

The 'Inevitable' Rise of the Administrators
A Sociological Study on Control in ICT



Thesis submitted for the MA degree in Sociology
by
André van Cleeff

Supervisor: Dr. Hans Pruijt
Erasmus University Rotterdam
May 2006

Abstract

The research subject in this paper is the control structure inside of authoring and publishing systems. More precise, it is examined how systems such as blogs, forums and content management systems regulate what people can and cannot do with them. For example: How is it determined that someone can publish an article on the front page of a website? Is there any censorship? Does someone have to approve of the text that another person wrote? Who determines that approval must take place?

The subject of control in ICT is of great importance; the way control is modeled in authoring and publishing systems influences (or is influenced by) issues such as freedom of speech, copyright protection, non-proliferation and censorship. In fact, the Internet itself is one huge authoring and publishing system. As ICT itself proliferates, it can be expected that the issue of control (and security) will dominate the debate on ICT in the next decades.

Although ICT is for a large part a social construction, and thus worthy of examination by sociologists, most sociological research on this subject does not focus on ICT itself but on its context. Organizations are studied, not software. The result is a gap in our understanding of ICT. In this thesis, an attempt is made to close this gap by studying ICT itself, apart from its context. The research indicates that, although there are potentially many ways to structure control, most control structures share common characteristics:

- Control is hierarchical;
- Situated at the top of the hierarchy, there is an all-powerful administrator who gives authorizations to others;
- Within a system, people can *cooperate*, but the cannot *act* jointly. No two persons can ever click 'OK' together.

Combined, it can be concluded that individuals (as opposed to groups) exercise control inside ICT, in hierarchically structured networks, with administrators having full control. This is the cause for authoring and publishing systems, but the results can easily be generalized to other systems.

With the rise of the Internet, the importance of good computer security has arisen: Ill-protected systems are prone to viruses, worms and other malicious programs that loom on computer networks. Sensitive data stored on computers should not fall into the wrong hands. A lot of effort has been put in patching systems and developing tools as virus scanners. Yet this does not solve what appears to be the largest security hole *by design*: the lack of any checks and balances inside ICT. No system is protected from ill will of an administrator. Few, if any, checks and balances are *inside of* ICT. As ICT proliferates, more and more power is put in the hands of individual administrators.

In how far is this an undesirable situation? A counterargument is that many other technologies do not contain check & balances. A surgical knife (no matter how well designed) does protect neither the patient nor the doctor from fatal accidents. A car does not protect people from being overrun. However, ICT differs in three ways:

- The potential amount of damage is much bigger. A knife can injure only one patient at a time in one place. An administrator with too much power can disable the ICT infrastructure of an entire hospital within seconds, putting many more lives at stake.
- ICT has autonomous capabilities. A knife doesn't injure by itself, but a computer virus spreads itself in seconds across the globe, compromising many administrator accounts. The Internet has thus short-circuited the checks and balances that exist in the physical and social world.
- Ironically, ICT *does* contain a social structure. Many applications require people to identify themselves before gaining access; they can only do what the system allows them to. Knives lack such features. If we think these structures in ICT are necessary, it seems illogical to have all-powerful administrators who can negate all security measures.

What has caused this problem to arise?

A first observation is that safeguards against abuse are currently *outside* of ICT, in the embedding of administrators within organizations, their education and indoctrination. These complement the minimal safeguards inside of ICT.

A technological explanation is the way that ICT itself is structured, consisting of hardware, programs and running applications. A program must be installed on a piece of hardware to become a running application; only an administrator can do this. He then grants the authorizations to other people. There is thus an inherent tendency to centralize. This is an indication of technological determinism: It is not organizations that choose a particular configuration; rather their options seem limited by the technology itself.

Another explanation is evolutionary: Because early systems had administrators, no one ever saw the need to change this. It was easiest to copy and reuse the existing authorization structures.

How can we change this situation? A possibility for existing systems is to split them up into smaller ones: this potentially limits the power of a single administrator. Future systems could be equipped with more checks and balances inside. An especially interesting idea would be to implement group actions, actions that can only be performed by a group of people instead of a single user. A more radical idea is to embed democratic or juridical procedures in software. For example, virtual elections can be held in which users vote for their administrators, who must be re-elected after a certain period (and might be recalled earlier). Time and space could also be reintroduced, an application would be spread over multiple locations and actions could take a week to complete instead of just milliseconds.

If implemented at a sufficiently low level, such measures would also reduce the dangers of existing security flaws. In this scenario, a malicious program could possibly take over one user's account, or one location – only to find that it needed someone else's cooperation to gain full access, and wait a whole day before it could do any real damage.

Table of contents

Abstract.....	2
Table of contents.....	4
Preface.....	7
1. Introduction	8
1.1 Introduction	8
1.2 Motivation	8
1.3 Aims.....	10
1.4 Outline.....	10
Part I: Theoretical Framework.....	11
2. ICT Research in the Social Sciences	12
2.1 Introduction	12
2.2 Research orientations.....	12
2.3 Meta study on power and information technology research.....	12
2.4 Conclusion	14
3. Social Aspects of ICT.....	15
3.1 Introduction	15
3.2 The rationality of computer science	15
3.3 The relation between ICT and the social world.....	16
3.4 Conclusions	18
4. Technology	19
4.1 Introduction	19
4.2 Society's evolution and it's causes	19
4.3 Technology?	19
4.4 Technology model.....	20
4.5 Technology as enabling and constraining	21
4.6 Conclusion	21
5. ICT	22
5.1 Introduction	22
5.2 What is ICT?	22
5.3 ICT model	22
5.4 Conclusion	25
6. Research Approach	26
6.1 Introduction	26
6.2 Research requirements.....	26
6.3 Sociological elements: Enabling and constraining	27
6.4 ICT: Authoring and publishing systems.....	27
6.5 Conclusion	28
7. Control (Enabling and Constraining).....	29
7.1 Power and control in sociology.....	29
7.2 Identifying power and control in ICT	29
7.3 Identifying control elements in ICT	30
7.4 Implementing control in ICT	30
7.5 The feature - user matrix	34
7.6 Value of enabling and constraining in ICT.....	36
7.7 Conclusion.....	37
8. Authoring and Publishing Systems.....	38
8.1 Introduction	38

8.2	Definition	38
8.3	Identifying hierarchies and control in the publishing process	38
8.4	The Internet as one huge authoring and publishing system	40
8.5	Conclusion	42
Part II: Empirical Research		43
9.	Research Questions	44
9.1	Four research questions	44
9.2	Initial hypothesis	44
10.	Sampling	46
10.1	Introduction	46
10.2	Sampling	46
10.3	Overview of cases	48
10.4	Examination process	51
10.5	Conclusion	51
11.	Structure	52
11.1	Introduction	52
11.2	Control structure	52
11.3	Workflow	53
11.4	Content structure	53
11.5	Initiation / Installation procedure	54
11.6	Organizational choice?	56
11.7	Conclusion	57
12.	Effects	59
12.1	Introduction	59
12.2	Security	59
12.3	Conclusion	59
13.	Causes	60
13.1	Introduction	60
13.2	Social explanations	60
13.3	Technological explanations (including physical and conceptual)	62
13.4	Evolutionary explanations	63
13.5	Conclusion	65
14.	Alternatives	66
14.1	Introduction	66
14.2	Cooperation	66
14.3	Splitting up	67
14.4	Audit trail	68
14.5	Conclusion	68
Part III: Conclusions		69
15.	Discussion: Rise of the Administrators?	70
15.1	Finding a suitable name for the ICT revolution	70
15.2	Three views on the administrator	70
16.	Conclusion	72
	Reflections on the research approach	72
	Key insights	72
17.	Recommendations	73
17.1	Introduction	73
17.2	Methodology	73

17.3	Subjects.....	73
18.	References	74
18.1	Printed publications.....	74
18.2	Online publications	76
18.3	Interviews & conversations.....	77
18.4	Software.....	77
Appendix A: Tables.....		78
Appendix B: Figures.....		79
Appendix C: Database modeling.....		80
	Database diagramming techniques	80
Appendix D: Structure		81
	Control structure.....	81
	Workflow	84
	Content structure	86
Appendix E: Causes		92
E.1	Design of Silva	92
E.2	Design of Talmon CMS.....	93
E.3	Design of EcoGrid	93
E.4	Usage of Silva at Erasmus University, Faculty of Social Sciences.....	96
E.5	Usage of Open Market CMS at NRC.....	97
E.6	Usage of phpBB at HeliportOnLine.....	98

Preface

Just after my graduation in computer science (Leiden University, 2001) I was overcome by the feeling that somehow I was not finished studying. But instead of pursuing a PhD in computer science I decided it was time for something completely different. Thus, I took up sociology in September 2001 at Erasmus University Rotterdam, while working full-time as a software engineer for an IT company. (A 'dotcom' or 'startup' as it was known at the time).

Already in 2001 it became clear that these were indeed totally different worlds, with completely different views on software. Generalizing, computer science is about theoretical modeling of software; amongst other things, sociology is about understanding the role of software in society; software engineering is about creating usable software in a neat, efficient and controllable way. Few ideas managed to get across the boundaries of these disciplines; there obviously was an opportunity to improve this situation.

One of the most interesting problems I had to tackle as a software engineer had to do with authorizations: Who is allowed to do what? This is relatively simple if you have an application that only has users, but it becomes difficult when you deal with people who are employed by different organizations, with all the connections that exist between them. For example: Is an employee E working for company C allowed to place an order O for product P at company D with a 10% discount? Is he even allowed to see that such a discount is given?

A solution to this problem also has to include assigning authorizations: At some point in time, someone has to grant employee E the right to perform this task; unwanted persons have to be kept out. From an engineering point view this is about security: how to build software that unauthorized persons can never gain access to; or detect that this has happened; In sociological terms it is the issue of control or more broadly power.

It leaves little doubt that this problem had not been addressed properly; As a software engineer I could not find any existing solution for my authorization problems and had to build my own; In fact, it turned out that even for the simplest cases there were no off-the-shelf solutions. From a computer scientific point there was (and to my knowledge still is) no such a thing as adequately secure software; the stream of news items about viruses, worms and patches continues. From a sociological perspective there seemed to be little meaningful sociological research on this subject. All in all, I got the idea that it was an interesting research subject.

Although multidisciplinary in nature, the orientation in this paper is still sociological: I investigate power and control in software from a sociological perspective, using my knowledge of computer science and software engineering. I have tried to limit the usage of terminology and applied them only where it served clarity. Some definitions, expressions and diagrams could be considered somewhat informal; such is the price you have to pay for bridging disciplines while keeping a paper readable. I had no intention to mystify anything. Most of the technical content is located in the appendices.

Finally, I would like to point out that this paper – like so many others - is indirectly the product of many people. I hereby thank them for their contributions.

André van Cleeff
Rotterdam, May 2006

1. Introduction

1.1 Introduction

In this thesis it is examined how people have structured control inside ICT. Control means here the ability to restrain (or allow) people to perform certain actions. As such there is a clear link with the sociological concept of power in the sense that someone who controls also has power. For now, ICT is simply defined as information and communication technology, comprising of such things as computers and networks, software and running applications. Chapter 5 contains a more precise description.

The methodology differs from normal approaches: Most sociological research is done from the contextual perspective of organizations that *use* software; here software *itself* is studied, before looking at the context. The motivation for this thesis is discussed in this first chapter –in terms of the subject, the research approach and the relevance.

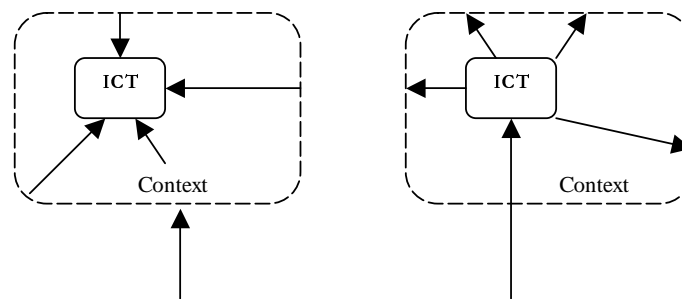


Figure 1-1: Two approaches for the study of ICT

1.2 Motivation

The state of existing sociological research on ICT

Several years ago, Orlikowski and Iacono stated that information system researchers - among whom many sociologists - have not dived too deeply into their core subject, namely ICT (Orlikowski & Iacono, 2001). Instead, they focused on ICT's context. As will be shown in Chapter 2, this is probably still true today. Given this situation, should ICT be examined at a closer level? Positively formulated, there are several reasons why this question can be answered with 'yes':

1. The simplest reason is that ICT itself is a social construction: behind the façade of hard technology it is in fact quite soft and malleable, open to influences from society. Therefore a sociological view on ICT has the potency to deliver more valuable insights than a pure technical view ever could. ICT research should not be the sole domain of computer scientists.
2. It also follows that the validity of existing information system research can be called into question: The research likely missed important sociological elements in ICT. Obviously there is a great opportunity to improve the validity of information system research. But such a view can only be developed if one does not evade the subject.
3. ICT provides an easy research medium to study society. With some technological expertise you can easily investigate the social world (or its digital equivalent). Because everything inside ICT is measurable, it is easy to gather hard data that is difficult to get elsewhere; it should not be necessary to bother research subjects with interviews or surveys. In other words: ICT research is non-obtrusive.
4. ICT proliferates more and more: An increasing part of our social life revolves around ICT. In fact, ICT research increasingly becomes *social* research per se: Or put more formally: its external validity keeps increasing.

The timeless subject of control

Now that it has been concluded that ICT itself (and not its context) should be the focus of the research, what should be the subject? Clearly, it should be a subject that can be studied outside of its context. We should not study how a software program was used in such and such an organization, at least not in the first place. Ideally, it should be a timeless subject that also has a clear link to sociology. In this respect control is a good subject: The aspect of control is always present, it can be studied apart from its context and it is linked to the sociological concept of power.

The increasing importance of control

In fact, concerning the sociological relevance, it can be expected that as ICT proliferates the issue of control will dominate the debate on ICT the next decades. A few decades ago the key threat to global security was a nuclear conflict between the United States and the Soviet Union. This particular threat is no more, but the proliferation of weapons of mass destruction is still a major concern. Worse, 9/11 has made it clear that non-nuclear weapons can also be very effective, and these turn out to be very hard to counter. In all this, the Internet plays a key role. First, the rise of the Internet created a whole new category of vulnerabilities for society, computer viruses can potentially wreak havoc to any organization that is connected. Secondly, the Internet is an excellent platform for the proliferation of potentially dangerous information. For example, the genetic sequence of the 1918 influenza virus (that killed 50 million people at the time) was published on the Internet in the GenBank database (Kurzweil & Joy, 2006). Knowing the genetic sequence, people can relatively easily recreate the lethal virus. Worse, as science progresses the amount of information that can be misused increases likewise. Dismantling existing atomic bombs is hard enough, but the removal of dangerous information from the public domain is totally infeasible. Once someone puts a text on the Internet, which is picked up by other users within seconds, it becomes impossible to remove. There is no cure, only prevention, hence the need for complete control of the publishing process will arise sooner or later.

In fact, the Internet is also an opportunity for surveillance as has never existed before. Even for the East German Stasi it used to be impossible to keep track of everybody's whereabouts, but such bookkeeping is feasible today. One need only to look at the wiretapping power of agencies such as United States' NSA or the effort the Chinese government is putting in the creating of the 'Great Firewall of China'. These are all examples where some form of control over content is exercised.

The question of how the Internet is managed, i.e. the question of power, how we deal with issues of copyright protection, freedom of speech, proliferation of potentially dangerous information and censorship is fundamental.

Control in authoring and publishing systems

In what systems should control best be examined? In the last section we have seen that present and future issues revolve around the ability to create, publish and read texts. Thus, it would be logical to study systems that provide such functionalities. In this paper, these systems are called authoring and publishing systems.¹

This category includes programs such as content management systems and discussion forums. There are several additional advantages to studying such systems: There is a great variety between such systems, from applications in use by individuals (weblogs) to large corporations (content management systems). They also have a relatively long history. The World Wide Web, of which they are a part, has existed for over 10 years. This makes it possible to study their evolution. Last but not least, they are relatively simple systems: It doesn't take much additional expertise; many people use text editors, search the Internet for information and publish their adventures on weblogs.

¹ This term was derived from M. Faassen's description of the program 'Silva', an 'authoring and publication system' (Faassen, 2005). The term accurately captures a whole range of systems that are usually not seen as belonging together.

1.3 Aims

This thesis tries to investigate ICT itself (and not just its context) from a sociological perspective. As such, the goal for this thesis is to prove that such an approach

- Is feasible and sometimes even preferable over other approaches
- Can lead to new insights that are not easily obtained by other means

The particular point of interest is control or more broadly power (as one of the most important issues in sociology) and how it is channeled in ICT.

Hopefully it will demonstrate to sociologists that

- ICT is for a large part a social construction, not a purely technical artifact and thus worthy of examination;
- It is possible to gather relevant sociological data simply by studying ICT itself;
- Therefore, there is no compelling need to study ICT from a pure process view (development, implementation, usage);
- The only requirements for ICT research are simple (but powerful) theories combined with some technological expertise.

As for computer scientists and software engineers it could help them

- To realize how they reconstruct the social world and that therefore, their responsibility is not limited to creating models or “getting things to work” with the tools at hand;
- To realize that in fact, their toolkits are also social constructions, and question the validity and usefulness of those toolkits and not take them for granted.
- To create software that is more secure

Or to put it simpler: The aim of this thesis is to create “a sociology of control” for ICT, useful in various disciplines.

1.4 Outline

The paper consists of three parts:

Part 1: Theory

In this part some recent sociological research on ICT is examined, to see where it falls short. Next a small theoretical framework is constructed to guide the research. It also argues why *control* is such a promising research field. (This is an elaboration of the ‘motivation’ section of this chapter.)

Part 2: Empirical Research

Here the lessons from the first part are applied and tested in practice. It also contributes to the understanding of ICT itself.

Part 3: Conclusions

In the last part conclusions will be formulated

- About how ICT related research should be done in general
- About the nature of ICT, its causes and effects

The paper ends with some general implications.

Part I: Theoretical Framework

2. ICT Research in the Social Sciences

2.1 Introduction

Originally, work on this thesis started with a literature study on software² and power. As it turned out, the scope of the subject was potentially huge. Software in relation to power can be a research subject in various disciplines, ranging from sociology, information sciences, STS (science, technology and society) studies and business administration, just to name a few. Remarkably, no article was found that actually examined *software* in relation to power; Many research articles featured only small glimpses of the technology that they were examining. There seemed to be a gap in the research. Of course a high level view can be very useful. It should not be necessary to understand ICT from the first to the last bit. But such a view has drawbacks; important details can be obscured; it would at least be prudent to mention why a certain level of abstraction was chosen.

It would be very interesting to research the way social scientists research technology itself. However, that is not the main goal here: The aim of this chapter is to illustrate how easy it is to perform an elaborate study on technology and still miss important aspects of it. From this we can draw conclusions about how it can be done better. But first, we differ between several types of research questions.

2.2 Research orientations

Auguste Comte, one of the founding fathers of the sociological discipline, divided sociology in two parts: statics (on structures) and dynamics (on processes) (De Jong, 1997). In his thinking Comte was influenced by successes of the scientific method in other disciplines such as mathematics and physics; For example, the movements (dynamics) of the planets (statics) could be predicted with great accuracy. Although the scientific method was never as successful in sociology as it was for physics, the distinction remains useful. When seeing society as a dynamic structure it follows that we can ask two types of questions:

- Process oriented questions such as, “How does an organization evolve over time?”
- Structural questions such as “What is the structure of an organization?”

In reality these questions are often combined; For example the book ‘Structures in Fives’ (Mintzberg, 1992), on designing effective organization structures, does not only describe organization structures but also explains how an organization changes its shape. This doesn’t mean that there is not a logical order in which the research should be done: In general it’s hard to examine change if you do not know what it is that’s changing. In quantitative research, structural questions logically predate process questions. When someone creates a conceptual model, where different concepts influence each other, the first step is to define those concepts (Neumann, 2000).

2.3 Meta study on power and information technology research

An exemplary study on software and power was done by Jaspersen e.a. (2002). The article (Review: Power and Information Technology Research: A Metatriangulation Review) is a meta study of 82 other research articles that deal with power and information technology. In the introduction the authors conclude that

² Chapter 5 contains more precise descriptions of ICT and software.

"In general, MIS (Management Information Systems, AvC) researchers have focused on three broad study topics:

- (1) Impact of IT³,*
- (2) Development, deployment, and use of IT, and*
- (3) Organization and management of IT resources including centralized / decentralized governance structures (Orlikowski and Barley 2001).*

We incorporate these three topical areas by examining research that investigates IT Impacts, Deployment or Development, Management or Use (ITIDMU)."

This might seem an adequate and broad description of IT issues. But on closer examination one could notice that something very fundamental is missing: What it says is that basically, IT itself never was the study of research, only it's impact, development, deployment, use, organization and management was considered. (Indeed, a managerial perspective, but a vision that is most definitely not limited to managers.) Next, Jaspersen e.a. state:

"Our purpose is to

- (1) Explore the paradigms that have been used to investigate power and ITIDMU,*
- (2) Describe patterns that have emerged in previous research on power and ITIDMU, and*
- (3) Use the disparity and complementarity across paradigms to develop metaconjectures about the relationship between power and the specific constructs within ITIDMU. "*

Again, not power and IT is researched, but power and ITIDMU.

Subsequently, the authors move on to the description of the 'lenses' through which the research is being done, or more simply put, points of view. Two pairs of lenses are used, the first deals with technology, the second with power. The lenses are divided between technological, organizational and emergent lenses. All three deal with the causal relations between technology and organizations:

Lens	Orientation/Viewpoint
Technological	The impact of technology in an organization. Is there evidence of technological determinism? ⁴
Organizational	How an organization chooses the technology it uses
Emergent	Usage of technology as a process determined by both organizations and the technology itself

Table 2-1: Research lenses

Again, to the inattentive reader this is as an adequate set of lenses. In general, given two topics IT and O we could examine whether any one of

IT → O

IT ← O

IT ↔ O

holds true and in which conditions.

But regardless of the relation one assumes, IT is never studied. No one uses IT as his or her research unit. The articles examined in the meta study do *not* study IT and neither does the meta review.

What is actually ICT, is it the computers, the software, the IT department, the Internet and the people who are using it? We cannot say. If one is to understand anything about IT, then it must at some point be the focus of research itself.

³ IT can be considered equivalent to ICT.

⁴ See section 4.2 for more information on technological determinism

Another problem is that research that doesn't examine IT itself has the built-in tendency to focus on larger organizations. These have naturally

- More software
- A larger IT budget (possibly spent on social research!)
- More IT related problems (having to integrate all these software packages)

The Jaspersen paper is again a good example. Although this article is intended as a meta study and should offer a wider view on power and information technology it still suffers from what could be called the "large organization and management perspective" bias. People at home use software, as well as do small business and sport clubs. From this viewpoint, many studies are lacking.

Some further research seemed to confirm that IT is seldom the core subject of research, see for example (Orlikowski & Iacono, 2001):

"The field of information systems is premised on the centrality of information technology in everyday socio-economic life. Yet, drawing on a review of the full set of articles published in Information Systems Research (ISR) over the past ten years, we argue that the field has not deeply engaged its core subject matter--the information technology (IT) artifact. Instead, we find that IS researchers tend to give central theoretical significance to the context (within which some usually unspecified technology is seen to operate), the discrete processing capabilities of the artifact (as separable from its context or use), or the dependent variable (that which is posited to be affected or changed as technology is developed, implemented, and used). The IT artifact itself tends to disappear from view, be taken for granted, or is presumed to be unproblematic once it is built and installed."

Orlikowski & Iacono provide the following solution to counter this problem:

We believe that to understand these implications (intended and unintended, for individuals, groups, organizations and society - AvC) we must theorize about the meanings, capabilities and use of IT artifacts, their multiple, emergent and dynamic properties, as well as the recursive transformations occurring in the various social worlds in which they are embedded. We believe that the lack of theories about IT artifacts, the ways in which they emerge and evolve over time, and how they become interdependent with socio-economic context and practices, are key unresolved issues for our field and ones that will become even more problematic in these dynamic and innovative times...

We will differ from this view, because it can be read as yet another call to evade IT: It is not the IT artifact that needs to be studied; it must be *theorized*; With this in mind we can sum up the conclusions.

2.4 Conclusion

There are several pitfalls to avoid when researching technology from a social perspective. In general researchers have a tendency to focus on context, rather than ICT itself and to research processes rather than structures. The research unit is almost never ICT itself and large organizations are favored. This does not lead to an adequate understanding of ICT. It follows that this situation can be improved if we examine the structure of ICT itself. Only then can we study its effects. A small theoretical framework can be helpful in the research, but it should be noted that the most progress is possible only if we focus on ICT rather than on theory.

3. Social Aspects of ICT

3.1 Introduction

In Chapter 2 we saw there is a tendency in sociological IT research to focus on contextual processes rather than IT structures. Thus, social research seems to favor ‘soft facts’ over ‘hard facts’.

In this chapter we look in the opposite direction: It is argued that computer science and other related disciplines (such as software engineering or information sciences) focus on hard facts rather than soft facts; However, such a view isn’t necessary better than the social view: Many ‘hard’ disciplines are not purely exact sciences. For some part they are – as many other disciplines – social constructions, although their constructedness is hidden behind a façade of rationality. The same holds true for the application of their knowledge in software. It follows that social scientists researching ICT must also study software in detail: It is not a pure technological artifact.

3.2 The rationality of computer science

Earlier in section 2.2 we saw that Comte envisioned sociology as a real science, like physics or mathematics, but that this idea was never fulfilled. We can ask the same question about computer science and related studies: in how far are they in line with positivist ideas of science, in how far are they rational?

There are several arguments to support this hypothesis:

- 1 Computer science is applied science, basically an extension of mathematics
- 2 Computer science is about performance, efficiency, improving algorithms, all of which are – to some extent - quantifiable and testable.
- 3 Software creation is nowadays a solid, well-defined process resulting in standard applications that fit into a neat multi-tiered architecture.

These will be addressed in the next sections.

Argument 1: mathematical basis

Unlike sociology, computer science has a solid mathematical basis in the form of the Church-Turing Thesis, developed in 1936 by Alonzo Church and Alan Turing. (Wood, 1987) It claims that a Turing Machine, a theoretical type of computer, can carry out every effective computation. Modern computers are equivalent to Turing machines in the sense that every computer can simulate any Turing Machine and each Turing Machine can simulate any other computer.

This equivalence has some far running implications:

- All algorithms can (at least theoretically) run on any type of computer and we need not concern ourselves with any particular type of system or structure. Of course, some computers are faster than others, or have more storage capacity than others, but these differences are not fundamental.
- A second consequence of the Church-Turing Thesis is that connecting two computers in a network does not really generate a new type of computing – so much for the ‘real’ Internet revolution.
- Another consequence is that the difference that is usually made between hardware (the physical items that make up a computer) and the software (the bits and bytes representing the data and programs) is blurred. We can use a computer from the 1970s to run a program created on brand new computer – and vice versa. A whole range of software called *emulators* or *virtual machines* is available to simulate old hardware. Hardware is in this sense very *soft* and software can be as hard as hardware.

To summarize: this mathematical basis is so solid that on top of it everything else could be considered a human construction.

Argument 2: performance and efficiency

A refutation of the first argument could be that some ways of computing are more efficient than others. The Turing machine has no predefined speed or storage capacity but these *do* matter in real life. If we want to sort a collection of numbers there are various algorithms to do so⁵. Logically those that are faster and more efficient should be more widely used. To some extent this is true – but for most applications, the immense increase in computing power over the last decades have put less emphasis on this type of efficiency. Other goals have become more important, such as maintainability and security. For example: Compare the creation of a database with the creation of a hotel booking system. The first can be examined in a strict manner: how fast is it, how efficient? For the latter one has to take into account the various practices of hotels and travel agencies. These practices can sometimes be rational, but sometimes less so.

Argument 3: software development process

The last argument can be rejected on the grounds that in many cases, software development starts with the user requirements⁶: These originate in the social world and thus most software development is driven by needs in the social world.

General counterargument: evolution

A final counterargument is that software itself evolves and this needn't be a purely rational process – or outcome. Computer programs generally do not come out of thin air. More precisely, people create software; they have been building millions of programs in the last decades. Software is built using existing programs and tools: It is impossible to start from scratch. As an illustration– below are two graphical representations of a software program at two particular points in time (Brisset, 2001).

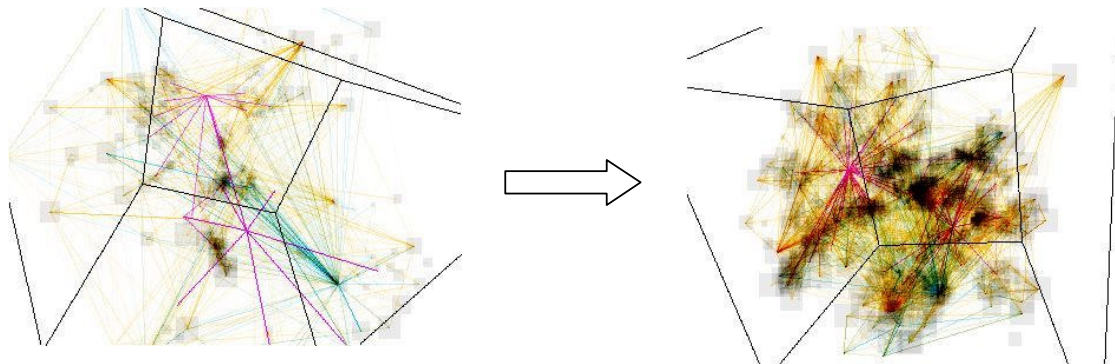


Figure 3-1: Graphical representation of the changing structure of a software program

These structures show the relations between parts of the program. The exact meaning of the lines and dots in the pictures will not be explained here, but hopefully the pictures convey the fact that software programs can be very complicated structures that evolve. A new version is created with the previous version as a point of departure. If the software is sufficiently complex, it is likely that most parts do not change; Change takes time, involves the risk of making errors and if a part is deemed adequate there is no need to change it. Thus software evolves slowly and programs are built in dependence of each other. How much rationality is there after so many decades of evolution?

3.3 The relation between ICT and the social world

In fact, there are very good reasons to examine software from a sociological perspective. Software has been evolving for several decades now – and we can do research about what sort of

⁵ Including bubblesort, quicksort, heapsort to name a few – the exact working is not relevant here

⁶ For more on the software development process, see Section 5.3

structure has been created during that time. Each piece of software could be seen as a sort of ontology, which results in more (and digitalized) knowledge about society, or more simply, how things work. As any social construction, this ontology is partly self fulfilling - *using software is to some extent accepting it's ontological points of departure*. No doubt some things are 'real', even before the age of computers there were employees and taxes, but others are less so. Along with computers new things as 'email' and 'database' and 'user' arrived. But for anyone who builds or uses software these days, users and email are very much real items. Thus the virtual becomes real, or "real virtuality" emerges. (Castells, 2000) People try to capture reality and depict ('map') it onto software that in turn becomes part of reality itself.

How much do ICT and the social world differ from each other? As said before, ICT transforms physical processes to digital ones. Arguably these do not correspond completely.

Let's first look at the similarities: Inside algorithms some form of information or knowledge⁷ is encoded, possibly related to the social world. This can also be seen as a social process: Every software program has a creator(s), who holds certain ideas about society and how it works or should function. The program is constructed to perform certain tasks in that same society. During this construction, a bit of the creator's knowledge about the outside world slips into the program: it is required to let the program successfully complete its task. For example, a payroll system must contain some knowledge about employees and tax regulations. There must be some similarity between social and digital processes: You cannot create a program for filing you tax form without any knowledge about laws and regulations. A spell checker has to contain knowledge of the user's language. In other words: The program is a (limited) model of the real world.

This doesn't mean though that there is a need for an exact 1:1 correspondence between physical (social) and digital (social) processes. If there were an exact match, why would someone build something (spend time and money) to recreate something that already exists? It would be more logical to try to optimize existing physical processes, make them more efficient, better manageable or simply develop something that is totally impossible in the normal physical world.

Secondly it is likely that the properties of ICT itself tend to change physical process. For example processes are always speed up, simply because the possibility is there. Slow social processes thus could become faster— even if this is not beneficial.

ICT is thus located between two extremes:

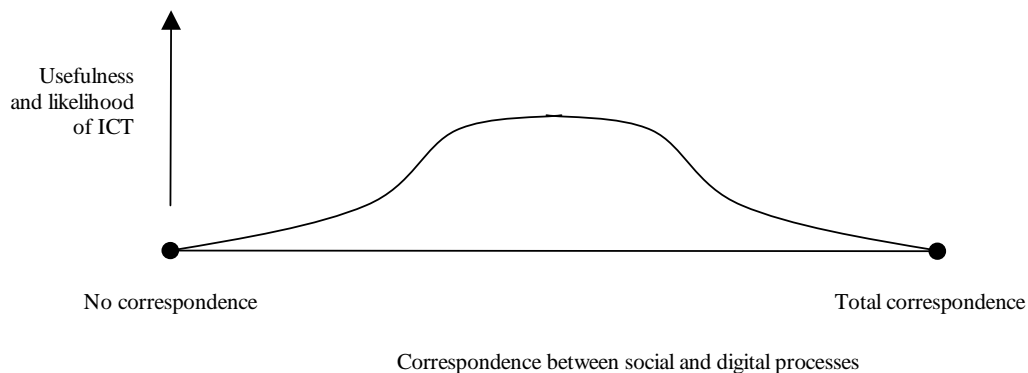


Figure 3-2: Correspondence between social and digital processes

⁷ The exact differences between data, information and knowledge are not deemed important here

3.4 Conclusions

We have seen that ICT is not just a technical artifact but contains a lot of sociological aspects. In fact, on top of a solid mathematical basis everything could be considered a social construction. Software evolves out of existing software and changes are weighed for their usefulness in society. Thus it becomes a relevant research subject for social scientists.

4. Technology

4.1 Introduction

In the previous chapters, we concluded that ICT is seldom the subject of social sciences' research, but that it should be, because there are many sociological aspects in ICT.

We can now try to avoid the pitfalls and build a small theoretical framework for the study of technology (and later especially for ICT). It is important to focus on structure and not only on process; this cannot be done without a proper understanding of the concept of technology itself, of which ICT (information and communication technology) is a part. This chapter examines the concept of technology and its relation with society. But first we start with some views on society's evolution.

4.2 Society's evolution and it's causes

Obviously, society has changed enormously in the last 10.000 years; it is also clear that technological developments are an essential part of this process. Society has it's origins in the past, evolving from societies based on hunting and gathering to horticultural and pastoral, agrarian, industrial and finally into the post-industrial information based societies that millions of people live in today (Macionis & Plummer, 1997). But does that mean that technology *causes* changes in society? The three main sociological paradigms (functional, conflict and symbolic interaction) each look differently (and thus explain differently) to what drives these changes: Symbolic interaction sees *ideas* as the cause for society's change from a traditional to a rational one. For conflict theory it is the *technological and social process of economic production*, for functional theory it is an *expanding division of labor and the needs of society as a whole*.

Macionis & Plummer hold an intermediate position and see technology as essentially neutral artifacts that can be put to different uses. They refer to studies done by Gerhard Lenski and Jean Lenski, who have researched technological changes and call the focus of their work 'sociocultural evolution', the process of change that results from a society's gaining new information, particularly technology. According to them, technologies create *preconditions* for different ways of organizing society (this is somewhat between the 'emergent' and 'organizational' perspective we saw in Section 2.3), but it does not mean that technology *determines* society. The latter view is more commonly known as *technological determinism*. But what is technology, actually?

4.3 Technology?

The Oxford Dictionary of Sociology (1998) defines technology as

'A term used rather loosely in sociology, to mean either machines, equipment, and possibly the productive technique associated with them; or a type of social relationship dictated by the technical organization and mechanization of work.'

This definition doesn't exactly pinpoint technology. What it does very well though, is to illustrate the problem that sociologists have when trying to understand technology, and reason with it in terms of cause and effect.

Example: car technology

As an example, let's try to apply the definition to cars. Obviously cars are a form of technology, but so are the people that use them for transportation or the factories where they are constructed (the productive technique associated with cars).

Using the previous definition, a sentence such as "the use of car technology allowed people to migrate to suburbs" is problematic: The car technology is inseparable from the people that use them. If we try to eliminate people from the car technology (and end up with just cars) we must deal with the problem that the existence of cars alone cannot have any effect, they must be used by people; people must choose to use them. Does it mean that people ultimately determine society and not technology? A counterexample is that, living in suburbs with little

public transport; people have no choice but to use cars. Their lives are *determined* by the need for transportation, they are dependent on it. To make matters more complicated we could also point out that technology does not only *determine* but is possibly *determined* itself: The human need for transportation might have led to the development of the car technology. Which makes technology a human construction and not a hard physical thing. Oppositely, we could also see the physical properties of cars (the steel, the engines) as a determinant for car development. Since technological developments have been going on for thousands of years we can also see the development of society as an evolutionary process of the continuous interaction of technology (the narrow definition) with the people that use them.

Ideas and technology

So far, technology was seen as a physical artifact but also as something that is located in usage. Does this mean that something that is not (always) used and has no physical properties cannot be considered technology? For example, how should we view ideas?

Ideas as technology

Can ideas be a form of technology as well? Some authors concur with this, for example (Stern, 2004). In 'Terror in the name of God', a book on religious terrorism, the author reaches the conclusion that religion is a sort of technology, "making good people better and bad people worse". In this sense, Stern could be considered a proponent of the theory of technological determinism.

But unlike cars, ideas do not have physical properties. Oppositely it can be argued that technology is somehow related to ideas: without the concept of wheels, the idea of the combustion engine and the blueprint for the rear suspension there can be no cars. Ideas are not exactly technology, but they are surely an essential part of it.

Ideas as causes and effects of technology

We will now discuss the concept of ideas in relation to technology further. In Section 4.2 it was mentioned that the symbolic interaction paradigm sees ideas as the driving force behind change in society. If this is so then ideas also have an effect on technology. Can technology also determine ideas? It seems possible but we have to separate the idea of technological determinism with technological determinism itself. This is made clear by (Smith & Marx, 1994) People can believe that technology influences them – but that doesn't mean there' is technological determinism per se.⁸

Two extreme positions

It appears that there are two extremes positions for the definition of technology. If we use a narrow definition of technology than there is a significant difference between society and technology; If we use a broad definition, they are roughly the same, with the consequence that it's hard to break events down in causes and effects. Intuitively, technology should be somewhere between these extremes: It is neither a pure technical construction – nor is it inseparable from human and physical influences. In the next section we will try to create a model that reflects these properties.

4.4 Technology model

Precisely because it's very hard to pinpoint technology, maybe it's better to think of technology as inherently fractured, consisting of interacting parts. This leads to two models, one structural and one causal (process) model:

⁸ In their book "Does Technology Drive History?" the authors argue that from its earliest beginnings, the United States had a strong and positive *belief* in technology. Applying and inventing technology was supposed to have beneficial effects. Other good examples can be found in advertising. Most campaigns support some sort technological deterministic thought by emphasizing the beneficial effects of a product on its buyers.

Structural model of technology

Technological element	Description
Physical	The real technical parts
Conceptual	The ideas, the design, the blueprint
Social	The usage, the human interaction

Table 4-1: Structural model of technology

Causal model of technology

Next, we can identify five types of causes and effects.

Cause/Effect	Description
Physical	Physical causes and effects
Conceptual	Ideas as causes and effects
Social	Human (inter) action as cause and effect
Technological	The combined physical, conceptual and social causes and effects
Evolutionary	The causes and effects seen as a evolution

Table 4-2: Causal model of technology

Note that in these models an idea is not technology itself, but it can be a part of technology, influence technology, or be created because of technology.

4.5 Technology as enabling and constraining

Now that we have seen the difficulty of grasping the concept of technology, it's time to introduce two ever-present properties of technology: enabling and constraining (Metselaar, 2000). Technology makes things possible but limits at the same time. These properties are often two sides of the same coin: Cars make it possible to travel between home and work easily, but constrain the options of urban planners at the same time, because they must include parking facilities for each new block that is created. Note that the constraining is not limited to physical actions but can also be applied to thought processes: If you have a car you might never consider moving closer to work. These properties are thus an alternative way of looking at the effects of technology.

4.6 Conclusion

The concept of technology is difficult to grasp for sociologists, narrow and broad definitions can be used. In this paper technology is seen as a construct of three different parts: physical, conceptual and social.

Causes and effects of technology are divided in five:

- Physical, conceptual, social
- Technological: the combined causes/ effects of the physical, conceptual and social parts of technology
- Evolutionary: the causes/ effects of the continuous interaction between the parts over time

Technology always includes the causal effects of enabling and constraining: It allows you to do things, but limits you as well.

5. ICT

5.1 Introduction

In Chapter 3 we saw that ICT contains a lot of social elements; in Chapter 4 a model of technology was created that included social elements. We will now continue with a specific model for ICT. But first ICT is defined.

5.2 What is ICT?

Terminology

ICT is an acronym for information and communication technology: Basically, information technology allows complex calculations to be performed very quickly; Communication technology allows data to be exchanged very quickly. As such, ICT is an accelerator, which speeds up physical processes after they have been transformed into digital ones. During this transformation, processes are altered and thus digital processes differ somewhat from the physical ones. Other common terms used in conjunction with ICT are IT and software. These terms will not be used here, IT more or less equals ICT nowadays and 'software' is comparable to the term 'program'.

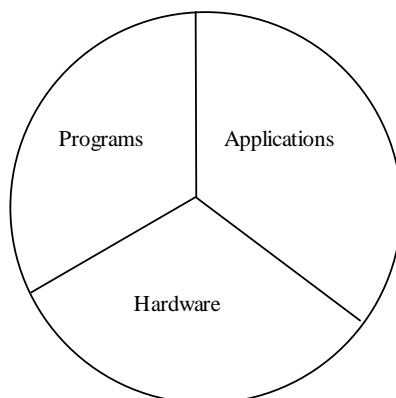
5.3 ICT model

How should ICT be modeled? There is no shortage of models in relation of ICT. In fact, modeling itself seems an important aspect of creating programs. Many types of models can be used, for example representing the inner workings of applications (class diagrams), the network structures, program components and their interaction (Booch, Rumbaugh, Jacobson, 1999) or the evolution of ICT in an organization (Nolan, 1979). Some of these models are quite complicated, possibly unnecessarily so. Thus, we will first enumerate the requirements for the model. This allows us to leave out the unnecessary parts. In fact there are just two requirements:

- The model should allow us to locate (parts of) ICT
- The model should allow us to locate ICT cause and effect relations

Structural models

Based on the general model of technology we can split the ICT into three distinct parts, namely programs, hardware and applications.



Part	Technological part
Hardware	Physical
Program	Conceptual
Applications	Social

Figure 5-1: Structural model of ICT

- Programs consist of algorithms, executable code, basically collections of bits and bytes residing on some sort of medium that specify calculations. A more common term is of course *software*.
- Hardware is the physical component that is capable of executing programs.

- Applications are programs running on hardware within a social context. That is, humans use them.

In its turn, programs consist of features, different parts that could be executed separately if the program was executing. A feature could be anything, for example an editor has a feature to create a new document or to print an existing document.

Program		
Feature A	Feature B	
Feature C	Feature D	Feature E
Feature F	Feature G	Feature H

Table 5-1: Schematic overview of a program consisting of different features

Why splitting applications into features is essential we be examined in the next section.

An additional benefit and justification of this model is that it corresponds to the way value is created in the IT industry: Organizations can make a profit selling hardware (such as processors, harddisks), software (operating systems, spreadsheets, games) and applications (software that runs online, such as search engines). The latter way of creating value is called the ASP model, (short for Application Service Provider), or more recently SaaS (Software as a Service). The application is a service that customers can use; yet the customer doesn't *own* the application itself; he is given a usage license for a particular period (a year) and/ or a transaction volume (storage capacity).

Software development and architecture

To be precise we must mention that the previous model is somewhat lacking: ICT is not simply a 'three part monolithic whole'. In any given setting there is never one program, one piece of hardware and one application. There are multiple pieces, each interacting with each other. This doesn't become a mess though, because most ICT is developed in hierarchical layers. Dividing ICT into distinct parts (modules, libraries) is an effective way to reduce (and conquer) the complexity. Separate parts that focus on special tasks are easy to replace, develop and test. A high level architecture is the so-called three-tier model (Fowler 1999):

Layer	Task	Typical Program
Presentation	User interaction	Webbrowser
Application ⁹	Logic, calculation	Webserver
Data source	Data storage	Database

Table 5-2: Logical application tiers

Each of these layers should only communicate with its nearest members: The presentation layer doesn't communicate directly with the data source. This makes it easier to replace parts without disrupting the whole. In fact the entire hierarchy has two more parts: The actual hardware and the operating system as a sort of abstraction for the communication with the hardware. The complete picture is given below:

Tier	Typical Program /Level
Presentation	Webbrowser
Application	Webserver
Database	Database
Operating system	Operating system

⁹ The term 'application layer' is confusing: In normal speech (as elsewhere in this paper) an application consists of all three layers (data, application, presentation)

Hardware	Hardware
----------	----------

Table 5-3: Complete architecture

Process models

The models described previously are not exactly causal models, in the sense that Part A or event A *causes* Part B or event B. Instead A always *predates* B because of some reason.

Technological model

Hardware is used to create programs and applications. Naturally, a program predates an application. In the software development life cycle the requirements guide the development of programs. As for the effects of ICT on society, programs are supposed to have an effect in so far they are used within a social context. If no one uses a program than it can have no effect. Effects 'work' through features: A feature that is not present in a program cannot be present in any application. The reverse is possible though: it could be disabled in a certain situation.

Software development life cycle model

How should the software development process (the life cycle) be organized most effectively? There are dozens of versions of the software development life cycle; take for example (Ghezzi, Jazayeri, Mandrioli, 1991) below is the (ideal) process of the so-called waterfall model:

- 1) Feasibility study
- 2) Requirements analysis and specification
- 3) Design and specification
- 4) Coding and module testing
- 5) Integration and system testing
- 6) Delivery and maintenance

A key footnote to this model is that software development generally does not halt at phase six; several iterations of this process (or cycles) might be necessary to achieve the desired results. Every iteration can provide useful information (feedback) for the next phase. Depending on the particular development method an iteration can be as short as few minutes and as long as a whole year. But in every iteration program are altered, which thus slowly evolve:

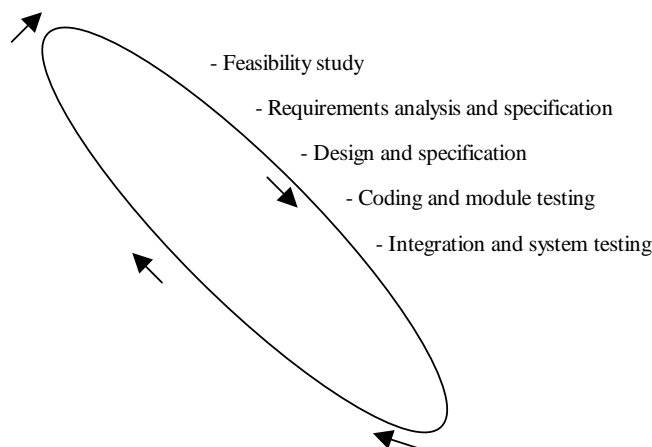


Figure 5-2: Graphical representation of the software development lifecycle

Additional benefits of the technological model

This model helps to understand several things: First it illustrates clearly why sociological views of ICT are often incomplete (as is the normal computer science view):

- Sociology tends to look at running *applications* within the social context of an organization, without looking at the underlying *program* structure.
 - Computer science (and related disciplines) look too much at *programs* without paying attention to their constructedness, ignoring that their evolution takes place through usage as *applications*.
- Secondly it identifies causal relation *inside* ICT itself. This property will be used later.

5.4 Conclusion

The physical, conceptual and social parts of technology also exist in ICT: They are hardware, programs and applications. Programs are created in iterations or cycles. ICT itself is organized in a hierarchical layered way.

6. Research Approach

6.1 Introduction

So far we have concluded the following: ICT research in the social sciences tends to ignore ICT itself and focus on its context. Ironically, a big part of ICT is in fact social in nature; this provides a good reason for social scientists to study ICT.

Then, we investigated what ICT actually is: We saw that, ICT, like other technologies, consists of three parts, physical (hardware), conceptual (programs) and social (applications). Logically, the next questions are about how ICT was constructed, and what its effect are on society. In this chapter we will determine how these questions should best be answered.

6.2 Research requirements

Previously, the causes and effects of technology were split in five parts (physical, conceptual, social, technological¹⁰ en evolutionary¹¹ causes and effects). We will first examine several important causes of ICT, to find out how we can best setup the research.

Technological / social causes and effects

The most interesting question that we would like to answer is about technological determinism: Does technology shape society? If there is no technological determinism, people are free to choose any type of technology that suits them (and they are not limited by it); on the other hand, if there is technological determinism their wishes are not relevant, because there's only a limited amount of options available to fulfill their needs.

If we want to assess this, we need to find a category of social requirements that is ever present but which contains a lot of variety. We could than investigate how the variation in requirements is transformed into ICT: If they are all mapped onto the same structure this could be evidence of some sort of technological determinism.

Evolutionary causes and effects

To research evolutionary determinants it would be important to study ICT artifacts with a long history. For example, if we research cell phone applications we effectively split the whole ICT era into three parts:

- A past when there were no such applications
- A present when there is such an application
- A (possible) future when such applications will not be there (or will be heavily modified)

We might derive that these applications allow the tracking of cell phone users – and possibly cause a reduction of privacy. But how can we generalize these conclusions into the past and the future? Do they not only hold for cell phone applications? It would be helpful to find an aspect that is somehow always present.

Other requirements

Concerning the usage of ICT, we can add that we should not only study ICT used in large organizations but also pay attention to the smaller ones, or even to ICT in use by individuals, this is not done often enough (see Chapter 2).

A final requirement is that the studied ICT artifact should be sufficiently simple: we do not want to spend more time on technicalities than necessary (and a too complicated subject makes it difficult to generalize).

¹⁰ The combined physical, conceptual and social causes and effects

¹¹ The causes and effects seen as a evolution

We conclude that we need to study ICT artifacts that:

- Are shaped by social requirements with sufficient variation
- Have a long history
- Are in use by large and small organizations
- Are sufficiently simple

All in all this approach should deliver results that can be easily generalized to other ICT artifacts.

6.3 Sociological elements: Enabling and constraining

Remembering that most ICT will be useful to some extent – what type of usefulness will remain over time? During your interaction with ICT you have to submit to its rules, no matter how loose they may be. These can be as simple and explicit as a certain workflow (or wizard) that you have to execute, or as subtle as a particular line of thought that is imprinted on you, for example that it's easiest to write everything in English because only an English spell checker is available. Such a feature *enables* – but it also *constrains*.

As said before, technology enables and constrains. In fact, it is precisely the constraining requirements of ICT that remain: If an application is used in a social context it's value can diminish if anyone can get unlimited access. This was as true in the past as it is now; and will be equally true in the future. It must be possible to turn features on and off.

A simple example is that of an application that keeps track of cars. Users can lookup a license plate and find out information about cars. Such a system can be very helpful to the police; it could *enable* them to find out whether a particular car has been stolen. However, the system will lose its value when everyone is capable of changing data inside it. As people with ulterior motives (or simply lacking a proper understanding of the application) gain access the data becomes less trustworthy. It is the *constraints* that create the added value. In fact it would also be possible to charge users for the information in the application; Someone who buys a car might be willing to pay a fee to lookup whether the car has sustained any damages during its lifetime. Again: limiting access creates value.

Note that access is between two extremes: If everyone gains access the application becomes valueless but this is also the case if no one gets access. Derived from the constraining requirement is the need for human interaction with the application. ICT does not work in a vacuum; it interacts with people in society. Now we have a sociological starting point for the study of ICT.

6.4 ICT: Authoring and publishing systems

To study ICT it would be best to choose simple, widely used programs that are similar in function and of which there is a great variety. Applications such as forums, content management systems and blogs are therefore good research subjects, especially once you consider the fact that the Internet as a whole is a huge authoring and publishing system. The constraining requirements vary quite a lot: Some programs strive mostly for control (content management systems) others for freedom (blogs, wikis). Furthermore they have a relative long history (the world wide web is over 10 years old).

Yet there are other reasons why authoring and publishing systems are such good subjects.

Let's begin with imagining that, for example, we were to study the guidance systems of cruise missiles. This would make research very difficult. The military nature of cruise missiles means that we would be confronted with issues such as non-disclosure agreements, background checks and probably a lot of politics. It is questionable whether we would be allowed to publish the full results.

Even more problematic, it would be doubtful if someone would understand the conclusions, since missile guidance systems expertise among most people is fairly limited.

Furthermore how would we generalize our results, would they not only apply to the cruise missile systems?

Turning to authoring and publishing systems we get a completely different picture. They are in use by almost everyone, be it active (writing, creating, publishing) or passive (reading, viewing), to look up the latest news, write articles or stories for study, work or maybe just for fun. There are various types of authoring and publishing systems including *portals*, *content management systems*, *wikis*, *forums* and *blogs*. This software usage is not limited to the military: Individuals use it, as do families, foundations, small business, multinational companies and so do governments as well as the UN.

The interesting thing is that authoring and publishing systems scale very well, from one individual to the biggest organization, from one page of text to a million. We could research perfectly well what happens when we increase the amount of texts in a system, or allow more people to work together on the same texts. In contrast, software that addresses the needs of a human resource department is typically used by organizations that have HRM departments in the first place, which usually means that they employ fifty or more people. This makes it hard to track how organizations function with fewer than fifty people, and what happens if they grow beyond this threshold.

Another reason to use authoring and publishing systems is that the publishing process they facilitate has been around for quite a while. Long before the age of computers people were writing, printing, distributing and reading texts. Looking at the proliferation of IT we can see what part of this process is altered or affected by IT - and possibly more important - which part is not. Of course, with some imagination one can compare missile guidance systems to ice-agersmen throwing javelins, but a lot of programs do not have a clear non-digital equivalent. (What is the equivalent of a web-browser, an operating system?)

Furthermore, the nature of certain software systems makes that for practical purposes there will only be a few of them. Auction software is a good example: The bigger an auction site becomes (i.e. the more items are offered) the more interesting it becomes for both sellers and buyers. This reduces the amount of subjects to do research on. It also means that the difference between the program and its usage blurs: the program *is* in fact the running application. There is not necessarily (or likely) another company that uses the same software program to fulfill different needs. Therefore we would not be able to research the impact of a certain program on an organization. In contrast, there's a plethora of authoring and publishing systems that are specially adapted to specific purposes, in use by many different organizations.

6.5 Conclusion

In this chapter we have formulated the research approach for the study: We should find a sociological theme that can be identified in ICT. The ICT artifacts that we study should have sufficient variation, have a history, be in use by large and small organizations and be sufficiently simple. It was concluded that focusing on the enabling and constraining properties of authoring and publishing systems would offer good changes of fulfilling these requirements.

7. Control (Enabling and Constraining)

We will now further examine the concepts of enabling and constraining. Obviously, enabling and constraining has to do with sociological concepts such as power and control: Who constrains also controls. To identify parts in ICT that are relevant to the topic we must first know how to identify them: Therefore we discuss a few sociological theories on power and control; with these in mind we can again look at the software.

7.1 Power and control in sociology

To examine the sociological perspective we will discuss two distinct views on power.

The first view is from Weber (Macionis & Plummer, 1997). Weber defined power as the ability to achieve desired ends despite resistance from others. Power can have its basis in force (physical or psychological) but these forms are not conducive to a society's stability. A stable society requires authority, meaning power perceived as legitimate rather than coercive. There are three basic sources of authority:

- Rational-legal, power legitimized by legally enacted rules and regulations;
- Traditional, power legitimized through long established cultural patterns;
- Charismatic, power legitimized through extraordinary personal abilities that inspire devotion and obedience.

Weber sees rational-legal power, as the basis of the bureaucracy and a successor of traditional and charismatic authority.

Lukes (1974) created another theory on power. His view is that power is not always observable; there is not always resistance from others that can be detected. Basically there are three levels on which power can be wielded.

1. Exercising of power through decision-making and observable behavior.
2. Applying power to prevent decision-making (a so called non-decision). For example by controlling the 'agenda', people prevent certain issues to be discussed.
3. Affecting other peoples' ideas and thoughts about a certain issue: Note that these might not even be aware that power is exercised over them.

The first dimension is obviously the most visible and the third is the most difficult to detect.

From these definitions it is obvious that some forms of power are subtler than others. Is resistance (as in Weber's definition) a fundamental aspect of power? If someone operates at Lukes' third dimension no one might ever notice that he is being coerced into doing something. And following Weber's own definition, you might be very much willing to abide to the wishes of a charismatic person.

In relation to enabling and constraining we can say that both enabling and constraining can be goals that someone wants to achieve. These do not have to be explicit – someone can be constrained without knowing it himself. The constraining part seems to be the best visible: If someone decides someone should *not* be able to do something than this is more visible than if someone *allows* someone to do something new. This visibility (or lack thereof) has implications for the research design: It's easiest to focus on structures that explicitly *constrain* people.

7.2 Identifying power and control in ICT

Both Lukes and Weber see power as a relation involving people, in the sense that person A has power over person B. A needs B to achieve his goals or A decides for B.

At a first glance such a line of thought seems logic, but on second thought it becomes problematic. For example, it could be the case that A finds a way in which he does not depend on B to achieve a certain goal. A is certainly not powerless in this case; you could even argue that he is more powerful, since he doesn't need B's compliance. Another possibility is that A substitutes B for a machine or device. In both cases there is no direct relationship between

person A and person B, but common sense tells us that A has power nonetheless. Does machine B also has some sort of power over A? Does B have any goals? In this paper we will choose an instrumental view of ICT: It has no goals itself but is used by people to enable and constrain others.

Note that when we examine ICT (instead of it's context) there are no actors, no goals and therefore power cannot be identified: There is only control over what can and cannot be done. Control can be examined: What is the structure of control? How does the control structure change?

Hierarchy of control

A key question is the hierarchical nature of control: Does one group control another group more than vice versa? How is a hierarchy constructed and how does it change?

7.3 Identifying control elements in ICT

First, let's theorize on what types of control could be exercised in ICT. As a starting point we can differ between four types of control. These types are not mutually exclusive (and thus somewhat arbitrary) but they do clarify what and how things can be controlled.

Control over tasks

At a certain moment someone might want to lookup a license number; or create a new entry for a car. We have now two tasks in our system:

- Lookup a license number
- Create a new entry for a car

These could either be allowed or denied to someone.

Control over data

We could also take another approach and look at the license plate information. For example the application contains information from various countries; Normal users can only lookup or change data from their own country;

Some people can only lookup data (such as Interpol) while others can only edit data. No one can every delete data.

Control over delegation

Another option is to investigate who can delegate control to others; someone must be able to grant another person access to the application or deny it in case of abuse. We could call this category "control over control": who controls the controller and who decides what controlling mechanisms are used?

Control over process

We could image a sort of flowchart of operations. A car can only be reported stolen by a person who is known to the system as a police officer. He has to fill in a detailed report of the circumstances. This report has to be authorized by another person. The car now gets the status stolen. If the car is found then another report must be made where the card was found and how identification was possible. It is not possible to remove cars from the database.

7.4 Implementing control in ICT

Control over tasks

Role based access control

How do applications constrain what people can and cannot do? A very common method is so called role based access control as, for example as described in (Ferraiolo, Kuhn, 1992 p. 4):

"A role can be thought of as a set of transactions that a user or set of users can perform within the context of an organization. Transactions are allocated to roles by a system administrator."

This is illustrated below.¹²

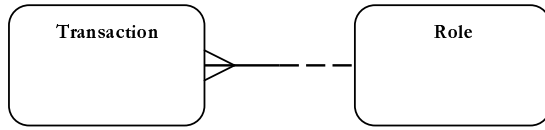


Figure 7-1: Role based access control

For example, in a hospital system we could have the roles of patient, doctor and nurse, which all perform different (and possibly overlapping) functions in the system.

A person is given a role, and based on that role he can perform several functions. This is done for practical and efficiency reasons: if we assign all the functions to individual users instead of groups it would be hard to change the system and the chance of errors would be higher. (Imagine an organization with a thousand employees and a hundred functions!)

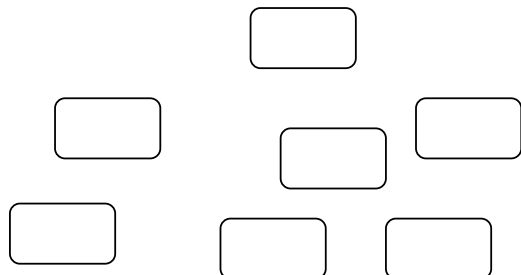
Notice people can be in multiple roles at the same time: a doctor can at some day become a patient. This can pose a problem: we don't want an accountant (who has to check a company's balance sheet for irregularities) to have the role of treasurer at the same time. This is a role conflict, and some systems can enforce rules on them. If you had role A you can never access anything that you could access using role B. This is called a 'deny'.

In practice another term is used, that of a group. One system's group is another systems role and vice versa. The difference is not very clear. In practice roles tend to be sort of static and groups more dynamic. An organization can add groups to an application but the roles usually remain the same.

Access control lists

An important question is at what level we apply these functions. As cited earlier: "A role can be thought of as a set of transactions that a user or set of users can perform within the context of an organization." Sometimes that's just not good enough, we might want to specify access to an *individual patient's medical record*, not for *all patients' records in the hospital*. This type of access specification is most common in file systems, where it's usually possible to specify access to an individual file. For example a document can only be changed the author, who can also specify access hierarchically and set the permissions on the 'thesis folder' that contains the paper. For more information on this specification type see (Custer, 1994).

Control over data



Does the content itself have to be included in the research about control? The answer is yes. To find out why this is the case we need to look close at content structures. First of all, let's assume that our world consists of pieces of text, which have no connection with each other.

Figure 7-2: Mesh of unconnected nodes

¹² The diagramming technique for this image is explained Appendix C.

No we start connecting those pieces, forming a whole network (in mathematical terms a graph).

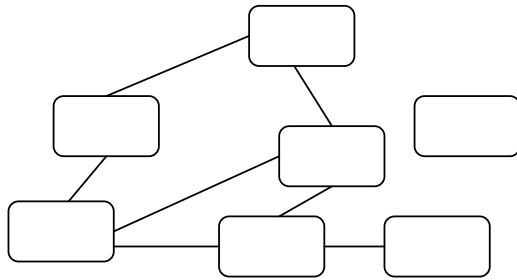


Figure 7-3: Graph

This is not necessary a hierarchical structure, in fact, a hierarchical structure is a special type of graph.

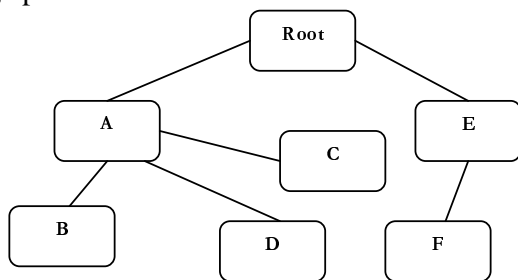


Figure 7-4: Hierarchical graph with root

Now we can remember the access control list type of control: Access that is specified for a certain item. In a hierarchical structure, if someone has access to the top level he is implicitly granted access to the nodes at the lower levels. Therefore a hierarchical content structure allows a hierarchical form of control. In the example: If one is given access to node A one automatically gets access to node B,C and D.

Let's now take a look at the depth of a hierarchy. In the first example, a book consists of chapters that are organized into paragraphs, resulting in a fixed three-layered structure.

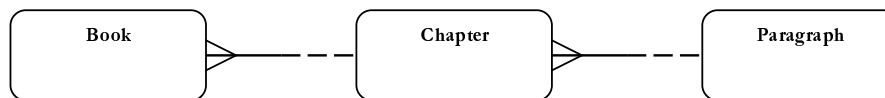


Figure 7-5: Fixed three-layer hierarchy

In some circumstances the depth of the hierarchy is less fixed. For example, we could have paragraphs (1.1 – 1.10) containing subparagraphs (1.1.1 – 1.1.10) containing other subparagraphs (1.1.1.1, 1.1.1...10) and so forth.

Two key differences are:

- In terms of graphs, in the first example we have three node types, in the last one there are only paragraphs.
- The first example does not allow a repetition of node types in its graph. A book cannot contain another book. But paragraphs can contain other paragraphs.

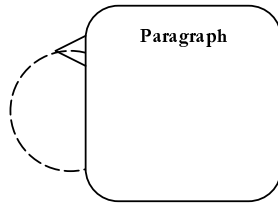


Figure 7-6 Unlimited depth hierarchy

Thus, some database structures (or parts of it) have a fixed hierarchical depth while others' depth is arbitrary.

Content structure revisited

To make matters more complicated we cannot simply state that content is either structured hierarchically or not. This is because ICT is layered (see section 5.3). For example, it would be possible for an application to read data from the database, convert it in some way and present it in a totally different way to the user. Or, because the webbrowser of the user resides at his computer, the user can determine the content presentation by himself. What if any – would be the 'real' content structure?

The answer is that although it's certainly possible to have a three (or more) layered structure where the content has a different representation at each level, in practice this doesn't happen that often and the content structure is determined at two levels:

- Database level: The database structure, where the content is split over different tables, containing records
- Application/ Presentation level: We would call this the 'textual level': Each piece of text in the database can itself contain a hierarchical structure. For example an application can put all texts into one table with no links to itself (and thus a non hierarchical structure) but store the text as a hierarchy:

Content	
ID (Primary key)	Text
1	<ul style="list-style-type: none"> • Land plants (embryophytes) <ul style="list-style-type: none"> ○ Non-vascular plants (bryophytes) <ul style="list-style-type: none"> ▪ <u>Hepatophyta</u> - liverworts ▪ <u>Anthocerophyta</u> - hornworts ▪ <u>Bryophyta</u> - mosses ○ Vascular plants (tracheophytes) <ul style="list-style-type: none"> ▪ <u>Lycopodiophyta</u> - clubmosses ▪ <u>Equisetophyta</u> - horsetails ▪ <u>Pteridophyta</u> - "true" ferns ▪ <u>Psilotophyta</u> - whisk ferns ▪ <u>Ophioglossophyta</u> - adderstongues ▪ <u>Seed plants (spermatophytes)</u> <ul style="list-style-type: none"> ▪ †<u>Pteridospermatophyta</u> - seed ferns ▪ <u>Pinophyta</u> - conifers ▪ <u>Cycadophyta</u> - cycads ▪ <u>Ginkgophyta</u> - ginkgo ▪ <u>Gnetophyta</u> - gnetae ▪ <u>Magnoliophyta</u> - flowering plants
2	

Table 7-1: Hierarchical content inside a database text

The database structure is normally not changeable by the user; it can only fill it with data. But it is possible to create one's own structures in every piece of text that is stored. So if we have a hierarchical database structure we can create or own non-hierarchical structure on top of it. And the other is also possible – we can potentially create a hierarchical structure on top of a non-hierarchical one.

Control over delegation

Some people must be allowed to do certain things, but how is this determined? Logic dictates that some persons must be able to delegate control and grant others access. This is usually implemented as a special role that someone can have or a task that can be executed.

Control over process

A complication of the role based access control and access control lists is that they lack the option to specify the process itself: while some functions may (or should not) be applicable at a given moment, others need to be executed in a specific order. For example, during a patient registration process in a hospital, the person who does the intake enters the patient's date of birth, name and address, and it's only possible to register the results of any blood tests after patient registration. We call this *workflow* and some applications can not only determine *what* but also the *order* in which users execute tasks by presenting (and enforcing) a so called work list.

The Workflow Management Coalition (TYPO3, 2005) defines workflow as follows:

A workflow, or workflow-process defines the predetermined succession of work steps, or activities, executed by various users of a common system who differ in terms of rights, tasks and access rights. The Workflow Management Coalition (WfMC, an international association of leading software vendors defining relevant standards and models) further defines workflow as "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules".

A standard content life cycle probably could contain the following steps for example:

Create todo – create instance - edit (copy of) existing content – edit copy – submit to predefined group/ role - review cycle 1-n – release/publication - live - archive – offline.

The most likely scenario for a workflow process will probably be the review cycles through which the system iterates the status of the content. This could possibly be any status of "edit, review, publish, and archive".

This is commonly called the "content life cycle".

Initiation

In fact we have forgotten something: in the previous section the implicit assumption was that we had an authoring a publishing system in the first place. We did not pay any attention to who puts the system into place: who installs it? This is not an irrelevant question, but an inherent part of the design: The program does not install itself and without installing it one can never use it. Therefore we must also examine the installation procedure.

Terminology

A note on terminology: A lot of different terms are used to indicate what a user needs to perform a certain function. Among those are roles, permissions, authorizations, transactions, capabilities, ... the important thing to remember is that groups and roles bundle actions together. In this paper permissions, authorizations, transactions, actions, capabilities all indicate features below the role level.

7.5 The feature - user matrix

Next we will theorize on control structure in applications. The objective is to characterize how they enable or constrain people. For this we will use the feature-user matrix: a table representing

application features and application users. A feature represents some action that a user can perform; in this sense, features loosely cover control over tasks, data, delegation and process. A user either has or does not have a certain feature. The matrix itself represents a particular point in time. Below is an example of 4 features and 4 users.

	User A	User B	User C	User D
Feature A		X		
Feature B	X	X		
Feature C	X	X		X
Feature D	X	X		X

Table 7-2: Example of a feature user matrix

In this case user A is allowed to execute the features B, C, D; user B can execute all features, user C none and user D only C and D.

Amount of possible feature user matrices

Since each user can either have or lack a feature, the total number of possible feature-user matrices in this case is $2^{(4 \times 4)} = 2^{16} = 65536$.

In general this is

$$2^{(\text{Features} \times \text{Users})}$$

For example, in a feature-user matrix with 2 features and 1 user we have:

	User A		User A		User A		User A
Feat. A		Feat. A	X	Feat. A		Feat. A	X
Feat. B		Feat. B		Feat. B	X	Feat. B	X

Table 7-3: feature-user matrix with 2 features and 1 user

Link with role based access control

Going back to role based access control, we remember that this approach ties features to roles: A user is not granted the execution of a certain feature, but gets a certain role that implicitly allows him to use the feature. We can name each role in a system by the features that it contains: for example the role ABC or the role AB, or AD.

How many of the available matrices are possible if roles are used? In general there are two cases:

- A user can only have one role
- A user can have multiple roles

One role for each user

If a user can only have one role, the amount of roles needed to 'cover' the entire feature user matrix is $2^{(\text{Features})}$. In the previous example this means we have the roles A, B, AB and (None). This situation is examined in the next example. We have three roles: ABCD, AB, CD.

An example of the user-feature matrix is:

	User A	User B	User C	User D
Feature A		X	X	
Feature B		X	X	
Feature C	X	X		X
Feature D	X	X		X

Table 7-4: Feature-user matrix using roles

In this case, it is not possible to grant user A only feature A: he automatically gets B. Therefore a fixed role system can limit the amount of available options to distribute features among users: the options are *predetermined*.

Multiple roles for each user

If a user can have more than one role, the amount of roles needed to ‘cover’ the entire feature user matrix is exactly the same as the amount of features. In the example we need only roles A, B, C, D. Each role corresponds with exactly one feature.

Hierarchies and roles

Looking from the perspective of hierarchies we wonder whether these structures have the tendency to cluster hierarchically. If a user has feature B he automatically gets features C and D; a feature at a high level (B) indicates that a user also has lower level authorizations. This is called a Guttman scale, and we can use it to identify hierarchies. Below is an example given the roles ABCD, BCD, CD and D.

	Role A	Role B	Role C	Role D
Feature A	X			
Feature B	X	X		
Feature C	X	X	X	
Feature D	X	X	X	X

Table 7-5: Hierarchical role structure (role-feature matrix)

An example that is not hierarchical is: ABC, BCD, CD and D. The role ABC and BCD each have an element (D and A) that the other one misses.

Workflow

Can we also use the feature - user matrix to model workflow? In fact, the previous examples were all situated at one particular point in time. Workflow works over multiple points in time so we can represent it like this:

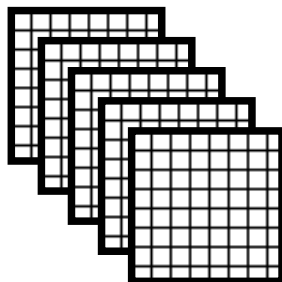


Figure 7-7: Workflow as feature-user matrices in time

Over time, several users are allowed to execute different features. At one time a user can edit a text, but after submitting it for review it becomes read-only.

7.6 Value of enabling and constraining in ICT

If a system is studied – how should we interpret its control structure? In the last sections it was shown how hierarchies could be identified in control structures and that role based control structures can limit the amount of options for the distribution of features over users. If this is the case, does it mean that such a software package is more limited than others? We can also restate ‘enabling’ and ‘constraining’ in terms of an application’s value, based on the available features or users’ actions.

More is better

More features, more data, more topics, more functions give more power to the user to do what he wants to do. Set aside issues of usability (can a novice user find his way through all these menus and options) such a view usually holds true for applications that are used on an individual

basis. Having a spell checker will automatically result in a text with fewer errors. Indeed, more is better.

	User A	User B	User C	User D
Feature A	X	X	X	X
Feature B	X	X	X	X
Feature C	X	X	X	X
Feature D	X	X	X	X

Table 7-6: 'Ideal' user / feature matrix (1)

Less is more

However, if several people use our application, questions of coordination, responsibility and trust tend to arise. This is especially the case in an environment where people have never met face to face, for example on an Internet forum. In such a setting, limits must be imposed to what an ordinary user is allowed to do, or the system will stop functioning properly at some point in time. There's always someone who could abuse the system in ways that make it unworkable as a whole. Out of all the options that users have some must be taken away to make the application function properly. Note that the perspective has changed: We now look at what the *entire* system achieves, not at an individual's contribution. To summarize: We started with a full set of options and have taken some away, thus less is more.

	User A	User B	User C	User D
Feature A		X	X	
Feature B			X	
Feature C	X	X		
Feature D	X			X

Table 7-7: 'Ideal' user / feature matrix (2)

As a result, a hierarchical control structure (as a special form of enabling and constraining) might be very useful in a certain context.

7.7 Conclusion

Enabling and particularly constraining is a good focus point for our research because they are always present in ICT. It is especially easy to see how ICT constrains people. There are four types of enabling and constraining in ICT: tasks, data, delegation and process. Enabling and constraining properties of a system can be modeled using feature user matrices. These help to identify hierarchies in systems and to assess how much potential freedom there is to distribute control. An implicit way of enabling and constraining is through the usage of roles: These are basically groups of features that can be allowed or disallowed for users.

8. Authoring and Publishing Systems

8.1 Introduction

In the previous chapter we saw that power was linked to control which in its turn could be seen as a form of enabling and constraining; The latter being easier to identify in ICT.

We will first look into authoring and publishing systems in greater detail. Next we examine the general publishing process and see what types of enabling and constraining we can find there – these might then also be present in authoring and publishing systems. To underline the relevance of the research, we conclude with a look at the World Wide Web as one huge authoring and publishing system.

8.2 Definition

How broadly should we interpret the term “Authoring and publishing systems?”

Authoring and publishing systems are programs whose primary purpose it is to facilitate the creation and publishing of written texts, accessible from the World Wide Web, by multiple users simultaneously.

This eliminates several types of systems:

- Applications designed for editing and publishing images, audio and movies
- Applications for sending E-mails, SMS messages, chat boxes, instant message applications: these are all forms of *personal* communications

A few examples of authoring and publishing systems that *are* included are given below:

Category	Overview
Forum	System that allows people to post messages (threads) to which other people can reply, resulting in whole discussions
Blog	Personal, online journal comprised of periodic articles and links
Wiki	System that poses no editing restricting for anyone
Content management system	System used to organize and facilitate collaborative creation of documents and other content

Table 8-1 Examples of authoring and publishing systems (Wikipedia, 2005)

8.3 Identifying hierarchies and control in the publishing process

How can control be identified in software systems? We will start by examining a general publishing process. Drawing upon this information issues of control can be identified.

Publishing process

Publishing is a well-established tradition and this makes it possible to compare the old-fashioned process of publishing a paper book with its pure digital counterpart and spot the differences that might be relevant (related to power). In software engineering practices it is not uncommon to write down an entire process in a so-called *use case* to identify the key actors, results and steps, before digitalizing it. There are special diagrams and techniques; see for example (Booch, Rumbaugh, Jacobson, 1999), but only a textual description is used here. The following section is based upon (Kraaijeveld, 2005) and (Huijzer, Peer, Pol, 2005):

The publishing process varies from publisher to publisher and the process of publishing a newspaper is quite different from publishing a book. The latter process is probably longer and we use it here, for clarity. The general process has the following steps.

- Initiation

- Concept creation
- Creation
- Marketing/ sales
- Realization
- Publication
- Delivery to customers

In the initiation phase people come up with new ideas about what should be published. Ideas can originate at the publisher self, a freelance author or even the publishers' customers.

Next, a selection is made from these ideas: some are better (general: more profitable) than others, not every idea can be realized.

In the creation phase one or more authors write the text. Editors help to improve it and make suggestions for improvements. Finally a decision is made whether or not to publish the book. Versions (called revisions) go back and forth between the author and the editor.

Even before the final printing of the book, the marketing and sales phase begins: the book is advertised in a brochure or a so-called dummy is created with only the cover and a back flap text. This helps to gather information from the market, about whether a book might be a success. Some data is already available, such as the price, number of pages and the date of publication.

If the book seems commercially viable and no other problems occur, it can be realized. Once the author and editor are agreed on the general text, the bureau-editor re-reads the text to correct style and typing errors that have gone unnoticed. These are mostly small changes; the author is informed of any important changes. Next the text's layout is determined by the typesetter and a proof is made. The bureau editor checks the proof for typesetting errors such as abbreviations and 'staircases'. If necessary the process is repeated. Sometimes but not always the author gets to see the results. The following stage is pre-press: Graphic workers do a final print (the plotter proof) to check printing errors such as missing pages. The publisher has to agree with this print: all subsequently discovered errors will be his responsibility. During the printing a specific copy is used to check the print quality (ink, paper).

Finally the books are delivered to the customers and the process starts all over.

Control in publishing

What does controlling publishing mean? We will pose several questions for each of the steps in the publishing process. Answering these will help to define what control for the entire publishing process means.

- Initiation

Initiation can come from multiple sources, who is allowed to initiate a publishing process?

- Concept creation

Who selects the concepts that will be published?

- Creation

Who creates the contents; who writes or changes the texts, creates drawings?

- Marketing/ sales

Who determines the price; who gets the publication under the users attention?

- Realization

Who decides when the publication is finished?

Who decides about the presentation, layout of the publication?

- Publication

Who distributes the publication to the customers, who determines who is allowed to read (or copy) it and for how long?

This list allows us to link the publishing process with the control structured in ICT. It essentially splits up the publishing process into distinct activities (or features) We can therefore look at who is allowed to perform a certain activity, for example determine a book prize. But we can also ask a

meta question: who determines the way the entire process is managed?¹³ This resembles the concept of workflow.

8.4 The Internet as one huge authoring and publishing system

To further build the case for authoring and publishing systems as a research topic, the Internet is essentially a very big authoring and publishing system, in use by millions of people, thousands of organizations and searchable through various search engines. Surely, it has changed from its early popularization starting with Gopher (Network Working Group, 1993) Usenet (Lost in Usenet – References, 2003) to the World Wide Web that we use nowadays, but the essential usage remains the same: facilitating communication, creating and sharing information. The question of control is also relevant on scale of the entire Internet: How we manage it, how we deal with issues of copyright protection, freedom of speech, proliferation of potentially dangerous information and censorship. All these issues have in some way to do with creating and publishing texts.

The original proposal for the World Wide Web

It has been over fifteen years since Tim Berners-Lee made his proposal for a distributed hypertext system at CERN, which formed the basis of the World Wide Web. (Berners-Lee, 1989) In his "*proposal concerning the management of general information*" he describes that although CERN is nominally organized into a hierarchical management structure, the actual shape of communications is that of a multiple connected web. (See illustration below).

According to Berners-Lee, tree-like systems that were in use at that time were inaccurate representations of the real world, because not everything fits into a hierarchy. For example in a hierarchically oriented discussion group system, some discussions might belong to several categories. It would therefore be better to create a system whose *method of storage does not place its own restraints on the information*. The original image of his proposal is presented below:

¹³ A side issue is whether the publisher exercises some form of control over its readers. Surely, publications contain knowledge that the readers consume, altering their ideas as they read the text. This goes too far, we will only look here at who's allowed to read or copy a text.

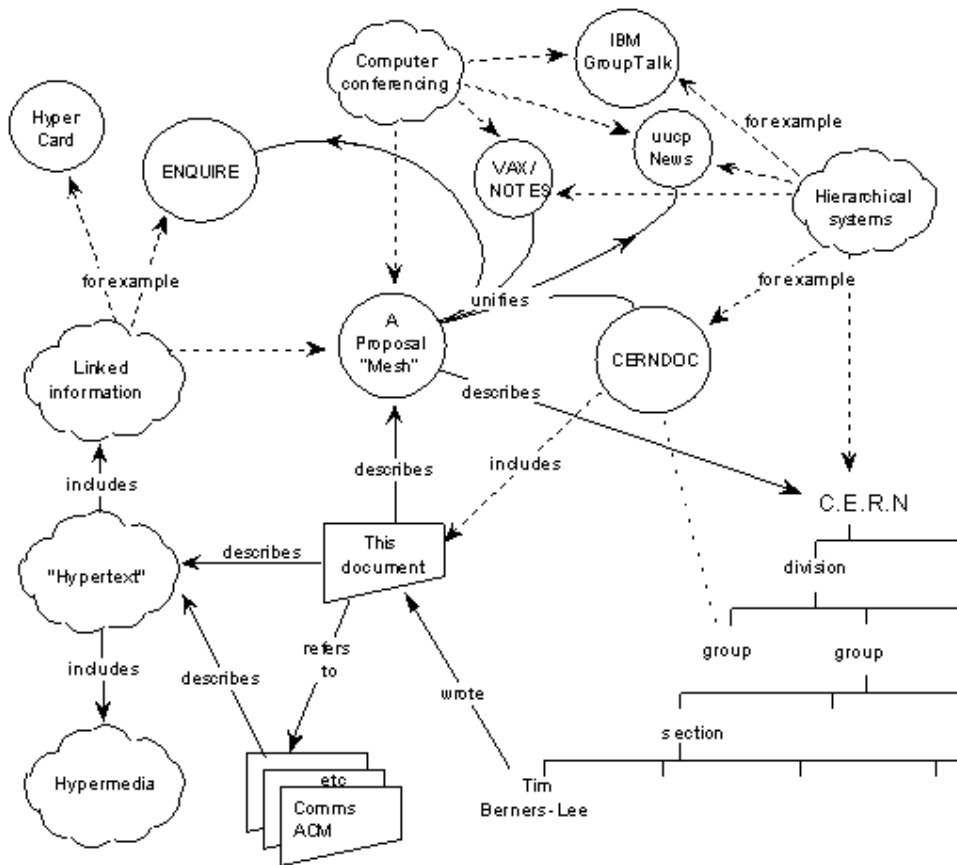


Figure 8-1: proposal by T. Berners Lee

At the right we see the neat hierarchical structure of CERN, to the left a more complicated 'mesh' structure. Is this non-hierarchical structure indeed a better representation of reality? If we see the World Wide Web as an experiment, it is tempting to agree. After all, the World Wide Web is a huge success since its introduction in 1989. An enormous amount of content is nowadays accessible via the World Wide Web. Nevertheless, some doubt remains, especially because the World Wide Web has changed significantly over time. Whereas CERN was forced to invent the Web itself, we nowadays have numerous applications available that enable us to create and share information. As technology has progressed more options are available to guide the content creation and publishing process, both enabling and constraining users options. Could it be that we have been steadily building hierarchies instead of meshes? Could the picture below be a more accurate (because more hierarchical) representation of the Internet?

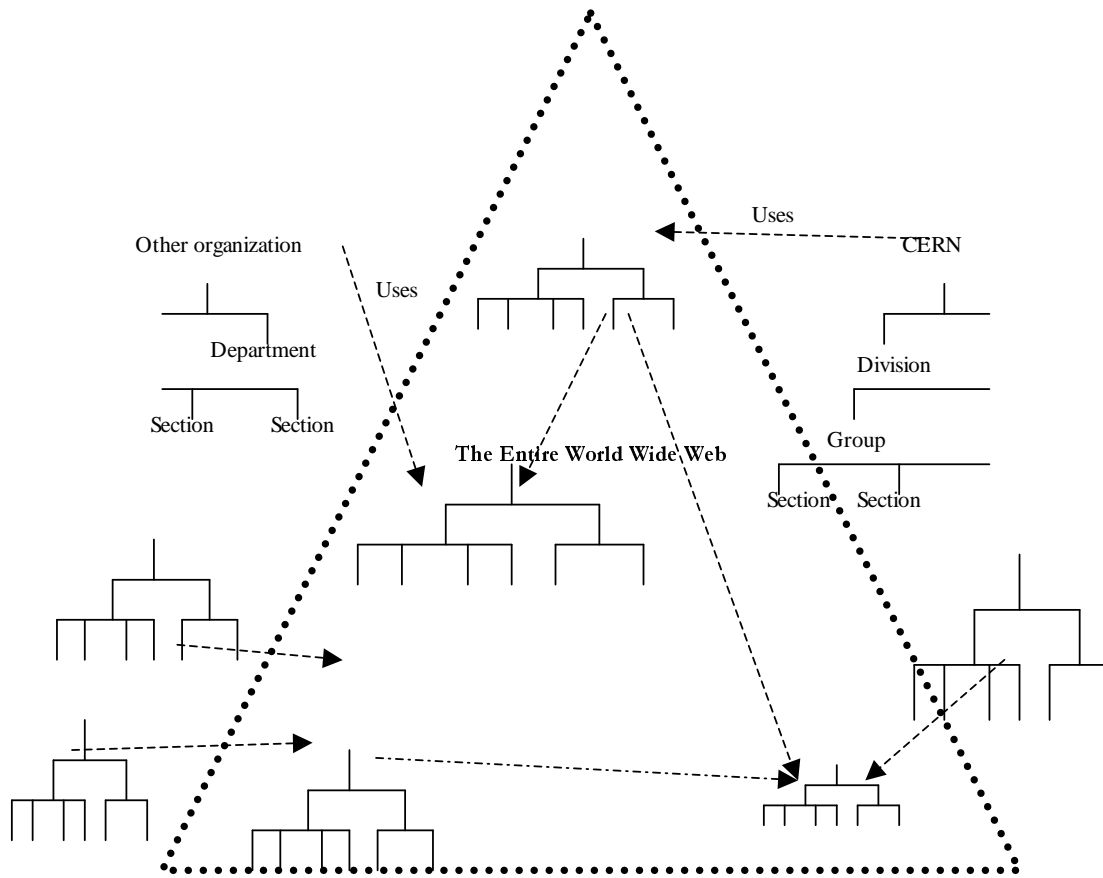


Figure 8-2: Alternative (hierarchical) representation of the World Wide Web

8.5 Conclusion

Authoring and publishing systems seem to be good research subjects for whoever wants to study software. They are widely used, easy to obtain and in use by many different types of users and organizations. In essence, the World Wide Web is one huge authoring and publishing system. In the context of this paper it is interesting to point out that it was specifically designed as a non-hierarchical structure. This reaffirms the idea that studying control hierarchies is a good approach for crossing the boundaries between sociology and computer science.

Part II: Empirical Research

9. Research Questions

9.1 Four research questions

In the previous chapters the research approach (study ICT structure before looking at the context) and topic (the control structure of authoring and publishing systems) were defined. We will now formulate the research questions. Earlier it was stated that the most improvements in sociological ICT research could be made if we focus on ICT rather than on theory. The first question is therefore descriptive:

Question 1 - On structure: How is control structured in authoring and publishing systems?

To address this first question we will research several authoring and publishing systems to find out how they deal with control issues.

We will especially look at the hierarchical nature of the control structure. Can users choose freely between hierarchical and non-hierarchical structures? Maybe Berners-Lee was right when he mentioned that it should be possible to store data non hierarchically – but does the same also hold for control?

Next we can examine what has caused these control structures to be created. A particular interesting question is about technological determinism: Do organizations have a choice in selecting a particular control mechanism?

Question 2 – On causes: How can the control structure be explained?

In the chapter on technology we saw several types of causes and effects. Among these are:

- Social causes and effects.
Software programs must work in society, therefore we can study society and examine what type of software is in demand and find out how the demand shapes the programs.
- Evolutionary causes and effects: The software development cycle shows that software is created in iterations: The study of many iterations could lead to an evolutionary explanation of how these systems evolved over time.
- Technological caused and effects: Also possible is that ICT has some inherent structural properties that naturally guide its evolution; there is little choice in this case.

Subsequently we can look for other ways of structuring control:

Question 3 – On alternatives: Would it be possible to structure applications differently?

If the answer to question 2 is mostly technological, there would be little choice between different structures. However, if the answer is primarily social or evolutionary it could be that some options have been overlooked.

Finally we ask what the effects are of these control structures on society.

Question 4 – On effects: Discussion based on a, b and c

How can the results be interpreted, what are the implications?

9.2 Initial hypothesis

For question 1 and 2, we will assume that the structure of a program can purely be explained by its usage or intended purpose. Based on this hypothesis, we estimate that a content management system (with the emphasis on management) will be more hierarchical than a forum, where individual contributions are deemed more important than control. This leads to the following table:

Type	Overview	Expected hierarchical score
Content management system	System that allows people to post messages (threads) to which other people can reply, resulting in whole discussions	++++
Forum	Personal, online journal comprised of periodic articles and links	+++
Wiki	System that poses no editing restricting for anyone.	++
Blog	System used to organize and facilitate collaborative creation of documents and other content	+

Table 9-1: Expected hierarchical score

10. Sampling

10.1 Introduction

Before any system could be studied, a selection had to be made: What authorization and publishing systems should be chosen? In this chapter we look at the sampling process and find out why certain systems were selected over others.

10.2 Sampling

There are hundreds if not thousands of programs that can be classified as authoring and publishing systems. It would be impossible to examine all of them. Since this is not a quantitative study there was no need for random sampling methods, non-random methods were applied. Neumann (2000) mentions several ways of non-probability sampling:

Type of sample	Principle
Haphazard	Get any cases in any matter that is convenient
Quota	Get a preset number of cases in each of several predetermined categories that will reflect the diversity of the population, using haphazard methods
Purposive	Get all possible cases that fit particular criteria, using various methods
Snowball	Get cases using referrals from one or a few cases, and then referrals from those cases, and so forth
Deviant case	Get cases that substantially differ from the dominant pattern (a special type of purposive sample)
Sequential	Get cases until there is no additional information or new characteristics (often used with other sampling methods)
Theoretical	Get cases that will help reveal features that are theoretically important about a particular setting/ topic

Table 10-1 Types of Non-probability samples

In retrospect all of these were used. At the end so-called theoretical saturation was reached—adding applications did not result in any new input, they seemed to converge into a general model.

We will now describe some issues that might create a bias towards a certain program structure and show this was compensated for

- Correlation with the software development model (the causes)
- Correlation with the software usage (the effects)

Software development ↔ software structure

Initially, there were two issues that had to be examined:

- The development method

Because open source software is freely available (and thus easier to research) the sampling was biased towards open source.

We will not dive very deep into the exact differences between these two methods. Instead, we only state that for the most part, they use different tools, methodology and that their perspective on what a program should be (or do) are different. The reader can find more information on the subject at (Open Source Initiative, 2005)

- Incorporation of changes in software over time

This issue is loosely related to the previous one. If the program is open source, everyone can modify the existing version to fit his needs. It is not uncommon for open source programs to

'fork': split into different programs if developers cannot agree on what should be included.¹⁴ The program gets multiple branches, it is not certain if changes in the 'leaves' are ever implemented in the 'root' of the program tree. Another important option is the availability of a plug-in architecture (which can be considered as a sign of technological maturity). In an ideal plug-in architecture customizations are implemented as an add-on ('plug-in') that must be installed separately from with the core version; it does not replace the core version but extends it. A plug-in architecture is common in large programs, both under open and closed source. Should plug-ins also be examined? For example, the core program could be considered very hierarchical, while a plug-in would negate this characteristic.

Regarding these issues two conclusions were reached:

- Plug-ins should not be studied. There could be too many of them – only the core program would be examined. If a program would have multiple forked versions just one relevant version was studied.
- Several open source programs should be compared with closed source applications. Two programs were studied: Sharepoint and a forum application in use by Microsoft about their web development platform called ASP.NET (Microsoft, 2005) They did not stand out significantly from the other systems but for practical reasons they are not included in this paper.

Software usage ⇔ software structure

Obviously, an organization will try to find a match between its own characteristics (or requirements) and a system's characteristics. It's easy to say that the sample population consists of programs and not of organizations, but that doesn't mean there's no relationship between them – and knowing that this relationship exists you might as well use it to be sure your sampling process was adequate.

Three special organizational characteristics are:

1) Organizational size

In general the requirements and needs of an organization change as it grows. Probably its website will become bigger and maintenance and coordination are more time consuming.

2) The importance that an organization attributes to the application contents

An organization that has a particular view about a topic and wants to express in public, will make sure that it displays a coherent set of texts, which possibly requires a stricter authorization or workflow model. Oppositely, there are also websites where this is not the case, instead whose owners try to generate a discussion *about* a topic without holding a particular view.

3) Organizational expertise

An organization that has its business in text writing itself (a publisher, marketing bureau etc.) has naturally more knowledge about authoring and publishing. It can therefore be expected to be able to formulate clearer, stricter requirements for its applications.

A bias was suspected towards applications in use by organizations that scored low on these characteristics.

¹⁴ Examples of these are Silva (Infrae, 2005) based on Zope (2005), CivicSpace (2005) derived from Drupal (2005), Project Minerva (2005) derived from phpBB (2005)

To assess the differences, the chief of the Internet edition of NRC was interviewed, a large Dutch newspaper (Benjamin, 2005) Secondly an interview was held with the Internet editor of the faculty of social sciences at Erasmus University Rotterdam (Maan, 2005). The latter's application was also examined. Interestingly but also fortunately, both organizations did not use any extraordinary applications or features that weren't present or comparable in the sample applications. (The Erasmus University's system was examined though.) This also indicates that the results can be easily generalized to other types of systems. Another form of stratification was by selecting applications of different categories such as blogs, forums, and content management systems.

Other sampling methods

Since this study is particularly interested in 'what is there' it was attempted to select systems that are in widespread usage. Furthermore, one case (MediaWiki) was selected for its theoretical relevance, because it was supposedly used in a non-authoritative way. Finally some cases were chosen because their creators and users were easily accessible.

10.3 Overview of cases

Below is a list of all programs that were studied. The appendix contains more information (vendor, URL, version number) of all the programs. Of each system the latest (obtainable) version was examined to make sure that the study was as up-to-date as possible.

Name	Category
Drupal	Content management system, generic
DSpace	Document management
Mambo	Content management system
MediaWiki	Wiki
Movable Type	Blog
phpBB	Forum
Silva	Content management system
TYPO3	Content management system

Table 10-2 Overview of cases

A short description of each system (taken from the systems' websites) is given below:

Drupal

Drupal (Drupal, 2005) is software that allows an individual or a community of users to easily publish, manage and organize a great variety of content on a website. Tens of thousands of people and organizations have used Drupal to set up scores of different kinds of web sites, including

- community web portals and discussion sites
- corporate web sites/ intranet portals
- personal web sites
- aficionado sites
- e-commerce applications
- resource directories

Drupal includes features to enable

- content management systems
- blogs
- collaborative authoring environments
- forums
- newsletters
- picture galleries
- file uploads and download

DSpace

DSpace (DSpace, 2005) is a groundbreaking digital repository system that captures, stores, indexes, preserves, and redistributes an organization's research data.

Jointly developed by MIT Libraries and Hewlett-Packard Labs, the DSpace software platform serves a variety of digital archiving needs.

Research institutions worldwide use DSpace to meet a variety of digital archiving needs:

- Institutional Repositories (IRs)
- Learning Object Repositories (LORs)
- eTheses
- Electronic Records Management (ERM)
- Digital Preservation
- Publishing
- and more

Mambo

First and foremost, Mambo (Mambo, 2005) is a Content Management System (CMS). It is the engine behind your website that simplifies the creation, management, and sharing of content.

The goal of the Mambo project is to meet most of the requirements highlighted in the above article. As each day in development goes by we are getting nearer and nearer, while at the same time building a solid core which can be extended by third party developers.

In the hands of a custom developer, this makes Mambo a powerful platform for a wide variety of Internet applications that go far above and beyond the simple creation of content.

MediaWiki

MediaWiki (MediaWiki, 2005) is the collaborative editing software that runs Wikipedia, the free encyclopedia, and other projects. It's designed to handle a large number of users and pages without imposing too rigid a structure or workflow.

Movable Type

Movable Type (Sixapart, 2005) is the premier blog publishing platform for businesses, organizations, developers, and web designers. Powerful customization gives you control over everything you publish and the elegant interface keeps things simple and clear.

PhpBB

phpBB (phpBB, 2005) is a high powered, fully scalable, and highly customizable Open Source bulletin board package. phpBB has a user-friendly interface, simple and straightforward administration panel, and helpful FAQ. Based on the powerful PHP server language and your choice of MySQL, MS-SQL, PostgreSQL or Access/ ODBC database servers, phpBB is the ideal free community solution for all web sites.

Silva

Silva (Silva, 2005) is a powerful CMS for managing content for the web, paper, and other media. Content is stored in a clean and future-proof format, independent of layout and presentation. Features include a multi-version workflow system, integral WYSIWYG editor (Kupu), content reuse in multiple publications, sophisticated access management, extensive import/export facilities, fine-grained templating, and hi-res image storage and manipulation.

TYPO3

TYPO3 (TYPO3, 2005) is a free Open Source content management system for enterprise purposes on the web and in intranets. It offers full flexibility and extendibility while featuring an accomplished set of ready-made interfaces, functions and modules.

10.4 Examination process

Triangulation (a mixture of research methods) was used as much as possible without any involvement of creators or users. This non-obtrusive approach made it possible to research more systems; to do so without bias and let the programs ‘speak for themselves’.

Before starting the examination it was necessary to install most programs beforehand, which turned out to be a non-trivial task. (In fact this was an important result - we will later return to the issue of installation). Each program was used for a while, some data was entered, users were added, and permissions were changed, to find out how it worked. Manuals were examined – if they were available. If possible the underlying database or source code was studied, these formed a good check on the user interface – as it turned out some applications were structured in a whole different way than you would expect from using it alone. This also helped to spot similarities between applications that would otherwise have gone unnoticed.

For each system, things that seemed to be connected to control and hierarchies were noted, keeping in mind the different aspects of content structure, authorization structure and workflow. After studying one system the next one was examined, sometimes finding out that something was missed in a previously examined application and it was necessary to go back.

10.5 Conclusion

In this chapter we described the sampling process. The sampling population consists of many authoring and publication systems. Several programs were selected, using non-probability sampling techniques. Effort was made to correct for biases towards certain program structures. Finally, eight programs were selected. Each program was examined using various methods, without involvement of users or creators.

11. Structure

11.1 Introduction

We will now examine the way in which authorization and publishing systems were structured. Earlier the functionality of authoring and publishing systems was split into four different parts: control structure, workflow, content structure and initiation, this division is maintained in the conclusions. How hierarchically is control structured in applications?

11.2 Control structure

Ranking the control structures

We will now look at the roles that programs provide. In Chapter 7 it was concluded that role based security can be hierarchically and that it can also limit the options to distribute authorizations among users. How should we interpret the results? We use a scale for the hierarchical score:

Score	Interpretation
-	Role are non-hierarchical or absent
+	An administrator is present, basically a two level hierarchy
++	Roles are hierarchical but the role structure is conceptually split in two sections
+++	More than two roles exist which form a strict hierarchy

Table 11-1: Interpretation of control structure scores

The table below shows the scores for each program. In this table (and in subsequent tables) the actual hierarchical score is shown next to the hierarchical score as derived from intended use (the latter is based on the hypothesis in section 9.2 that the intended usage determined the structure).

Name	Type	Roles	Hierarchical score as derived from intended use	Actual hierarchical score
Drupal	Content management system, generic	Anonymous Authenticated	++++	+
DSpace	Document management	Anonymous Admin	++++	+
Mambo	Content management system	Registered, Author, Editor, Publisher Manager, Administrator, Super Administrator	++++	++
MediaWiki	Wiki	User, Developer, Bureaucrat, Sysop	++	+++
Movable Type	Blog	-	+	-
phpBB	Forum	Ordinary user Administrator	++	++
Silva	Content management system	Reader, Author, Editor, Chief Editor, Manager Everybody, Authenticated, Viewer, Viewer+, Viewer++	++++	++
TYPO3	Content management	Ordinary user, Administrator	++++	+

	system			
--	--------	--	--	--

Table 11-2: Actual applications scores on control structure

From this viewpoint, MediaWiki is the most hierarchical application; Drupal and Movable Type are the least hierarchical. The hypothesis on the control structure does not seem to be supported by the evidence.

11.3 Workflow

It turns out to be more difficult to create a scale for workflow. Some application have workflow options, other do not. If we do have a workflow (consisting of two or more steps) we can check if the person who authorizes the content has more authorizations than the user who submits the content. If this is the case it's an indication that the workflow is hierarchical.

We will again use a scale for the hierarchical score:

Score	Interpretation
-	No workflow present
+	Workflow present, non-hierarchical
++	Both hierarchical and non hierarchical workflow possible
+++	Workflow present, hierarchical

Table 11-3: Interpretation of workflow scores

Name	Workflow features	Hierarchical score as derived from intended use	Hierarchical workflow score
Drupal	Moderation queue, voting Authenticated	++++	-
DSpace	Three step process, steps can be skipped	++++	+
Mambo	Two step process	++++	+++
MediaWiki	Two step process (lock and unlock pages)	++	+++
Movable Type	-	+	-
PhpBB	-	++	-
Silva	Two step process	++++	+++
TYPO3	Three or four step process	++++	++

Table 11-4: Actual applications scores on workflow

Of the five applications that have workflow functionality, only two have non-hierarchical workflow. The hypothesis on the control structure does not seem to be supported by the evidence.

11.4 Content structure

Each system allows its content to be structured in one way or another and all systems have some form of hierarchy. Earlier hierarchical content structures were linked to hierarchical control

structures: who has control over a top or root content element usually controls the underlying elements.

An application is therefore more hierarchical if the content structure

1. Is organized hierarchically
2. Is organized hierarchically and can be of unlimited depth (see Section 7.3)
3. Allows permissions on all of it's elements
4. Allows permissions to be propagated to lower nodes (In other words: a permission that is set on a high level node also determines permissions at lower levels)

Again, a scale for the for the hierarchical score; This time, we will count each of these items as one +.

System Name	Hierarchy	Propagation	Depth of hierarchy	Permissions on all levels	Hierarchical score as derived from intended use	Actual hierarchical score
Drupal	Yes	No	Arbitrary	No	++++	++
DSpace	Yes	No	Arbitrary	Yes	++++	+++
Mambo	Yes	No	Fixed	No	++++	+
MediaWiki	No (only textual level)	No	Fixed	No	++	
Movable Type	Yes	No	Arbitrary	No	+	++
phpBB	Yes	No	Fixed	No	++	+
Silva	Yes	Yes	Arbitrary	Yes	++++	++++
TYPO3	Yes	Yes	Arbitrary	Yes	++++	++++

Table 11-5: Actual applications scores on content structure

Thus, MediaWiki can be considered the least hierarchical application; Silva and TYPO3 are the most hierarchical applications.

The hypothesis on the control structure seems to be supported by the evidence, with the noted exception of Mambo.

11.5 Initiation / Installation procedure

The first finding was that installation procedures were in many cases far from trivial, even for someone skilled in software engineering. In spite of wizards and 'automatic' configurations they require a fair amount of knowledge.¹⁵ Another finding was that a typical installation procedure results in the creation of an initial administrator account. This administrator can create other users and delegate authorizations to these users. However, he remains in control for the duration of the application usage.

The only notable exception is Movable Type, which has a slightly different type of administrator: The first user can create other users, but can only remove (or administer) users he has created. All users have the same amount of permissions but run the risk of being deleted by those who created them.

Below are some screenshots and excerpts from manuals.

¹⁵ The technicalities of specific installation procedures are not discussed here. More information on installation problems can be found on the websites of the examined systems (see Section 18.4)

Welcome to your new **Drupal**-powered website. This message will guide you through your first steps with Drupal, and will disappear once you have posted your first piece of content.

The first thing you will need to do is **create the first account**. This account will have full administration rights and will allow you to configure your website. Once logged in, you can visit the **administration section** and **set up your site's configuration**.

Figure 11-1: Drupal installation

9. Create an initial administrator account:

```
[dSPACE]/bin/create-administrator
```

Figure 11-2: DSpace installation

URL	<input type="text"/>
Path	<input type="text"/>
Your E-mail	<input type="text"/>
Admin password	<input type="text"/>

Figure 11-3: Mambo installation

Sysop account name:	<input type="text" value="WikiSysop"/>	
password:	<input type="password" value="*****"/>	Must not be blank
again:	<input type="password" value="*****"/>	


Figure 11-4: MediaWiki installation

2. Log in with the author name **Melody** and the password **Nelson**.
3. The first thing you should do is change your author name and password. To do so, click **Edit your profile**, then change the author name and password there.

Figure 11-5: Movable Type installation

Admin Configuration	
Admin Email Address:	<input type="text"/>
Domain Name:	<input type="text"/>
Server Port:	<input type="text" value="80"/>
Script path:	<input type="text" value="/phpbb/"/>
Administrator Username:	<input type="text"/>
Administrator Password:	<input type="password"/>
Administrator Password [Confirm]:	<input type="password"/>

Figure 11-6: phpBB installation

Username and Password
Enter your username and password for the Zope Management Interface 

The username and password are required to secure the Zope Management Interface (ZMI) and the Silva Management Interface (SMI). If you forget your password, you can use the `zpasswd.py` Python script in the Zope directory to create a new one.

Username:

Password:

Figure 11-7: Silva installation

Apparently you have completed the basic setup of the TYPO3 database.
Now you can choose between these options:

- **Go to the frontend pages**
- **Go to the backend login**
(username may be: *admin*, password may be: *password*.)

Figure 11-8: TYPO3 installation

11.6 Organizational choice?

We will now reflect upon the issue of organizational choice. How much choice do organizations have to create their own control structures inside their applications?

We can try to answer this question by reexamining MediaWiki, which was found to be relatively hierarchical in structure, compared to other systems. However, MediaWiki is also used by Wikipedia, an on-line free encyclopedia, (Wikipedia, 2005) is based. With some exceptions, everyone is allowed to update articles at Wikipedia. Can a system be both hierarchical and open? The key to answering this question is to make a distinction between the *program* and the *application*: The MediaWiki software program is hierarchical in nature. Yet this does not tell us anything about how the application is used. The hierarchical structure of the MediaWiki is fixed, but its usage is not.

We can use the feature-user and feature-role matrices to illustrate this. MediaWiki has four roles: (Ordinary) User, Developer, Bureaucrat and Sysop.

	Sysop	Bureaucrat	Developer	(Ordinary) User
Feature 'Move page'	X	X	X	X
Feature 'Site admin'	X	X	X	
Feature 'Set user rights'	X	X		
Feature 'Protect page'	X			

Table 11-6: roles and features of MediaWiki

This is a clear hierarchical structure: Anyone with the Sysop role can use more features than with role Bureaucrat or role (Ordinary) User. Now consider two running applications at two organizations P and Q: Both have 4 users.

	User A	User B	User C	User D
Role Sysop	X	X		
Role Bureaucrat			X	X
Role User				

Table 11-7: Role distribution at organization P¹⁶

	User A	User B	User C	User D
Role Sysop	X			
Role Bureaucrat				
Role User		X	X	X

Table 11-8: Role distribution at organization Q

Both organizations cannot escape the hierarchical nature of the program: They both have hierarchical control structures. Yet the role distribution is different: the average user at organization P can use much more features than at organization Q. The structure of the program allows thus for some organizational choice.

Decentralization?

As a side issue: One of the returning questions about ICT is whether it leads to centralization or decentralization (Zuurmond, 1994) within organizations. Looking at the content structure of a program, a simple answer can be found: Within one program, decentralization is in fact a special feature of a hierarchical structure. You cannot decentralize before you have centralized; the feature can always be turned off. The tendency is thus to centralize.

11.7 Conclusion

Initial hypothesis on control structures

The hypothesis that content management systems would be more hierarchical than other types of applications was not proven. Most evidence does not support this hypothesis. For example, the MediaWiki program has a very hierarchical control structure; this was not expected.

General observations

In general we observe three things:

- 1) Within a program, control structures have a tendency to be hierarchical. Within the context of an application, decentralization should therefore be considered as a special feature that –if turned on – can always be turned off.
- 2) The transition from program to application (the installation procedure) almost always results in the creation of some sort of administrator who grants rights to other people. Installation procedures require a fair amount of knowledge and are not easily done by someone with little technological expertise.
- 3) Noteworthy is also the transition from persons to users. There is not necessarily a one-to-one link between persons and users, some persons share user accounts, people can have multiple accounts. Within applications groups of users can be attributed rights, but it is impossible for users to collective *do* something, collaborative actions are basically independent, individual actions. No two persons can ever click ‘OK’ together.

These conclusions are regardless of the original requirements (controlled or uncontrolled) behind the programs: It could be assumed that there is a shared underlying factor that causes this. (We will look for such a cause in Chapter 13.)

¹⁶ Obviously, a Sysop is also an (ordinary) user. To make diagram not too difficult to read only the most important role is shown.

Organizational choice

In many cases, the program determines the control structure. Thus, if an organization uses a certain program it has a limited amount of choice regarding the control structure.

12. Effects

12.1 Introduction

Chapter 11 ended with the conclusion that control structures in authoring and publishing systems share common characteristics, regardless of the requirements: control is hierarchical, with an all powerful administrator on top, who rules alone. The next two chapters deal with the causes and effects of such a setup. We will start with the effects because this suits the logical flow of the paper best.

12.2 Security

The biggest impact is probably on security: As ICT proliferates, the major implication of all this is that the correct functioning of fewer (but more critical and bigger) applications – at any point in time – is determined by the individual actions of all-powerful administrators. No limits are imposed on their actions. Society becomes more centralized and more vulnerable as a result.

As for software development, it is interesting to see that a lot of effort has been put in fixing security holes in software that can be exploited by malicious programs and users to circumvent existing security measures. But the biggest security holes are *by design*. No anti-virus or spyware solution can ever solve the administrator security hole.

Therefore, the net effects of applying present day technologies on improving security are probably quite limited. They can be even negative up to a point because ICT does not provide any safeguards to counter the administrator's power. In the information age, where knowledge equals power and administrators guard knowledge, this should be a cause for concern.

Or should it not be? A counterargument is that many technologies do not contain safeguards. A surgical knife (no matter how well designed) does protect neither the patient nor the doctor from fatal accidents. A car does not protect people from being overrun. Should ICT be protected from administrators? ICT differs in at least two ways from 'ordinary' technology:

- The potential amount of damage caused by abuse of ICT is much bigger. A knife can injure only one patient at a time in one place. An administrator with too much power can disable or misuse the ICT infrastructure of an entire hospital within seconds, putting many more lives at stake. ICT also controls many other forms of technology.
- ICT has autonomous capabilities. A knife doesn't injure by itself, but a computer virus spreads itself in seconds across the globe, possibly compromising many administrator accounts. The Internet has thus short-circuited many barriers that exist in the physical and social world.

Ironically, ICT *does* contain a social structure. Many applications require people to identify themselves before gaining access; they can only do what the system allows them to. Knives lack such features. If we think these structures in ICT are necessary, it seems illogical to have all-powerful administrators who can negate all security measures.

12.3 Conclusion

At any given time, the functioning of an application depends on the proper behavior of at least *one* user, the administrator. This setup has made society very vulnerable to abuse, by human administrators but also by computer viruses. ICT has two characteristics that make it different from most other technology that can be abused: Its potential impact is bigger; it has autonomous capabilities. In the next chapter we will try to find out how this could have happened.

13. Causes

13.1 Introduction

In the previous chapter, we examined the hierarchical control structure of several programs. After this descriptive part we will now attempt to explain how they evolved, using three types of explanations:

- Social explanations
- Technological explanations
- Evolutionary explanations

As stated before, software is created in so-called iterations. In each such period, software is developed before it is used; this usage leads to knowledge that forms the input for the next iteration. Therefore, two directions were followed:

- Looking at the design phase of the software development life cycle (Silva, Talmon, EcoGrid)
Data was gathered using several methods: by participating in the design of one specific program and by interviewing software developers.
- Looking at the usage phase of the software development life cycle (Erasmus University, NRC, Heliport)
For each, several users were interviewed who work regularly with authoring and publishing systems. The interviews were half structured and each took approximately an hour.

A brief description of the cases is given below:

Phase	Program	Program category	Organization
Design	Silva	Content management system	Infrae
	Talmon CMS	Content management system	Talmon
	EcoGrid	Mixed	UvA
Usage	Silva	Content management system	Erasmus University
	Open Market CMS	Content management system	NRC
	PhpBB	Forum	Heliport

Table 13-1: Examined programs and organizations

The remaining part of this chapter contains the aggregated results of the interviews as well as data from other sources. More data on the interviews can be found in appendix E.

The question that we try to answer here is: How did we end up with such hierarchical systems, over which individual all-powerful administrators rule?

13.2 Social explanations

Looking from a social perspective we find three explanations.

Disinterest in websites and content management

From the interviews, it becomes clear that website management has a sort of negative value. From the users' perspective, the people that were interviewed at NRC and EUR would love to involve more people in the content creation process, but this hasn't happened yet. In these organizations, the value that people attribute to authoring and creating content for websites

themselves is not very big.¹⁷ For Talmon as a software company, deploying content management systems allows their customers to change small things themselves – relieving them from of a lot of tedious work. This reduces their *webmaster bottleneck*.

For a part, this has to do with technological problems. The technology behind the World Wide Web is older than a decade – but a lot of applications are still difficult to handle. These problems start with difficult installation procedures and continue throughout the content creation process, where there is the risk of ‘breaking’ parts. This is something that webmasters like to prevent – but this attitude is also applicable for a lot of (potential) users. Looking from an evolutionary perspective this problem might even become bigger over time: as a website grows there’s simply more to break. Whether employees perceive websites as important or not, they still are essential communication tools for the organization. It could be that the users’ relative lack of interest and skills leads to centralization – in the end someone has to be responsible for the website.

This also affects the authorization process – since few people are really interested in content management the attitude is to keep the authorization structure very simple, both from the perspective of developers (who do not want to waste their programming time) as of the users, who do not like to be bothered with too much procedures. Of course: strict procedures become only essential until the amount of users grows beyond a certain threshold: when personal communication becomes difficult.

An indication that people do not take authorizations serious is that it is generally not possible to query users’ rights to perform a certain action. (In security terms: an audit) Indeed, applications have roles and users can be in roles that allow actions to be performed. But is it possible to find out what a particular user can or could do? Can user A edit this text? Could user A edit this text last week? Most applications make it very hard if not impossible to answer these questions. This makes implementing workflow particular difficult. If the administrator has decided that you cannot edit a text – but you just found a typo – to whom do you need to turn to have it corrected? If you can’t find out who can – it might never be fixed. A side effect could be that users have a tendency to be over-privileged, how can they otherwise work with the system?

Clear lines of responsibility

Maintenance is almost always necessary and requires someone with very high privileges. The idea that a single person should be able to perform maintenance is a reasonable requirement, especially for smaller organizations where technical expertise is limited. This results in a simple and hierarchical structure.

Keeping in control

More critical is the explanation that administrators want to retain their power: Once in control, they do not want to relinquish control to other users. The same goes for developers who create the programs themselves. Again, this results in simple and hierarchical structures.

But during the research for EcoGrid (see Appendix E) it seems that the opposite was the case: The developers would like to turn over control to the users – but fear the consequences of improper usage at the same time.

An incomplete informatization process

Can the structure of authoring and publishing systems be explained by looking at the informatization process as a whole and at software as a part of society? Certainly this process is far from finished. The authorization system of the examined systems is rudimentary. Why does it work? The negative value of website management can only be a partly explanation because sometimes content management *does* matter. The answer has to be that the checks and balances are somewhere else - in the ‘real’ world outside of ICT. An administrator has seemingly a lot of

¹⁷ (In the case of NRC it is to be expected that employees will be far more interested to have their article printed on the front page of the paper edition.)

power – but is still employed by an organization. If he wants to keep the job he cannot afford to make too many mistakes or abuse the system. Thus the results indicate that:

Socialization, training and indoctrination are the most important safeguards and complement the simplicity of current day authorization systems.

This hypothesis was verified by interviewing a database administrator (Anonymous, 2006) Because of the sensitivity of the information this was done on condition of anonymity. During the past five years, the interviewee he has worked on a project basis in the financial and telecommunications industry, for various large organizations. As such he deals with systems for which the security requirements are generally higher than for the average authoring and publishing systems examined here.

Typically, there are three tasks that a database administrator must perform:

- Operational management: For example, creating and restoring backups
- Database design/ layout: making sure that the database performance is optimal
- Database programming: creating programs that work with the data

For each of these tasks it is not necessary to get full administrator access, but usually this is possible though. In general he describes the security measures as 'insufficient'. Available security protocols are not properly implemented. Data is not encrypted (any administrator can read information from the database). Audit trails are setup but seldom examined. Many older (legacy) applications are not designed according to today's security standards.

However, gaining entrance to these databases is a more complicated process. A candidate must demonstrate that:

- He has the proper training: a certification or diploma for the database that must be maintained)
Obviously, one of the elements of this training is dealing with and securing sensitive information.
- He has been of good conduct (no criminal offences)
- He has good references

Additional examinations are done using intelligence and psychological tests. For extra sensitive database background checks are performed on family and friends. Initially new employees are carefully watched - but this practice is abandoned once someone had demonstrated that he does his job properly. It can also be said that additional loyalty is bought - a good database administrator can earn a substantial amount of money. In general the working conditions are good. Once you have setup your working environment properly (right database configurations, usage of the right tools) there's in fact little to do. Overall, few incidents have happened - he doesn't know of incidents in the past years where data was compromised or abused by administrators.

13.3 Technological explanations (including physical and conceptual)

First of all, let's point out that theoretically there can be no real limitations - any piece of hardware can run programs that are equivalent to a Turing Machine (see Section 3.2). In this sense, ICT cannot be limited by itself. However, the Church-Turing thesis doesn't state anything about how Turing Machines come about and how they are supposed to interact with the outside world. Limits in it's creation and communication would have no real meaning for a running Turing Machine; but they sure could have for those who use them.

When a program is installed on a single piece of hardware (and becomes an application) there is always one (and exactly one) administrator who does the installation. The underlying hardware and software forbid that two people can install a program together. True cooperation

during the installation requires special hardware and/ or software (something like the proverbial two keys that must be turned to launch a nuclear missile.)

Without such special hardware or software it would still be possible to create a program that requires two or more installations (possibly on different hardware). The applications would then work together, forming a new application on top of the existing ones. Multiple installations would negate the power of a single administrator. This is evidence against technological determinism. But such a setup could be very complicated and costly.

Once an application is available other users must be granted access. The easiest way to do this is to have an administrator grant those users access. This automatically creates a hierarchical role structure. When an administrator is present it doesn't make sense to constrain people too much or require them to cooperate. (In the end the application's weakest link is the administrator. Thus ICT always *enables* more than it *constrains*.)

13.4 Evolutionary explanations

How can the control structure of authoring and publishing systems be explained? Earlier we mentioned: *“As any social construction, this ontology is partly self fulfilling - using software is to some extent accepting it's ontological points of departure.”* From this point of view software represents the way things are – and is thus self explaining. However this does not take into account that software has evolved. There were no content management systems in the early days of the World Wide Web, but they are here now.

Evolutionary explanations

ICT consists of hardware, programs and applications. These are essential building blocks for the creation of ICT that can only be put together in a certain way: A program is installed on a piece of hardware and becomes an application. Once this setup was created it became (and still is) very difficult to escape the idea of a single administrator, for reasons of logic, cost and standardization. The existing programs and application serve as inspiration and basis for new programs and applications: the idea of an administrator has been around for a long time: everyone knows such a setup: deviating from the standard “line of thought” becomes difficult. There could have been other possibilities – but only this one survived.

Website evolution

Why did we end up with authorization and publishing systems in the first place? Below is a small table adapted from (Gijzel, 2003) that describes the evolution of a non-specific website into a website based on a content management system.

	Phase	Add content	Visitor feedback	Workflow management
1	Brochure website	None (File level access by vendor)	None	None
2	Extended website with webmaster	File level access	Email form	None
3	Database driven website	Special page protected by user login	2+ comments	None
4	Web content management	3+ Special page protected by user login	3+ Discussion forums	Rudimentary
5	Adaptive business communication	4 + various sources, possibly automatic	4+ Collaboration on documents	Extended

Table 13-2 Website phases

In the first phase the organization outsources website creation and maintenance. The website contains no more than a brochure and the organization's postal address and phone number.

Next the organization retakes control (possibly realizing the strategic possibilities of the website) and hires a webmaster. The webmaster controls the structure, contents and layout of the website. Everything is stored in flat HTML files that must be uploaded to the webserver. Sometimes the webmaster writes texts himself or others ask him to publish a text on their behalf. This creates the so-called 'webmaster bottleneck': as the site's size increases it becomes more and more difficult to maintain. Dead links, different layout of web pages created a tough job for the webmaster. The solution is found in phase three: separate the content (text) of its presentation that are intertwined in traditional HTML files. Now that the content is stored in a database people can write texts without worrying (more than necessary) about the way they were presented. Users are encouraged to edit their own texts, relieving the webmaster who only has to determine the generic structure and layout.

In the 4th stage the workflow starts to take hold: users can no longer just publish anything, contents has to be approved by an editor.

The final phase is reached when the system is no longer seen as an extension of the website but an integral part of the organization, with backend connections such as payment processing.

Note that the sites have become more and more interactive over time. With phase three the organization starts using the three-tier model described in Section 5.3.

Benefits of content management systems

Hartman Communicatie (Hartman Communicatie BV, 2005), a consultancy for information architecture and content management strategy, mentions four positive effects from content management systems:

- 7) The website is easier to update and therefore more up-to-date
- 8) The website is more consistent because style and layout can be enforced throughout the website
- 9) The website can be personalized – information can be tailored for specific users

10) Overall maintenance costs go down because less webmaster time is required

Since there were no content management systems in the early days we can see this process as essentially technology driven: as the technology to improve things becomes available it is applied. Because the paper deals with content management systems in the broad sense – do the same reasons hold up for other applications like forums or blogs? Another important advantage of content management over traditional HTML file websites is the ability to have more people contribute to the website, even if they do not have an account at the webserver itself. Content management allows for more collaboration.

Authorization evolution and hierarchy

How did the authorization structure evolve into what it is now? In the early days, users were given full control of (a part of) the website. They uploaded their HTML files onto the website, using a special user account. Unfortunately this account (typically created for a webmaster) normally allowed access to the entire site (or part of it). Therefore it was unsuitable to use in collaboration with other people, especially when their trustworthiness was unknown. To counter this a new system was built on top of this, and a whole new sets of accounts was created in a database.

But the webmaster account is still in use: whoever decides to use a content management system must first install it on the webserver, which up to this day means uploading files on the webserver. Because content is now added via the application the old webmaster account is used less and less.

In a sense it was duplicated in the administrator role in a typical content management system (and in fact, a similar role must exist in the database as well).

If it were possible to do away with the administrator account of an authorization and publishing system, this would not make much sense because the all-powerful webmaster account still gives access to all files and content. At some point in time it must be re-used, for example to install an update of the authorization and publishing system.

Given a number of features in an application, people can be granted the right to use them. But there has to be someone who grants them this right and this creates the need for a sort of administrator who can oversee the entire system. To some extent this negates the fear of someone breaking the system: the omnipotent administrator can destroy the system whenever he wants – and this obviously doesn't happen very often or the system could have been altered. And indeed – most ordinary users behave quite decently.

13.5 Conclusion

Why are control structures are so hierarchical? Possible social explanations are the general lack of interest in websites and content management; the importance of clear lines of responsibility; the wish of developers and administrators to stay in control; the existence of checks and balances outside of ICT. A technological explanation is that it's very hard to create a system without some sort of initial administrator. Finally an evolutionary explanation shows us why and how authoring and publishing systems have evolved over time.

14. Alternatives

14.1 Introduction

In the previous chapter we tried to find explanations for the evolution of the examined systems by researching their development and usage. Dozens of iterations consisting of designing and using software have resulted in a wide array of systems, but this doesn't necessarily mean that there are no alternatives, that these applications are somehow the logical conclusion of automation and informatization. Maybe they could be structured in a whole different way. Can software be non-hierarchically? Can we create a system without administrators?

14.2 Cooperation

One of the findings was that it is generally not possible to perform an act *together*. In a typical scenario for an organization, one day the company board decides that the organization will start using a software package. This is the last real collaborative action in the whole software deployment chain. Certainly, it is nowadays possible to collaborate on a document, if necessary with thousands of other users – but this *collaborative* result is created by *individual* efforts. Workflow does nothing to counter this: it consists of a series of steps done by individuals, not real collaboration.

Actions are always attributed to users, not to groups: Groups can be given permissions, but a group cannot jointly *do* something. This is visible in every system log. It never mentions that a group has done something *together*. It is always user *X* performed action *Y*.

This is easy to understand if one searches for “Last edited by” in a search engine. There's a big chance that you will only find user names. There's never a group that publishes or changes something. Some people have realized that this can be incorrect – a statement could be issued from an organizational unit rather than a person. They will then create a specific *user* for this purpose, for example (Tzoumas, 2005), or hide the person that last edited the entry from view. And let's not forget that the installation creates a single *individual* administrator. How would we imagine a system where people act together? We can reuse the feature-user matrix for this purpose. Initially we have the following situation:

	User A	User B	User C	User D
Feature A		X		
Feature B	X	X		
Feature C	X	X		X
Feature D	X	X		X

Table 14-1: Example of feature user matrix

Next we could create a ‘cooperation matrix’ where we describe the actions that are only jointly possible.

	Group	Group	Group	Group
Feature A	A	B	C	D
Feature B	AB	DC		
Feature C	ABCD			
Feature D	AB	AC	BC	

Table 14-2: Example of cooperation matrix

For example, User A, B, C and D can perform function A alone, but they must do feature C all together. And for feature D they users A, B and C need the cooperation of one other user. There is no reason why such a system could not be implemented. Why isn't it done? One would suspect that it takes extra effort. Maybe hierarchies are simply more efficient.

14.3 Splitting up

One system, different parts

Because of the negative attitude that was perceived towards website management, authoring and publishing systems might not be the best case to look for alternatives. An interview was arranged with a system administrator (Govers, 2005). It was learnt that really critical systems (such as banking applications) are split up into different parts that communicate with each other. As a whole, this creates a less hierarchical application. In a sense this is a sort of workflow, but because each system is separated from the other the result is somewhat different. The cashier can accept the money from the customer and the truck driver can transport the money to a central safe, but there is no way that the truck driver can ever become cashier or vice versa. Another solution is to split-up a system altogether and have the parts run completely independent from each other. These three situations are illustrated below. First we see one huge hierarchically structured system:

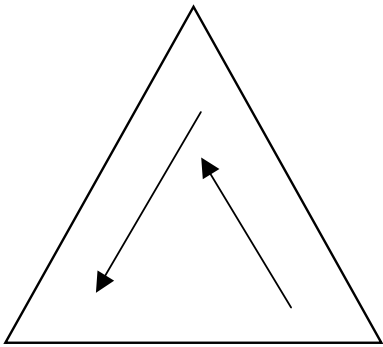


Figure 14-1: Monolithic hierarchical system

Next we split the system in two different (but still hierarchical) systems that communicate with each other.

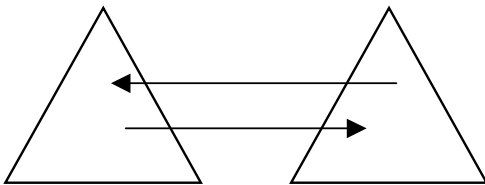


Figure 14-2: Multiple hierarchical systems with communication between them

Finally we sever the ties between these systems, which now function completely independent of each other.

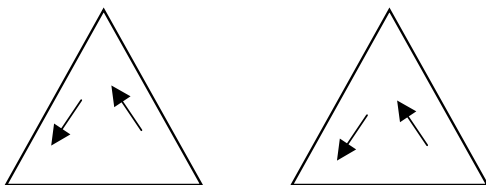


Figure 14-3: Multiple systems without direct communication

14.4 Audit trail

In Chapter 11 we examined the on-line encyclopedia Wikipedia, and mentioned that almost all users are allowed to edit its articles. How can such a system function at all– why do vandals not destroy the Wikipedia? In fact, vandalism does occur, but it is repaired. Wikipedia contains an extensive audit trail of all the changes that articles undergo. If someone adds advertisements or profane language, others can restore the article to a previous version easily. Most users do not have direct database access – and are therefore not capable of changing the audit trail. (Again - this functionality is limited to people higher in the hierarchy).

Thus, a good audit trail, combined with version management can reduce a system's vulnerabilities, without extensive authorization rules. But the systems is therefore not as open as one might think – and more hierarchical at the same time. More importantly, such a feature is only useful in an environment where the information that is stored is not very sensitive, and changes have a limited effect on the physical world. If changes also affect the physical world they could be impossible to roll back (consider someone publishing the blueprint for an atomic bomb).

14.5 Conclusion

There are indeed alternatives for the way that current authorization and publishing systems are set up. Splitting up hierarchies is something that is already implemented in other systems. If the information is not particularly sensitive an audit trail combined with version management can do the job as well. What is not yet available is the option to perform group actions, to act as a collective.

Part III: Conclusions

15. Discussion: Rise of the Administrators?

15.1 Finding a suitable name for the ICT revolution

How should the last decades of ICT proliferation be described? Castells (2000) coined his book “Rise of the network society” and indeed our society has become a network of interconnected nodes. But this is nothing new - already hundreds of years ago there were networks of cities trading goods with each other. In that sense nothing fundamentally has changed – or will ever change. It’s also interesting to point out that most networks in this paper turn out to be hierarchical. In a lot of circumstances a network seems too loose a structure to be useful. Maybe ‘Rise of the Networked Hierarchical Society’ might be a more accurate description. But were networks not always hierarchical?

But although ICT is ubiquitous, some expectations did not come true. The arrival of a new intelligent species as envisioned in the movie “2001: A Space Odyssey” has not happened. It is also way too early to speak of a “Rise of the Robots” to recall another one. Only recently have we witnessed the first vehicles that were capable of driving themselves autonomously through the desert (DARPA, 2005). We are a long way from intelligent machines. To the contrary, we have seen that old-fashioned people – the administrators – are in total control.

Marx’s theory that the industrial revolution *caused* the capitalist class is questionable, but ICT certainly gave rise to a whole new group of people, the administrators. And just as the capitalist class replicated itself every generation so does the administrator class in some way: The operating system administrator creates a database administrator, who in turn creates an application administrator.

Maybe the best description of what is happening would be: “Rise of the Administrators”. If this paper makes anything clear, it is the ‘inevitable’ rise of a new class of people who rule the digital – and the physical – world.

15.2 Three views on the administrator

Within an organization, administrators are situated in the techno structure. As such, they are less visible than line managers, but over the years they have managed to control a vital part of our society that we cannot live without.

Who is the administrator? He has many names (application manager, sysadmin, admin, root, network manager, IT manager, systems manager to name a few) and this makes it hard to describe his position. Is he the all-powerful operator in a system that he completely controls? (Constraining) Or is he the trouble-shooter who has to fix a system in the middle of the night if something goes wrong? (Enabling) It appears that we can look at the administrator in many ways. We will now describe three distinct views on his position.

The administrator in Lukes’ 3rd dimension

When asked, an administrator will probably deny that he has any power – emphasize his subordinate position in the organization, point out that he doesn’t work alone and consults others before taking action. (Anonymous, 2006) But an administrator is in many cases the sole and supreme authority in ICT. Ordinary users can only suspect this. For example it is often assumed in organizations that administrators can read everyone’s email (and gather a substantial amount of information)– but in many cases these remain suspicions (Anonymous, 2006). Using Lukes’ theory on power: They seem to operate in the third dimension.

The administrator as a street-level bureaucrat

Where should we place the administrator in an organization? Could he be described as a street-level bureaucrat (Lipsky (1983), who has much more discretionary power than intended? Lipsky argued that in bureaucracies, policy is often made at the lowest levels where bureaucrats interact with clients. Their power is especially great if performance is hard to measure, there are limited resources, clients are forced to use the service and if there are no clear goals for what should be

done. These situations can indeed occur for administrators. We should however differ between several cases:

- The clients are within the organization itself;
In this case they often need to use certain services.
- The clients are outside of the organization and use the application voluntarily.
If the application is not critical then clients can simply opt out. For example, if a blog administrator changes the policies then someone can simply signup at another blog.

Thus the administrator's real power depends on the exact circumstances – but this doesn't change anything about the underlying control structures inside the application.

Metaphor of the digital dive

The administrator's position can also be understood when we use a metaphor, and see ICT usage as taking a dive into a submerged digital world. For deep sea diving, it holds that a dive cannot be done safely if no one remains on the surface. A line tender is needed who controls the line to which the diver is attached. If something goes wrong the line tender can 'simply' pull up the diver. But the diver is also completely dependent on the line tender: if he would sever the line, the diver would be in a lot of trouble.

The administrator is in a similar position. His position is not on the surface right between air and water, but right between the digital and the physical world. In case of trouble – he can rescue the users. Without him, using ICT is not safe – but at the same time he has complete control over them.

The line tender is also not the most important person on the diving team – but his position makes him important nonetheless, the same thing holds for the administrator of an organization. He does not rival the CEO or even the CIO, but does perform an essential and critical function. Seen from this point of view, administrators are necessary when groups of people attempt a *digital dive* together. His position flows naturally from any type of technology that allows physical processes to be transformed into digital ones. It also shows that ICT has an inherent physical component – of which it cannot free itself. As such there is an inherent limit as to what can be digitalized.

16. Conclusion

Citing the introduction, the aim was to investigate ICT itself (and not its context) from a sociological perspective and prove that such an approach

1. Is feasible and sometimes even preferable over other approaches
2. Can lead to new insights that are not easily obtained by other means

We will now reflect on whether this has been the case.

Reflections on the research approach

Indeed, it was possible to study ICT apart from its context: It is very well feasible to investigate programs and find out about their structure, rather than organizations. But such an approach must be complemented by other means: If we want to explain why control structures are the way they are, we must also look at their development and usage – process that take place in society. While trying to explain ICT we can therefore not ignore context.

Key insights

ICT is structured hierarchically and – for now - cannot do without all-powerful administrators. Ultimately, this makes ICT very insecure.

ICT's control structure can be explained in three ways:

A technological explanation is the way that ICT itself is structured, consisting of hardware, programs and running applications. A program must be installed on a piece of hardware to become a running application; only an administrator can do this. He then grants the authorizations to other people. There is thus an inherent tendency to centralize. This is an indication of technological determinism: It is not organizations that choose a particular configuration; rather their options seem limited by the technology itself.

A social explanation is that safeguards against abuse are currently *outside* of ICT, in the embedding of administrators within organizations, their education and indoctrination. Socialization complements the minimal safeguards inside of ICT.

Another explanation is evolutionary: Because early systems had administrators, no one ever saw the need to change this. It was easiest to copy and reuse the existing authorization structures.

Options to mitigate (and possibly eliminate) the critical role of administrators include the creation of true cooperation within applications, the splitting up of applications and the creation of audit trails.

The results can be used for further research and to improve the security of both existing and new applications.

17. Recommendations

17.1 Introduction

This chapter gives several recommendations for future research projects.

17.2 Methodology

Version control systems

To help the research of authoring and publishing systems evolve, an important step would be to do research with the aid of version management systems. In this paper, state of the art programs were examined and a number of interviews were taken to research their evolution. However, many organizations keep their software versions in separate, so called version management systems.

If properly used, these contain the entire history of a software package and this data can be used to study the software's evolution.

One program, multiple applications

In this paper several programs were examined. Another approach would be to examine one program and see how several organizations make use of the control structure. Do organizations differ in this respect? How much organizational choice is there?

Experimental systems

One practical research approach could be an attempt to create an administrator-less system, where all users are equal. Another idea would be to implement a sort of democratic process inside an application, where users can choose their administrators for a predetermined period.

17.3 Subjects

High risk systems

Although the examined systems had authorizations and audit trails, probably no one designed these systems to be completely secure. Has this lead to a bias – would other types of systems have yielded different results? To complement this paper, more critical systems could be examined, for example systems used by banks, intelligence organizations, pharmaceutical companies.

Initially, the hypothesis could be that these systems also suffer from the administrator vulnerability, but that they have tighter social and/ or physical control mechanisms in place.

Former communist systems

All examined systems were designed in the western world. Could there be a cultural or temporal bias in this? For example, we could contrast them with systems designed under communist rule.

The Great Firewall of China

How is this filtering system controlled? Who decides what is to be filtered and how? How do the censors know what must be censored if they cannot find it themselves?

Control over the entire Internet

How is the entire Internet governed? For example, who controls the domain name system (DNS), the protocols etcetera? How hierarchical is the entire Internet really?

18. References

18.1 Printed publications

Booch, G., Rumbaugh, J., Jacobson, I. (1999)
The Unified Modeling Language User Guide
Addison-Wesley

Brookshear, J. G. (1991)
Computer Science: an overview. 3rd ed.
Benjamin/ Cummings

Castells, M (2000)
The Rise of the Network Society Volume I
Blackwell Publishing

Custer, H. (1994)
Inside the Windows NT File System
Microsoft Press

Ferraiolo, D., R. Kuhn (1992)
Role-Based Access Control
National Institute of Standard and Technology
Gaithersburg, Maryland, presented at proceedings of 15th National Computer Security
Conference

Fowler, M. (1999)
Analyses patterns
Addison-Wesley

Ghezzi, C., Jazayeri, M., Mandrioli, D. (1991)
Fundamentals of Software Engineering,
Prentice Hall

Gijzel, F. (2003)
Content Management systemen, een onderzoek naar succesfactoren,
UvA, doctoraalscriptie

Huntington, Samuel P. (1998)
The clash of civilizations and the remaking of world order
Simon & Schuster

Jaspersen et al. (2002)
Power & IT Research
MIS Quarterly Vol. 26 No. 4, pp. 397-459

Jong, M.-J. de (1997)
Grootmeesters van de sociologie
Boom

Lipsky, M (1983)
Street Level Bureaucracy
Russell Sage Foundation

Lukes, S. (1974)
Power: A Radical View
Macmillan: London

Macionis, J., Plummer, K. (1997)
Sociology - a Global Introduction
Prentice Hall

Metselaar, C. (2000)
Sociaal organisatorische gevolgen van kennistechnologie
SIKS

Mintzberg, H (1992)
Structure in Fives: Designing Effective Organizations
Prentice Hall

Neuman, Lawrence W. (2000)
Social Research Methods 4th edition
Allyn and Bacon

Nolan, R. L. (1979)
Managing the Crises in Data Processing,
Harvard Business Review, April/ March

Orlikowski, Wanda J., Iacono, Suzanne C. (2001)
Research Commentary: Desperately Seeking the "IT" in IT Research--A Call to Theorizing the IT
Artifact
Information Systems Research, Vol. 12, Issue 2

Roe Smith, M., Marx, L. (1994)
Does Technology Drive History?
MIT Press

Wood, D. (1987)
Theory of Computation
Wiley

Zuboff, S. (1985)
Automate/ Informat: The Two Faces of Intelligent Technology
Organizational Dynamics, Autumn

Zuurmond, A (1994)
De Infocratie: een theoretische en empirische heroriëntatie op Weber's ideaaltype in het
informatietijdperk
Den Haag, Phaëdrus

Gordon, M (1998)

Oxford Dictionary of Sociology
Oxford University Press

18.2 Online publications

Berners-Lee, T (1989), Information Management: A Proposal, from *World Wide Web Consortium (W3C)*, Retrieved August 18, 2005, <http://www.w3.org/History/1989/proposal.html>

Brisset, P. (2001) A 3D animation of Linux source code development, Retrieved August 18, 2005, <http://perso.wanadoo.fr/pascal.brisset/kernel3d/kernel3d.html>

Chandler, D (2005) Technological or Media Determinism, Retrieved August 18, 2005, University of Wales, Aberystwyth, Media
<http://www.aber.ac.uk/media/Documents/tecdet/tecdet.html>

CivicSpace (2005) Retrieved August 28, 2005, <http://civicspacelabs.org>

DARPA Grand Challenge 2005, Retrieved August 28, 2005, <http://www.grandchallenge.org/>

Drupal (2005) Retrieved August 28, 2005, <http://drupal.org/>

Haque, A (2003) Information Technology, GIS and Democratic Values: Ethical Implications for IT Professionals in Public Service,
Retrieved August 18, 2005, University of Alabama at Birmingham
http://images.main.uab.edu/psychology/gps/GIS_Ethics_Haque.pdf

Hartman Communicatie BV (2005) Retrieved August 18, 2005, <http://content.hartman-communicatie.nl>

Huijzer, D, Peer, P van, Pol, Huub van de (2005) *Wegwijzer in de Uitgeverij*
Retrieved August 18, 2005,
<http://www.wegwijzerindeuitgeverij.nl/wegwijzerindeuitgeverij/pdf/Wegwijzer-H1-7.pdf>

Infrae (2005) Retrieved August 28, 2005, <http://www.infrae.nl/>

R. Kurzweil, B. Joy, Recipe for Destruction (2005) Retrieved April 8, 2006,
<http://www.nytimes.com/2005/10/17/opinion/17kurzweiljoy.html?ex=1287201600&en=29351015130c0ebf&ei=5090&partner=rssuserland&emc=rss>

Lost in Usenet – References (2003), Retrieved August 18, 2005, from *Internet FAQ Archives*,
<http://www.faqs.org/usenet/>

MediaWiki (2005), Retrieved August 28, 2005, <http://en.wikipedia.org/wiki/MediaWiki>

Microsoft (2005), Retrieved August 28, 2005, <http://www.microsoft.com/>

Network Working Group (1993), RFC 1436 - The Internet Gopher Protocol, Retrieved August 18, 2005, from *Internet FAQ Archives*, <http://www.faqs.org/rfcs/rfc1436.html>

Open Source Initiative (2005)
Retrieved August 28, 2005, <http://www.opensource.org>

phpBB (2005) Retrieved August 28, 2005, [http:// www.phpbb.com/](http://www.phpbb.com/)

Princeton University (2005) Wordnet
Retrieved August 19, 2005, [http:// wordnet.princeton.edu/ perl/ webwn](http://wordnet.princeton.edu/perl/webwn)

Project Minerva (2005) Retrieved August 28, 2005, [http:// www.project-minerva.org](http://www.project-minerva.org)

TYPO3 (2005) Workflow description
Retrieved August 27, 2005, [http:// TYPO3.org/ development/ projects/ workflow-
engine/ page/ 2/](http://TYPO3.org/development/projects/workflow-engine/page/2/)

Wikipedia (2005)
Retrieved August 27, 2005, [http:// en.wikipedia.org/](http://en.wikipedia.org/)

Zawinski, J. (2005) Groupware Bad. Retrieved July 25, 2005,
[http:// www.jwz.org/ doc/ groupware.html](http://www.jwz.org/doc/groupware.html)

Zope (2005) Retrieved August 28, 2005, [http:// www.zope.org/](http://www.zope.org/)

18.3 Interviews & conversations

Benjamin, J. (2005). Interview with the author on June 28, 2005. Rotterdam. [Notes in possession of author]

Faassen, M. (2005). Interview with the author on March 10, 2005. Rotterdam. [Notes in possession of author]

Govers, V. (personal communication, July 20, 2005)

Kraaijeveld, J. (personal communication, July 12, 2005)

Linden, A.J. van der. (personal communication, July 11, 2005)

Maan, I. (2005). Interview with the author on June 21, 2005. Rotterdam. [Notes in possession of author]

Tzoumas, D (2005). Interview with the author on July 10, 2005. Rotterdam. [Notes in possession of author]

18.4 Software

Name	Version	Retrieved from URL
Drupal	4.6.0	http:// www.drupal.org/
DSpace	1.2.2	http:// www.DSpace.org/
Mambo	4.5.2.3	http:// www.mamboserver.com/
MediaWiki	1.4.0	http:// sourceforge.net/ projects/ wikipedia/
Movable Type	3.16	http:// www.sixapart.com/ movabletype/
PhpBB	2.0.11	http:// www.phpbb.com/
Silva	1.1	http:// www.infrae.com/
TYPO3	3.8.0	http:// www.TYPO3.com/

Appendix A: Tables

Table 2-1: Research lenses.....	13
Table 4-1: Structural model of technology.....	21
Table 4-2: Causal model of technology.....	21
Table 5-1: Schematic overview of a program consisting of different features.....	23
Table 5-2: Logical application tiers.....	23
Table 5-3: Complete architecture	24
Table 7-1: Hierarchical content inside a database text.....	34
Table 7-2: Example of a feature user matrix	35
Table 7-3: feature-user matrix with 2 features and 1 user	35
Table 7-4: Feature-user matrix using roles	35
Table 7-5: Hierarchical role structure (role-feature matrix)	36
Table 8-1 Examples of authoring and publishing systems (Wikipedia, 2005).....	38
Table 9-1: Expected hierarchical score	45
Table 10-1 Types of Non-probability samples.....	46
Table 10-2 Overview of cases	48
Table 11-1: Interpretation of control structure scores.....	52
Table 11-2: Actual applications scores on control structure	53
Table 11-3: Interpretation of workflow scores.....	53
Table 11-4: Actual applications scores on workflow	53
Table 11-5: Actual applications scores on content structure	54
Table 11-6: roles and features of MediaWiki.....	56
Table 11-7: Role distribution at organization P.....	57
Table 11-8: Role distribution at organization Q	57
Table 13-1: Examined programs and organizations	60
Table 13-2 Website phases.....	64
Table 14-1: Example of feature user matrix.....	66
Table 14-2: Example of cooperation matrix	66

Appendix B: Figures

Figure 1-1: Two approaches for the study of ICT	8
Figure 3-1: Graphical representation of the changing structure of a software program	16
Figure 3-2: Correspondence between social and digital processes	17
Figure 5-1: Structural model of ICT	22
Figure 5-2: Graphical representation of the software development lifecycle	24
Figure 7-1: Role based access control.....	31
Figure 7-2: Mesh of unconnected nodes.....	31
Figure 7-3: Graph	32
Figure 7-4: Hierarchical graph with root.....	32
Figure 7-5: Fixed three-layer hierarchy	32
Figure 7-6 Unlimited depth hierarchy.....	33
Figure 7-7: Workflow as feature-user matrices in time.....	36
Figure 8-1: proposal by T. Berners Lee.....	41
Figure 8-2: Alternative (hierarchical) representation of the World Wide Web	42
Figure 11-1: Drupal installation	55
Figure 11-2: DSpace installation	55
Figure 11-3: Mambo installation.....	55
Figure 11-4: MediaWiki installation	55
Figure 11-5: Movable Type installation.....	55
Figure 11-6: phpBB installation	55
Figure 11-7: Silva installation.....	56
Figure 11-8: TYPO3 installation.....	56
Figure 14-1: Monolithic hierarchical system	67
Figure 14-2: Multiple hierarchical systems with communication between them.....	67
Figure 14-3: Multiple systems without direct communication	67

Appendix C: Database modeling

Database diagramming techniques

In ICT various diagramming techniques are used to illustrate and document how programs are structured. On such structure is the database structure: The part where all the data is stored that is used by (and/ or created by) an application. There are various database types, but the *relational* database is most popular. Logically, it stores everything in tables, below is a small example representing two tables “Publisher” and “Book”.

Publisher	
ID (Primary key)	Name
AW	Addison-Wesley
PH	Prentice Hall

Book			
ID (Primary key)	Name	Author	Publisher (Foreign key)
CS	Computer Science: an overview.	Brookshear	AW
SOC	Sociology, a Global Introduction	Macionis, & Plummer	PH

Table C-1 Tables Publisher and Book

Each table has a unique field called the *primary key*. Other tables can *reference* this field with a so-called *foreign key*. In the example a book *references* a publisher.

A common notation is the crowfoot notation:

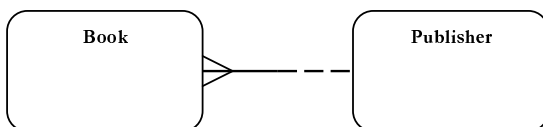


Figure C-1 Example of crowfoot notation

This notation is used in the next sections and is interpreted as follows:

- From left to right: a book has only one publisher
(Going from the *three lines* at the book you end up with *one* publisher)
- From *right* to *left*: a publisher can publish multiple books
(Going from the *single line* at the publisher you end up with *multiple* books)

Appendix D: Structure

Control structure

Drupal

The only principal in Drupal is the user. A user can be in one or more roles. Roles can be granted permissions, based on those permissions users can perform certain actions. The installation defines two basic roles but it is possible to add new roles.

Role	Definition
Anonymous user	A user that has not identified itself
Authenticated user	A user that has logged in

Table D-1: Roles in Drupal

The installation procedure creates one account. The first user of a Drupal site automatically receives all permissions, no matter what role that user belongs to.

User	Permissions granted
First user	All permissions
All other users	Role based

Table D-2: Users and permissions in Drupal

DSpace

DSpace defined two basic groups:

Groups	Definition
Administrators	Can do anything in a site
Anonymous	Not identified

Table D-3: Groups on DSpace

It is however possible to create other groups.

Mambo

Mambo has two main hierarchies: one for access to the Front-end (so users can log in to the web site and view designated sections and pages) and one for Back-end Administration access. A user can only have one role.

Hierarchy	Role	Description
Front-end	Registered	This group allows the user to login to the Front-end interface.
Front-end	Author	This group allows a user to post content, usually via a link in the User Menu.
Front-end	Editor	This group allows a user to post and edit any content item from the Front-end.
Front-end	Publisher	This group allows a user to post, edit and publish any content item from the Front-end.
Backend	Manager	This group allows access to content creation and other system information.
Backend	Administrator	This group allows access to most administration functions.
Backend	Super Administrator	This group allows access to all administration functions.

Table D-4: Roles in Mambo

MediaWiki

The only principal in MediaWiki is the user. Users can have certain rights, based on those rights users can perform certain actions. The MediaWiki term 'right' is best understood as a role, for example a user has the role 'developer' or 'sysop'. Users can be in one or more roles.

Role	Granted by
User	All
Developer	-
Bureaucrat	Developer
Sysop	Bureaucrat

Table D-5: Roles in MediaWiki

Roles are hierarchical as the table indicated: a developer creates a bureaucrat creates a sysop. During the installation of MediaWiki a special user account (with the role of sysop) is created that can perform various maintenance tasks. (This basically means that the top-level roles bureaucrat and developer are not available via the user interface – they must be set in the database themselves).

Movable Type

The only principal in Movable Type is the user. Users can be either registered or non-registered.

User	Permissions granted
Registered user	Depending on each user
Unregistered user	Only option: allow comments

Table D-6: Users in Movable type

Registered users can be granted permissions, based on those permissions users can perform certain actions. The installation procedure creates one account with the user name *Melody* and the

password *Nelson* who has all permissions. Melody can create blog authors, who, in their turn, create yet other authors.

phpBB

Users can be granted the administrator role.

User	Permissions granted
Administrator	Administrator
Non-Administrator	Depending on other settings

Table D-7: Users in phpBB

The principals in phpBB are users and groups.

Silva

Silva has several predefined roles split into two hierarchies.

Action (feature)	Reader	Author	Editor	Chief Editor	Manager
Read, preview, Copy content	+	+	+	+	+
Create, edit, delete, Unpublished content		+	+	+	+
Submit for publication		+	+	+	+
Create editable version of published content		+	+	+	+
Approve, publish content			+	+	+
Define, change time frame			+	+	+
Close, delete published content			+	+	+
Create new editors, authors, readers, viewers				+	+
ZMI actions, add users, add External Sources, refresh content					+

Table D-8 Silva roles and permissions

Action	Everybody	Authenticated	Viewer	Viewer+	Viewer++
View public content	+	+	+	+	+
View authenticated content		+	+	+	+
View restricted authenticated content			+	+	+
View IP controlled content	Optional	Optional	Optional	Optional	Optional

Table D-9 Silva roles and permissions (continued)

TYPO3

TYPO3 is divided into a front-end and a backend section, with corresponding user types. Only content contributors are allowed access to the backend - the administration of the website. A special permission is the admin permission. Admin users have full permissions. Users can be a member of one or more groups and get their permissions from these groups. Groups are organized hierarchically, a group can be a member of another group, receiving all its parents permissions.

User	Permissions granted
Administrator	Full permissions
Non-Administrator	Depending on other settings

Table D-10: Users in TOPY3

Workflow

Drupal

Drupal has a moderation feature: If applied, users that lack a specific role must have their contents approved by others. Meanwhile the contents reside in a *moderation queue*. According to the documentation: The queue provides a way for your users to vote on submitted content. Users can moderate a post up (give it a point), or down (subtract a point). Several 'thresholds' give you control over how many points are required for the status of a post to be automatically changed.

Threshold	Description
Post threshold	When a post gets this number of moderation points, it is promoted to the front page automatically.
Dump threshold	When a post drops below this number of points, its status is changed to unpublished.
Expiration threshold	When a post gets this number of points, its status is changed to unpublished.

Table D-11: the threshold system of Drupal

DSPACE

DSPACE has a three-step workflow process, and is one of the few systems in which workflow is mentioned as a separate subject. From the manual:

A collection's workflow can have up to three steps. Each collection may have an associated e-person group for performing each step; if no group is associated with a certain step, that step is skipped. If a collection has no e-person groups associated with any step, submissions to that collection are installed straight into the main archive.

In other words, the sequence is this: The collection receives a submission. If the collection has a group assigned for workflow step 1, that step is invoked, and the group is notified. Otherwise, workflow step 1 is skipped. Likewise, workflow steps 2 and 3 are performed if and only if the collection has a group assigned to those steps.

When a step is invoked, the task of performing that workflow step put in the 'task pool' of the associated group. One member of that group takes the task from the pool, and it is then removed from the task pool, to avoid the situation where several people in the group may be performing the same task without realizing it.

The member of the group who has taken the task from the pool may then perform one of three actions:

Workflow Step	Possible actions
1	Can accept submission for inclusion, or reject submission.
2	Can edit metadata provided by the user with the submission, but cannot change the submitted files. Can

	accept submission for inclusion, or reject submission.
3	Can edit metadata provided by the user with the submission, but cannot change the submitted files. Must then commit to archive; may not reject submission.

Table D-12 DSpace workflow steps

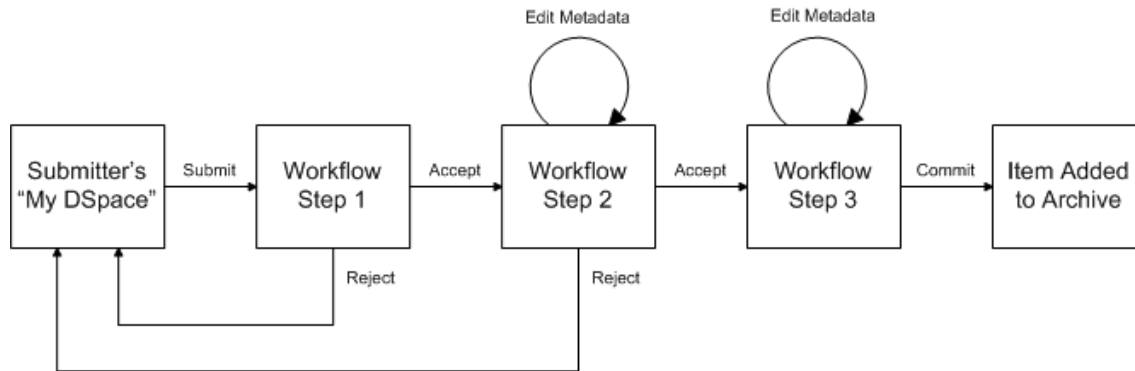


Figure D-1: Submission Workflow in DSpace

Mambo

In mambo, workflow is a two-step process: Authors and editors can submit content that is published by a publisher.

MediaWiki

MediaWiki has no enforceable workflow. Anyone with sufficient access can change any page anytime. It is possible though for administrators to protect pages (make them read-only) and change them upon request of other users. Another option is to add pages to a 'watch list', a list of pages that are tracked for changes by others. Users can periodically view this page to check for updates.

Movable Type

Movable Type has no real workflow. In fact, one of its main features is called QuickPost that enables 'one-click publishing'. However, an entry can be in one of three statuses:

Status	Description
Draft	Unpublished
Future	To be published at a future point in time
Publish	Published

Table D-13 Movable type publication statuses

It is also possible to keep track of new items on other sites and vice versa via a so-called 'track back' system.

phpBB

phpBB has no real workflow. It allows the user to get an email notification when a reply is posted to a certain topic. Furthermore it has a feature to automatically delete topic after a number of days after they have not been posted to (auto pruning). Somewhat incorrectly phpBB has a role called moderator - but there is no submission queue to be moderated. In fact the moderator is just a plain role that can be granted permissions for each forum.

Silva

Silva has a two-step workflow process:

Authors can submit content for publication, which must then be approved by an editor via his To-Do list. Each piece of content has a publication an expiration date. If content is approved, it

will become publicly visible between these dates. A content element can have different versions - only one of these versions can be published at any time.

A content element (and version) can be in one of five statuses:

Status	Description
Draft	The content is waiting for approval
Pending	The approval task is on some Editor's To-Do list.
Approved	The content has been approved, but is not published yet.
Published	The content is visible for the public.
Closed	Indicates that there is no new version and the public version is closed

Table D-14 Silva publication statuses

Silva also has an email subscription feature; visitors can subscribe and receive email notification when a part or parts of the site are updated.

TYPO3

Typo3 has very elaborate workflow features to allow for a variety of workflow models to be implemented. Specific workflows can be defined for existing or new content, for specific content types (page content, alternative page language, guestbook, news, internal note), assigning steps to users or groups. But in fact there are only two options for workflow:

[Editor] -> [Author] -> [Editor]

This involved two persons.

[Editor] -> [Author] -> [Reviewer] -> [Editor]

This adds a third person to the process.

Content structure

We will now link the *content* structure to the *control* structure.

Drupal

Drupal has several content types (called nodes), including stories, polls, blogs, forums and books. Nodes can be hierarchically ordered. The core Drupal application doesn't have ACL functionality but there are modules such as 'node privacy by role' (Drupal, 2005) to determine which roles can view or edit a node.

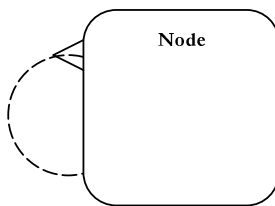


Figure D-2: Drupal node structure

DSpace

The basic content element is an item that consists of bundles and bit streams (files in fact). Items can be part of one or more collections that belong to a community. Communities can have sub-communities. Access can be specified at each level.

Access type for element
Read

Write

Table D-15: Access levels in DSpace

Mambo

Mambo organizes text in content elements. Content elements belong to a category and a section. A special feature is to put content directly on the front page. For each content element the access level can be specified.

Access level	Users
Public	All users
Registered	Registered user that is logged in
Special	Any user created as Author, Editor, Publisher, Manager, Administrator or Super Administrator is considered a Special User.

Table D-16 Access levels in Mambo

MediaWiki

MediaWiki is organized into pages; everything is grouped at the same level.

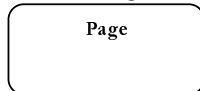


Figure D-3: MediaWiki page structure

It has also a category grouping system, to help users find things hierarchically (see for a live example <http://en.wikipedia.org/wiki/Category:Botany>) but this system allows to group items in more than one category: there is not necessarily a single hierarchy. Links are created at the text level.

Control of pages is done via 'page protection':

Page type	Description
Protected	Only users with sysop role can change the page
Unprotected	Everyone can change the page

Table D-9 Page protection in MediaWiki

Movable Type

The basic content element is an entry that is placed in a blog. People can comment entries. Entries can be organized into categories; an entry can belong to multiple categories but has only one primary category. Each user can be given different access rights for each blog:

User rights
Post
Upload File
Edit All Posts
Edit Templates
Edit Authors & Permissions
Configure Blog
Rebuild Files
Send Notifications
Create and Edit Categories
Edit Address Book

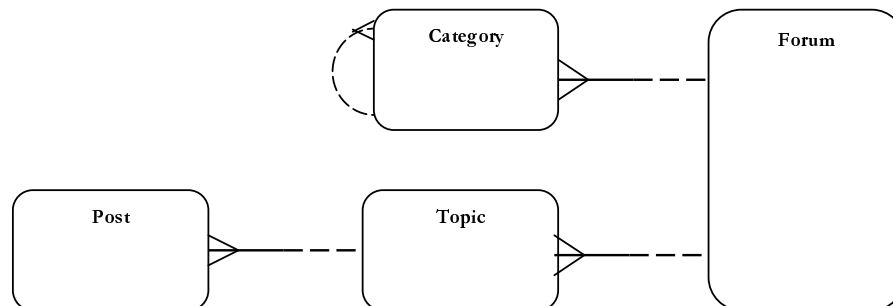
Table D-18 Movable Type user rights for blog

phpBB

The basic content element is a posting. There are several kinds of postings:

- a) Normal
- b) Sticky (before any other topics in the form)
- c) Announcement (even before sticky topics)
- d) Locked: only moderators can send messages to locked topics. Locked topic can also be unlocked

There's also the option to create a poll. Postings are organized into topics, topics in forums and forums into categories. Registered users can send private messages to each other that can only be read by the sender/ receiver.



Users can be granted permissions at the forum level. (They can also be granted permissions via the group to which they belong). For each forum, the easiest way is to use the simple 'permission levels'.

Permission level	Description
Public	Anonymous users can read and post. Registered Users can additionally edit their posts, and create and vote in polls. Moderators and administrators can make stickies and announcements.
Registered	Anonymous users can read the forum. Registered Users can additionally post, reply, edit their posts, and create and vote in polls. Moderators and administrators can make stickies and announcements.
Registered [Hidden]	Anonymous users may only register. Registered Users can read, post, edit their posts, and create and vote in polls. Moderators and administrators can make stickies and announcements.
Private	Non-Private users may only see the forum. Private Users can read, post, reply, edit their posts, and create and vote in polls. Moderators and

	administrators can make stickies and announcements.
Private [Hidden]	Only Private Users may see the forum. Private Users can read, post, reply, edit their posts, and create and vote in polls. Moderators and administrators can make stickies and announcements.
Moderators	Anonymous and Normal users can only see the forum. Moderators and administrators can read, post, reply, edit their posts, create polls, vote in polls, and make stickies and announcements.
Moderators [Hidden]	Normal users cannot see the forum. Moderators and administrators can read, post, reply, edit their posts, create polls, vote in polls, and make stickies and announcements.

Table D-19 phpBB simple permission levels

Next to the simple permission levels, users or groups can be given certain permission levels.

Permission levels	Description
All	Every user in the board is in this permission level. This level is used particularly to grant permissions to users who are not registered and/or logged in.
Registered	A user is in this permission level if he/ she is both registered at the board, and is currently logged in.
Private	There are two parts to being a private member of a forum. In the forum permissions, there must be at least one permission type set to the permission level PRIVATE. Additionally, in the User Permissions or Group Permissions panel, the user or group must be "Allowed Access" to the private forum (or have permission types set to "ON" in advanced mode.)
Moderator	Someone is in this permission level if they are a moderator of the forum.
Administrator	Board administrators (and no one else) are in this permission level.

Table D-20 phpBB group permission levels

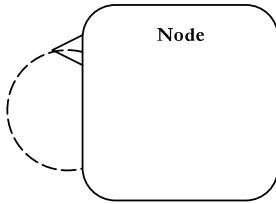
Notice that at the database level, there's an odd similarity between the forum application phpBB and the blog application Movable Type: They use the exact same content structure for different purposes. Even the access specification is at the same level: phpBB specifies access to a forum, Movable Type to a blog.

Application	Element 1	Element 2	Element 3	Element 4
phpBB	Category	Forum	Topic	Post
Movable Type	Category	Blog	Entry	Comment

Table D-21 Side-by side comparison of phpBB and Movable Type content structure

Silva

Every node in Silva is part of one hierarchical structure. Roles (as mentioned before) can be specified for each node. Users can be given local roles, meaning roles for a specific node.



The complete list of node types is given below and illustrates Silva's extensive features:

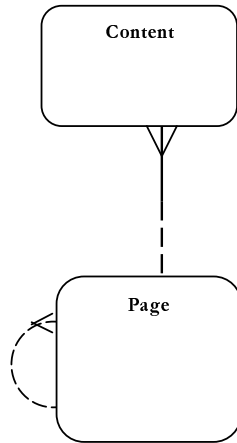
Accelerated HTTP Cache Manager	Silva Group
Browser Id Manager	Silva IP Group
DTML Document	Silva Image
DTML Method	Silva Indexer
Docma Service	Silva Layout Service
External Method	Silva Link
File	Silva Link Version
Filesystem Directory View	Silva Message Service
Folder	Silva Multi View Registry
Folder (Ordered)	Silva Publication
Formulator Form	Silva Renderer Registry Service
Image	Silva Root
Mail Host	Silva Sidebar Service
Page Template	Silva Simple Member
Parsed XML	Silva Simple Member Service
RAM Cache Manager	Silva View Registry
ReStructuredText Document	Silva Virtual Group
Script (Python)	Site Error Log
Session Data Manager	SiteRoot
Set Access Rule	Transient Object Container
Silva AutoTOC	User Folder
Silva CodeSource Charset Service	Version
Silva Container Policy Registry	Virtual Host Monster
Silva Document	Vocabulary
Silva Document Version	XMLWidgets Editor Service
Silva Editor Support Service	XMLWidgets Registry
Silva Extension Service	Z Gadfly Database Connection
Silva File	Z SQL Method
Silva Files Service	Z Search Interface
Silva Folder	ZCTextIndex Lexicon
Silva Ghost	ZCatalog
Silva Ghost Folder	ZODB Mount Point
Silva Ghost Version	Zope Tutorial
	kupu editor

Table D-15 Node types in Silva

Note that these are not all content related, for example the 'User Folder' or the 'kupu editor'. A special feature is the option to link an existing hierarchy onto another node (a ghost folder). Every user can be granted a role at each node (so called local roles).

TYPO3

TYPO3's text is organized into pages that contain content.



For each page, 5 basic permissions can be set for:

- The owner of the page
- The group of the page
- Everyone¹⁸

Permission	Description
Show page	Show/ Copy page and content.
Edit content	Change/ Add/ Delete/ Move content.
Edit page	Change/ Move page, e.g. change pagetitle etc.
Delete page	Delete page and content.
New pages	Create new pages under this page.

Table D-4 TYPO3 permission types

¹⁸ This is probably derived from the Unix/ Linux file system where each file has permissions for its owner, group and everyone else.

Appendix E: Causes

E.1 Design of Silva

Introduction

According to the website of Infrac (Infrac, 2005)

Silva is a powerful CMS for managing content for the web, paper, and other media. Content is stored in a clean and future-proof format, independent of layout and presentation. Features include a multi-version workflow system, integral WYSIWYG editor (Kupu), content reuse in multiple publications, sophisticated access management, extensive import/export facilities, fine-grained templating, and hi-res image storage and manipulation.

To understand how Silva was designed Martijn Faassen was interviewed. (Faassen, 2005) Together with Kit Blake he started the company Infrac a few years ago and Silva is one of their most important products. Their business model is open source and Silva is freely available from their website. Silva is built on top of Zope, a framework for content management systems.¹⁹

Content and authorization structure

Zope has a completely hierarchical database system that contains everything in the application, text, images, users, editors etc. This posed some challenges for Infrac, as there can be a mismatch between the underlying application structure, the authorization structure and finally the way things are presented to the user. For a particular customer they created so called *ghost directories*, a way to attach content to multiple locations in the tree, without duplicating it.

As they started creating Silva they did not choose the Zope authorization structure because it was deemed inadequate. Initially Zope did not have user groups; neither did it support local user roles. A user could only have a global role throughout the application (for example manager) but they wanted to add a context to the role, to give a user the option to be manager of only a part of a website. Silva started with the following roles: Reader, Author, Editor, Chief Editor, Manager and Viewer. On a customer's request they added the *Viewer+* and *Viewer++* roles. These can be used for fine-grained control of sensitive documents.

Workflow

In the interview, while talking about publication permissions and the fact that only one user has to authorize a publication, Faassen pointing out that the paper was actually about *workflow* and *groupware* features: Ways to prescribe how people collaborate on a given task. He was quite skeptical about this and thought it wasn't of much practical use. It's something that managers want when they buy something, but it doesn't help making actual users enthusiastic for the product (nor developers – not unimportant in the open source world), according to (Zawinski, 2005). It reminded of the *Viewer+* and *Viewer++* roles – his idea was that in practice they were never used. Not surprisingly, Silva has only modest workflow features. He also pointed out that Plone has a somewhat different workflow pattern compared to Silva (see the Silva section on Workflow): Create => Publish => Retract => Modify => Publish. With this structure, you can't edit a web page that's currently published; you have to retract it first.

User management

For some organizations identification is done with the help of another system. This way, Silva can use the username and password that users already have, instead of creating new logins. As it turned out this Zope feature wasn't fast enough and they had to rewrite this feature.

Overall development

Looking back at the development of Silva his idea was that it went very gradually. In roughly chronological way, Silva first had only users, followed by groups, IP address groups, audit trail features and version management. There were no new authorizations features planned.

¹⁹ Another CMS built on top of Zope is Plone

E.2 Design of Talmon CMS

Talmon CMS is a content management system created by Talmon Communicatie (www.talmon.nl) and is sold to customers for whom Talmon has built websites. Via email, developer Aart Jan van der Linden was interviewed over the development of this system. (Van der Linden, 2005).

As became clear, website development is not the core business for Talmon, their focus is on creating multimedia presentations. As such, website maintenance has a somewhat negative value - the main reason that they created the CMS was to let the customer update his own site, relieving them from a lot of work.

Nevertheless, the authorization system development was demand driven: if an organization would request a new feature it was built – but the overall idea was to keep it as simple as possible. Talmon once had the idea of implementing workflow – but this was never in demand and has never been realized. Different roles such as Editor and Chief editor were also deemed unnecessary: for most websites, there are only a couple of people who actually put information on them. Most of the time they have the same rights. There is however a difference between Editors and Administrators – Editors cannot ‘damage’ the site (delete pages, create dead links etc.) but administrators can. This allows less experienced users to participate. For one specific customer they created an extra role, in order to grant a specific user rights to edit the ‘Careers’ section of the website.

E.3 Design of EcoGrid

In this section we will discuss the development the EcoGrid application. EcoGrid is not strictly an authoring and publishing system. Unlike other applications in this paper it does not handle or produce written text, instead it deals with observations on the Dutch flora and fauna. Developed by the UvA (University of Amsterdam) it aims to create a joint platform for data collection and mining. Various organizations that gather information about plant and animal life in the Netherlands will provide the necessary input that – combined with other records such as soil usage and weather - will create an enormously valuable set of data with observations on thousands of locations and species, in some cases with a time span of a hundred years. In May 2005 Floris Sluiter (one of the developers) asked me (AvC) to contribute to the design of the authorization scheme; this helped to get a case study. At this stage of the development there was only a prototype. Its main purposes were to

- Find and solve technical problems before the actual program was being built
- Show the possibilities to potential participants

Not much time had been spent on authorization issues yet and the proposal was accepted. ECOGRID seemed to be a very interesting case for a couple of reasons, also because it deviates from the previously described systems.

1) Strict procedures need to be implemented:

- For data entry: Any observation done in the field should be validated to prevent database corruption.
- For data retrieval:

It is not desirable that anyone can access the database at will. If a very rare bird or plant is sighted this might attract lots of people wanting to observe it for themselves, or possibly try to take it with them.

Furthermore, all participating organizations are still the legal owners of the data they collected and want - if possible - to be paid for it's usage.

However, the entire set of data is not owned by a specific organization, creating a lot of issues with regard to the overall management of the system.

2) The authorization model that is needed to fulfill the above needs is more complicated

It involves more than one organization, creating inter-organizational and organization-person relations. An organization can grant certain rights to other organizations, for example to perform certain queries on the database. A person can be an employee of an organization but only a sponsor of another.

The question of “what can someone do with this data” could therefore be determined by:

- The role that one fulfills within your organization
- The organization’s rights with regard to the data

Next an overall introduction to the system was given; its goals and requirements. During a couple of months a conceptual authorization scheme was developed.

Application architecture

In advance it was decided that each organization should be given its own database to store its data. This seemed logical because they were the legal owner. If it - for one reason or another - would decide to step out of the program it could be given back its own data. It also allowed the organization to appoint its own administrators for maintenance purposes.

User management

A complication was the user management. A significant part of all users would need access to databases owned by different organizations. Having each organization maintain its own users would lead to data duplication (with all the problems of having to remember multiple passwords, inconsistent address data etc.) Therefore it was decided that there should be a central database that would store the users.

This would make the registration process easier but created other problems at the same time:

First the user management is now centralized and becomes critical for upholding the authorization scheme of the system: but who is responsible for its maintenance?

Secondly: How would users be identified? For example if a user J. Doe was registered centrally how would the participating organizations know that J. Doe really was J. Doe and not someone else? If possible, a direct physical link has to be made between the *account* of J. Doe and the *user* J. Doe.

Each organization should be given several ways of identifying each person individually, either by phone, email or home address. If a John Doe would be registered centrally and ask permission to view certain data, the owner could look into John’s records, and identify John by phone as a volunteer for a project. This successful authentication would be registered in the central database and be visible to other organizations, and based on this information, allow other organizations to make judgments about J. Doe’s reliability.

Authorization requirements

There were a couple of basic requirements for the overall authorization scheme:

- (1) It should be simple enough to allow users with limited IT skills to perform basic authorization tasks.
- (2) Minimal centralization. If it was feasible to do something at a local level it should not be centralized. (It should adhere to the subsidiary principle)
- (3) To reduce overall system complexity (and thus development and maintenance costs) authorization features should be implemented in the same manner for all organizations and for all items stored in the database.
- (4) It should allow for optimal data integrity (data entry) and protection (data retrieval).

Obviously these requirements bite each other: Dealing with authorizations in a uniform way (3) contradicts (2). To make matters more complicated, if an organization has its own database should it be given its own administrative password for maintenance? This could compromise (4) while reflecting the legal facts. (To solve this problem it was suggested to give each organization a closed envelop with the administrator password, automatically expiring all guarantees for proper data entry and retrieval as soon as someone used it!)

Workflow

No one can just enter data into EcoGrid; there are strict rules that must be followed.

For example, if a certain type of bird is sighted at a given location the likelihood of the observation is checked against the database with previous observations and corrected for seasonal influences etc. If it's a rare bird further data must be filled in by the observer, before someone else can validate the observation. This improves the overall validity of the data, while at the same time restricting users in their way of working. Observations are done in visits of a particular area, resulting in a form that contains all species that are either observed or not. Forms are part of larger projects that can span years.

Forms can have four types of species:

- Species that must be counted (with a significant 0 meaning none was sighted)
- Species that can be counted (with a 0 meaning none was counted)
- Species that should not be counted
- Free fields for other species

Audit trail

Typically no data should be removed from the system because this would break the audit trail. An observer could suddenly realize that for the past 5 years he has mistakenly identified bird species X for species Y. In that case the system should be able to come up with all observations of this person, correct them, make an entry in the log and preserve the invalid data.

Proposal

Based on all this information a small prototype was created that showed how the authorization structure could be implemented. Users were given roles at two levels:

- On the organizational level
E.g. employee, observer, administrator
- On the level of a particular item (project, form, observation)
E.g. owner, observer, project leader

The roles on the organizational level were a function

$$F(\text{action, role}) \Rightarrow \{\text{yes, no}\}$$

Depending on their role users could request a particular set of data or perform a certain action. An administrator can get a list of all projects or create a new project. An employee can view a list of all other employees. For the roles at the item level hold that the possibilities also depend upon the state of the item itself (a form of workflow). If a project is active, a project leader can add a form to a project, but not if it's status is closed.

Some other rules may also apply, for example a user can see data because his employer has paid for it. This results in a function

$$F(\text{item, action, status, role, other rules}) \rightarrow \{\text{yes, no}\}$$

This forms a generic structure that can be used throughout the application. The disadvantage of this solution was that the definitions of all these functions had to be defined before the system was used and it would be difficult to change while it was being used. In most situations it would be possible to change data via the user interface, there would typically be no need for an administrator to bypass application security and change something in the database itself. Changes would always be logged. The other option would be to create a non-generic solution and implement authorizations on the fly. This results in a shorter development time before deployment, but might create loops in the system that would be difficult to fix and also require an administrator to fix things in the database.

E.4 Usage of Silva at Erasmus University, Faculty of Social Sciences

The Erasmus University (www.eur.nl) uses the software package Silva for its website. On June 21st, interviewed Ilse Maan was interviewed, the webmaster of the Faculty of Social Sciences (FSW) (Maan, 2005)

Content and site structure

The first topic that we discussed was the structure of the website itself. Although the Erasmus University as a whole uses Silva, some parts of it do not. For example the Faculty of Economic Sciences (FEW) uses its own application and so does Psychology. User can also have their own website, e.g. R. Veenhoven with the *World database of Happiness*. These sub-sites are 'hung' into the Silva tree of items.

There are some general agreements about the structure of the FSW website – it should follow the guidelines set by those who manage the top Erasmus University website. The central automation department of the EUR does the maintenance.

Identification

People can login onto Silva via their universal Erasmus Account called ERNA, which is a separate system; this allows users to remember only one password.

Authorization

The role distribution for FSW is as follows:

Role	Users	Cumulative
Manager	1 (Maan)	1
Chief editor	1	2
Editor	4	6
Author	8	14

Table E-1 Role distribution for FSW

Statistics are for FSW only: Maan can only access the Social Sciences part of the website. Note that Silva allows local roles: you can be an editor in one section of the site and a manager in another one, only the highest roles are listed; the cumulative column indicates the effective roles: A manager is automatically a Chief editor. The Viewer+ and Viewer++ roles are not used. Note that the overall majority of FSW employees (100+) do not have access to Silva.

Workflow

The I&A staff creates web pages. People can request the creation of a new page. The webmaster, sometimes advised by the web editors, makes the decision. No one has access by default – people can request access to perform certain tasks. First they get the role of author and if they appear to know how to use it they are promoted to editor. This means that they can publish content directly.

Cooperation usually takes place in Maan's room, sometimes people email text for placement onto the website. She would like more people to use the system. It would save her a lot of work if people would maintain their own pages. Furthermore it would be good for the University as a whole if the web presence of the well-known staff persons was improved.

Page responsibilities are divided by theme, for example Education and Research. Each member of the Web editors has a special theme to look after.

The information officers are responsible for the main pages. She manages those for FSW, adds announcements and press releases, normally once a day. There is no replacement for her – if she's on holiday old data remains on the site.

Before the usage of Silva the site has been hacked but this is long since. There is no real censorship; the only agreement is that the employees' home pages should not be too personal. No one was ever banned from using the site.

History

The website of the EUR is about 10 years old. Until Silva in 2002 was introduced, they used Notepad and FrontPage (flat HTML files). FSW didn't want to maintain its own webserver and agreed to use the central managed Silva system. The deployment of Silva was slowed down by performance problems. These have been solved now, but limited the widespread usage and the creation of new 'value adding' features. Now that the 'window of opportunity' is passed the central automation department started looking at alternatives.

E.5 Usage of Open Market CMS at NRC

The Dutch newspaper NRC uses Open Market CMS (hereafter OCMS) as a content management system. On June 28th 2005 Jan Benjamin was interviewed, the chief editor of the NRC Internet edition. (www.nrc.nl) (Benjamin, 2005) He manages the daily website affairs and reports to the main editing board (*the hoofredactie*). He also writes articles himself.

Content and site structure

A lot of features are available via the website, most of which are actually separated systems, built apart from Open CMS. There's a web quiz, blogs, an archive and a reading club (see below). The site maintenance is outsourced to Pink Roccade, a company that also maintains websites for other newspapers that are owned by PCM (including *de Volkskrant* and *het Algemeen Dagblad*). OCMS is a front-end itself for Coyote, an application used to author and publish the paper newspaper.

The policy is not to throw anything away so that the URLs are preserved. If someone saves an article it should still be accessible from that URL after a couple of years. The program that is used to register people on the website (whether they have a subscription to the real newspaper or not) is separated from OCMS. Approximately 70.000 unique visitors visit the site each day.

Authorization

NRC currently employs about 200 editors. Of those only ten people can access OCMS. Five are the real Internet editors (including Benjamin) and five others who maintain files and lookup information support them.

According to Benjamin, using OCMS properly requires a lot of technical skills. If more people were given access it would be questionable if someone would make use of them. It would be nice though; if for example foreign editors would maintain the pages with foreign news. This is not the case at the moment.

NRC gets articles from all sorts of sources. Some of these sources – such as freelance editors – do not want their articles published on the Internet. These articles are filtered out during the export from Coyote to OCMS.

It is always possible to change data. If an error is detected (say a typo) the policy is to fix it as soon as possible. There is no revision system – previous versions could be lost (note that in most circumstances a copy is preserved in Coyote).

All five editors have the same authorizations, but at any time, one person has the overall responsibility for the website.

The site has not been hacked for a long time, a few years ago something was 'changed' in files on the Middle East.

Workflow

There's a daily publishing cycle, in line with the paper edition. At 13:30 the paper edition is finished for printing. About 12-15 articles are selected for publication on the web. Links to external sources and NRC web archives are added. Web publication has to wait until 16:00 (shortly after this time the paper edition is delivered to the subscribers). Between 13:30 and 16:00 changes can still be made to make the edition as up-to-date as possible.

History

The NRC has a web presence since 1995. They first used flat HTML files before switching to OCMS in May 2001. (A decision made by PCM). It was seemed as an improvement, but maybe things were simpler before OCMS. PCM is currently planning the usage of a new system that combines several features that are now split between Coyote, OCMS and other applications. The paper edition, website, archive and layout functions will all be done from one integrated system.

Reading club (*de leesclub*)

Every few months NRC compiles a list of books that people can read and discuss on a separate part of the website, based on phpBB. A subscription to NRC is not required, one only has to register. The phpBB censorship feature is on; messages are checked, but always afterwards. There have been few problems with this procedure. Some people write very long articles –these are removed. It helps if you ask direct questions rather than general comments.

E.6 Usage of phpBB at HeliportOnLine

In the centre of Rotterdam, owners and tenants of an apartment block called *Heliport* have organized themselves on the Internet. The website www.heliportonline.net serves as a means of communication and contains a lot of useful information about Heliport in general. On July 10th Dimitri Tzoumas was interviewed (Tzoumas 2005), the webmaster. He has been living there for several years and is currently active on the board of HBB (*Heliport Bewoners Belangen*), the uniting organization for tenants and owners. About a thousand people live in Heliport. At the time of the interview, 67 people were registered at the website.

Content and site structure

The application uses CuteNews for some web pages, but most content is located in a forum built using phpBB. Within phpBB there are nine separate forums, divided in three categories. The board of HBB and Dimitri have access to webserver, they can access all content.

Authorization

Dimitri, Peter van Amen, and the HBB board are administrators in phpBB. There are no private forums. All messages are readable, but the user HBB moderates four forums. Dimitri and Daan Schultz use this account together to post messages that should appear to originate from the board. Users can post anonymous messages, censorship is on but limited.

Site abuse does happen, phpBB is quite common on the Internet and there are scripting tools that allow malicious users to post advertisements on the site. Dimitri blocks their usernames and IP addresses and this solves the problem temporarily. Unsigned messages of scolding users are also removed. A problem with phpBB is that it's hard to figure out what a user actually can do on the forum.

Workflow

Dimitri gets many emails whose texts he posts under the HBB account, but most people post messages onto the forum themselves.

History

In 2002 Dimitri was a member of the communication committee of HBB and started building the site, together with Peter van Amen. Dimitri chose phpBB because he had previous experience with this application.