Topic Paper presented at the Seventh IFIP World Conference on Computers in Education.WCCE2001, Copenhagen, Denmark, 2001.

Changing Code - Notes on the Interdependency Between Informatics and Society

Carsten Schulte, Didactics of Informatics, University Paderborn, Germany
Universität Paderborn, FB 17, Fürstenallee 11, 33102 Paderborn
Germany. Phone: (049) 5251/60-6340, E-mail: carsten@uni-paderborn.de

Keywords: Didactics, Higher Education, Catalyst for Adoption/Change, Social, Teaching Methods

Abstract:

For some kind of software, change is an inherent feature. Such a software is embedded in a surrounding context of use. According to Unesco/IFIP curriculum from 1994 the typical context changes as the development shifts from automation phase, informatics phase to communication phase. The latter is the reason why in cyberspace—as Lessig puts it—code becomes law and regulates behavior. This has implications on the way software *is*, the way it is constructed, and it has implications on the surrounding environment. To know this is an essential key in order to understand how Information and Communication Technologies (ICT) and society are in an interdependent process of change. This concept is in contrast to a so-called technological determinism and is topic of a *system oriented didactics of informatics*. To build a bridge between the insights won through the observation of a large software system over a period of several years and the reduced examples which can be used at school the method of *deconstruction of an informatics system* is supposed. Deconstruction puts a piece of software, that was transported into the classroom, virtually back to a typical or original environment, in order to show the interdependencies. Attention widens from the construction of an algorithm and the construction of a software 'from scratch' to the evolutionary process in which software is constantly changed. Thus young persons can develop an understanding of the interdependencies between ICT and society as well as an understanding of different roles of ICT. The term deconstruction points out the embodied values, the connected practices of use and change which can not be found directly in the code, which can not be reconstructed through a whatever sharp analysis. Instead, they are result of certain views, opinions, judgments and interpretations, hence must be deconstructed. Methodically, deconstruction means primarily re-engineering and assessment of a given software; therefore object orientation seems especially suitable for such a method.

An Example for the Role of Information and Communication Technologies (ICT)

In 1999 Lawrence Lessig, Professor of Law, published 'Code and other laws of cyberspace', a study about how the internet may be transformed from a free, unregulable place to a regulated market-place. He argues, that the commercial interests of the big players are leading them to alter the software of the internet. Among other examples he compares the use and evolution of different online-communities (chapter six ‘Cyberspaces’, pp.63-84). He ranks the software systems by one dimension, “their amenability to outside control” (Lessig, p. 82). “[A]s we move from .law.cyber to CC to LambdaMoo to AOL, the ability to enforce a norm on the group from the outside increases. [..] Our look at these contrasting spaces should give depth to the idea that architecture matters and highlight different ways in which the code of a cyberspace might

enable or disable certain forms of life" (Lessig, p.83). The point—Lessig states—is, code is law (chapter one 'Code Is Law', pp.3-8): "Too many miss how different architectures embed different values, and that only by selecting these different architectures—these different codes—can we establish and promote our values" (Lessig, p.58).

Before developing conclusions for the study of informatics I want to comment the debate about that book (see e.g. slashdot). The discussion centers around law, policy and commercial influences. It seems appropriate to keep these issues in mind while forming oneself an opinion whether Lessig's concerns are realistic or not. In other words, if one wants to discuss the interdependencies between technology and society one can not reduce the debate to technological questions. Lessig notices: "It is not important that you understand the underlying technologies, though it would be great if more people did. What is important is that you get a hint of the purposes to which these technologies can be turned, and the consequences of their power" (Lessig, p.36).

From the view of an informatics expert Lehman also discusses changing code: "[E]volution is an intrinsic, feedback driven, property of software" (p.1067). Due to the 'softness' of software changes are easily, and due to certain pattern or laws he identifies, changes are frequently made (see Lehman). So the task for software engineering is, as Lehman argues in 1980: "Programs must be made more *alterable*, and the alterability *maintained* throughout their lifetime. The change process itself must be *planned* and *controlled*" (p.1061). The "Laws of software evolution" are deeply connected with Lessig's observations on the internet. Both aim at a special kind of software that Lehman calls "e-programs" (in later articles he uses the term 'e-type'). In short, "they are programs that mechanize a human or social activity". The consequence is, that the program becomes "*a part of the world it models*, it is *embedded* in it". This causes a feedback loop: The original situation the program was designed for has changed, so that the program is likely to be changed in order to fit—in whatever sense— into the new situation. But when the new version is installed the situation has changed again .. (see Lehman, pp.1062f).

So the first meaning of 'changing code' is here: For a special kind of software, the 'embedded' one, change is an inherent feature. This has implications on the way software *is*, the way it is constructed, and it has implications on the surrounding environment. To know this is an essential key in order to understand how ICT and society are in an interdependent process of change. Secondly, it is useful knowledge for the informatics and software engineers constructing software on the one hand and also for the users of software and—maybe more passive—members of the changing society on the other hand.

Interdependencies

Which aspects of informatics should be studied in order "to prepare young people to play a full role in modern society and to contribute to wealth creation" (Unesco94, p.5)? The Unesco/IFIP curriculum from 1994, for example, mentions the point 'C9 Social and ethical Issues' which is divided into three sub-objectives:
"1. the benefits and drawbacks of computer use to society in general;
2. the economic advantages and disadvantages of the use of computers;
3. the ethical questions which have arisen as a result of computer use"(Unesco94, p.41).
This is an accepted view, in which often terms like 'impacts on society', 'chances and dangers' tend to suggest a conceptualization of the relationship between technology and society that can be called technological determinism (see Chandler). It means that technology is causing impacts on society. Technology thereby becomes an active force changing a passive society. Alternatively one could describe the role of social groups—think of the mentioned big players by Lessig—as the driving force directing technological development. In the extreme form, this

could lead to “study the impacts of society on technology” (Pannabacker, p.2). Therefore, Pannabecker (p.2) concludes: “The immediate task is not, [..] to find a single alternate metaphor but to recognize that there are different ways of approaching the study of technology and society” (Pannabecker, p.8).

It seems useful to conceptualize the relationship as an interdependent one. Lessig, elegantly avoiding the term society, sees four different modalities directing (individual) behavior: Law, norms, the market and architecture. We focus on architecture, described as “the way the world is” or the “built environment”. Some constraints of the architecture may be absolute, like physical “limit to the speed at which we can travel”, others may be certain features of a software: “But whether absolute or not, or whether man made or not, we can consider these constraints as a single class—as the constraints of architecture, or real space code” (Lessig, pp.235f). While the four modalities are different, they may be seen all the same from an individual perspective and experience. Lessig stresses the importance of seeing the interdependencies between the different factors: “if we relativize regulators—if we understand how the different modalities regulate and how they are subject, in an important sense, to law—then we will see how liberty is constructed not simply through the limits we place on law” (Ibid. p.239).

In an ICT curriculum only one part of these four modalities—the code of software—is subject to study. So, if it is true that different influences are experienced as only one (see Lessig, p.237), then a certain way of studying the above cited sub objectives could lead to a certain mental model, in which the different modalities are conceptualized as one technological modality, a mater of code. Such a model would be supported by the cited terms suggesting a kind of technological determinism. By suggesting technology as a force leading to changes in society, this deterministic view deepens and confirms the mental model, that all constraints and modalities can be summarized under ICT. Even worse, the intended aim to develop a thinking strategy often called 'algorithmic thinking' in order to improve 'problem solving skills' in this context shifts to a potential drawback. It also fits in the developing mental model, it also confirms and deepens it: The problems to figure out the ethical, economical and other general implications of computer use thereby might be 'solved' by describing simple cause-effect models, in which a certain ICT Infrastructure causes certain effects in society. In other words, the problem reduces to the problem to find the right (deterministic) algorithm predicting the resulting social, economical and ethical issues of a certain kind of computer use.

Such a mental model wouldn't allow to acknowledge different assessments of a given technology. With such a model it is nearly impossible to see a difference between the artificial and the natural. As Lessig (Draft, p.4f) puts it with regard to Cyberspace: “It is constituted by a set of code [..]. This code imbeds certain values; it enables certain practices; it sets the term on which life in cyberspace is lived, as crucially as the laws of nature sets the terms on which life in real space is lived. Now most think about this architecture—this code that defines cyberspace—as given. [..] as if it is simply defined. As if it has a nature, and that nature can't change. As if god gave us cyberspace, and we must simply learn how it is.”

Lehman says that requirements gathering, specification, design, implementation in and for the “E-situation” “involve[s] extrapolation and prediction of the consequences of system introduction”. For example, as users become familiar with a software system they will change “their behavior to minimize effort or maximize effectiveness” which leads to pressure for system change—the feedback loop of e-programs (see Lehman, p.1063).

So, the second meaning of changing code is to change the view upon the patterns of software construction and use in the studying of informatics. Attention should shift—or better: widen—from the construction of an algorithm in order to solve a problem and the construction

of a software 'from scratch' to the evolutionary process in which software is constantly changed so that young persons can develop an understanding of the interdependencies between ICT and society as well as an understanding of different roles of ICT.

System oriented didactics of informatics, deconstruction and object orientation

In section four (pp.17-21) the Unesco/IFIP curriculum describes different roles of ICT in a country. Development shifts from the “automation phase” and the “information phase” to the “communication phase”. Although these phases are primarily meant to describe the educational sector, they also describe the role of ICT in society. And as the curriculum “has been specified for countries in the information phase” (Unesco94, p.7) it is obvious that Lessig's “Laws of Cyberspace” belong to the communication phase. In this phase “computers are in networks and use is characterized by collaboration between users and informatics is part of the essential infrastructure” (Ibid.). The ICT infrastructure has become part of everyday life so that it is possible to think of it like an architecture in the sense Lessig uses the term. So the above mentioned drawbacks in handling the 'Social and Ethical Issues' rather result from the changing role of ICT in society than from changes in informatics.

For young persons growing up in a society in the communication phase with informatics as an essential part of their life in private, at school and in economy the ICT infrastructure *is* given; and it *is* a natural part of the real world. So the primary objective of ICT education is a trivial one: to stress that software is man made instead of given or natural.

Here, the didactics of informatics faces two transfer problems: At school it is not possible to observe the mentioned effects of ICT directly. Therefore some complex software systems, their use and change over a longer period of time would have to be assessed. A concept is needed, which enables one to show and teach this with the help of an example with reduced complexity and in a shorter period of time, so that the pupils can transfer the learned insights upon real life systems. While this is a kind of near transfer, the second is a problem of general transfer to other subject areas: It is the question, how and to what extent informatics education could—and should—be a kind of a general technology education.

Finally, these are the questions and topics of the *system oriented didactics of informatics*.

The concept is based on theories like Ropohl's socio-technical systems: “The starting point for designing a theory of socio-technical systems is the observation that hardly anybody has a general understanding of the technical society; this applies to laypeople as well as to specialists. Particularly, engineers tend to ignore the social concerns of their work, and social scientists, on the other hand, do not know very much about technology and are reluctant to consider the artificial reality of technical objects. [..]So I take the systems model to describe both social and technical phenomena, persons and machines, the technization of society and the socialization of technology ”(Ropohl, p.66). This leads to the notion of (socio-technical) informatics systems. They consist of hardware, software, the practices of use, the users, .. - in short: the surrounding environment, in which the software is embedded. To build a bridge between the insights won through the observation of a large software system over a period of several years and the reduced examples which can be used at school the method of *deconstruction of an informatics system* is supposed. The term deconstruction points out

? the alternating sequences of engineering and re-engineering, hence construction and deconstruction;

? the embodied values, the connected practices of use and change which can not be found in

the code, which can not be reconstructed through a whatever sharp analysis. Instead, they are result of certain views, opinions, judgments and interpretations, hence must be deconstructed.

Deconstruction acknowledges that different, sometimes contradictory criteria has to be used to assess an informatics system and the embedded software. These criteria are in part software quality criteria. For example: correctness, efficiency; and connected with the inherent likeliness to change: readability (of the code), reusability, flexibility and extensibility; and connected with the surrounding environment: usability, appropriateness and productivity. But all these quality criteria do not question the means of the software, they see the aims of the software as given, as the given requirements from the customer. Here are different criteria possible: According to Cyberspace and Lessig's statements we could note the terms regulation, privacy and liberty. In other areas we might note 'automation and unemployment'. Deconstruction tries to show the changing role of ICT, the change of criteria to describe the role and the resulting interdependent development process of society and informatics. The way Lessig works is an example for these method: He *compare*s *different software architectures* by certain *criteria*. He interprets and compares resulting *use cases*. And throughout the chapter “is versus ism” one find reflections about the *changing code* building the Internet: The development or the *evolution* of the Net over time and by forthcoming *software versions*.

Although compatible with all common software development paradigms, deconstruction seems to be especially useable in conjunction with object oriented notions. This is the third and last meaning of changing code: To change code to object orientation. For our purposes, uses cases, re-engineering and visualization (UML) are the most interesting aspects of object orientation.

Use cases “describe the behavior of a system from a user's standpoint by using actions and reactions. They allow the definition of the system's boundary, and the relationships between the system and the environment” (Muller, p.95). While from an informatics viewpoint a human user, external hardware or an outside system are all seen as the same actors, we focus on human actors, so that use cases scenarios show how the software is used in the context of a given task.

This leads to the notion how the software constrains behavior. Describing use case scenarios for different environments shows that the assessment of software depends on the context—and on the chosen criteria. The role of deconstruction here is to put a piece of software, that was transported into the classroom, back to a typical or original environment. Therefore the use of videos, showing the use of the software in detail and excursions to firms are supporting teaching methods.

On the other hand this external view on use-cases has to be completed by the internal processes. One can use UML sequence diagrams, starting with an actor to show the internal processes of the software. Here, animation systems are worthwhile (see as an example Mr. Dobs, a graphical debugger and part of the Fujaba tool).

The notion of different environments, or new interests leads to re-engineering questions in order to adjust the software. Changing code therefore means to describe new use-cases—from the users external view and from the systems internal view. To change the design (or: architecture) of the system, visualization is needed. It promotes the discussing, understanding and changing of the architecture of the system.

Re-engineering therefore becomes a task where the software has to be fitted in a new situation or environment, which has to be understood. The relevant aspects and criteria has to be described. And at least, the software has to be altered according to the desired aims, which presupposes an understanding of the architecture of the given software. So, not really surprisingly, the object oriented software engineering serves as a good training unit for the

relevant cognitive skills required in order to see and evaluate the interdependencies: One has to find several different criteria, has to see tradeoffs and interdependencies between them, has to rank them according to their relevance.

Methodically, deconstruction means primarily the re-engineering and assessment of a given software. It depends on object oriented software systems with open source code suitable for use at school and it depends on visualization and re-engineering tools. Here the development just started, further experiences and empirical evaluations are necessary (see Hampel/Magenheim/Schulte).

References:

Chandler, David: Engagement with media: Shaping and being shaped. In: Computer Mediated Communication Magazine, February 1996. (http://www.aber.ac.uk/media/Documents/short/determ.html)

Fujaba: Fujaba: From UML to Java And Back Again. (http://www.fujaba.de)

Hampel, Thorsten; Magenheim, Johannes; Schulte, Carsten: Dekonstruktion von Informatiksystemen als Unterrichtsmethode – Zugang zu objektorientierten Sichtweisen im Informatikunterricht. In: Schwill, Andreas: Informatik und Schule, Springer 1999, pp.149-164.

Lehman, M. M.: Programs, Life Cycles, and Laws of Software Evolution. In: Proceedings of the IEEE, Vol. 68, No.9, September 1980.

Lessig, Draft: Lessig, Lawrence: Architecting for control. Draft 1.0, 2000 (http://cyber.law.harvard.edu/works/lessig/camkey.pdf)

Lessig, Lawrence: Code and other laws of cyberspace. Basic Books, New York, 1999.

Muller, Pierre-Alain: Instant UML. Wrox Press, Birmingham. 1997.

Pannabecker, J. R.: Technological impacts and determinism in technology education: Alternate metaphors from social reconstructivism. Journal of Technology Education, 1991,Vol 3, Nr.1 (http://scholar.lib.vt.edu/ejournals/JTE/v3n1/pdf/pannabecker.pdf)

Ropohl, Günter: Philosophy of socio-technical systems. In: Techné: Journal of the Society for Philosophy and Technology, Volume 4, Number 3, 1999. (http://scholar.lib.vt.edu/ejournals/SPT/v4_n3pdf/ROPOHL.PDF)

Slashdot: Michael Sims: Review of Code and other laws of cyberspace. (http://slashdot.org/books/99/12/05/1637235.shtml)

Unesco94: Unesco/IFIP 1994: Informatics for Secundary Education. A Curriculum for Schools. Unesco, Paris, 1994.