RC23015 (W0311-158) November 21, 2003
Computer Science

IBM Research Report

Towards a Metric of Software Interface Complexity

Joshua I. Dulberger
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Towards a Metric of Software Interface Complexity

Joshua I. Dulberger

IBM Thomas J. Watson Research Center

P.O. Box 704

Yorktown Heights, New York 10598

Abstract:

The interaction between humans and computers increasingly governs productivity. There have been largely uncontroversial advances in interface technology, such as the original Macintosh operating system and GUIs (graphical user interface), yet the methodology of HCI is currently a confined one; it relies on the intuition of designers in conjunction with lengthy often expensive empirical study. The primary point of interest in human-computer interaction lies in what is now termed "usability," but in the past has also been called "complexity." Software engineers, developers, and designers are merely able to gesture at a nebulous conception of interface complexity, often disagreeing not only about what this alleged complexity amounts to, but also whether it is necessarily a positive or negative attribute. This paper aims to make a clean distinction between usability and complexity, and with this distinction to gain insight into the views concerning interface complexity. We hope to begin a discussion of a conception of interface complexity that is precise, quantifiable, and useful. With the further development of a complexity metric, a powerful tool will be placed in the hands of those interested in the improvement of software productivity.

1. *Introduction: Complexity Science and Interface Complexity*

Before delving to the subject of this research, for the sake of clarity we should sort some laundry, mainly with a brief discussion of a field known as complexity science, which seeks to understand and explain dynamical, nonlinear systems and randomness. A brief outline of the goals and methods of complexity science will help us see the relationship, should there even be one, between measuring software complexity and complexity science. It is my (informed) hunch, that such a relationship is at best obscure right now, but I would like to flesh out why this is the case before moving onto the central topic of our discussion.

Some things in our world are, at least primae facie, simple. One reading of this claim of simplicity is that there exist primitive things. In language for example, the letters of the alphabet are generally accepted as primitive, which is to say that they permit not further analysis. Words might also be simple, though they are not primitive because they do permit analysis: into letters. The natural sciences, and physics in particular, has by and large sacrificed its commitment to primitives. Physicists think that primitives do not exist, it is just an appearance that some objects are primitive, and this appearance is dispelled upon further empirical study. To understand the brain, we look for chemicals, to understand chemical we look for particular molecules, to understand molecules we look to atoms, to understand atoms we look for electrons, protons and neutrons, and to understand these entities we now look to even small things called quarks. It is doubtful that anyone thinks quarks will be primitive, and already there are likely a slew of parts we can break quarks into. This view, that to understand (or explain) x one must understand (or explain) the parts of x, is one of many that falls under the surname "reductionism," as it seeks to understand all phenomena by means of understanding the parts that make up the phenomena.

These parts are then to be understood via the same method, presumably (though rarely explicitly) ad infinitum.

One might have some doubts as to the soundness of this method (if explaining *x* always requires explaining *x*'s parts and there exist no primitive, one must be skeptical about explanation in general), but a critique of this brand of reductionism is not the project of this paper. In fact, it seems that synthetic methods tend to be applied once reductionist methods fail. It is our hope to establish some primitives in the study of interface complexity, and that by breaking the interface into natural primitives we can understand, measure, and ultimately decide what to do with it.

Much of the scientific and philosophical interest in complexity is directed at the relation between different levels in a hierarchical reality. It is thought that some phenomena occurring at a higher level, say psychological, reduce to lower level phenomena, say chemical or physical. However, complexity is often employed as a term to describe relations between levels that somehow disable reduction. In short, this is not the brand of complexity, call it inter-level complexity, is not the subject of this paper. We are interested in what might be called intra-level complexity, that is given a level of analysis, are there properties that might make some things at that level "more complex" than others.

2. ***Defining the Problem of Interface Complexity:***

Software engineers simultaneously struggle to combat "complexity" and bolster product functionality. Yet without a rigorous definition of "complexity," the former part of this goal is a

moving target, and so we will try to establish a definitive conception of complexity that provides insight into the design and implementation of software.

To begin, let's take a concrete example, the Windows desktop. Now how might we decide how complex the Windows desktop is? Well, the answer to this question will depend on what "complex" is taken to mean. In this context complex might mean one of the following things (this is not necessarily a comprehensive list):

1. How visually complex is the desktop?
2. How difficult is it to perform some task (i.e. Open a program)?
3. How many lines of code is it composed of?

There are probably a great many variation on these different views of complexity. Our project, however, is not directly concerned with questions of type (3), because the we are primarily concerned with what the user sees and interacts with, but not the code the determines this interaction. Different programs might produce the same interface, or at least very similar interfaces, and so long as the user cannot distinguish between these interfaces, they are functionally equivalent. Because of this functional equivalence, this project is not concerned with the eloquence of coding, but rather we are looking at the interface as a functional unit in its own rite. In other words, an interface is a black box, and so we hold the interface to be, in some sense, primitive in our question. As such, in our study of complexity, we will not endorse a metric that analyzes our subject into parts and then simply counts the number or types of parts.

So, our conception of complexity might still lie in (1) or (2). There is an appropriate worry that accompanies the observation that (3) appears to be the only question to which there is an

objective answer. That is, the number of lines of code is the same, no matter your perspective, no matter who is counting and from where she is standing. Both (1) and (2) seem both to be subjective, that is they seem to ask "how complex is the desktop for x?" If both (1) and (2) are subjective, then the desktop might be more complex for me than it is for you, and if I informed you that I measured Windows at 90 complectiles, it would be equivalent to me telling you that it was hard for me. While you would know how I feel about Windows, the problem is that this would not provide you with information about how complex Windows is for you, and so we might wonder why attaching a number to it would be of any help at all.

It might be the case that the complexity Windows is completely subjective, which is to say that it is determined by its context or its user. If this were the case, then the metrics of complexity would have to shift from looking at the Windows interface in itself, to looking at the relationship between users and the interface. Windows itself would not be measured, in this case, but the relationship would be measured. For the sake of clarity, I will call the measurement of this relationship *Usability*. I think this appropriately captures the relative nature of this measure, when we ask how usable Windows is, we certainly have in mind that usability is contingent upon the user and there is no usability without a specified user.

What we are concerned with in this project is whether there is an interesting kind of complexity that is not relativistic and can be measured of an object, i.e. An interface, in itself. If we could develop such a concept, then the question "how complex is Windows?" could be answered while both treating the interface as primitive, while rendering the "for whom" question irrelevant.

Before further developing this non-relativistic sense of complexity, consider why we might think there would be such a thing. Take a common example: from a young age, most human beings are quite good at recognizing faces. Yet human beings are often poor at arithmetic involving more than two numbers or more than a few digits in those numbers. Computers are quite good at adding and subtracting, without respect to the amount of digits or number of numbers involved in the calculation. Yet it is quite difficult to have computers recognize faces, a project which meets only limited success after many years of intensive research.

Now, in terms of complexity, we might say that humans are quite good at some complex tasks, while quite bad at others. Conversely, we might also say that humans are poor at task that are not very complex, but good at others. We might worry that "good at" and "bad at" is simply in relation to computer counterparts, but this need not be the case. Given one human, suppose there are complex things she is better at doing than less complex things. If this supposition is coherent, then we have some non-relativistic conception of complexity at work. It is important to notice that while the thing in itself may be more complex than another thing in itself, the user is irrelevant in this calculation, just as when one object is heavier than another, the person reading the scale is irrelevant.

One way to dig a bit further into what we might mean by intrinsic complexity is to think of a map that looks like a branching tree, and a person starting at a node A and trying to get to another node, B. The first time the person tries to navigate from A to B, she might take a few wrong turns, take time to think about each choice, and eventually find her way to B. But with some practice, she will probably remember many of the turns and perhaps even the entire route.

The number of turns between node A and node B might be one way of getting a handle on how complex the route is. That number will not change from person to person not is it a matter of perspective. Yet some people will be much better at getting from A to B, some with think it easy and some might think it is difficult. In all likelihood, experience will alter their impressions of how difficult the route is. Yet there are things about the route that are constant, and if there is a coherent non-relativistic notion of complexity, this will be one of those constants.

This thought experiment takes question (2) “how difficult is it to perform a task?” As the relevant question, but demands that difficulty be objectively defined here: How difficult, in our sense, is it to travel a route? It takes 8 turns to get from A to B. This leaves question (1) “how visually complex is the interface?” by the wayside. While visual complexity is of particular importance to the user, if the desktop has too many visual foci it may be very inefficient to navigate, it is not the material of intrinsic complexity. Because the visual aspect enables users to accomplish tasks with greater ease, it is certainly a component of usability. Yet the visual aspect is but a representation, often metaphorical, present in the interface.

This brings up the question of what exactly is an interface? HCI traditionally defines the user interface as ”a display for the user (usually on a computer monitor) that allows the user to interact with the system (hyperdictionary.com).” Alternatively an interface is often construed as a representation of an algorithm. But for our project is important to understand the interface more broadly, not just as a screen frozen in time. We are interested in the complexity of a process, and so we must understand interface as a representation that connects user inputs with

system outputs, that is the interface is a representation of a process that allows users to accomplish their goals.

One way of accomplishing this task is to hold the left click of the mouse over the folder and then move the mouse such that the folder is moved to another folder labeled “trash.” The obvious metaphor is that of picking up some garbage on your desk and throwing it away. Windows employs this metaphor to represent the process of inputting a command such as delete<folder’y’.exe> and connecting this input to its output. The Windows interface thus makes deleting files, among other things, easier for novice users who are unfamiliar with programming language. It purchases this ease by using metaphors, but the use of such metaphors potentially creates tortuous routes for experience users. For those who are familiar with languages, presumably proficient typists, simply typing a delete command is faster and involves fewer steps than employing the Windows metaphor. Furthermore, for those who know that files can be deleted by right clicking on the mouse and choosing delete, the click and drag metaphor is generally obsolete - requiring more time and steps to accomplish the same task.

So we are concerned with the intrinsic complexity of the dynamical interface. While it is plausible to talk of just the intrinsic complexity of a process, what is of more interest is the comparative complexity of processes with the same goal. If complexity is correlated with some combinatory of steps to accomplish a given task and the choices at each step, then all else being equal, more powerful software will have more choices or more steps. All else being equal, this addition of steps and choices is just what will make some software more powerful.

As such, know that that the interface of a CAD program is “more complex” than the interface of Netscape Navigator is hardly helpful in determining how either interface performs. To make use of a complexity metric, we must first specify a goal, or a set of tasks. Given this set of tasks, we can then compare the complexity of one interface with that of another. For example, in Windows there are a few different processes by which one might delete a file: drag and drop it into the trash, right click and choose delete, draw a box around it and drag the box, click on it and just key delete.

As I hinted earlier, there are two primary ways to analyze intrinsic complexity. First, we might count the number of steps required to accomplish a set task. Second, we might consider the difficulty of choosing one action from a group of actions. To make this more concrete, consider how one might setup the Windows desktop so as to be able to easily open a file. One, rarely take, approach is to litter the desktop with every file. Then in the course of a double click, any file could be opened. But when you consider the mess on the screen, it would be quite difficult to find any file. So while you are only one step from opening the file, the choice is exceedingly difficult. To deal with this difficulty, we might categorize files and place them in folders. We might go further, creating a hierarchical taxonomy, in which folders are within folders, and within those folders are files. But then we see that the open file is more than a mouse click away, first one must navigate the filing system.

Intrinsic complexity is a combination of the number of steps required to accomplish a given task and the number of choices at each step. Neither of these two approaches are sufficient. A simple way to see this is to imagine that either were our sole indicator of complexity. To reduce the

complexity of an interface with many steps, we could simply subtract steps and replace them with more choices at each step - eventually winding up with something like our desktop littered with every file. Conversely, to reduce the complexity of the interface with many choices, we could simply subtract choices, nesting them into steps, eventually reducing the desktop to two folders, and then two within each of those, on and on. In that case, finding a folder could require hundreds of mouse clicks.

3. *GOMS*: *Revisiting the early 80s*

Forming a definition of complexity that is narrow enough to be unambiguously measured, but still able to capture a relevant and useful concept is a problem first formulated in the late seventies and early eighties. Tom Moran (1981) and David Kierias (1983), among others, put a great deal of effort into the creation and development of GOMS (Goals, Operators, Methods, Selection) models, which were aimed at assessing what the authors called "usability." Friends of GOMS, or those who at least wanted to see it get off the ground, posited that "usability" could be measured by the amount of information required to operate an interface. The problem of a "usability" metric is then transferred to a problem of how to measure the amount of sufficient information. The GOMS people conclude that one way to do this is to build a model, in particular a Turing machine, to carry interact with the interface, and then to quantify the information in this model.

In my view, the GOMS framework can be improved with a distinction between complexity and usability. If reserve the term "complexity" for objective properties, then it appears as though the GOMS information criteria captures a good chunk of this objectivity. "Usability" we can then

employ as a label for the gap between the user's information and the amount of sufficient information to operate the interface. The more information a user knows the more usable the interface will be, yet the amount of requisite information will be the same regardless of the particular user.

The skeletal notion that interface complexity (on our understanding of complexity) could best be understood via modeling human-computer interaction. One hallmark of models is that they allow us to make simplifying assumptions in order to identify the relevant features of a phenomenon. Given a set of assumptions, a model can be generated and then compared with actual occurrences. To test the model, we need simply (more or less) compare the model with the real world. If there is a strong resemblance between the model and the actual occurrences, then the model is said to be good one. Good models identify essential features of given phenomena.

That said, the idea behind GOMS is to model a user interacting with an interface. In short, this involves writing a program that is capable of carrying out a given task, for example cutting and pasting in Microsoft Word. The complexity measure is the amount of information required to perform the cut and paste procedure. The information can be quantified as lines of codes, commands, or bits. Goals, Operators, Methods, and Selection criteria must be identified in order for the GOMS model to get off the ground, hence its name. The complexity of the interface then, is defined by the information required to navigate it. Moreover, destinations in the interface must be identified, (i.e. Goals) and complexity is relative to a task. When we ask precisely what is complex on this account, the answer is a task; the complexity is the amount of information to

accomplish a given task. The amount of information is determined by the sub-goals, or operators, methods and selection process required to carry it out.

Interest in GOMS has died down in the last decade or so, primarily as this particular modeling process is an arduous one. In most cases, GOMS requires far more work and generates unsurprising or even obvious conclusions about human-computer interactions. But practical problems aside, there are two principled problems with the GOMS model. First, the GOMS judges complexity by counting steps. Yet steps cannot be the only measure of complexity, because there is often (perhaps always) a trivial way to reduce the number of steps; that is to add more options. This generates the disarray of a desktop with all possible tasks available with one click. Second, one might be concerned as to how well GOMS really models human-computer interaction. To take GOMS seriously, we must draw the analogy between the Turing machine and a person executing a cut-and-paste operation. Those critical of GOMS tend to think there is little or no resemblance here, thus concluding the model is a bad one. While we might still draw conclusions about the model, these critics think those conclusions simple do not apply to people in the real world.

A fix to the second of these problems would be to employ better modeling techniques, borrowing from artificial intelligence. As of yet, AI appears to have limited insight into modeling the human mind, so this solution is not yet available. The problem of transferring steps into choices is more fundamental, and might be the most serious problem in quantifying complexity. If complexity is measured by counting steps, and steps can be turned into choices, then there is either a trivial way to reduce complexity, or complexity must count both steps and choices.

When we include choices, we must consider whether the difficulty of making a choice is an entirely subjective matter. On the face of it, there is little reason to think this difficulty anything but relative to the amount of information a user has about an interface. If you have comprehensive information about cutting and pasting, then there are no hard choices; steps are the limiting factor. For cutting and pasting at least, there is a basic constraint on the precede of moving text: the text must be identified and it's new location established. This procedure requires two steps. Yet if one lacks adequate information, it is common to labor over choices, and the limiting factor is how obvious the correct choice is. Indeed, it is hard to imagine how this might be unambiguously measured.

4. ***An Alternative View: Biology and the Insight of Instruction Manuals***

Some biologists have considered methods for measuring the complexity of different organisms. This is in response to contemporary disregard for the notion that evolution brings "progress" - moral or otherwise. While most deny that evolution implies "progress," even loosely defined, some philosophers and biologists think that evolution has brought an increase in average organism complexity. Armed with a hard metric of complexity, the question transforms into an empirical one. Presumably the fossil record could demonstrate the truth or lack thereof of the claim of increased mean complexity.

But as of yet, only preliminary attempts have been made to quantify biological complexity. This section focusses on the intuitions driving one such preliminary attempt. Dawkins suggests that to compare the complexity of two organisms, we needs imply to write two books; one describing

each. The comparative lengths of the books will determine which is more complex. Of course, for this method to work, the books much go into the same amount of detail. It would be clearly misguided to compare a book about canaries with a book about paramecia when the former details to the level of the cell and the latter details to the level of the molecule. On the face of it, this is a practical problem, but it could turn out to be a fatal one if we cannot find a principled way to standardize levels of description.

Incorporating this view into our previous discussion, we can devise alternative approach: to measure interface complexity, measure the lengths of the instruction manual for different programs. I will call this the *instruction-manual-comparison* method. To refine this idea, we probably would not want to look at existing instruction manuals for Microsoft Word or Lotus Word Pro, but as they are likely to contain varying levels of detail. There is also always a trivial way to make manuals longer, just explain something in a circumnavigate manner. To make manuals shorter, one could just rip out some pages, leaving tasks to be figured out by users. So the manual-comparison-method needs work. More specifically, this method requires a systematic way to construct instruction manuals that will have the same level of detail regarding each program task.

It seems that the problem of measuring complexity is prior to a host of other HCI problems, and a complexity metric would be a tool for work on a variety of solutions. But some work on these other problems may shed light on ours. Bhavnani (1997, 1999, 2000) is primarily concerned with the problem of teaching users to efficiently use complex interfaces, like those found in many CAD programs. He has argued for the introduction of a set of intermediate principles that do not

inform users of the specific steps in accomplishing a given task, but rather provide general strategies for using an interface. Bhavnani asks the question, what five (or seven or ten) strategies would make a user generally more proficient and efficient in the context of a program or set of programs. To answer this question, Bhavnani tries to determine what computers do better than humans. The short answer is iteration. Humans are sloppy at copying and repeating the same task over and again, whereas computers repeat most tasks quickly and with very little error. Hence copy and pasting is a better strategy than retyping the same sentence.

Bhavnani's research might seem only tangentially related to the project of complexity metrics. His work posits the complexity of an interface, and then asks how users can cope with it. There is no pretense at measuring objective complexity. But it is clear that this work could create a new method of writing instruction manuals. These new manuals might contain only the general strategies and some examples, rather than a task by task list of procedures. These new manuals are of interest if we adopt the comparative-manual-approach.

New manuals are but one side of Bhavnani's insight. Few would object to the production of shorter and better program manuals, but there is also something to be said about interface design. If we adopt Bhavnani's view in conjunction with the comparative-manual-method, then all else being equal, designers should aim at interfaces that allow for shorter better manuals, that is, interfaces that can be navigated with only a few general strategies.

5. *Complexity, One Component of Usability*

Thus far we have discussed three distinct views on complexity, though this might not be reflected in the literature. The skeptical and contemporarily popular view of interface complexity holds that complexity is a subjective matter, determined entirely by the user and there is no conception of complexity that can be usefully applied to the interface itself. The second views is the GOMS and information theoretic inspired view, which states that complexity can be quantified as the amount of information required to operate the interface. The thirds view comes from Bhavnani's work in the previous section holding that complexity can be measured in terms of the (number of) general rules governing the interface, or the strategies that would help any user.

Each of these views has its problems. The skeptical view contradicts our intuitions that people are good at some complex things and bad as some simple things, mainly because it determines what is complex as what we are bad at. The GOMS analysis counts steps, but fails to address choice difficulty, and as we saw, steps can often be converted into choices. Friends of GOMS use "mental operators" as band-aids, but such operators can be analyzed into guess work or stand in need of empirical support. Despite its virtues, even the *instruction-manual-comparison* method has serious practical problems; how can we specify a standard level of analysis and provide a fair method of comparison?

The remainder of this paper seeks to reconcile the skeptical view with a GOMS/information theoretic view of complexity. While Bhavnani's work and the *instruction-manual-comparison*

methods are of great interest, at this time I see no resolution to the practical problems it faces. However, we can include its insight in our analysis, given an IOU implementing it. This is possible as it does have a conceptual pertinence to our analysis. The insight Bhavnani provides us is a reminder that not all information is of the same kind. His work suggests that the information required to efficiently operate an interface is sometimes compressible; it can be abbreviated. That is to say, there are patterns running through the information, and if users can be made aware of these patterns, the interface will become easier to use. Information theory would say that what is relevant to complexity is the compressed information. Yet in our case, the compression is non-trivial, some users might do well in that realm, but for those who have the uncompressed form, that is the particular procedures for particular tasks, the interface will be less usable.

Below is one possibility to determine each matrix in the Usability equation.

[Sum Complexity of Program Tasks]
- Steps * Choices for tasks, (100-1000)
- Complexity Primitives (Drop down Menu, Scroll Choices, Right Click, etc.)
- For Tasks:
  - Cut and paste
  - Build table
  - Recall last webpage
  - Etc...

[Information Compressibility]
- Generalization compresses information by some factor [0 - 1]

[Task Weights]
- i.e.
- Open file = 0.1
- Save file = .1
- Cut = .05
- Copy = .05
- Paste = .1
- Change font = .05
- etc...

[User Fxn]
- U_x =
- general literacy
- domain literacy
- Experience
- etc...

Then,

[Interface Complexity] [Compressibility] [Task Weight] [User Fxn] = Usability

The User function might take into account a number of users, weight some users more heavily than others, or it might simply assess an individual to determine her relationship with the interface. With a categorization of users, less information would need be known about a given user to determine usability, but a successful categorization would likely depend on empirical data.

Even if this is an accurate depiction of the user interface, the question arises to whether it is an interesting one. The answer to this question turns on whether we can determine a concept of complexity that makes the account worthwhile. For example, if we are merely interested in the structural complexity of a dynamical interface, this will not inform us as to whether such complexity should be minimized. This is to say that even should we have two interfaces that accomplish comparable tasks, more structural complexity does not imply the inferiority of either interface. If we look at the structural complexity of the Microsoft desktop where files are arranged in alphabetized folders and compare it with a desktop where all files are scattered, the former has more structural complexity. Presumably it is the more usable desktop, provided the user possess the relevant information (in this case the ordered alphabet).

The moral here is a straightforward: more structural complexity does not mean less usability. Structural complexity is an apparatus that can be harnessed for the sake of usability, mainly if it employs information with which the user is familiar or its structure is such that users find it natural or intuitive. One structure in and of itself is no more natural that another, but some structures more naturally accommodate things in the world. So, if we want to use a usability equation of the sort previously mentioned, the measure of structural complexity is not a useful one.

6. Conclusions:

At this juncture it might be helpful to step back from our discussion, which has gradually descended into particulars, to see at least one of the larger pictures here. Computer Scientists are

interested in user-interface complexity because it seems to limit the productivity of users and thus the appeal of the software. If we accept this premise, then our implicit goal in measuring complexity is to provide insight into minimizing it. While measuring complexity would not inform us as to how to reduce it, a metric would be a valuable tool in deciding when we were in fact successful at reducing it. With a soft or merely intuitive understanding of complexity it is difficult to definitively determine whether one interface is more complex than another - arguments take the form of "I feel this" and "you feel that."

So our broad and somewhat vague sketch of this landscape begins with the distinction between usability and complexity. We call usability the relationship between user and interface, and it is a goal of software engineers to maximize usability. Complexity, as we define it, is a property of a dynamical interface, and this property is one of a set of determinants of usability. The other elements of usability consist in the relative importance of each task and the information a particular user possesses. Some tasks are clearly more commonly required and more important, while others are more obscure, and so the final usability measure should take this into consideration. Because usability measures a relationship, the information a user brings to the interface, for example language and experience, is critical in determining how efficient the user-interface relationship will be.

The specifics of both task weight and user information are contingent matters to be decided empirically; the way to find out whether a task is important is to look at what users actually need to accomplish, the way to find out what information users have is to ask them (i.e. Test them, study their behavior and draw inferences, etc.). In assessing the big picture of usability, these

empirical matters are not in the least bit trivial, and a large sector of HCI is devoted to a further understanding of these factors. There is a conception of interface complexity, however, that does not require empirical study, rather it is suited to analytic inquiry. It is our hope that we have at least begun the project of understanding what methodology might permit such an inquiry. |

Acknowledgments:

This work benefited from the inspiration and enthusiastic tutelage of Alfred Z. Spector (IBM, VP Services and Software), and would not have been possible without the generosity of IBM's TJ Watson Research Center. I am indebted to Wendy Kellogg, Brent Halpern, Tom Moran, Tracee Wolf, and Martin Wattenberg among many others at IBM research for their energy, conversations, valuable insights, and feedback. I also thank Bill White, for without his assistance I surely would have been an academic casualty of IBM's logistics and bureaucracy.

Bibliography

Balbo, S. *Automatic evaluation of User Interface usability: Dream or Reality.* Proceedings of the Queensland Computer-Human Interaction Symposium (1995).

Bhat, J. Kumar, A. *Measuring and Improving Process Capabilities - Best Practices*. Infosys.

Bhavnani, S.K., Flemming, U. Forsythe, D.E., Garrett, J.H., Shaw, D.S., Tsai, A.. *CAD Usage in architectural office: from observations to active assistance*. Automation in Construction. 5 243-255 (1996).

Bhavnani, S.K., John, B.E.. *The Stategic Use of Complex Computer Systems*. Human-Computer Interaction, Vol. 15, 107-137 (2000).

Card, S., Moran, T., Newell, A. The Psychology of Human-Computer Interaction. Hillsdale, NY: Lawrence Erlbaum Associates, Inc (1983).

Church, K and Braake, G. *The Future of Software Development*. IRMA International Conference, (2001).

Curtis, B., Carleton, A. *Seven +- Two Software Conundrums*.

Flemming, U., Bhavnani, S.K., John, B.E.. *Mismatched Metaphor: user vs system model in computer-aided drafting.* Design Studies 18 (1997) 349-368.

Ivory, M., Hearst, M. *The State of the Art in Automating Usability Evaluation of User Interfaces*. p. 470-516. ACM Computing Surveys, Vol. 33, No. 4, (2001).

Kellogg, W. A., Breen T.J.. *Evaluating User and System Models: Applying Scaling Techniques to Problems in Human-Computer Interaction*. HCI + GI (1987).

Kierias, D. *A Guide to GOMS Model Usability Evaluation using GOMSL and GLEAN3*. (1999).

Kieras, D. *A Guide to GOMS Model Usability Evaluation using NGOMSL*. In M. Helander and T. Landauer (eds.) The handbook of human computer interaction (2nd ed.). Amsterdam: North Holland (1997).

Kieras, D. *Using Keystroke-Level Model to Estimate Execution Times*. (1993).

Kieras, D. *An Approach to the formal analysis of user complexity*. International Journal of Man-Machine Studies 22, 4, 365-394 (1985).

Langton, C., Taylor, C., Farmer, J., Rasmussen, S. (ed). Artificial Life II. Addison-Wesley (1992)..

Leveson, N. G. (1997). Software Engineering: Stretching the Limits of Complexity. Communications of the ACM, Feb. Vol. 40, pp. 129-131 (1997).

Nicolis, G. and Prigogine, I. Exploring Complexity, An Introduction. R. Piper GmbH and Co (1989).

Parush, A., Nadir R., and Shtub, A. *Evaluating the layer of graphical user interface screens: Validation of a numerical computerized model*. International Journal of Human Computer Interaction, 10; 4. 343-360 (1998).

Polk, T. A.& Rosenbloom, P. S. *Task-independent Constraints on a Unified Theory of Cognition.* Handbook of Neuropsychology, Vol. 9 (1994).

Rauterberg, M. *How to measure and to quantify usability of user interfaces*. In A Ozok and G. Salvendy, Eds. Advances in Applied Ergonomics, 00 429-432, West Lafayette Publishing (1996).

Rauterberg, M and Aeppili, R. *Learning man-machine systems: the measurement of behavioral and cognitive complexity*. In Proceedings of the IEEE Conference on Systems, Man and Cybernetics (Vancouver, BC, Oct) pp 4686-4690 (1995).

Sears, A. *AIDE: A step towards metric-based interface development tools. In Proceedings of the eighth ACM Symposium on User Interface Software and Technology* (Puttsburgh, PA, Nov), pp. 101-110. New York, NY: ACM Press (1995).

Zettlemoyer, L. S., St. Amant, R. S., & Dulberg, M. S. *Agent control through the user interface*. In Proceedings of the International Conference on Intelligent User Interfaces (Redondo Beach, CA, Jan.) P. 31-37. New York, NY: ACM press (1999).