On the dimensions of software documents – An idea for framing the software engineering process

E. Kindler, H. Baumeister, A. Haxthausen, and J. Kiniry
DTU Informatics
DK-2800 Lyngby, Denmark
{*eki,hub,ah,jkin*}*@imm.dtu.dk*

***Keywords*-software engineering documents; software development process; software engineering theory;**

I. Introduction

In a call for action [1], Jacobson, Meyer, and Soley, together with many other signatories, encourage the software engineering discipline to "re-found" software engineering based on a "solid theory" [2]: The SEMAT initiative. In a soon to be published book [3], the "The Essence of Software Engineering" is presented by "Applying the SEMAT Kernel". This SEMAT kernel identifies the essential concepts or "things" that need to be kept track of in order to successfully develop software, the so-called *alphas* (α). This way, SEMAT conceptualizes the "things" going on in the software development process, independently from a specific software development approach, methodology or philosophy. The alphas allow us to talk about what things need to done and monitored, discussed and taught in software engineering independently from how they are done in a specific development approach. This agnostics when identifying the alphas is one of the strength of SEMAT's conceptualisation.

Surprisingly enough, the artefacts that are used for software development seem not to be of primary concern in SEMAT: documents describing the software in some form or the other. In this paper, we understand *software documents* in the broadest possible sense, which would subsume single paragraphs with the product objective, product definitions, systems specifications, source code, binary code, tests (executable and not), all kinds of UML and non-UML diagrams, formal models, user stories, GUI definitions, and handbooks; in short, any written or graphical artefact we encounter during the software development process (be it on paper or in electronic form).

We can only guess as to why software documents do not play a more prominent role in the SEMAT kernel; one reason might be that discussing any of these software documents specifically, would introduce a bias towards some specific development approaches – SEMAT would not be agnostic anymore. When discussing specific documents – and in particular when defining specific structures and how they should be written – we might introduce a bias towards how things should be done, and this way towards a specific software development philosophy.

Still, we believe that software documents are way too important not to be a primary concept of a theory of software engineering. In this paper, we will have a first glance at the space of software documents and their characteristics – independently from a specific software development philosophy. In order to understand this space, we identify some first dimensions that span the space of all software documents with their different characteristics; we give a glimpse of how these dimensions could be used to better understand what should be done during the software development process, which, in particular, would help teaching software engineering. Moreover, the way and order in which different software development approaches create documents with their specific characteristics in this space – i. e. the project's software document trajectory in this space – might characterise specific software development approaches and provide insights into the way they work.

In this paper, we will discuss some ideas of how this could look like. This paper, however, does not provide the answer yet – we do not even dare to fix the most essential dimensions yet. The dimensions and examples discussed in this paper, should demonstrate that it is worthwhile investigating the dimensions, and that, eventually, these dimensions could be an ingredient to the theory of software engineering.

II. Dimensions and their purpose

Next, we discuss some first candidates for some of the dimensions of software documents, and how they reflect on the development process.

A. Some dimensions

Figure 1 depicts three dimensions, which – from our teaching experience – seem to be important for software development. For lack of a better name[1], we call the first one the *"What-How" dimension*; the idea of this dimension is that in the early phases it should be defined "what" the final software product should do, in contrast to "how" this

[1] A Sofa Seminar discussion of the Software Engineering Section of DTU Informatics resulted in a proposal to call this dimension the Abbott-Costello dimension after the famous "Who's on first?" performance from 1945.

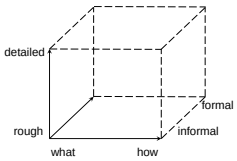

Figure 1. Three dimensions of software documents

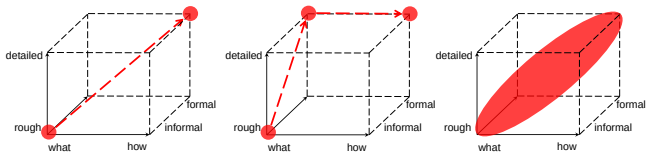

Figure 2. Process trajectories

is finally technically realized and implemented. The second dimension is *level of detail*, which runs from "rough" to "detailed" – we will see later, that the *level of detail*, probably, can be decomposed into two independent dimensions. The third and, for now, last dimension is *formality*, which runs from "informal" to "formal". Also for *formality*, it appears that it can be decomposed – at least its is entangled with another dimension, *executability* (see Sect. II-C).

Of course, there are more dimensions; which ones are the most relevant and helpful ones, is still an open issue. For getting a grip on the issue, we will start some form of wiki or open document, where all interested people could contribute their perspective. A reasonable schema for defining a dimension could consist of a *name*, an (informal) *definition* or characterisation, and a "*litmus test*" for identifying on which side of a dimension a software document would be located; in some cases, there could be even some *metrics* for measuring documents with respect to the dimension; most importantly, there should be a set of *examples* that show which kind of document would be at which end of the resp. dimension. For example, the "product objective", which typically is a single sentence or paragraph of what should be achieved with the product, would be about the "what", "informal", and "rough"; by contrast, the handbook would be about the "what", more or less "formal", but "detailed". The result of an object oriented analysis would still be about the "what", would be more "formal", and more "detailed". The code – remember that we also consider code as a document – would be about the "how", would be "formal", and "detailed". It is a worthwhile exercise to place more kinds of software documents in this space.

B. Dimensions and development process

Now, let us have a brief look at how software engineers would navigate through the space of software documents. Figure 2 shows three cases. The left one, is where there might be a *rough product idea* or *objective* initially, and an *implementation* finally. The middle once shows one iteration of agile development: it goes from an initial *user story*, via an (automated) *test*, to the final *impelementation*. The right one, shows a more waterfall-like process, which more systematically covers all stages. Being agnostic about the process, we do not prefer one over the other – it certainly depends on the kind and size of software what is better. Anyway, Fig. 2 suggest that the trajectory of a process in the space of software documents tells something about the underlying process – for the left one, there almost certainly is no handbook, since this would imply that the "detailed" "what" area is covered.

As mentioned, the middle trajectory shows one iteration of a agile development process only. When going through different iterations the collected user stories and implementation would cover more and more features of the final software. This observation, actually gives rise to another dimension: *coverage*, which is not yet shown in Fig. 2.

C. More dimensions and entanglement

The *coverage* mentioned above seems to introduce another dimension of software documents (or in the case of agile a collection of documents). Somehow this is related to the *level of detail* – just organized according to the product's features or functions. The *level of detail* seems to have two independent components: *coverage* and *abstraction*, which however needs more investigation.

Likewise, there are other dimensions like *non-executable/executable*, which, however, is entangled with (i. e. is not full independent of) *formality*, since *executability* implies some form of *formality*. And there are more dimensions, that should be discussed before ultimately deciding on *the dimensions* of software documents: "textual/graphical", "imprecise/precise", etc.

III. Conclusion

In this paper, we gave a glimpse of the dimensions of software documents – barely enough to see that it might be a worthwhile endeavour to better understand these dimensions, which then could be a part of software engineering theory. In this endeavour, existing characterisations of kinds of software documents such as the one discussed by Bjørner [4] should be taken into account.

References

[1] I. Jacobsen, B. Meyer, and R. Soley, "The SEMAT initiative: A call for action," Dr. Dobb's Journal, Dec. 2009, http://www.ddj.com/architect/222001342.

[2] I. Jacobsen and I. Spence, "Why we need a theory for software engineering," Dr. Dobb's Journal, Oct. 2009, http://www.ddj.com/architect/220300840.

[3] I. Jacobsen, P.-W. Ng, P. E. McMahon, I. Spence, and S. Lidman, *The Essence of Software Engineering*. Addison Wesley, 2013, pre-publication draft.

[4] D. Bjørner, *Software Engineering 1*. Springer, 2006.