THE NEED FOR AN APPLIED COMPUTER ETHICS HANDBOOK

Daniel Salber
CLIPS-IMAG, University of Grenoble
B.204 / BP 53
38041 Grenoble Cedex 9
France
Email: daniel.salber@imag.fr

ABSTRACT

Information technology is developing at an astounding pace. Computers are now common and their use is switching from number-crunching tools to "information and communication appliances". The growing success of the Internet is just an example of how information and communication facilities are now available to the general public. But today's Internet is just the precursor of more innovative applications. New advances in computer science disciplines such as communications networks and mobile devices are suggesting radically new uses of information technology.
These new uses may raise new ethical issues. We give a few examples and show that these issues are not explicitly dealt with in existing ethical frameworks such as Mason's PAPA or Huff's ImpactCS. We also discuss the usability of these frameworks by software designers and suggest that information technology practitioners need more explicit tools such as handbooks to help them understand and deal with ethical issues.

INTRODUCTION

This paper presents a computer scientist's perspective on ethical issues that may arise in future computing environments. We believe that ethical issues should be given thought very early, even before building a prototype or developing a new computing paradigm. Following software engineering discipline, high-level requirements such as ethical concerns raised by features of the system should be considered in the very first phases of systems development. If ethics are taken into account as an afterthought, the new system or paradigm may break a number of ethical principles and may be very difficult and costly to modify. Ethical concerns should help shape tomorrow's computing environments.
We first describe a few examples of systems where ethical concerns had to be taken into account after the fact. We then present some examples of current active research areas which question existing ethical frameworks. We then argue that IT practitioners need better tools to help them understand what and where ethical issues are.

ETHICAL ISSUES IN RECENT AND FUTURE IT DEVELOPMENTS

Information Technology (IT) is developing fast. The World-Wide Web concept, for example, was first implemented only three years ago and is now widely popular among Internet users. The Java language now extends the World-Wide Web concept from an hypertext document base to a collection of documents and executable programs. Java has brought answers to old computing problems for which solutions where not readily available less than a year ago. Examples include cross-platform portability of executable code or secure execution of remote code. These capabilities bring to front ethical issues related to transaction security, authentication and privacy and also to software and networks reliability.
In a first paragraph, we present examples of commercial systems where privacy has obviously not be taken into account in the design phase. These systems had then to be redesigned. In the second paragraph, we discuss some issues related to software reliability. In the third paragraph, we introduce two examples of recent technological advances which raise serious ethical concerns. Let us state right now that we don't intend to be exhaustive but we focus on areas that we have rarely seen mentioned in the computer ethics literature.

Two Privacy-Related Examples

Many new privacy-related issues surround the recent developments of the Internet and computer-mediated communication. We first give two examples of systems which have been designed without caring for ethical issues and which had to be modified due to users' concerns. In the first example, the designers recognized they didn't think about the issues. The second case raises questions about the designers' intent.

Magic Cap Business Cards

Magic Cap is an operating system running on mobile personal digital assistants. Magic Cap provides facilities for communication between users through phone, e-mail or fax. With this system, each user has a personal electronic business card in an address book which is used for two purposes. First, information stored in the business card may be used by the system, for example for phone call or email forwarding. Second, the electronic business card is sent to other users, in the same way one gives a business card to a colleague who asks for contact information. In the first version of Magic Cap, giving a business card to someone also meant giving away any personal information that was stored on it. Thus, users either were reluctant to give away their business card which contained fields such as "private personal phone number" or wouldn't fill out personal information and thus couldn't use some advanced features of the system. Due to users' requests, these two uses of the electronic business card had to be separated. In the newer version of the system, each field of information has an "unlisted" checkbox which allows the user to prevent private information to appear on the business card as related by Jerney (1995).

In this case, the mistake is blatant and was even recognized by the designers. Personal information should be dealt with carefully and the electronic business card function should be handled differently from the personal information function. Designers should be better warned about the risks of a system dealing with user's personal information as is the case with most personal digital assistants today. We'll show in the next paragraph that mobile systems also bring up new privacy issues.

This example shows that customer concern over privacy may lead software designers to redesign software, and, of course, at a cost. The next example brings up a different issue: if the system includes a mechanism that deals with some user personal information, the user should be made aware of this fact.

Netscape Cookies

Netscape "cookies" are a software mechanism through which World-Wide Web servers can store information on the computer of the visiting client browser. This information may be any code or string sent by the server as a cookie. Although the cookie mechanism is intended for harmless uses such as a server remembering a client's last visit, it may be diverted and thus raises ethical concerns. The most worrying potential use of this capability is the possibility for a server to "know" if another given page or server has been visited by the client. There is one condition though for this to be possible: the previously visited server or page and the server which asks for the information should be in the same "URL range" (URL stands for Uniform Resource Locator, i.e., the unique address of a web page). But an URL range may be as large as "all sites ending with acme.com" as defined in Netscape (1996).

This possibility raises a privacy risk if used improperly: a server may get information about which sites a user has previously visited. Until recently, cookies transactions were not even noticeable to the user. Everything happened "behind the user's back", giving him absolutely no control over the information that was stored in the cookies file. This design choice, when discovered, was questioned by the Internet users community: cookies are a mechanism which may benefit servers, for example for tracking customer fidelity or for presenting different advertisements every time a user connects to a server. Why did the designers hide this feature from the user, especially if it may involve privacy risks? This speculation has remained unresolved, but in the new Netscape Navigator version 3, Netscape added the option for the user to be notified of new cookies and to allow or not cookies to be stored on his/her machine. Although this option is disabled by default thus leaving the novice user ignorant of cookies, the informed user has a way to track the storage of cookies and can even forbid them. We'll show in section 3 that a basic human-computer interaction principle could have helped analyse this problem.

It is interesting to note that the cookie blocking capabilities of Netscape version 3 were not considered convincing by Internet users. Systematic confirmation/denial of cookies is required from the user and is largely intrusive (some web documents send several cookies, requiring user confirmation/denial for every cookie). This poor design has triggered reactions: on the Apple Macintosh platform alone, six different freeware or shareware utilities are available to clean up cookies after every Netscape session.

These examples show that computer-mediated communication software is prone to involve privacy risks and that software designers should be better informed of the issues. But the possibilities brought by computer-mediated communication are rapidly changing the whole software industry. A worrying aspect of this change is the lower reliability of some software distributed over the Internet.

Software reliability

Beta Software Is Unreliable Software

A recent trend on the Internet is the wide distribution of unfinished software also known as "public beta". In software engineering, the word "beta" to qualify the development stage of a software product has a precise meaning. Beta software is considered feature-complete by the development team and has no known serious bugs.

According to Marciniak (1994), "the purpose [of beta-testing] is to find problems that show up in actual operation before many copies of the system are released". When a product is in beta stage, it is usually distributed to a group of technically-aware users ("beta-testers") with the aim of testing the product on a wide range of software and hardware combinations to detect possible incompatibilities that the quality assurance team couldn't have found under normal conditions. Beta-testers are asked to "stress-test" the software and report technical details, should they find a remaining bug.

With the advent of "public betas" freely available on the Internet, the situation has changed radically. Any user, whatever her/his technical proficiency, can download and use a beta version. Most users aren't even aware that they use beta software, because beta software is not often clearly labeled so, except for the version number which usually contains a "b". Moreover, most "public betas" software is at a development stage that would rather be called "alpha" (feature-complete but known bugs still remaining) or even "development release" (not feature-complete and still in development).

The reasons for this fury over beta versions are outlined in a recent issue of an electronic newsletter Duncan (1996). In an environment where communication is fast and ubiquitous, software companies are making every effort to get attention and distributing new beta versions of their software every few weeks is a way of making news. However, this behaviour has some consequences for users and may be damaging to the software industry in the long term. Every user is now a potential beta-tester, even if s/he lacks the necessary technical background, and most of all, the willingness to be one. Actually, development or alpha versions of software may even contain dangerous bugs. The user is thus left at risk on his own, since most software products are distributed with a license agreement by which the software company declines responsibility in case of software misbehaviour. Users are now commonly using software that is known to be defective, since they won't always upgrade to the final version if they have a public beta. In the long term, this strategy may be disastrous: confidence in software is already lower and lower and "crashing software" is a common experience among users. Giving away buggy software to inexperienced users may be rewarding for a company's publicity but carries the risk of diminishing seriously the confidence users have in software and the software industry in general. New approaches in building software such as the component software approach we discuss next may make this problem even worse.

Reliability of Component Software

Maner explains that a unique characteristic of information technology lies in its complexity Maner (1995). Because of the large number of interacting components, even in a basic system, hardware and software are prone to errors and bugs which are difficult to isolate and correct. A current trend in software engineering aims at building a software product from smaller software parts (or components) put together. This approach, named "component software", has some definitive advantages from a software engineering point of view. Small pieces can be designed to be highly reusable across a variety of software projects, thus reducing development time and cost. Java or OpenDoc are examples of this approach.

However, one may wonder at the implications of this approach with regard to software reliability. Increasing the number of components increases the complexity of software and thus the un-foreseeable or unwanted interactions between components. Although some approaches, for example the use of mathematical tools such as formal methods, alleviate these risks, these methods are costly and used only for critical systems such as flight decks or power plant control systems.

We don't deny the clear advantages of the component software approach: it allows reusability of ready to use software components and facilitates standardization of software. However, we believe it is misleading to advocate this approach for cost reasons. If component software actually cuts software coding costs, it requires a much more thorough testing approach and thus costs a lot more in testing. We should emphasize that software testing and stronger software engineering practices are the only key to software reliability.

Future Computing Environments and Ethical Concerns

When looking at trends of computer science technology, one may wonder at the ethical implications of some recent developments. In this paragraph, we outline a few of these recent advances and the ethical issues they raise.

Who is Responsible for "Intelligent Agents"?

"Intelligent Agents" are a new computing paradigm. It aims at changing the way the user works from "doing" to "letting do". Most current systems follow the traditional "direct manipulation" paradigm of graphical user

interfaces: users point at objects (such as text paragraphs or shapes of a drawing) and act on them. Similarly, when using the World-Wide Web, users point and click on links they want to visit and have to elaborate information retrieval plans when looking for specific information.
With agents, users are able to delegate tasks to software agents that will then act on behalf of the user. For example, the user may delegate the task "make plane and hotel reservations for a two-day trip to London on Monday next week". Based on the user's preferences or habits, the agent would then roam the network, visiting the airline company's and the hotel's servers and making appropriate reservations, then getting back to the user to inform her/him of the results. General Magic's Telescript system described in White (1995) already allows for the use of such agents on a dedicated network.
The agents paradigm raises issues related to privacy and responsibility. A software agent wandering on a network carries a great deal of information on the user whose behalf it is acting on: personal information such as name or credit card number for handling electronic transactions, but also information required to carry the agent's task, for example the user's daily schedule or airline and hotel preferences. Attention should be given to the privacy of this information. Security mechanisms should prevent an hostile host from "hi-jacking" a software agent or from gathering data from visiting agents. Another issue is responsibility: if an agent performs unwanted transactions whose responsibility is involved? The difference here with the traditional software paradigm of "doing" is that the agent acts following instructions given by the user. One should expect users to make errors when giving instructions to agents and also software agents to be sometimes faulty. Should the user or the software company or the visited host be held responsible in case the agent misbehaves?
Another area of concern is the use of mobile systems. New risks to privacy occur due to the very nature of mobile devices.

Ubiquitous Computing and Privacy of Location

With mobile systems, such as portable phones or personal digital assistants (PDA), users are now free to use information technology while on the road and in different places. The "ubiquitous computing" paradigm proposes the implantation of fixed infrastructures in buildings that allow mobile systems to communicate, whatever their location inside the building or whatever building they are in.
This paradigm raises serious issues with regard to privacy. Whenever a mobile device establishes a communication through the fixed infrastructure, the location of the entry point nearest to the user is known to the system. Thus, it is possible for the system to know the user's location with great accuracy: a typical entry point range is a few meters or hundred meters. A similar problem already occurs with cellular phones: the cellular phone infrastructure divides geographical space in small cells. It is then possible to know which cell a user carrying an active cell phone is in. Not only can users be located, but more worryingly, they are often not aware of this possibility. Once again, system designers should inform the users of ethical issues raised by the use of their systems.
These examples cover a broad range of issues but are symptomatic of a common problem. Software designers and computer science researchers aren't aware of ethical issues that may arise in the systems they develop. Practical tools are needed to help designers identify and assess potential ethical issues.

TOWARDS AN IT PRACTITIONER'S COMPUTER ETHICS HANDBOOK

A possible first step towards a better understanding of ethical issues is a categorization of the areas of ethical concern. Ethical frameworks such as PAPA in Mason (1986) or ImpactCS in Huff (1995) are certainly adequate to classify ethical issues into categories. But their intent is not practical use in software design. ImpactCS is intended for teaching ethics and therefore, as the authors recognize, requires a good knowledge of the ethics literature to identify issues and refine the framework's categories. PAPA suffers some similar limitations: without previous exposure to computer ethics, these frameworks are of little value to the practitioner. In the case of new technologies, even previous experience with computer ethics may not be sufficient. For example, the problem of privacy of location raised by the use of mobile systems is not commonly dealt with in the computer ethics literature.
Computer ethics codes, such as the ones proposed by the ACM or IFIP certainly are valuable to the practitioner. But here again, their scope is too broad for them to be practically usable in software design. They present general principles but are of little help when confronted to a software design decision.
Another approach is to establish a bridge with human-computer interaction. Human-computer interaction deals with the design of user interfaces. The user interface of a computer system mediates the interaction between the user and the system. In the terms of Norman, the user interface presents to the user the image of the system from

which the user builds a cognitive model of the system behaviour and functions Norman (1986). User interface design relies heavily on collections of rules often devised by ergonomics specialists, which practitioners apply when designing a system. We first introduce a simple basic rule of human-computer interaction and show how computer ethics could provide input for its use. We then show how it applies to an example of the previous section.

The Observability Rule in Human-Computer Interaction

The rule of observability states that the user interface of a system should inform the user about the current internal state of the system that is relevant to the user's current task (Abowd (1992)). This rule is sometimes called "feedback principle" because it implies that any action from the user must trigger a response from the user interface (a user action modifies the internal state of the system). The difficult part in applying this rule is deciding what part of the internal state of the system is actually relevant to the user's task. Computer ethics could help decide what information should be considered relevant and thus be presented to the user.

In the example of Netscape cookies, observability applies easily. In this case, the system to be considered is the user's browser. When handed a cookie by a server, the client browser's internal state is modified: an information is received and then stored in a file. When a server asks for a previously stored cookie, the internal state is again modified since this information is fetched in the file by the browser and then transmitted to the server. Is this information relevant to the user's task? When Netscape first didn't give any possibility to the user to know about cookie transmission, they considered that it was not relevant. In the current version of Netscape's browser, information about cookies is presented to the user only if a server hands in a cookie. If the server fetches a previously stored cookie, the user is not informed about it. There is here a clear discrepancy between the presentation of both operations to the user. If privacy concerns were taken into account consistently, both operations would be considered relevant to the user's task and the user interface of the browser would notify the user in both cases.

To be of value to practitioners, computer ethics could provide them with practical rules to be taken into account when designing systems. These rules could build on the observability rule.

Towards A Practitioner's Computer Ethics Handbook

To take into account computer ethics in system design, designers need clearly outlined issues expressed in terms of their practical concerns instead of broad categories of ethical concerns.

A first step could be to recommend the use of basic human-computer interaction principles. Human-computer interaction prescribes a "user-centered" approach to design. It emphasizes that computers are designed for users to accomplish tasks in a human way, and not for users to adapt to the peculiarities of the software they use. The observability rule should also be presented. It doesn't prevent a system to break computer ethics principles such as privacy, but it requires the user to be informed about the capabilities of the system. It forbids a system to perform actions "behind the user's back". This applies to Netscape cookies but also to the problem of privacy of location we have mentioned above. If a mobile system is able to transmit the location of the user, at least the user should be made aware of this.

Practical recommendations would be helpful to designers. For example, they should be warned about the potential issues of systems dealing with the user's personal information such as software agents or mobile systems. Not only should the system inform the user when personal information is about to be disclosed, but the system should provide a way of keeping personal information really personal if the user wishes so. Systems permitting remote execution such as software agents should also be mentioned as raising issues related to privacy of personal information and responsibility.

Finally, emphasis should be put on software quality and the risks of unreliable software. Software engineering could certainly be helpful here, but better software quality really requires a profound change in current software development practices.

CONCLUSION

We have presented a few examples of recent systems and technologies that raise ethical issues. We have also shown how practitioners need to be better educated about ethical concerns. There is definitely a need for practical recommendations that software designers could apply without possessing a background in computer ethics. Human-computer interaction rules could provide a first basis for taking computer ethics principles into account in software design and disseminating them efficiently to designers.

REFERENCES

Abowd, G., Coutaz, J. & Nigay, L. (1992) "**Structuring the Space of Interactive Systems Properties**", EHCI'92, IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction, Ellivuori, Finland, pp. 113-130.

Duncan, G. (1996) "**Waiting with Beta'd Breath**", TidBITS, n° 328, Vol 1. <http://www.tidbits.com/tb-issues/TidBITS-328.html>.

Huff, C. & Martin, C.D. (1995) "**Computing Consequences**", Communications of the ACM, n° 12, Vol 38, pp. 75-84.

Jerney, J. (1995) "**Magic Cap: Straight from Andy Hertzfeld**", Mobilis, n° 8, Vol 1, pp. 1-9. <http://www.volksware.com/mobilis/august.95/hertz1.htm>.

Maner, W. (1995) "**Unique Ethical Problems in Information Technology**", Proceedings of ETHICOMP95, An International Conference on the Ethical Issues of Using Information Technology, Leicester, UK.

Marciniak, J.J. (1994) "**Categories of Testing**", Encyclopedia of Software Engineering. New York: John Wiley, pp. 90-93.

Mason, R.O. (1986) "**Four Ethical Issues of the Information Age**", MIS Quarterly, n° 1, Vol 10, pp. 486-498.

Netscape (1996) "**Persistent Client State HTTP Cookies**", Netscape Communications Corporation. <http://www.netscape.com/newsref/std/cookie_spec.html>.

Norman, D. & Draper, S. (1986) "**User Centered System Design: New perspectives on human- computer interaction**", Hillsdale, NJ: Lawrence Erlbaum Associates.

White, J.E. (1995) "**Mobile Agents**", General Magic Corporation. <http://www.genmagic.com/Telescript/Whitepapers/wp4/whitepaper-4.html>.