The precautionary principle in a world of digital dependencies

Wolter Pieters and André van Cleeff
University of Twente, Enschede, The Netherlands

Abstract

According to the Jericho forum, security in information technology can no longer be achieved by building a digital fence around the organisation. We investigate the consequences of this "de-perimeterisation" for software engineering ethics. "Being in control" is no longer feasible for certain types of risk, and therefore software engineering ethics cannot only be based on foreseeable consequences. Instead, ethics could be based on the precautionary principle, which states that lack of certainty shall not be used to postpone cost-effective measures. We translate this principle from the domain of safety to the domain of security, in which we are interested in precaution with respect to people's actions rather than natural disasters. Following the example of Napster as well as recent results in philosophy of technology, we present a guideline for ethical software development, focused on inviting desirable use.

1 Introduction

Traditionally, protection of information has been directed at securing an organisation at the perimeter of its systems and network, typically in the form of a firewall. The implicit assumption behind this approach is that the inside of the security perimeter is more or less trusted, whereas the outside is not. Due to changes in technologies, business processes and their legal environments this assumption is not valid anymore. Many organisations are outsourcing part of their IT processes, and employees demand that they can work from home. Business are collaborating on an unprecedented scale, forming complex networks around the globe and putting more trust in third parties than in their own networks. Mobile devices can access data from anywhere, smart buildings are being equipped with small microchips that constantly communicate with each other (and their headquarters). With

so-called cloud computing, organisations can rent virtual PCs by the hour. This leads to even more complicated systems, or systems of systems, which span the boundaries of multiple parties and cross the security perimeters that these parties have put in place for themselves.

Following the Jericho Forum [1], we call this process *de-perimeterisation*: the disappearing of boundaries between systems and organisations, which are becoming connected and fragmented at the same time. The most obvious problem of de-perimeterisation is how to re-organise our security. But de-perimeterisation implies not only that the border of the organisation's IT infrastructure becomes blurred, but thereby also that the border of the organisation's accountability fades. If an organisation outsources its data-processing, who is morally responsible for maintaining the privacy of its customers? And if an organisation insources another organisation's processes, how can it be sure that it meets all the obligations?

In this paper, we investigate the consequences of de-perimeterisation, the disappearing of boundaries between systems and organisations, for software engineering ethics. The discussion is on the level of ethics of system design. We take inspiration from the lawsuits against the peer-to-peer network Napster to discuss responsibilities for indirect consequences, based on the so-called *precautionary principle*. This principle has mainly been applied in environmental ethics, to be on the safe side with respect to environmental damage caused by new technologies. But what if a particular design decision, somewhere in the chain of technologies enabling electronic health records, invites employers to check people's health before hiring them? And what if Internet voting slowly undermines the idea of the secret ballot? These examples give some intuition of the relevance of precaution in information technology as well. In the following, we will investigate the idea more systematically.

2 Accountability in a de-perimeterised world

De-perimeterisation has consequences for moral and legal accountability of people and organisations that develop software. In a complex chain of events or systems, many people will have had a share in an action that leads to undesirable consequences. As such many people will also have had the opportunity to prevent these consequences, and therefore no-one can be held responsible. This has been described as "the problem of many hands" [2].

This is certainly the case if IT systems are developed that depend on other systems, which in turn depend on other systems, et cetera. In fact, making this possible – or actively promoting it – is the design philosophy

behind the service-oriented architecture (SOA) and the associated SOA governance, where all functionality consists of services which can be aggregated into larger applications performing functions to end-users. In such cases, the networked organisational and technological structure makes it difficult to determine who is responsible if the final result turns out to be wrong. And could we have known at all that something undesirable might happen? An ethics requiring that consequences can be unambiguously attributed to a single person or organisation is bound to be useless in such a situation. This is illustrated by the lawsuits against Napster, which we will discuss later: the responsibility for copyright infringement is *shared* between Napster and its users.

Worse, it is not only unclear where the border of accountability lies, it is even unclear what influence the organisation has and therefore what it can accomplish in terms of consequences. If the organisation makes a decision to apply a certain data protection policy in its software, the data may in fact be managed by a different organisation. How will the organisation that actually manages the data implement this? And how can this be verified? If the accountability becomes unclear, so do the consequences of the organisation's own actions. This means that the organisation will have to deal with the risks in a different way. Apart from the problem of many hands, de-perimeterisation also leads to the issue of uncertain risk.

If we develop a service, we may not know exactly which other services we are dependent on. Conversely, we may not know exactly which other services will start using our service. We cannot make reasonable estimations of probabilities of unwanted events in such a case. As an example, consider Skype, which went off-line for some time in 2007, following an automatic update from Microsoft. The update caused many Skype's customers to re-login at the same time, leading to a denial-of-service. The main *direct* dependency, i.e. Skype running on Windows, seems to be trivial, and must have been known in the design phase. But before the incident, it was unknown to both Microsoft and Skype that there was also this *indirect* dependency between their two applications, in this case related to the timing of the updates. Where direct dependencies are part of the design, the complex structure of services may also cause such indirect dependencies.

Traditional approaches of risk assessment have focused on the probability of failure and the severity of the consequences of failure, usually expressed in terms of costs. This approach has been criticised for various reasons [3, 4]. Most importantly, it assumes that the probabilities of events and the costs associated with them are known or at least objectively determinable. This is usually not the case when we have the many dependencies of a de-perimeterised setting.

In order to address these issues, we need to be more precise about what

we mean with risk. In a recent publication [5], the Dutch Scientific Council for Government Policy (WRR) distinguishes between four types of risk problems:

1. simple risk problems: problems that can be addressed by standard risk assessment and risk management procedures;
2. complex risk problems: problems in which the relations between causes and effects are subject to scientific discussion;
3. uncertain risk problems: problems in which there is a lack of knowledge about possible effects;
4. ambiguous risk problems: problems in which the desirability of effects is subject to discussion.

According to the WRR, the third and fourth type of problems require a different approach than classical risk management. When the effects themselves are unknown, risk management, according to the Council, should be based on the so-called *precautionary principle*. Risks in software development in a de-perimeterised world, as we have shown, are at least uncertain, possibly even ambiguous. The precautionary principle may therefore help in ethically designing networked services, since a) it does not assume that the risks can be objectively assessed and b) it does not focus on consequences directly attributable to the action of a single person or organisation. In the following, we discuss how to define and implement the precautionary principle for software engineering ethics.

3 The precautionary principle

The precautionary principle is a moral and political principle stating that a) parties should refrain from actions in the face of scientific uncertainties about serious or irreversible harm to public health or the environment possibly caused, and b) the burden of proof for assuring the safety of an action falls on those who propose it (adapted from [6]). In the Rio Declaration on Environment and Development from 1992 this is formulated as follows: "Where there are threats of serious or irreversible damage, lack of full scientific certainty shall not be used as a reason for postponing cost-effective measures to prevent environmental degradation."

The usefulness of the precautionary principle depends on two conditions. Firstly, there must exist a resource which can be damaged beyond repair. If there is no such resource, there is no reason to apply the principle. In that case, if a policy fails it can be changed with limited cost and its effect

reversed. Secondly, the usage of the resource is compulsory: it cannot be substituted. If it could be substituted, it would be possible to switch to the backup resource in case of damage, or to avoid using it in the first place.

In the European Union, the principle has been adopted in the Maastricht Treaty in Article XVI-130r from 1992. One of the areas in which the principle has been used is for decision making about genetically modified organisms. The EU has delayed the introduction of genetically modified corn from the United States for several years, fearing that it would be unsafe and would lead to pollution of the corn reserves.

The precautionary principle has been extensively criticised. Among the major objections are that it is unscientific and/or impractical, and that it does not take the costs of missed opportunity into account. The EU has also been criticised for applying it opportunistically, to protect its agricultural industry from the US. Despite these criticisms, it still is one of the cornerstones of European environmental law.

The precautionary principle incorporates the unknown into the ethics of risk assessment. The principle has been related to the legal concept "duty of care" [7], which implies that one can be liable for damages if one fails to exercise care in relation to other people or their properties. Care is thus a central issue in the precautionary perspective. Rather than demanding control, the concept of care points to relations and dependencies, which is precisely what we have to deal with in a de-perimeterised information society.

4 The precautionary principle in IT

For the precautionary principle to apply, threats of serious or irreversible damage need to be present, and there should be no possibility of substitution. By now, substituting the world's digital resources would be next to impossible. Although small, independent systems may easily be replaced, the increasing amount of dependencies reduces the number of systems that can effectively be called "small". That serious damage can be done by software to assets both inside and outside the digital world need not be explained in today's world of bugs, worms and patches. That this damage can be harmful to humans or other moral subjects follows from our dependency on digital assets. Showing that damage in the digital world can be irreversible on a large scale, however, is not so trivial. Although serious damage by itself would justify precaution, it is helpful to understand how digital irreversibility can happen.

Obviously, the digital world of the Internet is not a physical space, it holds no living beings that can fall ill or die because of pollutants. As

such the "infosphere" does not resemble nature's biosphere. Still, there are good reasons for considering them in a similar way. We see families of systems, that produce "offspring": new systems that resemble the old ones. Programs called viruses roam the Internet and can infect these systems; anti-virus software is needed to protect them. Just as in nature, monocultures can be dangerous, because it makes them easier prey for viruses. The many dependencies may lead to propagation of problems to other digital "species". In short, the digital world is a complex of dependencies similar to nature, and small disturbances at one point may have major and irreversible consequences at others.

An important source of irreversibility is the so-called *function creep*. Even if a system is initially being developed for a limited purpose, chances are that requirements and uses will grow over time: a system that is designed for one purpose, may be judged to be very useful for another, even if that other purpose was explicitly indicated as undesirable when the system was first designed. For example, a database with biometric data of citizens may be designed for authentication purposes, but may then turn out to be very helpful for crime investigation. Society may then become so dependent on this mechanism that un-implementing the system is not an option. The speed in which such dependencies develop increases with the de-perimeterisation of systems and organisations. Because services depend on each other, it is hard to remove a service once it is up and running.

Thus, IT satisfies both the conditions of serious or irreversible damage and lack of possibilities for substitution. If it is justified to apply the precautionary principle to IT, which we have argued here, the question becomes how the principle can be applied in an effective way, and which specific characteristics of IT demand adaptation of the principle for this domain. A major difference between IT and the common application areas of the precautionary principle is that the principle has mainly been applied in contexts of safety. This means that the probabilities of damage occurring are determined by nature, or by unintentional human error. If a genetically modified organism spreads in the environment and wipes out a native species, there was probably nobody that actively tried to promote this effect.

However, in IT many of the threats are related to people's intentions, be it the intention to illegally copy an MP3 file or the intention to make a server unreachable. In the context of security, where we have to deal with active adversaries, the situation is therefore different: in this case, the probability that a problem occurs does not only depend on natural causes, but also on the intentions and perseverance of people that have access to the system. This does not only include hackers, but also for example the users of Napster, who may change their behaviour based on the design of

	environmental safety	**digital security**
environment	nature and health	society and information
origin of risk	nature and human error	intentional human actions
solution	safety engineering	anticipating human behaviour

Table 1: Differences between the traditional and new application domain of the precautionary principle

the system. What does this mean for the precautionary principle?

Here, the precautionary principle is not only about preventing accidental unintended effects. It is also about unintended effects that are caused by other people's intentions. Therefore, the precautionary principle in software engineering should not only address issues similar to keeping a drug off the market as long as there is no scientific certainty about possible side-effects. It should also cover keeping a drug off the market because people may use it for undesirable purposes, such as recreational or terrorist ones. Although safety is important in IT as well (as in the Skype example), the security dimension makes it impossible to implement the precautionary principle unmodified. An overview of the differences between the traditional application area (environmental safety) and the target area (digital security) is given in table 1.

Thus, the de-perimeterised context of software development leads us to apply the precautionary principle to software engineering ethics. Since damage in the digital world can be serious and irreversible, this application is justified. However, to account for a context in which people rather than nature play the main roles, we have to take their intentions into account. After discussing a practical case, we will develop a terminology for this specific context.

5 The Napster case

In some examples, the ethical responsibility of software developers is rather straightforward. If your software damages assets of customers due to bad design or programming, you are morally responsible for the consequences, even if it has been legally asserted that the software is provided without any warranty. This is just ethics of consequences: if you would have designed the software better, the damage would not have happened.

A different type of responsibility occurs if you willingly put assets of your customers at risk, for example by designing software in such a way that you

damage these assets for your own good. This trivially applies to writing viruses, but in some cases the issue is more subtle. For example, when Sony included DRM software on their audio CDs and put the users at risk to certain attacks on their computer, was this a case of bad programming or of intentional misuse? Especially relevant is the informed consent question: did Sony really install the software even if the customer declined the licence agreement? [8]

The examples above have clear boundaries between responsibilities: if the designer's action leads to damage and the designer could have prevented the damage, the designer is morally, if not legally, responsible. Here, the effects are directly caused by the software, developed by a clearly defined organisation. The effects also do not involve intentional actions by others. Responsibilities for such issues can be relatively easily covered by traditional risk management and associated ethics, even though the duty of care may still apply.

What happens with de-perimeterisation is that the boundaries between responsibilities become blurred. The trail of actions leading to a particular service may be such that several people or organisations could have prevented the damage, and hold each other responsible.

In such a case, the precautionary principle may help to establish accountability. The major example in which the precautionary principle has already been applied in practice concerns the lawsuits against peer-to-peer network applications. Using such software, both legal and illegal content can be shared between computer users. In case the content is illegal, it may be argued that serious damage can ensue, both to the copyright owner and to the enforceability of intellectual property rights in general. The question is whether the designers of such an application are responsible for what people do with it.

The first lawsuit concerning peer-to-peer software occurred in 2000 against Napster. Later, other organisations such as Grokster were sued as well, for similar reasons. What was at stake is what is called *indirect liability* [9] for copyright infringement. Were the users themselves responsible for their unlawful behaviour, or could the provider be held liable because of indirect impact on the actions? In the latter case, it would be much easier to enforce the copyright, by a lawsuit against the provider rather than against individual users. Could the provider successfully point to the many hands that contributed to the infringement to counter the charge of liability?

Under US doctrine, third parties may be indirectly liable for copyright infringement if a) they knowingly contribute to the infringement (contributory infringement) or b) have control over the infringer and enjoy direct financial benefit from the infringement (vicarious liability). Peer-to-peer

software fits in the former category. The questions that determine liability in the peer-to-peer case are therefore a) whether the company has a meaningful capacity to prevent or discourage illegal use and b) whether there is substantial possibility for non-infringing use. In the end, Napster was held liable, because information about the shared files was centrally available, so that Napster could have known about and even prevented the infringements.

With the lessons of Napster in mind, other peer-to-peer operators made sure that information about shared files was decentralised and the data streams between the nodes were encrypted. In effect, their solution was to make accountability more difficult. Since neither the company providing the solution nor the ISPs could know about the infringements, there was no easy way to block the application any more, apart from targeting individual users. This is not merely a matter of refusing to implement measures to prevent certain behaviour of users, which may be a justifiable point of view; it is intentionally designing the technology in such a way that these measures are hard to implement. Apparently, the current moral and legal framework makes avoiding accountability more attractive than including ethics in system design.

What the example of peer-to-peer applications shows, is that in a deperimeterised situation, people or organisations may be held accountable for consequences *that would not have occurred if others had acted differently*. This is necessary to keep accountability in situations where multiple actors contribute to an action. Thus, even though the action *cannot be attributed to the person or the organisation*, the fact that the action was made possible or likely is sufficient to be held accountable. This accountability follows the reasoning of the precautionary principle, in the sense that precaution against unintended and undesirable use is demanded. In a deperimeterised setting, chains of contributory actions may be much longer, for example when many services depend upon each other and collectively cause damage. We can demand precaution there as well, but the consequence of organisations intentionally limiting accountability may need to be avoided.

A question raised by the example is how indirect liability or the demand of precaution can be prevented from inhibiting innovation and the development of new services. If I may be held liable for selling a new type of technology because other people use it for illegal purposes, I may be tempted not to invest in developing the technology at all. For copyright issues, this problem was addressed by the Digital Millennium Copyright Act (DMCA), passed in 1998. No matter what one thinks about the contents of the law, which have been subject to controversy, it illustrates how precaution can be applied without asking the impossible of designers. Specific

requirements are laid down in this law, such that if a company meets these requirements, it is safeguarded against indirect liability. The requirements focus on acting upon knowledge or notification of infringement. Such practical legal requirements can form the basis for a generalised application of the precautionary principle in IT. In order to do so, we first need some terminology to describe the precautionary approach.

6 A philosophical vocabulary

The fundamental challenge of de-perimeterisation is one that has already been acknowledged by philosophy: actions, and therefore morality, cannot be ascribed to a single person or organisation, but only to a complex network of cooperating entities [10]. In such a network, each actor's behaviour may influence the behaviour of others, and can as such contribute to the morality of the whole network's actions. The classical case here is the weapon: if someone shoots someone else, this may not have happened if he would not have had a gun. In this sense, the gun can be held partly responsible for the action, since it invited the action of shooting. Whether this is merely a metaphor or something more substantial will not be discussed here.

In such a network of cooperating entities, the design of technology plays a crucial role. Technology, in modern philosophy of technology, is neither just a simple instrument used by humans for their own purposes, nor an unmanageable force that is taking over society, as traditional philosophies by for example Jacques Ellul and Martin Heidegger have argued. Instead, technology may invite people to act in a certain way, or, conversely, inhibit people from acting in certain ways, and can be *designed* to do so. Technological artifacts come with their own implicit "invitations to action", called *scripts* [11]. For example, if you drive more slowly because of a speed bump, the reason that you do something ethically desirable is not only your own, but also the speed bump's. In the same sense, the reason that people reveal privacy-sensitive information on social networking sites such as Facebook is partly due to the design of the application inviting such behaviour (which allows it to sell very precisely targeted advertisements).

This role of technology in desirable or undesirable actions is called *technological mediation* and has been developed in the context of a so-called postphenomenological approach to philosophy of technology [12]. Instead of considering actions as completely determined by either the people or the technology, technology may *invite* or *inhibit* actions of people. These notions of cooperative action come with their counterparts in cooperative experience: technology may amplify or reduce certain aspects of people's

	desirable	**undesirable**
amplify experiences that are	+	-
reduce experiences that are	-	+
invite actions that are	+	-
inhibit actions that are	-	+

Table 2: Using postphenomenology for software engineering ethics. Pluses indicate what should be encouraged, minuses what should be discouraged.

experience. Just as binoculars amplify part of the world while preventing you from seeing the remainder, a social networking site may amplify certain aspects of friendship and reduce face-to-face contact in the experience of its users.

From a postphenomenological perspective, the following questions need to be asked when applying the precautionary principle to security domains such as software (see table 2):

- does your design *amplify* or *reduce* aspects of people's experience? are those desirable or undesirable?
- does your design *invite* or *inhibit* certain actions? are those desirable or undesirable?

Which experiences of actions are considered desirable or undesirable will depend on moral consensus in society; the precautionary principle only argues for precaution with respect to these values.

This terminology allows us to formulate the precautionary principle for software engineering ethics. Apart from accountability in terms of programming errors that cause safety or security issues, the precautionary principle allows us to define accountability for impacting people's intentions. This is essential for the combination of precaution and dealing with people rather than nature. As in the case of peer-to-peer applications, one can no longer focus on direct consequences of action in software engineering ethics. Instead of asking whether you did something morally wrong, the question becomes whether you or your design *invited* someone to do something wrong or *inhibited* someone from doing something ethically desirable.

A Dutch voting advice website (Stemwijzer) turned out to log the IP addresses of potential voters, along with their political preference. There is not much doubt that this constituted a violation of privacy. But what if a social networking website includes a field "political preference" in your profile? It looks like there is the choice not to provide the information, but it certainly constitutes an invitation. In the Napster case, the question that

can be asked is whether a peer-to-peer system invites copyright-infringing behaviour. If this is the case, then a software engineer may choose to add measures to the system that limit this invitation.

Just as the hotel manager will have to attach bulky rings to the keys to make sure that her guests return them, the software engineer will have to add measures to her application that invite morally sound use and inhibit undesirable or controversial actions. More concretely, the software engineer will be responsible for implementing measures that actually mediate the interaction of the software with other entities in such a way that the other entities will be discouraged from performing morally undesirable actions.

Apart from asking whether your service invites or inhibits actions, the question should also be asked whether your service amplifies or reduces aspects of experience. For example, in the case of DRM, making copies for use by the owner may not be possible either, thus reducing the desirable experience of enjoying the music in the car. An Internet voting system may reduce the desirable experience of voting as a public ritual. This may be compensated by creating a similar public environment online, in which the voting can take place.

The preceding analysis leads us to propose the following definition:

Definition 1: The *precautionary principle for de-perimeterised software design* states that lack of certainty about the use of software shall not be used to refrain from implementing measures in software design that invite desirable behaviour, inhibit undesirable behaviour, amplify desirable experiences and reduce undesirable experiences of users.

7 Operationalising the principle

When the precautionary principle is applied to software engineering, the most important issue is how much effort should be put in identifying the invitations, inhibitions, amplifications and reductions. How much are we able to do to establish precaution in the face of uncertainty? When an organisation *could have known* about the undesirable effects, it may be held morally responsible, but how does one determine if that is the case?

Researchers have pointed out that problems in information security arise mainly because issues were not known or even knowable at the time of design. These issues have sometimes been termed "unknown unknowns" or "monsters". This might imply that many of the mediating effects of a software system may be unknowable at the time of design. How, then can the precautionary principle be applied?

First of all, only awareness of the principle will already lead to better identification of possible effects. If software engineers focus on indirect consequences of their technology next to direct effects such as damage due to bugs, many of the indirect effects may be identified in an early stage. This may even become a success factor when legislators would extend indirect liability, which is not that unlikely in a de-perimeterised setting.

Secondly, conceptual tools need to be developed to support reasoning about indirect consequences in terms of invitation, inhibition, amplification and reduction. Such tools can help software engineers traverse the social context in which their design will operate, thereby enabling them to identify how the script of their technology may interact with the users' intentions. The tools may be an operationalisation of the notion of "could have known", in the sense that if they are used appropriately, this may limit ethical and legal responsibility for unexpected indirect consequences. This is analogous to the provisions of the Digital Millennium Copyright Act, in which companies cannot be held indirectly liable if they follow the appropriate procedures. Drawing up such procedures for the precautionary principle in general requires extensive future research, and may draw upon work in the areas of logic and policy specification.

8 Conclusions

As organisations become de-perimeterised from the perspective of information security, this also requires a new paradigm in software engineering ethics. One can no longer rely on an ethics of consequences, as consequences may not be foreseeable, their desirability may not be unambiguously assessable, and they cannot be directly ascribed to actions of a single person or a single organisation. Instead, the precautionary principle allows for a more extensive moral framework for software engineers. The Napster case showed that this principle has already been around in the form of indirect liability, and that specific legal requirements can be put in place to clearly specify the requested amount of precaution. In this paper, we showed how a postphenomenological vocabulary from philosophy of technology can be applied to implement the precautionary principle in software engineering ethics. Using this framework, the focus is not on identifying direct consequences, but rather on identifying what kind of actions the software to be designed invites or inhibits. This approach can complement traditional risk management procedures.

References

[1] Jericho Forum. *Jericho whitepaper*. Jericho Forum, The Open Group, 2005.

[2] H. Nissenbaum. Computing and accountability. *Communications of the ACM*, 37(1):73–80, 1994.

[3] S. Jasanoff. The political science of risk perception. *Reliability Engineering and System Safety*, 59:91–99, 1998.

[4] D. Gotterbarn and S. Rogerson. Responsible risk analysis for software development: Creating the software development impact statement. *Communications of the Association for Information Systems*, 15:730–750, 2005.

[5] Wetenschappelijke Raad voor het Regeringsbeleid. *Onzekere veiligheid: verantwoordelijkheden voor fysieke veiligheid*. Amsterdam University Press, Amsterdam, 2008.

[6] C. Raffensperger and J.A. Tickner. *Protecting public health and the environment: implementing the precautionary principle*. Island Press, 1999.

[7] M.D. Rogers. Scientific and technological uncertainty, the precautionary principle, scenarios and risk management. *Journal of Risk Research*, 4(1):1–15, 2001.

[8] J.A. Halderman and E.W. Felten. Lessons from the Sony CD DRM episode. In *Security 06: 15th USENIX Security Symposium*, pages 77–92, 2006.

[9] W. Landes and D. Lichtman. Indirect liability for copyright infringement: Napster and beyond. *Journal of Economic Perspectives*, 17(2):113–124, 2003.

[10] B. Latour. *Reassembling the social: an introduction to actor-network theory*. Oxford University Press, Oxford, 2005.

[11] M. Akrich. The de-scription of technical objects. In W. Bijker and J. Law, editors, *Shaping Technology - Building Society*, pages 205–224. MIT Press, Cambridge, MA, 1992.

[12] P.P.C.C. Verbeek. *What things do: Philosophical Reflections on Technology, Agency, and Design*. Pennsylvania State University Press, 2005.

Biographies

Wolter Pieters is a postdoc researcher in the DIstributed and Embedded Security group as well as the Information Systems group at the University of Twente. He is working in the VISPER project on de-perimeterisation. His research focuses on access control and social aspects of de-perimeterisation. Contact him at w.pieters@utwente.nl.

André van Cleeff is a PhD student in the Information Systems group at the University of Twente. He is working in the VISPER project on de-perimeterisation. His research focuses on cloud computing and virtualisation. Contact him at a.vancleeff@utwente.nl.

Acknowledgements

This research is supported by the research program Sentinels (www.sentinels.nl). Sentinels is being financed by Technology Foundation STW, the Netherlands Organization for Scientific Research (NWO), and the Dutch Ministry of Economic Affairs. The authors wish to thank Roel Wieringa and Eric Luiijf for very helpful comments.