering of quality, however this step is not long enough.

When compared to the Software Engineering Fundamentals KA with its 250 core contact units (simply speaking, hours of lecturing) or even to the Software Design KA with its 78 units, the Software Quality KA with its 21 units does appear to be a "Cinderella" rather than a full member of the software engineering family. Even though software quality is addressed in many other KA's of the SEEK, a plea can certainly be made for a more prominent presence of this topic.

When analyzing the content of the Software Quality KA proposed by SEEK, one can easily notice how modern its profile is. Introducing *Software Quality Concepts and Culture* is def-

[3] École de Technologie Supérieure in Montréal, Canada, is in the process of enhancing its undergraduate software engineering program to cover software quality in a more exhaustive way.

Proceedings of the 10th International Workshop on Software Technology and Engineering Practice (STEP'02)
0-7695-1878-8/02 $17.00 © 2002 IEEE

initely innovative but requires further refinement, especially in the area of definitions and quality models and techniques.

The presence of *Software Quality Standards* indicates that standards have finally become part of modern software engineering education, however the documents proposed as seminal do not always represent the state-of-the-art. From a software engineering standpoint ISO 9000 should be replaced by ISO 9001 [ISO 9001, 2000], ISO/IEC 12207 [ISO 12207, 1995] (Software Life Cycle Process) should be discussed together with ISO/IEC 15288 [ISO 15288, 2002] (Life Cycle Management – System Life Cycle Processes), ISO/IEC 15504 [ISO 15504, 1998] (Software Process Assessment) should be enriched with two suites of other standards: ISO/IEC 9126 [ISO 9126, 2000 2003] (Software Product Quality) and ISO/IEC 14598 [ISO 14598, 1998 2000] (Software Product Evaluation) and the very basic standard for software measurement – ISO/IEC 15939 [ISO 15939, 2002] should find its place in the discussion.

Finally, there are subjects that did not yet find their place in SEEK's Software Quality KA, subjects that will have to be taken into consideration for SEEK to become the worldwide-accepted reference for state-of-the-art software engineering education. These subjects, when present, may also result in a change of the name of this KA to Software Quality Engineering KA, for they would discuss engineering areas of software quality:

- *Software Quality Management and Evaluation* would allow the future engineers to apply proper management, measurement and evaluation processes in software development. See also the discussion on measurement in section 3.

- *Software Quality Implementation* would teach the students that quality might easily become the art of tradeoffs (such tradeoffs also drive the architectural stage of development), that the implementation is the project per se, demanding its specific techniques and methodologies and, last but not least, that software quality implementation will shortly become the development process's watchdog.

To summarize, on behalf of the average software user tired by *blue screens* and *your-computer-will-have-to-restart-nows*, the authors would like to ask the following, full of hope question: "*Software quality is the hot subject of the past and present century, but still more in words than in facts. Why don't we use SEEK to change this proportion?*"

The practical response to this question would strongly recommend that SWEBOK and SEEK remain in a constant synchronization loop allowing "mutual upgrading". SWEBOK, as the domain's body of knowledge is being continuously used, verified and monitored by the industry, and this process reflects desired changes in the evolution of software engineering, including software quality. SEEK, on the one hand, seeks to provide the knowledge that will allow universities to produce professionals able to stay abreast with the fast moving industry, but on the other hand it also adds a scientific and innovative flavor to so called "best practices". These elements have to find their way to SWEBOK in the same manner as SWEBOK's evolution should be reflected in SEEK's education programs. The "mutual upgrading" would then take place for the benefit of both, so important for software engineering initiatives.

In the light of such a statement the STEP2002 workshop on SEEK-SWEBOK makes an excellent beginning, but only a beginning for something that should become an institutional cooperation platform grouping more frequently than once a year industry experts, teachers and scientists.

5 Human Computer Interaction

The area widely known as 'Human-Computer Interaction' (HCI) historically grew up as a distinct discipline on the boundary of computer science, cognitive science and several other areas. Over the years, however, there has been a gradual recognition among an enlightened subgroup of software engineers that many aspects of HCI should be practised and taught in an engineering way, and that software engineers should know as much about user interface development as they know about the development of algorithms, architectures and other aspects of an engineered software product.

In order to incorporate HCI into soft-

Proceedings of the 10th International Workshop on Software Technology and Engineering Practice (STEP'02)
0-7695-1878-8/02 $17.00 © 2002 IEEE

ware engineering, the terms 'user interface design/development', 'usability engineering' and 'user-centered design' have also appeared. In this paper we will, however, not play the nomenclature game and stick with the term HCI for simplicity.

HCI is reflected both in SWEBOK and SEEK, but in quite different ways. In SWEBOK, it is considered a 'related discipline', whereas in SEEK it is embedded in a variety of units and knowledge areas that we will detail below. A third approach is possible: it could have been treated as an entire knowledge area ... neither BOK was prepared to make such a bold move though.

The following are the HCI units and topics found within SEEK:

> REQ.ma.5 Analyzing quality (non-functional) requirements (e.g. ... usability ...)
>
> DES.con.6 Design goals (e.g. ... usability ...)
>
> DES.hci Human computer interface design
>
> VAV.tst.9 Testing across quality attributes (e.g. ... usability ...))
>
> VAV.hct User Human computer interface testing and evaluation

Two of the above are not just topics, but entire units, breaking down as follows, and giving 15 HCI topics in total.

> DES.hci.1 General HCI design principles
>
> DES.hci.2 Use of modes, navigation
>
> DES.hci.3 Coding techniques and visual design
>
> DES.hci.4 Response time and feedback
>
> DES.hci.5 Design modalities (e.g. menu-driven, forms, question-answering,), etc.)
>
> DES.hci.6 Localization and internationalization
>
> VAV.hct.1 The variety of aspects of usefulness and usability
>
> VAV.hct.2 Heuristic evaluation
>
> VAV.hct.3 Cognitive walkthroughs
>
> VAV.hct.4 User testing approaches (observation sessions etc.)
>
> VAV.hct.5 Web usability; testing techniques for web sites
>
> VAV.hct.6 Formal experiments to test hypotheses about specific HCI controls

In the next couple of sections we will discuss some of the criticisms that can be levelled against both the SWEBOK and SEEK approaches.

SWEBOK was criticized quite heavily by some reviewers for leaving HCI as a related area. Some arguments in favour of their decision are: a) Software engineers are typically not knowledgeable about HCI and would not be able to review the knowledge to the same level of competence as other areas, b) software engineers have enough to know without having to learn HCI ... they can leave that to HCI experts, and c) including HCI in SWEBOK or SEEK would be akin to usurping another discipline and calling it our own.

Counter-arguments that oppose the SWEBOK approach (and to some extent favour the SEEK approach) are as follows:

1. Many other areas of software engineering also have very specialized knowledge, and employ experts who know that field better than average software engineers, but know much less about other areas of SE: For example, there are software process experts who do not feel expert in design, and software designers who do not feel expert in process. We respect the convention that all software engineers should know something about both design and something about process, and furthermore that some people can be experts in one or the other. It therefore seems reasonable that all software engineers should have to know something about HCI with some people being the experts.

2. Including HCI knowledge in a software engineering BOK says that:

   - all software engineers should know the basics of this subset of HCI knowledge,
   - it is being catalogued here from a software engineering perspective, and

Proceedings of the 10th International Workshop on Software Technology and Engineering Practice (STEP'02)
0-7695-1878-8/02 $17.00 © 2002 IEEE

- some software engineers could call themselves HCI experts.

However, this does not preclude the parallel existence of a separate HCI field with HCI professionals who *don't* consider themselves software engineers. We see an analogy with psychology and psychiatry: Both disciplines treat the mind, with one doing it from a medical perspective, and having experts that must know the basics of the rest of medicine.

If we conclude from the above that a software engineering perspective on HCI should be included in some SE body of knowledge, three questions arise: 1) Should SEEK and SWEBOK *both* take this approach? 2) How should the knowledge be organized – as a separate area, or distributed throughout the BOK? And 3) at what level of detail should HCI topics be presented?

We believe the answer to question 1 is clearly yes. We have argued above that all software engineers should know HCI from a software engineering perspective; SEEK needs an even broader perspective since it is intended to have a wider, albeit shallower, scope than SWEBOK, answering the question: "what should be taught to an undergraduate SE student", not just "what *SE* knowledge should be taught").

The answer to question 2 is controversial. SEEK has chosen to *distribute* the HCI knowledge as listed earlier, in a similar way that knowledge of measurement is distributed. This makes sense because human issues can be seen to cut across many other areas of software enginering. Alternatively, SEEK could have created an HCI knowledge area and taken the 'mixed' approach for HCI that it takes with most other knowledge areas (e.g. it has a small Process knowledge area to draw attention to the main concepts, but also lists many Process matters elsewhere as topics under KAs such as Design and Requirements). Both the distributed and mixed approaches have drawn fire from critics who argue either: "the area isn't given enough attention because it is too distributed", or "the main KA is too minimal" (and because this is all they see, they don't notice the additional distributed topics.) In our opinion, the answer to all this is to add a very clear section in SEEK that guides the reader to all the places where cross-cutting topics like HCI are found.

Question 3 is the most difficult: No matter whether the decision is to have a separate KA, or to distribute HCI knowledge entirely in other areas, it will be hard to satisfactorily choose an appropriate subset of HCI to call, "essential for all undergraduates" (in the case of SEEK) or "essential for all professionals" (in the case of SWEBOK). Any subset chosen will draw fire from people who feel that some other subset is better. The SEEK DES.hci area has drawn fire from people looking at its choices and saying: "SEEK has not gone far enough towards embracing HCI", merely because SEEK was forced to limit the topics so as to not overburden the "essential" curriculum with more than can possibly be taught. We believe that there is one solution to this: Rather than listing 15 topics that can be criticised as arbitrary, developers of SEEK could instead have created two sets, one being a superset of the other. The superset would contain many more than 15 topics, with a previso that at least some number (say 15) be taught. The subset would be many fewer than 15, and would provide a central core that must always be taught. Implementing this approach would complicate SEEK considerably as it would have to be applied to areas other than HCI as well for consistency. We recommend not trying such an approach at the current time, so that SEEK can stabilize and be reasonably simple; however, maybe it should be considered in a few years when SEEK next comes up for modification.

6 Maintenance or Evolution

There are two body-of-knowledge issues related to software maintenance and software evolution: whether maintenance or evolution should be treated separately, and whether or not maintenance/evolution should be treated as a separate knowledge area. Both SWEBOK and the SEEK draft treat maintenance/evolution as the same area, although interestingly enough, one of them (SWEBOK) labels the area maintenance while SEEK calls it evolution. (Both artifacts (SWEBOK in particular) appear to be saying that one of these topics is a subset of the other.) One solution for this problem would be to call the

Proceedings of the 10th International Workshop on Software Technology and Engineering Practice (STEP'02)
0-7695-1878-8/02 $17.00 © 2002 IEEE

knowledge area "maintenance and evolution", which would provide a broader description of the topic.

The more far-reaching aspect of this discussion is whether maintenance/evolution should be treated as a special case or not. SWEBOK and SEEK have taken the more traditional route of discussing these issues separately as, for example, ISO/IEC 14764 [ISO 14764, 2000]. However, there is a compelling argument that the engineering of software has for many years been more about maintenance and evolution than it has about development. It is also true that there has been a disproportionate amount of knowledge disseminated about maintenance and evolution i.e. there has been insufficient emphasis on these topics in the literature.

The question, then, is how SWEBOK and SEEK should address these concerns; potential remedies for the two artifacts will be considered separately here. In traditional engineering disciplines, maintenance has generally been covered as a separate subject. This makes sense, since in these disciplines defects in delivered products are usually an abnormality rather than the norm. Although this has not been true for software products, much of this is because of the fact that software engineering is not yet a mature enough discipline for this to be the case. Since one of the stated goals of software engineering is to produce a highly reliable product, perhaps it is best that maintenance remain classified as a separate area in SWEBOK.

Evolution is a different issue, however. Due to its non-physical nature, the continuous evolution of existing software products is something that will pervade all areas of software engineering for the foreseeable future. So, it would make sense to have the discussion of most (if not all) knowledge areas from both development and evolution points-of-view. Certainly, evolution issues are currently not covered in much detail in the SWEBOK chapter on maintenance, so this would provide a method for remedying that situation in the most effective manner possible. (However, question of where to put an introduction of items such as Lehman's Laws of Software Evolution – a more general evolution topic – would still need to be addressed; in the case of Lehman's Laws, the software process area might be the best fit.)

The issues involving SEEK are somewhat different. The SEEK Evolution knowledge area is disproportionally small – only 10 contact hours, or about one-quarter of a standard course devoted to maintenance/evolution issues would be required of an undergraduate software engineering curriculum. Furthermore, all ten of these hours are recommended at only the knowledge level of Bloom's taxonomy – in other words, SEEK is recommending that a small amount of general knowledge about maintenance and evolution is sufficient for undergraduate studies before entering the workforce, with any capability and application of those areas being at best optional. This is a matter of great concern which needs immediate addressing. (SWEBOK has an appendix on Bloom's taxonomy, but unfortunately does not include the Maintenance knowledge area in that section.)

However, a solution like the one proposed for SWEBOK (evolution covered throughout the knowledge areas, with maintenance as a separate topic) would not be easily implementable in actual curricula. First of all, students are initially introduced to software from a development point-of-view in a computer science introductory programming sequence, and this is not likely to change in the near future. Next, in the software engineering courses and projects, it would be difficult to find sufficient artifacts to do maintenance and evolution throughout the curriculum. However, some schools have successfully used ongoing maintenance/evolution projects in the capstone sequence, although such projects by their nature are usually in-house rather long-term projects for industrial clients. With this in mind, it might be best to discuss both maintenance and evolution separately within a software engineering curriculum, although the number of contact hours should be closer to 40 than the 10 that are currently proposed, and some of those hours should be definitely be devoted to mastery at the capability and application levels of Bloom's taxonomy.

7 Conclusion

In an environment which changes, processes have to be reflexive to stay successful. Topics such as software process improvement and the capability maturity model have a natural

Proceedings of the 10th International Workshop on Software Technology and Engineering Practice (STEP'02)
0-7695-1878-8/02 $17.00 © 2002 IEEE

place within software engineering because of its changing nature. In a similar vein, SWEBOK and SEEK need reflexion. The full-day workshop on this topic held at STEP 2002 provided an excellent forum to discuss the alignment between the two initiatives. In particular, this workshop identified a number of issues that warrant further thought and discussion:

- Software architecture has become one of the central topics in software engineering, linking quality and design.
- As software engineering approaches maturity, it should demonstrate so, for example through the sound use of measurements.
- Software quality issues gradually shift from creating and engineering solution to satisfying the stakeholders, necessitating an engineering approach to quality.
- User Interface Design can be seen as a related discipline, or as part of software engineering, and both perspectives have their advantages and disadvantages.
- Should maintenance/evolution be treated as a separate topic, given the fact that, in daily practice, greenfield development is the exception and maintenance is the default mode of development?

This paper summarizes our thoughts on these issues. We suggest that a regular reflexion on the status of both SWEBOK and SEEK, for example in the form of a workshop as held at STEP 2002, be institutionalized.

References

[Abran *et al.*, 2001] A. Abran and J.W. Moore, Ex. editors, P. Bourque, and R. Dupuis, editors. *SWEBOK: Guide to the Software Engineering Body of Knowledge: Trial Version 1.00*. IEEE, 2001.

[Abran *et al.*, 2003] A. Abran, A. Khelifi, W. Suryn, J. Rilling, and A. Seffah. Consolidating the ISO Usability Models. In *submitted to the 11th International Software Quality Management Conference and the 8th Annual INSPIRE Conference*, 2003.

[Bass *et al.*, 1998] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice*. Addison-Wesley, 1998.

[Bevan and Bogomolni, 2000] N. Bevan and I. Bogomolni. Incorporating user quality requirements in the software development process. In *Proceedings 4th International Software Quality Week Europe, Brussels*, pages 1192–1204, 2000.

[Clements *et al.*, 2002] P. Clements, R. Kazman, and M. Klein. *Evaluating Software Architectures: Methods and Case Studies*. Addison-Wesley, 2002.

[Fenton and Pfleeger, 1997] N.E. Fenton and S.L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. PWS Publishing, Boston, second edition edition, 1997.

[Fox and Rakes, 1997] C. Fox and W. Rakes. The Quality Approach: Is It Delivering? *Communications of the ACM*, 40(6):25–29, 1997.

[Halstead, 1977] M.H. Halstead. *Elements of Software Science*. North-Holland Publishing Company, 1977.

[ISO 12207, 1995] *ISO/IEC 12207 – Software Life Cycle Process*. ISO, 1995.

[ISO 14598, 1998 2000] *ISO/IEC 14598 – Software Product Evaluation, Parts 1-6*. ISO, 1998-2000.

[ISO 14764, 2000] *ISO/IEC 14674: Software Engineering – Software Maintenance*. ISO, 2000.

[ISO 15288, 2002] *ISO/IEC 15288 – Life Cycle Management – System Life Cycle Processes*. ISO, 2002.

[ISO 15504, 1998] *ISO/IEC 15504 – Software Process Assessment, Parts 1-6*. ISO, 1998.

[ISO 15939, 2002] *ISO/IEC 15939 – Software Measurement Process Framework*. ISO, 2002.

[ISO 9001, 2000] *ISO 9001: Quality management systems – Requirements*. ISO, 2000.

[ISO 9126, 2000 2003] *ISO/IEC 9126 – Software Product Quality, Parts 1-4*. ISO, 2000-2003.

[Kitchenham *et al.*, 2002] B.A. Kitchenham, S.L. Pfleeger, L.M. Pickard, P.W. Jones,

Proceedings of the 10th International Workshop on Software Technology and Engineering Practice (STEP'02)
0-7695-1878-8/02 $17.00 © 2002 IEEE

D.C. Hoaglin, K. El Emam, and J. Rosenberg. Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering*, 28(8):721–734, 2002.

[Shaw, 1988] M. Shaw. Toward Higher Level Abstractions for Software Systems. In *Proceedings Tercer Simposio Internacional del Conocimiento y su Ingerieria*, 1988.

[Sobel, 2002] A.E.K. Sobel, editor. *Computing Curricula Software Engineering Volume, First Draft.* CCSE Steering Committee, `http://sites.computer.org/ccse/` (December 6, 2002), 2002.

[Suryn *et al.*, 2002] W. Suryn, A. Abran, P. Bourque, and C. Laporte. Software Product Quality Practices, Quality Measurement and Evaluation using TL9000 and ISO/IEC 9126. Technical report, STEP 2002 position paper, 2002.

[Tichy *et al.*, 1995] W.F. Tichy, P. Lukowitz, L. Prechelt, and E.A. Heinz. Experimental Evaluation in Computer Science: A Quantitative Study. *Journal of Systems & Software*, 28(1):9–18, 1995.

[Wang and Patel, 2000] Y. Wang and D. Patel. Comparative software engineering: Review and perspectives. *Annals of Software Engineering*, 10:1–10, 2000.

[Zelkowitz and Wallace, 1998] M.V. Zelkowitz and D.R. Wallace. Experimental Models for Validating Technology. *IEEE Computer*, 31(5):23–31, 1998.

Proceedings of the 10th International Workshop on Software Technology and Engineering Practice (STEP'02)
0-7695-1878-8/02 $17.00 © 2002 IEEE