Assessing Use Complexity of Software: A Tool for Documentation Designers

Brigit van Loggem

Open Universiteit, Valkenburgerweg 177,
6419 AT Heerlen, The Netherlands
brigit.vanloggem@ou.nl

Abstract. One way to support end users of software is to provide documentation materials such as user manuals and online Help. As not all software is equally difficult to master, documentation designers need to determine the quality and quantity of the information to be included in the user documentation. A first step towards this end would be to assess the complexity of the software from the user's point of view. This paper suggests one approach to such an assessment, based on the idea of use complexity as a multi-dimensional construct. A consideration of width, depth and height of use complexity can help designers determine documentation requirements.

Keywords: user documentation, use complexity, documentation design.

1 Introduction

Among the practitioners who work to support users of software are the designers of user documentation. Since manuals and Help systems contain information that is recorded and stored before the software is used in naturalistic environments, documentation designers cannot observe the software being applied to real-life tasks. Matching the quantity and quality of the information to the user's needs, without being able to assess those needs directly, constitutes a major challenge when designing user documentation. Documentation designers need to couple their own in-depth knowledge of the software to be documented with a methodical approach; so that even if they can carry out only abstract analyses, these yield valid starting points for design.

This paper briefly outlines one such approach, in which a series of abstract analyses is carried out to assess software complexity from the user's point of view. This is referred to as *use complexity*: that part of task complexity that originates from the software rather than from other elements in the task environment, including the user. Unlike design complexity, which is the degree to which a program exhibits emergent behavior [1], use complexity is a measure of the learning requirements (and therefore the documentation requirements) for a particular software system.

As user documentation aims to support users in applying software to real-life tasks, its design must be task-oriented rather than system-oriented [2]. To do so, it must contain more than straightforward procedural instructions on how to interact with the system, and cover the whole of the "User Virtual Machine" or UVM; which is defined

M. Winckler, P. Forbrig, and R. Bernhaupt (Eds.): HCSE 2012, LNCS 7623, pp. 267–274, 2012.

as “not only everything that a user can perceive or experience (as far as it has a meaning), but also aspects of internal structure and processes as far as the user should be aware of them” [3]. The visible part of the UVM is what these authors refer to as the “perceptual interface” and what is more commonly referred to as the user interface; but the UVM as a whole is a much larger conceptual machine that exists in the user’s mind. A software system is a self-contained “world” with its own objects (think of the Clipboard in many operating systems; of templates, style sheets and fields in a word processing environment; or of layers in an image editor). These software-specific objects, with their mutual dependencies and the rules governing their behavior, are as much part of the UVM as is the interaction layer through which they are accessed. A user needs a thorough understanding of the UVM to gain complete mastery of a particular software tool, and apply it successfully to every task it can possibly be applied to. Such understanding is fostered through meta-communication in the shape of documentation or training [4].

Use complexity describes the UVM, providing a measure of what the UVM consists of rather than how a user interacts with it. As such it is not directly or inversely related to usability, which focuses on the user interaction component of the UVM as indicated in the ISO 9241-11 standard by the definition of usability as *the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use*. Usability is prescriptive, in that high usability is desirable; whereas use complexity is descriptive, in that high use complexity is not necessarily undesirable. Use complexity is a necessary consequence of versatility [5, 6] and a complex system may, but need not, have high usability just as a simple system may, but need not, have low usability.

2 Dimensions of Use Complexity

Complexity is a multi-dimensional construct [7, 8]. The quality and quantity of the learning required to achieve full mastery of a particular software tool can be analyzed from a number of different viewpoints. It is then possible to take the idea of dimensions literally and visualize the results of the separate analyses, leading to an image of use complexity offering pointers for the design of documentation.

2.1 Width: Novelty

The first dimension of use complexity to look at is involved with quantity of things to be learned. Learning is required when the software brings novelty to the task environment. Novelty comes in the form of hitherto unknown concepts, with their associated rules and interdependencies, that are exposed to the user and with which he may choose to interact. Where the user is exposed to many novel concepts that he can choose to interact with, and many choices are “wrong” in that they will not lead to an optimum end result, there is much to learn before full mastery is reached. Where at the opposite end of the spectrum there is no novelty to choose, or every choice is equal to every other choice, there is hardly any requirement for learning at all.

This quantitative dimension, related to the number of meaningful choices open to the user that originate from novelty in the software world, can be pictured as the "width" of the use complexity. *The width of the use complexity is proportionate to the number of hitherto unknown concepts (with their associated rules and interdependencies), brought to the task environment by the software, that are exposed to the user and with which he may choose to interact so that his choice makes a difference*.

Several theoreticians have independently presented a hierarchical description of interactive systems. Moran [9], for example, distinguished four levels, or layers, at which a computer tool can be described, one of which (the interaction layer) describes physical interactions with the computer hardware such as: "key presses and other primitive device manipulations". In the context of standard computer use, at least for fully-grown users who are not physically restricted and who possess basic computer skills, the interaction layer can a priori be assumed to be fully mastered. This leaves three layers at which to describe software:

- The *task layer* is that aspect of a software tool that describes the possible end results of the user's interaction with the software.
- The *semantic layer* is that aspect of a software tool that describes the intermediate steps that the user may carry out to realize a certain result.
- The *syntactic layer* is that aspect of a software tool that describes a user's choice of commands with which he directs the software's behavior.

The use complexity is not necessarily equally wide at the syntactic, the semantic and the task layers. At the task layer the user chooses to work towards a certain end result. At the semantic layer he chooses the next step to realize the result. At the syntactic layer, finally, he chooses which screen area to click or touch, which key or keys to press, which sequence of characters to enter, or which command to vocalize.

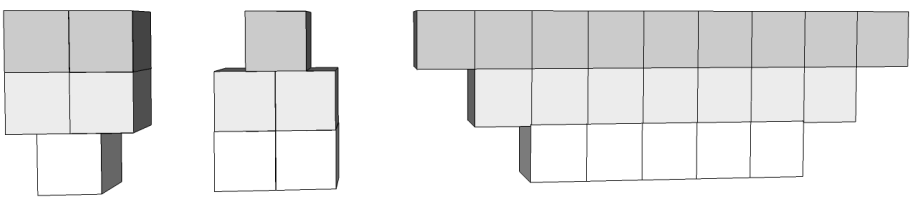

Fig. 1. Width of use complexity sketched for Notepad (left), WordPad (center) and Word2007 (right)

When analyzing the layers one by one it is important to take all novel opportunities for choice into account, as there is no way of knowing which are important and which are not. Since the higher layers build upon the lower ones, even misunderstandings at the syntactic layer can lead to grief. Concepts such as pressing the Enter key on the keyboard resulting in a paragraph being created, or a space character not being nothing can, when not thoroughly understood, make laying out even the simplest text document very difficult. Conversely, novel concepts at the higher layers may provide meaning to the lower ones. For example, the presence of the concept of "wildcards" at

the semantic level may lead to unexpected results for an uninitiated user attempting to search his text for the occurrence of an asterisk.

In Figure 1 the width of the use complexity is roughly sketched, using a relative scale of 1 ("very little") to 10 ("very much"), for three different text processing applications that have all been marketed by Microsoft®. The base corresponds to the syntactic layer, then comes the semantic layer, and the top represents width of the use complexity at the task layer.

Notepad is judged to have very little novelty to interact with at the syntactic layer: its interface is uncluttered and most of the options open to interaction are well known. At the semantic layer a bit more novelty is found, for example in Notepad's dealing with word wrapping and the use of variables in its page setup; while at the task layer its uses are acknowledged as a tool for holding and converting electronic text that was never intended to be printed. WordPad on the other hand holds a little more complexity at the syntactic layer, mostly due to its formatting capabilities, but less at the task layer as it is mainly geared towards producing a document that is to be printed as laid out on the screen. For Word2007 much more novelty is found at all three layers, more so as we move up towards the task layer.

2.2 Depth: Mapping

The difficulty brought to an activity by a software tool can be seen as stemming from an experienced gap between the software world and the non-software world. To close the gap, the novel concepts that are open to interaction must be reconciled with meaningful ideas in the pre-existing task world [10, 11]. This is relatively easy when there exists a correspondence between the novel software concepts on one side of the gap and known concepts in the outside world on the other side. Yet software can bring to an activity not only new ways to do things, but also new things to do.

Where novelty in the software corresponds to pre-known ideas in the pre-existing task world, it allows for new ways of doing things. The knowledge that needs to be acquired then is mostly procedural: specifying how to do something. Yet where the software exposes its users to novelty that does not map directly onto pre-known ideas there is more of a problem. There are now new things to do, and a user who is not even sure what it is that he should be doing in the first place, is unable to frame his interactions in such a manner as to use the software to its full potential. To achieve full mastery, the user will need to acquire more than just procedural knowledge.

This distinction constitutes a qualitative dimension of use complexity that can be visualized as its "depth". The less tightly that the novelty brought to the task by the software is coupled to pre-known concepts, the deeper the use complexity. *The depth of the use complexity is proportionate to the degree of mapping between the novelty brought to the task by the software and pre-known concepts.*

The degree of mapping between software novelty and pre-known concepts can most readily be determined by considering the outcome of the user's interaction with the software. When the outcome of the interaction lies within the software only, there is no direct mapping. This is the case, for example, at the task layer of software for creating websites and at the semantic layer of text editing software that allows for regular expressions in searching and replacing. In other cases, the outcome of the

interaction lies partly outside and partly inside the software. The software then holds a model of something in the physical world or in the user's mind, and it is the model that is modified. The mapping is then much more straightforward. Think, for example, of a kitchen planner such as offered by many home furnishing stores, whose task layer is involved with constructing a model of your new kitchen; or a calculator program where at the semantic layer the memory functions M+, M- and MR mimic a scratchpad holding intermediate results.

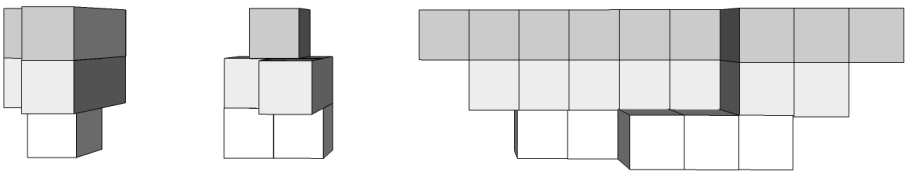

Fig. 2. Depth of use complexity added to the analysis for Notepad (left), WordPad (center) and Word2007 (right)

Figure 2 shows the sketches from the previous figure enhanced with depth to indicate novelty for which there is no straightforward correspondence with the pre-known world. Approximately half of Notepad's novelty at the semantic and task layers is considered not to correspond to pre-known concepts; whereas WordPad is judged to expose the user to non-mapping novelty only at the semantic layer, and a significant proportion of Word2007's novelty at all layers has meaning within the software world only.

2.3 Height: Impact

Turning our attention now to the use that is made of software, we find in Activity Theory a different three-tiered hierarchy; one which is fluid rather than fixed, user-centered rather than tool-centered. Central to the theory is the idea that all human activity is mediated by tools and context and that these fundamentally change the activity. Activity Theory covers many different aspects and offers a rich framework for the development of theory and practice alike [see 12, 13]. For the purpose of assessing use complexity of software, however, further details of the activity-theoretical framework can be largely ignored, taking from it only the following decomposition of human activity:

- An *activity* is a sequence of actions, undertaken one after the other in order to achieve an object, which provides the overall intention to do the work.
- An *action* is a sequence of operations, undertaken one after the other in order to achieve a goal within the wider framework of the activity's overall intention.
- An *operation* is a routinely carried-out observable behavior.

The task layer in any given software tool does not necessarily correspond to the activity, nor the semantic layer to actions or the syntactic layer to operations. Task layer, semantic layer and syntactic layer are tool-centered and objective. To determine

what they consist of, it is sufficient to consider only the software. Activities, actions and operations on the other hand are human-centered and subjective. They change over time and from one performer to another and to determine what they are, it is sufficient to consider only the user. Aligning the two hierarchies, the human-centered and the tool-centered, exposes the relative roles of the two partners in a man-machine system.

These are not always the same [14]. The software component does not necessarily affect all the levels in an activity: situations are easily imaginable in which only part of the activity is computer-mediated. Software that affects the activity only insofar as it changes the nature of operations is less complex than that which touches upon actions, or even changes the nature of the activities that are possible. As the software's mediation reaches further up into the activity, it adds more uncertainty to constructs that were not very well defined to begin with: the goals of actions and the intention of the activity. This third dimension can be visualized as the "height" of the use complexity, enhancing the impact of the width and depth. *The height of the use complexity is proportionate to the degree to which the software can affect an activity (in the activity-theoretical sense of the word).*

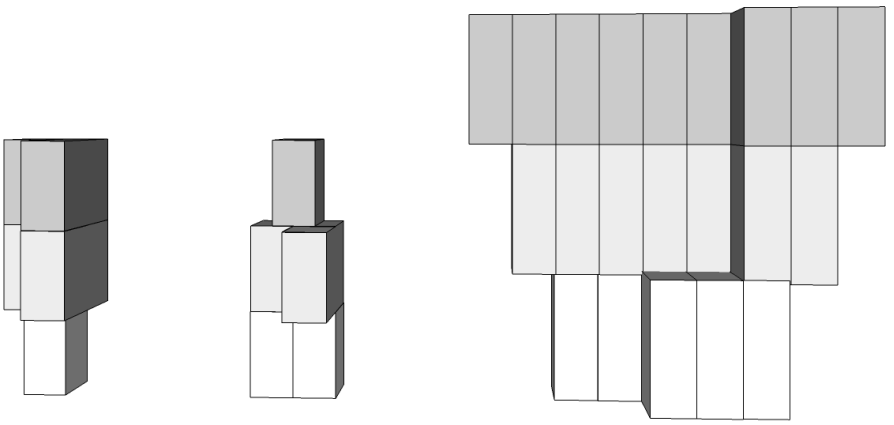

Fig. 3. Height of use complexity added to the analysis for Notepad (left), WordPad (center) and Word2007 (right)

To determine the height of the use complexity, first ask whether the outcome of the task layer could ever be imagined to constitute a valid object, providing the intention to sit down to work. If so, the software reaches up into the level of the activity; if not, its application constitutes no more than the goal of an action and it reaches no further than into the level of the separate actions. This is reflected in the visualization by "pulling up" the use complexity to the level of the actions (left and center in Figure 3) or all the way to the overall activity (right).

The outcomes of Notepad's and WordPad's task layers are relatively modest, and in this example they are not seen to provide more than the goal for an action such as writing a short note or making a shopping list. The outcome of Word2007's task layer on the other hand could well be imagined to be an object in itself, such as the

complete production and layout of a camera-ready manuscript or the programming of a word processing environment for third parties to use.

3 Conclusions

Use complexity is a promising concept to structure and direct documentation design efforts. However, much more work is required before it can be fruitfully deployed in sound engineering practice and is scalable for different types of software. A stepwise course of action must be developed for documentation developers to assess use complexity in a methodical manner, and measurable indicators must be established for all three dimensions. How to determine when a concept in the software world contributes to width and possibly depth of the use complexity, and by how much? How to decide whether the task level of the software reaches up into the intention for an activity or merely constitutes the goal of an action? To allow for learning the approach and to limit inter-rater variance, stringent definitions and ordinal scales on all three dimensions must be developed, unambiguously illustrated with standardized examples.

Therefore, further research is very much needed (and indeed planned), preferably in close cooperation with the direct stakeholders: the designers of user documentation.

These are uncharted waters. The field of documentation design tends to focus on the effects of specific interventions rather than underlying mechanisms or methodological approaches to the design process [2]. Task analysis in HCI (human-computer interaction) is usually carried out before a new system or version is developed, while CSCW (computer-supported cooperative work) considers tasks and working practices emerging from new technology for groups of people working together, rather than individuals. Usability, as we have seen, refers to only part of the User Virtual Machine and there is no consensus as to its metrics or even its constituent parts [15]. Dix et al, for example, see usability as having three constituents: learnability or the ease with which new users can begin effective interaction and achieve maximal performance, flexibility or the multiplicity of ways the user and system exchange information, and robustness or the level of support provided to the user in determining successful achievement and assessment of goals. Each of these constituents is in this framework seen as a function of underlying aspects, resulting in predictability, synthesizability, familiarity, generalizability, and consistency; dialog initiative, multi-threading, task migratability, substitutivity and customizability; and observability, recoverability, responsiveness and task conformance as factors to usability [16]. Comprehensive as this overview seems, it bears little likeness to the definition in the ISO 9241-11 standard. Indeed, other authors (for a more in-depth discussion, see [15]) present different indicators for the same concept of usability. And although some of the many aspects mentioned (e.g. familiarity) reduce use complexity, others (e.g. flexibility) may actually lead to increased use complexity. Use complexity is thus a concept that, appropriately operationalized to become a practical tool rather than a theoretical construct, can provide practitioners in documentation design with much-needed guidance.

Acknowledgment. I am indebted to Gerrit C. van der Veer and three anonymous reviewers for their insightful comments on the first draft of this paper. They pointed me to relevant literature, and helped me distinguish the concept of use complexity from its practical applicability and from other, related concepts.

References

1. Aiguier, M., Le Gall, P., Mabrouki, M.: A Formal Definition of Complex Software. In: Proceedings of the 2008 The Third International Conference on Software Engineering Advances, pp. 415–420. IEEE Computer Society (2008)
2. Schriver, K.: Dynamics in document design. John Wiley & Sons, New York (1997)
3. van der Veer, G.C., van Vliet, H.: The Human-Computer Interface is the System: A Plea for a Poor Man's HCI Component in Software Engineering Curricula. In: Ramsey, D., Bourque, P., Dupuis, R. (eds.) Proceedings of the 14th Conference on Software Engineering Education and Training, pp. 276–286. IEEE Computer Society (2001)
4. van der Veer, G.C., Tauber, M.J., Waern, Y., van Muylwijk, B.: On the interaction between system and user characteristics. Behaviour & Information Technology 4, 289–308 (1985)
5. Norman, D.A.: The Invisible Computer. The MIT Press, Cambridge (1999)
6. Norman, D.A.: Living with Complexity. The MIT Press, Cambridge (2010)
7. Nadolski, R.J., Kirschner, P.A., van Merriënboer, J.G., Wöretshofer, J.: Development of an Instrument for Measuring the Complexity of Learning Tasks. Educational Research and Evaluation 11, 1–27 (2005)
8. Kim, S., Soergel, D.: Selecting and measuring task characteristics as independent variables. Proceedings of the American Society for Information Science and Technology 42, n/a–n/a (2005)
9. Moran, T.P.: The Command Language Grammar: a representation for the user interface of interactive computer systems. International Journal of Man-Machine Studies 15, 3–50 (1981)
10. Payne, S.J., Squibb, H.R., Howes, A.: The Nature of Device Models: The Yoked State Space Hypothesis and Some Experiments With Text Editors. Human-Computer Interaction 5, 415–444 (1990)
11. Payne, S.J.: On Mental Models and Cognitive Artefacts. In: Rogers, Y., Rutherford, A., Bibby, P.A. (eds.) Models in the Mind: Theory, Perspective and Application. Academic Press, London (1992)
12. Bødker, S.: Through the Interface – A Human Activity Approach to User Interface Design. Lawrence Erlbaum Associates, Hillsdale (1991)
13. Nardi, B.A. (ed.): Context and Consciousness: Activity Theory and Human-Computer Interaction. The MIT Press, Cambridge (1996)
14. Mirel, B.: "Applied Constructivism" for User Documentation. Journal of Business and Technical Communication 12, 7–49 (1998)
15. van Welie, M., van der Veer, G.C., Eliëns, A.: Breaking down Usability. In: Interact 1999, Edinburgh, Scotland, vol. 99, pp. 613–620 (1999)
16. Dix, A., Finlay, J., Abowd, G.D., Beale, R.: Human-Computer Interaction. Pearson (Prentice-Hall), Harlow (2004)