



# THE PRECAUTIONARY PRINCIPLE IN A WORLD OF DIGITAL DEPENDENCIES

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

As organizations become deperimeterized, a new paradigm in software engineering ethics becomes necessary. We can no longer rely on an ethics of consequences, but might instead rely on the precautionary principle, which lets software engineers focus on creating a more extensive moral framework.

**T**raditionally, information protection has been directed at securing an organization 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Due to changes in technologies, business processes, and their legal environments, this assumption is no longer valid. Many organizations now outsource part of their IT processes, and employees demand to work from home. Businesses collaborate 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, and smart build-

ings are being equipped with microchips that constantly communicate with each other and their headquarters.

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

Following the Jericho Forum,<sup>1</sup> we call this process *deperimeterization*: the disappearing of boundaries between systems and organizations, which are becoming connected and fragmented at the same time.

The most obvious problem deperimeterization poses involves how to reorganize our security. Deperimeterization implies not only that the border of the organization's IT infrastructure becomes blurred, but also that the border of the organization's accountability fades. If an organization outsources its data processing, who is morally responsible for maintaining the privacy of its customers? Further, if an organization insources another organization's processes, how can it be sure it will meet all those obligations?

## ACCOUNTABILITY IN A DEPERIMETERIZED WORLD

Deperimeterization has consequences for the moral and legal accountability of people and organizations that

develop software. In a complex chain of events or systems, many people will have shared in any action that leads to undesirable consequences. As such, they 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.”<sup>2</sup>

This is certainly the case if IT systems are developed that depend on other systems, which in turn depend on other systems, and so on. Making this possible—or actively promoting it—is the design philosophy behind the service-oriented architecture and the associated SOA governance, where all functionality consists of services that can be aggregated into larger applications performing end user functions.

In such cases, the networked organizational 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 that something undesirable might happen? An ethics requiring that consequences can be unambiguously attributed to a single person or organization is bound to be useless in such a situation. The resistance to Napster offers inspiration to discuss responsibilities for indirect consequences based on the so-called *predictability principle*. The responsibility for copyright infringement is shared between Napster and its users.

Worse, not only is it unclear where the accountability border lies, it is even unclear what influence the organization has and therefore what it can accomplish in terms of consequences. If the organization makes a decision to apply a certain data protection policy in its software, the data may in fact be managed by a different organization. How will the organization that actually manages the data implement and verify this?

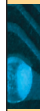
If the accountability becomes unclear, so do the consequences of the organization's own actions. This means that the organization must deal with the risks in a different way. Apart from the problem of many hands, deperimeterization also leads to the issue of uncertain risk.

If we develop a service, we might not know exactly which other services we depend on. Conversely, we might not know exactly which other services will start using our service. We cannot make reasonable estimations regarding the probabilities of unwanted events in such a case.

As an example, consider Skype, which went offline in 2007, following an automatic update from Microsoft. The update caused many Skype customers to relogin at the same time, leading to a denial of service. The main direct dependency—Skype running on Windows—seems 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 update's timing.

Whether or not direct dependencies are part of the design, the complex structure of services might also cause such indirect dependencies.

Traditional approaches to risk assessment have focused on the probability of failure and the severity of failure's consequences, usually expressed in monetary terms. This approach has been criticized for various reasons.<sup>3,4</sup> 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 so when we have the many dependencies of a deperimeterized setting.



**If the accountability becomes unclear, so do the consequences of the organization's own actions.**

To address these issues, we must be more precise about how we categorize risk. In a recent publication,<sup>5</sup> the Dutch Scientific Council for Government Policy (WRR) distinguished between four risk problem types:

- *Simple*. These problem types can be addressed by standard risk-assessment and -management procedures.
- *Complex*. These problem types occur when the relations between causes and effects are subject to scientific discussion.
- *Uncertain*. A lack of knowledge about possible effects arises when these problem types occur.
- *Ambiguous*. These problem types arise when the desirability of effects becomes subject to discussion.

According to the WRR, the third and fourth problem types require an approach that differs from classic risk management. When the effects themselves are unknown, risk management should, according to the Council, be based on the so-called *precautionary principle*. Risks in software development in a deperimeterized world are at least uncertain, possibly even ambiguous. The precautionary principle might therefore help in ethically designing networked services, since it does not assume that the risks can be objectively assessed and does not focus on consequences directly attributable to the action of a single person or organization.

## **THE PRECAUTIONARY PRINCIPLE**

A moral and political imperative, the precautionary principle states that parties should refrain from actions in the face of scientific uncertainties about serious or irreversible harm to public health or the environment.

It further holds that the burden of proof for assuring the safety of an action falls on those who propose it.<sup>6</sup> The 1992 Rio Declaration on Environment and Development formulated this 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” ([www.un.org/documents/ga/conf151/aconf15126-1annex1.htm](http://www.un.org/documents/ga/conf151/aconf15126-1annex1.htm)).

The usefulness of the precautionary principle depends on two conditions. First, there must exist a resource that 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.

**The digital world offers a complex of dependencies similar to nature—small disturbances at one point could have major and irreversible consequences elsewhere.**

Second, use of the resource is compulsory: it cannot be substituted. If it could be substituted, it would be possible to switch to the backup resource when damage occurred, or to avoid using it in the first place.

In the European Union, the principle has been adopted in the 1992 Maastricht Treaty, in Article XVI-130r. One area in which the EU has used the precautionary principle is to make decisions about genetically modified organisms. For example, the EU has delayed the introduction of genetically modified corn from the US for several years, fearing it would be unsafe and lead to pollution of corn reserves.

The precautionary principle has been extensively criticized. Major objections include that it is unscientific or impractical and that it does not take the costs of missed opportunity into account. The EU has also been criticized for applying the treaty opportunistically, to protect its agricultural industry from the US. Despite these criticisms, it remains 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,”<sup>7</sup> which implies that someone who fails to exercise care in relation to other people or their properties can be liable for damages. 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 must deal with in a deperimeterized information society.

## THE PRECAUTIONARY PRINCIPLE IN IT

For the precautionary principle to apply, threats of serious or irreversible damage must be present, with no possibility of substitution. By now, substituting the world’s digital resources would be next to impossible. Small, independent systems can easily be replaced, but 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 Internet’s digital world is not a physical space and holds no living beings who can fall ill or die from pollutants. As such, this *infosphere* does not resemble nature’s biosphere. Still, there are good reasons for considering them similar. We see families of systems that produce offspring: new systems that resemble the old ones. Viruses roam the Internet and can infect these systems, forcing us to employ antivirus software to protect against them. Just as in nature, monocultures can be dangerous because they become easier prey for viruses. The many dependencies might lead to propagation of problems to other digital species. In short, the digital world offers a complex of dependencies similar to nature—small disturbances at one point could have major and irreversible consequences elsewhere.

So-called *function creep* provides an important source of irreversibility. Even if a system is initially being developed for a limited purpose, chances are requirements and uses will grow over time: A system designed for one purpose might be judged useful for another, even if developers explicitly indicated as undesirable that other purpose when first designing the system.

For example, a database with biometric data about citizens might be designed for authentication purposes, but then become helpful for crime investigation. Society could become so dependent on this mechanism that unimplementing it is no longer an option. The speed at which such dependencies develop increases with the deperimeterization of systems and organizations. 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 the lack of possibilities for substitution. If applying the precautionary principle to IT is justified, as we have argued here, the question becomes how the principle can be applied in an effective way, and

**Table 1. The precautionary principle's traditional and new application domains.**

Process	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 behavior

which specific characteristics of IT demand adapting the principle for this domain.

That the principle has been applied mainly in safety contexts constitutes a major difference between IT and the common application areas of the precautionary principle. 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, almost certainly no one actively sought to promote this effect.

However, in IT many threats relate to people's intentions, whether that intention is to illegally copy an MP3 file or to make a server unreachable. In the context of security, where we must deal with active adversaries, the situation is therefore different. In this case, the probability that a problem occurs depends not only on natural causes, but also on the intentions and perseverance of those who have access to the system. This includes not only hackers, but also, for example, Napster users who might change their behavior based on the system's design. What does this mean for the precautionary principle?

Here, the precautionary principle focuses on more than preventing accidental unintended effects. It also addresses unintended effects caused by other people's intentions. Therefore, the precautionary principle in software engineering should consider not only issues such as 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 might use it for undesirable purposes, such as recreational or terrorist applications. Although safety is important in IT as well (as in the Skype example), the security dimension makes it impossible to implement the precautionary principle without modification. Table 1 shows an overview of the differences between the traditional application area (environmental safety) and the target area (digital security).

Thus, the deperimeterized 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 must take their intentions into account, as the following example shows.

### ASSET DAMAGE

In some examples, software developers have a rather straightforward ethical responsibility. If their software

damages customer assets as a result of bad design or programming, they are morally responsible for the consequences, even if it has been legally asserted that the software is provided without warranty. This is the ethics of consequences: Had they designed the software better, the damage would not have occurred.

A different type of responsibility occurs if developers willingly put their customers' assets at risk by, for example, designing software so that they damage these assets for their 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 its audio CDs that, in turn, put users at risk of being attacked by Sony's hidden code, was this a case of bad programming or intentional misuse? The informed consent question is especially relevant to this case: Did Sony install the software even if the customer declined the license agreement?<sup>8</sup>

These issues assert clear boundaries between responsibilities. If a designer's action leads to damage that could have been prevented, that designer is morally if not legally responsible. Here, the effects are directly caused by the software, developed by a clearly defined organization. The effects also do not involve intentional actions by others. Traditional risk management and associated ethics can cover responsibilities for such issues relatively easily, even though the duty of care might still apply.

Deperimeterization causes the boundaries between responsibilities to blur. The trail of actions leading to a particular service might be such that several people or organizations could have prevented the damage, and each holds the others responsible.

In such a case, the precautionary principle might help establish accountability. In practice, the leading example of how the precautionary principle has been applied in practice is the lawsuits against peer-to-peer (P2P) network applications. Using such software, both legal and illegal content can be shared between computer users. In cases where the content is illegal, it can be argued that serious damage might ensue, both to the copyright owner and to the enforcing intellectual property rights in general. This poses the question, Should designers of these applications be held responsible for what people do with them?


### NAPSTER LAWSUIT

The first lawsuit concerning P2P software occurred in 2000 against Napster. Later, other organizations such as Grokster faced lawsuits as well, for similar reasons.

At stake was the principle of *indirect liability*<sup>9</sup> for copyright infringement: Were the users themselves responsible for their unlawful behavior, or could the providers be held liable because of their indirect impact on these 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 can be indirectly liable for copyright infringement if they knowingly contribute to the infringement (contributory infringement) or if they have control over the infringer and enjoy direct financial benefit from the infringement (vicarious liability). P2P software fits in the former category. The questions that determine liability in the P2P case are therefore whether the company has a meaningful capacity to prevent or discourage illegal use and whether there is substantial possibility for noninfringing 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.



**While controversial, the Digital Millennium Copyright Act illustrates how precaution can be applied without asking the impossible.**

### PEER-TO-PEER PRESSURE

With the lessons of Napster in mind, other P2P operators made sure to decentralize information about shared files, then also encrypted the data streams between the nodes. In effect, their solution made 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 apart from targeting individual users.

This is not merely a matter of refusing to implement measures to prevent certain user behavior, which might be a justifiable point of view; it is intentionally designing the technology so 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.

P2P applications show that in a deperimeterized situation, people or organizations could 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 organization, 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 it demands precaution against unintended and undesirable use. In a deperimeterized setting, chains of contributory actions might be much longer when, for example, many services depend on each other and collectively cause damage. We can demand precaution there as well, but it might be necessary to avoid the consequence of organizations intentionally limiting accountability.

A question raised by this example is how indirect liability or the demand of precaution can be prevented from inhibiting innovation and the development of new services. If someone can be held liable for selling a new type of technology because other people use it for illegal purposes, they might be tempted not to invest in developing the technology at all. The 1998 Digital Millennium Copyright Act (DMCA) addressed this problem for copyright issues. The law's contents, while controversial, illustrate how precaution can be applied without asking the impossible. This law lays down specific requirements, such that if a company meets them, 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 generalized application of the precautionary principle in IT. To do so, we first need some terminology to describe the precautionary approach.

### PHILOSOPHICAL VOCABULARY

The fundamental challenge of deperimeterization has already been acknowledged by philosophy: Actions, and therefore morality, cannot be ascribed to a single person or organization, but only to a complex network of cooperating entities.<sup>10</sup> In such a network, each actor's behavior may influence the behavior of others and, as such, can contribute to the morality of the whole network's actions. The classic case here is the weapon: If someone shoots someone else, this might not have happened without a readily available gun. In this sense, the gun can be held partly responsible for the action, since it invited the action of shooting.

In such a network of cooperating entities, the design of technology plays a crucial role. Technology is neither just a simple instrument used by humans for their own purposes, nor an unmanageable force bent on taking over society, as some traditional philosophies have argued. Instead, technology might 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*.<sup>11</sup> For example, if someone drives more slowly because of a speed bump, the reason that they do something ethically desirable is not only their own, but also because the speed bump requires

doing so. In the same sense, people reveal privacy-sensitive information on social networking sites such as Facebook partly because the application's design invites such behavior, which in turn lets the site sell precisely targeted advertisements.

The 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 the philosophy of technology.<sup>12</sup> Instead of considering actions as completely determined by either the people or the technology, technology could invite or inhibit people's actions. As Table 2 shows, these notions of cooperative action come with their counterparts in cooperative experience: Technology can amplify or reduce certain aspects of people's experience. Just as binoculars amplify part of the world while preventing any view of the remainder, a social networking site could amplify certain aspects of friendship and reduce face-to-face contact with its users.

Plusses in Table 2 indicate what should be encouraged, minuses what should be discouraged. From a postphenomenological perspective, the following questions must be asked when applying the precautionary principle to security domains such as software:

- Does a design amplify or reduce aspects of people's experience? Are those desirable or undesirable?
- Does a 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 lets us 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 lets us define accountability for impacting people's intentions. This is essential for the combination of precaution and dealing with people rather than with nature. As in the case of P2P applications, we can no longer focus on the direct consequences of action in software engineering ethics. Instead of asking whether we did something morally wrong, the question becomes whether we or our design invited someone to do something wrong or inhibited someone from doing something ethically desirable.

For example, Stemwijzer ([www.stemwijzer.nl](http://www.stemwijzer.nl)), a Dutch voting advice website, logged 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 labeled "political preference" in its profile? It would look as if the site offers the choice of not providing the information,

**Table 2. Using postphenomenology for software engineering ethics.**

Actions	Desirable	Undesirable
Amplify experiences that are	+	-
Reduce experiences that are	-	+
Invite actions that are	+	-
Inhibit actions that are	-	+

but it certainly would constitute an invitation. In the Napster case, the question that can be asked is whether a P2P system invites copyright-infringing behavior. If this is the case, a software engineer might choose to add measures to the system that limit this invitation.

Just as hotel managers must attach bulky rings to their keys to ensure their guests return them, software engineers must add measures to their applications 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 software's interaction with other entities such that those entities will be discouraged from performing morally undesirable actions.

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

This analysis leads us to propose the following definition: The precautionary principle for deperimeterized 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 behavior, inhibit undesirable behavior, amplify desirable experiences, and reduce users' undesirable experiences.

## OPERATIONALIZING THE PRINCIPLE

When we apply the precautionary principle to software engineering, the most important issue becomes how much effort should be put into identifying the invitations, inhibitions, amplifications, and reductions. We must determine how much we can do to establish precaution in the face of uncertainty. Further, when an organization could have known about the undesirable effects, it might be held morally responsible, but we must first somehow determine this to be the case.


Researchers have pointed out that problems in information security arise mainly because issues were not

known or even knowable at design time. These issues have sometimes been termed *unknown unknowns* or *monsters*. This might imply that many of the mediating effects of a software system might be unknowable at design time, which poses the problem of applying the precautionary principle correctly.

To do so, we must first acknowledge that only awareness of the principle will lead to better identification of possible effects. If software engineers focus on the indirect consequences of their technology next to direct effects such as damage due to bugs, many of the indirect effects could be identified at an early stage. This could even become a success factor when legislators extend indirect liability, which is not that unlikely in a deperimeterized setting.

Second, conceptual tools must 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 might interact with users' intentions. The tools might be an operationalization of the "could have known" notion, in the sense that if they are used appropriately, it might limit ethical and legal responsibility for unexpected indirect consequences. This is analogous to the provisions of the DMCA, 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 might draw upon work in the areas of logic and policy specification.

**A**s organizations become deperimeterized—from the information security perspective—a new paradigm in software engineering ethics becomes necessary. We can no longer rely on an ethics of consequences, as these consequences may not be foreseeable, their desirability might not be unambiguously assessable, and they cannot be directly ascribed to actions of a single person or a single organization. Instead, the precautionary principle focuses on creating a more extensive moral framework.

The Napster case shows that this principle has already been established in the form of indirect liability, and that specific legal requirements can be put in place to clearly specify the requested amount of precaution. We have shown how a postphenomenological vocabulary from the philosophy of technology can be applied to implement the precautionary principle in software engineering ethics. Using this framework, the focus shifts to identifying what kind of actions the software to be designed invites or inhibits. This approach can complement traditional risk management procedures. 

## Acknowledgments

This research is supported by the Sentinels research program ([www.sentinels.nl](http://www.sentinels.nl)). Sentinels is financed by Technology Foundation STW, the Netherlands Organization for Scientific Research (NWO), and the Dutch Ministry of Economic Affairs. The authors thank Roel Wieringa and Eric Luijff for their helpful comments.

## References

1. The Open Group, Jericho Forum, 2005; [www.opengroup.org/jericho/faq-misc.htm](http://www.opengroup.org/jericho/faq-misc.htm).
2. H. Nissenbaum, "Computing and Accountability," *Comm. ACM*, vol. 37, no. 1, 1994, pp. 73-80.
3. S. Jasanoff, "The Political Science of Risk Perception," *Reliability Eng. and System Safety*, 1998, pp. 59; 91-99.
4. D. Gotterbarn and S. Rogerson, "Responsible Risk Analysis for Software Development: Creating the Software Development Impact Statement," *Comm. Assoc. Information Systems*, vol. 15, 2005, pp. 730-750.
5. Scientific Council for Government Policy, *Onzekere Veiligheid: Verantwoordelijkheden voor Fysieke Veiligheid*, Amsterdam Univ. Press, 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," *J. Risk Research*, vol. 4, no. 1, 2001, pp. 1-15.
8. J.A. Halderman and E.W. Felten, "Lessons from the Sony CD DRM Episode," *Proc. 15th Usenix Security Symp.*, Usenix, 2006, pp. 77-92.
9. W. Landes and D. Lichtman, "Indirect Liability for Copyright Infringement: Napster and Beyond," *J. Economic Perspectives*, vol. 17, no. 2, 2003, pp. 113-124.
10. B. Latour, *Reassembling the Social: An Introduction to Actor-Network Theory*, Oxford Univ. Press, 2005.
11. M. Akrich, "The Description of Technical Objects," *Shaping Technology—Building Society*, W. Bijker and J. Law, eds., MIT Press, 1992, pp. 205-224.
12. P.P.C.C. Verbeek, *What Things Do: Philosophical Reflections on Technology, Agency, and Design*, Pennsylvania State Univ. Press, 2005.

*Wolter Pieters is a postdoctoral researcher in the Distributed and Embedded Security group and the Information Systems group at the University of Twente, the Netherlands. He is working in the VISPER project on deperimeterization. His research focuses on access control and social aspects of deperimeterization. Contact him at [w.pieters@utwente.nl](mailto:w.pieters@utwente.nl).*

*André van Cleeff is a PhD student in the Information Systems group at the University of Twente. He is working on deperimeterization for the VISPER project. His research focuses on cloud computing and virtualization. Contact him at [a.vancleeff@utwente.nl](mailto:a.vancleeff@utwente.nl).*