Anecdotes

James Tomayko, Editor

There are several new initiatives to develop the history of software. Usually, the engineers do the work and the historians provide the context later. I asked Guy Tremblay to describe how a group of software engineers used a historical model to establish a context while the actual work was being done.

Software design knowledge and Vincenti's categories of engineering knowledge

Since January 1998, a concerted effort to develop a "Guide to the Software Engineering Body of Knowledge" (also known as the Guide to the SWEBOK) has been ongoing. The goal of this project, managed by a team from the University of Quebec at Montreal, is to develop a guide to the core knowledge in the software engineering field. Corporate support for this project is provided by the ACM, Boeing, the IEEE Computer Society, the National Institute of Standards and Technology, the National Research Council of Canada, Rational, Raytheon, and SAP Labs (Canada).

To attain the intended goal, 10 major knowledge areas (KAs) were first identified by a group of software engineers, researchers as well as practitioners (Guide to the SWEBOK, Strawman version). In a subsequent phase (Stoneman version), another group of specialists then developed detailed breakdowns and descriptions of these KAs. These descriptions have been extensively reviewed; for example, more than 40 individuals reviewed an early version of the software design KA description. The KA specialists then had to go through the various review recommendations and make any modifications they deemed appropriate; all decisions (acceptance, rejection, and so on) had to be formally documented in a public database (see http:/www.swebok.org). Each knowledge area description contains a brief definition of the KA, a breakdown and description of the KA's key topics, a rationale for the breakdown, a list of recommended references, and classifications of the KA topics based on two categories—Bloom's taxonomy of cognitive knowledge and Vincenti's categories of engineering knowledge.[1]

This article focuses on Vincenti's categorization. As will be explained, the categorization of engineering knowledge that Vincenti initially proposed has been obtained mostly from an analysis of historical data, thus its possible interest to *Annals* readers.

As a participant in the Guide to the SWEBOK project, I was asked by the editorial team on two occasions—first as the author of a jump-start document for the software construction KA, then as the KA specialist for software design—to classify the proposed KA topics on the basis of Vincenti's categories of engineering knowledge. The motivation for attempting this categorization was to better understand how knowledge topics from the software engineering field could be mapped with those from more traditional, established engineering fields.

Vincenti's categories of engineering knowledge

W.G. Vincenti, in his book *What Engineers Know and How They Know It—Analytical Studies from Aeronautical History* (Johns Hopkins University Press, 1990), examines how knowledge in the field of aeronautical engineering evolved. His book presents a number of historical case studies pertaining to various aircraft design problems for the period from 1908 to 1953. Problems include the design of wings, the development of control-volume analysis theory, and a few more, including why the design of flush-riveted joints turned out to be crucial for modern aircrafts.

A key chapter, "The Anatomy of Engineering Design Knowledge," attempts to draw conclusions about engineering knowledge in general by providing a tentative categorization of such knowledge based on the case studies and the author's own engineering experience. Although Vincenti admits that such a generalization may be risky since it draws mostly from the field of aeronautics, nonetheless, he believes that his classification might be universally applicable. The six categories of engineering knowledge he proposes are as follows:

- *Fundamental design concepts (FC):* These concepts consist of the operational principles of the devices[2] to be designed in addition to their normal configurations. Together, these two aspects define the normal technology and design approach, in contrast with radical design in which new concepts and techniques might be needed. Interestingly, Vincenti says that these concepts "may exist only implicitly in the back of the designer's mind [and that they] are absorbed … in the course of growing up, perhaps even before entering formal engineering training."

1058-6180/01/$10.00 © 2001 IEEE

Table 1. A classification of the software design KA topics based on Vincenti's categories.

Software Design Topic	FC	CaS	TT	QD	PC	DI
I. Software design basic concepts						
General design concepts			X			
The context of software design						X
The software design process						X
Basic software design concepts	X		X			
Key issues in software design			X		X	
II. Software architecture						
Architectural structures and viewpoints			X			X
Architectural styles and patterns (macroarchitecture)			X			X
Design patterns (microarchitecture)			X			X
Design of families of programs and frameworks			X			X
III. Software design quality analysis and evaluation						
Quality attributes		X			X	
Quality analysis and evaluation tools			X			
Metrics			X	X		
IV. Software design notations						
Structural descriptions (static view)						X
Behavioral descriptions (dynamic view)						X
V. Software design strategies and methods						
General strategies						X
Function-oriented design						X
Object-oriented design						X
Data-structure centered design						X
Other methods						X

FC: Fundamental design concepts. CaS: Criteria and specifications. TT: Theoretical tools. QD: Quantitative data. PC: Practical considerations. DI: Design instrumentalities.

- *Criteria and specifications (CaS):* This type of knowledge allows an engineer to "translate the general, qualitative goals for the device into specific, quantitative goals couched in concrete technical terms." The goal is to assign specific numerical values or limits to some technical criteria. However, according to Vincenti, the key knowledge here is the selection of the appropriate set of criteria.
- *Theoretical tools (TT):* Such tools range from mathematical models and theories useful for quantitative analysis and design, as well as intellectual concepts useful for qualitative conceptualizing and reasoning ("concepts for thinking about design"). Such concepts and tools include general scientific knowledge as well as what Vincenti calls "phenomenological theories"—ad hoc techniques, approximations, or theories useful solely for engineering calculation.
- *Quantitative data (QD):* This is empirically obtained data about the physical properties of the devices, typically represented in tables or graphs. The role of such data, which can be descriptive (how things are) as well as prescriptive (how things should be—for example, safety factors), is to help determine the details of the devices to be designed.
- *Practical considerations (PC):* This is practical, empirically derived knowledge learned mostly from on-the-job experience. Such knowledge is generally not formally codified but often represented by rules of thumb. According to Vincenti, when this type of knowledge becomes formally recorded and codified, it often becomes part of another category of knowledge.
- *Design instrumentalities (DI):* These are procedures (for example, hierarchical decomposition), ways of thinking (including visual thinking), judgmental skills, and knowledge of "how" to carry out tasks.

The software design knowledge area and Vincenti's categories

Table 1 presents a tentative classification, based on Vincenti's categories, of the software design KA description (as of version 0.70; see http://www.swebok.org for the complete and up-to-date description of this KA and a companion rationale). The whole KA has been divided into five top-level topics, and the

whole breakdown is three levels deep, although only the two topmost levels are shown here.[3]

Mapping each software design topic into Vincenti's categories was no obvious task—in fact, I made a number of revisions to the initial categorization while preparing this article. First, the categories are not mutually exclusive: This is both because some of the indicated topics contain a number of subtopics and because the frontiers between Vincenti's categories are sometimes, as he himself admits, relatively fuzzy. The following paragraphs explain, for a number of topics, why these categories were chosen, with additional details provided on the topics' content.

The "General design concepts" topic includes notions and concepts relevant to design in general—for example, goals, constraints, alternatives, representations, and solutions. An initial reaction might be to put this topic into the FC category. However, Vincenti seems to emphasize the idea that this category pertains to the "operational principles" of the devices to be designed, whereas the aforementioned KA topic describes concepts "about design," that is, intellectual concepts for thinking about design. Thus, this explains the choice of the TT category.

The "The context of software design" topic has been classified in the DI category. This KA topic addresses the question of how software design fits in the software development life cycle (requirements and specification versus design versus software construction). Since it describes knowledge about "how" to perform software development and design, it thus seems appropriate to include it in the DI category. This line of reasoning also applies to "The software design process" topic that, among other things, discusses architectural versus detailed design—again, process knowledge.

The "Basic software design concepts" topic is a hodgepodge of various subtopics, such as coupling, cohesion, encapsulation, interface versus implementation, and so on. Many of these are fundamental software concepts. What is less clear is whether these concepts are "operational principles" associated with "normal configuration" (FC), or whether they are "conceptual tools" that can help reasoning quantitatively or qualitatively about software (TT). Both options (FC/TT) seem valid.

This categorization contrasts with the next topic, "Key issues in software design," which addresses issues such as concurrency, distribution, exceptions, partitioning, persistence, and platform independence. Currently, knowledge about these issues seems to fit better in the PC category, although some of these concepts probably belong to the TT category. The notion of dialogue independence, for example, used in the development of interactive systems can be seen as a tool that helps address qualitative concerns about such systems.

The classification of the various topics related with design patterns, either macroarchitectural patterns (sometimes known as architectural styles), microarchitectural patterns (for example, patterns such as factory, singleton, adapter, proxy, and so on) or framework-level patterns (also known as program families) again is a hybrid. Initially, it seemed they should go in the PC category. However, as mentioned earlier, Vincenti views this category as knowledge "often not written down." This, clearly, is not the case here, since the whole pattern movement aims at exactly the opposite: making explicit and documenting the key micro- and macrostructures typically found in software.

So, TT seems a more appropriate category for the above topics, but note that the DI category has also been indicated. If we strictly interpret this category as "knowledge on how to carry out tasks," then DI might not be appropriate. On the other hand, if we more loosely interpret the DI category as "ways of thinking," as Vincenti does, then the pattern methodology—that is, the more general idea of pattern language by opposition with the use or description of any specific pattern—might be seen as falling into this latter category.

The next top-level section, "Software design quality analysis and evaluation," is probably one where software design and software engineering distinguish themselves from other, more mature fields of engineering. For instance, consider the "Metrics" topic. Although various quantitative metrics exist to estimate various aspects of the design size, structure, or quality, no clear set of rules to ensure a particular design's correctness and quality exists. Contrary to the more physical fields of engineering, then, pure quantitative data are scarce in software, be they descriptive or prescriptive. This is why "Metrics" has been categorized as QD as well as TT.

A slightly similar situation holds for "Quality attributes." Although such quality attributes could be considered part of the CaS category, many of these attributes are still rather informal—for example, when exactly does the cohesion of a module becomes so low or the coupling so high that the program "breaks"? They are also approximate, warranting their inclusion in the PC category as well as in TT (conceptual tools for qualitative reasoning). Finally, note that the "Quality analysis and evaluation tools" topic includes, among

other things, the use of reviews to help ensure the quality of design products as one of its subtopics. Thus, the DI category (know-how) could also have been indicated.

The "Software design notations" section pertains to the various notations and languages that can describe and represent software design artifacts. These include structural (static)—for example, class and object diagrams, and deployment diagrams—as well as behavioral (dynamic) descriptions, such as dataflow diagrams, sequence diagrams, state transition diagrams, and pseudocode.

Because the structural and behavioral descriptions did not really fit in any other category, I categorized them as DI. For example, although these tools may allow designers to specify various aspects and features of a software artifact, the tools do not help the designers decide what these features should really be, which is what the CaS category entails. In other words, these notations are relatively quality- and target-neutral: We can write bad (structured) pseudocode or draw senseless class diagrams. On the other hand, since these notations and techniques can be seen as ways of thinking about a design, making it possible to express various design facets, they have been listed in the DI category.

The final section of the KA is "Software Design Strategies and Methods." This was probably the easiest to categorize. Since all such methods are concerned with "how" to proceed during software design, all topics have been classified as DI.

One striking feature of the above categorization is that, probably contrary to more established fields of engineering, few elements of the software design KA belong to the CaS and the QD categories. Two reasons help explain this fact. First, the software design KA is only one of 10 KAs described in the Guide to the SWEBOK. For instance, we might expect the software requirements analysis and the software quality analysis KAs to have identified more topics in these categories. A second, and important, reason is that software design might not yet be considered mature, often more akin to art than science. As the field evolves and matures, additional topics related to these two categories of knowledge might emerge.

Mapping the different topics of the software design KA into Vincenti's categories of engineering knowledge has been an interesting exercise and challenge. The above categorization might well also be open to challenge in the initial sense of the word—that is, "dispute," according to my pocket dictionary—since different people can have slightly different interpretations of Vincenti's categories … almost as I did every time I reread Vincenti's book. I therefore gladly welcome any comments or suggestions on this classification.

Guy Tremblay
Dept. of Computer Science
University of Quebec at Montreal
Montreal, Que.
tremblay.guy@uqam.ca

References and notes

1. For an overview of the SWEBOK project and a more detailed presentation of the various knowledge areas, see *IEEE Software*, Nov./Dec. 1999, vol. 16, no. 6; or http://www.swebok.org to obtain the current version of the Guide to the SWEBOK.
2. Vincenti's work is grounded in traditional engineering. Thus, the common use of the notion of device is "Devices are single, relatively compact entities […]". By contrast, a system is simply an assembly of devices. As he admits, the frontier between the two is often tenuous.
3. In the most recent version of the Software Design KA description (version 0.90), the topic "Key issues in software design" has been turned into a top-level topic. Minor modifications to some of the topic names have also been made. The ensuing discussion, based on version 0.70 of the Guide, still remains valid for the new version.