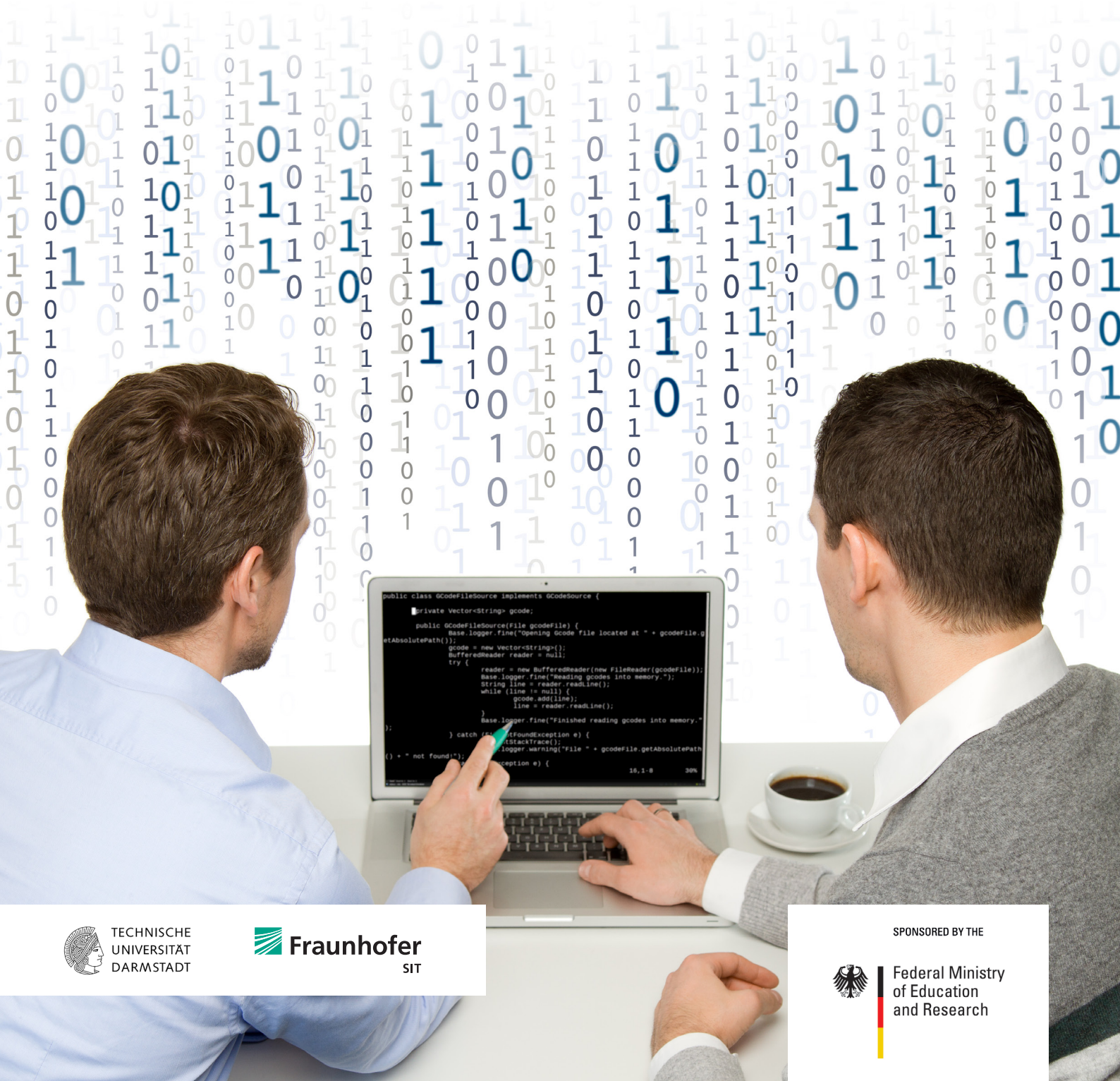
EC SPRIDE
EUROPEAN CENTER FOR
SECURITY AND PRIVACY BY DESIGN

EMERGING TRENDS IN SOFTWARE DEVELOPMENT & IMPLICATIONS FOR IT SECURITY: AN EXPLORATIVE STUDY

06/2014



TECHNISCHE UNIVERSITÄT DARMSTADT

Fraunhofer SIT

SPONSORED BY THE

Federal Ministry of Education and Research

EC SPRIDE
EUROPEAN CENTER FOR
SECURITY AND PRIVACY BY DESIGN

EMERGING TRENDS IN SOFTWARE DEVELOPMENT & IMPLICATIONS FOR IT SECURITY: AN EXPLORATIVE STUDY

CARSTEN OCHS
EC SPRIDE, TU Darmstadt
Mornewegstrasse 30, 64293 Darmstadt
www.ec-spride.de, www.informatik.tu-darmstadt.de

Ed. Michael Waidner

SIT Technical Reports
SIT-TR-2014-02

June 2014

Fraunhofer Institute for Secure
Information Technology SIT
Rheinstrasse 75
64295 Darmstadt
Germany

IMPRINT

SIT Technical Report
SIT-TR-2014-02
Emerging Trends in Software Development &
Implications for IT Security: An Explorative Study
Carsten Ochs
ISSN 2192-8169

Editor
Michael Waidner

Design
Elena Vituschek

Photograph credit
Fraunhofer SIT

Editor's address
Fraunhofer Institute for Secure
Information Technology SIT
Public Relations
Rheinstrasse 75
64295 Darmstadt
Germany
Telephone: +49 6151 869-399
Fax +49 6151 869-224
info@sit.fraunhofer.de
www.sit.fraunhofer.de

CONTENTS

EXECUTIVE SUMMARY

There have been numerous transformations in the interrelated realms of software development (SD) and IT security. To form a clear picture of the SD trends and account for their implications, we conducted an explorative study comprising 23 interviews with SD and IT security experts from industry, academia and regulating institutions. The analysis reveals six major trends:

1. SD's Working Environment: Dynamic, Flexible, Distributed

The acceleration, fluctuation of cooperating partners, distribution of production sites and complexity of supply chains call for finding ways to build continuity into SD processes. To guarantee developers have a shared vision, companies must invest significantly in proper communication and enable team members to build strong relationships so as to be clear about product security relevant requirements. Companies are well advised to establish long-term and trustable relationships with outsourcing partners and to abstain from outsourcing critical components. Harmonizing tools, frameworks and Integrated Development Environment (IDE) with the outsourcing partner may reduce compatibility issues. In general, the complexity of supply chains calls for improved techniques to measure and certify components.

2. Widespread Adoption of Agile Software Development

Agile SD is oftentimes portrayed as a way to adapt to a rapidly changing environment. As teams and team roles become flexible, agile SD raises the level of IT security expertise any individual developer needs to have. The latter must not be overburdened with security considerations, though. Therefore, agile SD makes it even more important to integrate security processes systematically and make security expertise scalable. While agile SD is about adapting to rapidly changing circumstances forced by market pressure, software systems tend to run for quite a while. This makes it mandatory to preserve expertise and knowledge and to trace decisions that went into system construction. Last but not least, usability should be taken seriously as a security factor. Agile SD allows the integration of »laymen expertise« right from the beginning by involving customers/end users.

3. Code Generation and Assembly of (Prefabricated) Code

There is a certain »democratization« of SD allowing developers without formal engineering education to create code. To make sure secure code is being generated by laymen, security must be factored into self-learning platforms and code generators. In industrial SD, security should be automated by integrating security features into Integrated Development Environments, frameworks and libraries. The latter devices must be equipped with tools that allow for testing and generating secure building blocks. Developers must be able to keep a consistent relation between modeling and source code level (round-trip engineering) and to make informed decisions when selecting a framework. This presupposes framework security certification. As there is more sharing of code (on the web, etc.), developers should have acquired the skills to check such code for security, whereas companies may consider formulating sharing policies. As the phenomenon of crowdsourcing software development gains a foothold, research is called upon to investigate security implications.

4. Compositional Systems and Modularization

Compositionality and modularization entails a profound paradigm change as regards security. Study participants mentioned a range of measures that companies may take to increase the probability of producing secure compositional systems, such as no integration of external components as a black box; rigorous testing of security relevant components and raising security relevant parameters; continuous integration of new features or code portions; a clear understanding of the integrated component's code; checking and safeguarding the compatibility of the in-house code base with external code; reducing the attack surface by deactivating the external component's functionalities that are not in use; etc. In the future it will be desirable to dispose of the formal specifications of the components' characteristics and automatable testing procedures and specifications of valid indicators concerning the static and dynamic quality and security attributes of compositional systems. This should be done in order to enable certification and legal guarantees of a component's security. Preferably, R&D manages to produce intrinsically secure building blocks/components guaranteeing secure compositional systems.

5. Distributed Systems and Intensified (Cross Domain) Networking

There are similar issues if it comes to distributed systems: there is a demand to develop techniques allowing one to determine the security level of distributed systems at runtime and to make a distributed system's components intrinsically secure. In matters of Cloud Computing – one specific form of distributed systems – techniques to separate sensitive from non-sensitive data as well as solving legal and liability issues are required. Furthermore, as distributed systems may network sub-systems from different domains, an amalgamation of perspectives and techniques from different domains is necessary. The aforementioned domains may include safety and security realms, which is why ways to fuse safety and security mechanisms must be explored. Especially the convergence of the world of information and the physical world (embedded devices, cyberphysical systems) renders it necessary to invest in R&D focusing on security and safety at once.

6. Legacy: The Complexity of Evolved Software Ecosystems

Dealing with the legacy requires one to integrate security into systems retroactively. This may be done pragmatically via isolating untrusted building blocks; encapsulating untrusted building blocks by building virtual software cages; or building security into legacy systems ex-post. All these measures presuppose the development of easy-to-manage techniques and tools. Quite in general, there is a need for tools and procedures to analyze security and safety in very large and complex systems; for scalable techniques and tools that allow complex systems to be broken down and analyzed in aggregates; for risk assessment techniques that go from the top all the way down; and for techniques to identify at-risk components stochastically. To avoid legacy in the future, however, there is a need to build security into systems proactively. Therefore, smart documentation techniques for expertise and knowledge capture as well as for preserving assumptions and design decisions are required, just as the development of novel methodologies that account for SD's »evolutionary«

character are necessary. While educational institutions are called upon to prepare developers for being confronted with loads of legacy, industry is well advised to harmonize in-house code production to keep code comprehensible and establish long-term perspectives on the systems they produce. This may be achieved by optimizing software architecture and by introducing product line engineering.

In turn, the SD trends lead to six challenges for IT security pertaining to different aspects of SD:

- **Education:** The flexibility of current SD calls for raising individual developers' expertise
- **Processes:** Ways must be found to render security processes systematic and scalable also in large, distributed, and unstable social networks
- **Methodologies:** The flexibility coming with the spread of agile values must be reconciled with building continuity into SD
- **Techniques:** The increase in networking distributed components brings up the need to encapsulate untrusted components and develop intrinsically secure building blocks
- **Metrics:** There is a dire need to find better ways to measure the security of components, systems, and processes
- **Tools:** To assist developers in integrating security in SD processes, tools must be equipped with automated secure engineering features

1. INTRODUCTION

In recent years, software development (SD) and IT security as well have undergone massive transformations. As regards the former, the perceived need for more flexible approaches to software development culminated in the 2001 »Agile Manifesto«[1] indicating a turning away from more linear, sequential methodologies as represented by the waterfall model. What is more, working environments (outsourcing, freelancing etc.) were rendered flexible in general, which certainly did affect the world of SD too. On the technical level, new sorts of products and services appeared, such as apps and Software-as-a-Service. Further, developers began to have more and more tools at their disposal when developing software, relying on Integrated Development Environments (IDEs) and the like.

As far as IT security is concerned, there was a steady rise in news stories reporting on security breaches. Especially in 2013, the public realization of US secret service National Security Agency (NSA) activities enjoyed considerable press coverage. Events even drove the former German Minister of the Interior Hans-Peter Friedrich to strive to enact an IT security law.[2] From a more general point of view, as the internet was somewhat expanded to a network, more and more things started to rely increasingly on the internet, including systems of the physical world (cyber-physical systems) that had previously run more or less isolated, as well as the workings of everyday life. As a result, while critical infrastructures have become integrated into digital networks, the latter have become critical infrastructures in their own right.

These rather general remarks may suffice to justify the intuition that (somewhat interrelated) major transformations in the realm of software development and in that of IT security have taken place in the last 10 to 15 years; moreover, there is no reason to believe that the dynamic of change in both these areas is going to come to a halt in the near future. We take this intuition as an opportunity to act empirically and pose the questions of what exactly the crucial transformations were that happened in the last decade or so in SD; which ones are to be expected in the near future; and how those transformations are likely to affect IT security.

Thus, the aim of this report is to indicate trends in software development that are currently emerging and their implications for IT security. Having said this, it is, of course, beyond all doubt that nobody is able to look into the future. However, it is still possible to elaborate possible future developments, the chains of cause and effect associated with them, and the related options for action (Steinmüller/Schulz-Montag 2004: 63). As regards the accurateness and exactness of trend analyses, sociological technology studies distinguish between visions and scenarios: while the former »are vague in their specifications of the technical features and the forms of use of the envisioned technology« (Schulz-Schaeffer 2012: 2,3), scenarios work by indicating rather concrete chains of cause and effect (Steinmüller/Schulz-Montag 2004: 63). In the study presented here, our aim is to provide a catalogue of the developments from the recent past or currently underway, respectively. Therefore, our approach comes closest to what is usually called an »explorative scenario« (ibid.).

1 *www.agilemanifesto.org*

2 *www.handelsblatt.com/politik/deutschland/koalitions-verhandlungen-friedrich-will-it-sicherheitsgesetz-durchsetzen/9022878.html*

We aim to achieve this by systematically absorbing the expertise of key figures in industry, academia and regulating agencies so as to generate an empirically funded picture of the trends underway (i.e. according to this expertise and at times to the additional material considered in the analysis). To do so, we harnessed qualitative sociological research methods. Exploring trends presupposes applying a method that exhibits a certain measure of openness. Furthermore, in our study we are interested in several influential factors, such as the social organization of SD, the technologies (e.g. tools) being used as well as the technology (i.e. the software) to be developed and so on. In this sense, our study is located at the intersection of sociological technology studies and organization studies. While it is quite usual in technology studies to focus on experts, in organization studies the method of the qualitative interview occupies a leading position (Kühl/Strodtholz/Taffertshoffer 2009: 19). For this reason we applied the method of the qualitative expert interview when conducting our study. In this context, experts are to be understood as decision-makers within some institutional setting; they are responsible for designing, implementing, and controlling organization-related interventions and solutions, and they have unrestricted access to sensitive information as regards organizational structures and decisions (Liebold/Trinczek 2009: 34). In the context of SD/IT security, different types of experts are to be considered:

- Professionals responsible for industrial software development, such as developers, team leaders, project managers, executives, consultants etc.
- Professionals responsible for IT security in industrial software development, such as chief security officers, chief information officers etc.
- Researchers at universities and semi-public research institutes concerned with either software engineering or IT security, or both (secure software engineering).
- Experts located in the institutional infrastructure that provides the framework of academic and industrial software development/IT security endeavors, such as delegates, regulators, public authorities' representatives, or auditors and evaluators operating product certification.

These experts are expected to pick up information regarding current and future developments in the area of software development/IT security: due to their position they can be considered as possessing an »empirical radar« that gathers information related to relevant trends in everyday practice. Hence, in-depth expert interviews with influential actors, i.e. with experts from all areas listed above formed the core of our analysis.

In the next section we will describe the research procedure before presenting the results in section 3 and 4. Finally, in section 5, we will map research challenges resulting from the trends identified.

2. RESEARCH PROCEDURE

We did not limit ourselves to conducting expert interviews. Instead, we followed a threefold research strategy:

- First, we systematically collected internal expertise by conducting several brainstorming sessions with scholars at EC SPRIDE and the Technical University Darmstadt.
- Second, from the internal expertise collected, we developed a questionnaire designed to guide »quick and dirty« interviews with developers conducted at the CeBit fair 2013. These interviews served two purposes: a.) validating the assumptions put forward by our internal experts; and b.) broadening the spectrum of issues raised by addressing not only academic experts but also practitioners »in the wild.« This allowed for developing a very robust questionnaire, which we relied upon when taking the main step of our investigation:
- Third, we conducted 21 in-depth expert interviews with 23 key figures (two »double interviews«) in industry, academia, and institutional settings.

Given the background assumptions we derived from the gathering of internal expertise, we developed an interview guideline including 14 questions[3]. This guideline concerned the following areas:

1. An entirely open question concerning the major transformations in software development in the last 10 to 15 years and transformations to be expected in the near future
2. Questions concerning making employer-employee relationships flexible and the potential implications for IT security
3. Questions concerning the geographical distribution of collaborating developers and firms
4. Questions concerning development methodologies (assessment, pros and cons, whys and wherefores, future evolution etc.)
5. Questions concerning the relation between software type (application type, legacy etc.) and development methodology
6. Questions concerning the relevance of and awareness for IT security
7. Questions concerning the role of tools and technologies in software development
8. An entirely open question asking the interviewee to identify relevant themes insofar as they were not addressed in the interview

In posing questions concerning these areas, we did not mean to impose our worldview on interviewees. In fact, one of the unique strengths of qualitative research is that it leaves plenty of room for study participants to differentiate, challenge or refute the assumptions the interviewer may hold and to feed information into the interview in a way that was not anticipated by the interviewer. We argue that this is precisely what renders expert interviews an appropriate method for conducting trend analyses; for as experts are expected to be the first to catch sight of indications for current and future developments, it is not possible for the non-expert interviewer to anticipate their expertise.

3 *This is the maximum number of questions that we posed; however, as not all questions were applicable to all interviewees' professional areas, the number varied between 13 and 14 questions. Moreover, as is common in semi-structured expert interviews, if in the course of the conversation further questions arose, still more questions were formulated ad hoc by the interviewer.*

So whom did we interview? The following gives an overview over our interviewees' professional areas:

- Software developers, team leaders, project managers: 5
  Interviewees' affiliation types: one global player in business software, one global player in consumer and industrial electronics, one worldwide operating bank, one major insurance company, one publicly-funded European high-tech institution.
- Security products & consultancy: 4
  Interviewees' affiliation types: three middle-sized and one large security software and consultancy firm.
- IT security experts in industrial software development: 5
  Interviewees' affiliation types: three global players in business software, one global player in consumer and industrial electronics, one global player in telecommunications.
- Researchers at Universities and semi-public research institutes: 4
  Two researchers located in Germany, one in the EU, one in the US.
- Experts located in the institutional framework: 5
  Interviewees' affiliation types: two delegates, one auditor, one public authorities' representative, one regulator.

Generally speaking, most of the respondents were based in Germany. Many of them have work experience abroad (especially in the US), and two of the Germany-based interviewees have a US and a UK background, respectively. Two further interviewees were US citizens living in the US. After conducting the interviews, we went on to categorize the answers. While doing so, we did not make use of a pre-defined category scheme. Instead, we developed categories from the interviews themselves so as to really be able to systematically grasp and group the interviewees' themes.

In the remainder of the paper, we will present the major trends to be identified from our interviewees' statements.

3. RESULTS I.: THE CURRENT SOFTWARE/ IT SECURITY WORLD ACCORDING TO EXPERTS

When analyzing the interviews, we identified a range of major themes addressed by our study participants. Having categorized those themes, we clustered the statements into six major trends. These trends concern the social and technical organization and accomplishment of software development as well as the IT security implications this brings about. Before providing an analysis of the trends, we will shortly summarize our experts' statements on the general character of current and future software systems, and on the current IT security situation as well. By doing so, we intend to give the reader an idea of the software/IT security world our interviewees referred to.

3.1. Software Characteristics

The account we will provide in this section is largely the result of analyzing the answers to the first, entirely open question we posed in the interview: »From your perspective, what are the major transformations that have taken place in the realm of software development in the last 10 – 15 years?« This is, of course, a very broad question that may refer to both the process and the product of software development. The purpose of posing it was to prompt interviewees to indulge in brainstorming (concerning recent and future trends in software development) without any specific prior priming (except for the general introductory briefing of study participants, that is). At this point, we are only interested in those answers that refer to the character of current and future software systems (i.e. software development's products). We clustered these answers into seven categories (in parentheses we indicate the number of participants who mentioned the topic)[4]:

a) Cloud Computing & SaaS: Browser-Supported Access via Midget Terminals (15)

By »Cloud Computing« experts [E1, E2, E4, E5, E7, E8, E9, E10, E12, E13, E14, E16, E19, E21, E22, E23][5] referred to a range of things, such as companies outsourcing storage to external services or individual end users making use of software that does not run on their device locally. Accordingly, the common denominator of all those who, one way or another, addressed Cloud Computing is the increase in using software systems that do not run on the user's (be it a professional organization or an individual) own hardware (be it servers, desktop PCs or mobile devices) but are external to the entity that is using its device. Experts highlighted that the role of hardware will be ever more reduced to possibly small-sized terminals allowing access to external infrastructures featuring computing capacity and software via browser technology. Some interviewees stressed that, in a sense, Cloud Computing marks the return to huge systems, such as those that were ran in the pre-90ies on mainframes. The turning away from these systems that occurred by introducing PCs in the 1990ies is somewhat reverted; the current return to huge systems features a novelty insofar as Cloud Computing infrastructure nowadays comes in a networked fashion.

4 *Please note that we only include topics that were mentioned by at least three study participants. Three actors, according to sociological definition, is the minimum number of what can be called a »group.« In this sense, we only present topics here that were identified by a group of people.*

5 *When analyzing the interviews, each of the 23 experts received a randomly assigned number between 1 and 23. We will thus indicate to whom a particular consideration refers by stating »E« for »expert« and the respective random number in squared brackets.*

b) Distributed Systems and Intensified Networking (11)

Another key theme experts [E1, E2, E4, E5, E6, E10, E12, E17, E19, E20, E21, E22] brought up was the distribution and networking of systems. Furthermore, interviewees believed that there would still be more cross-domain networking, for they assumed that in the future virtually everything will be equipped with sensor technology and embedded software, including the ever more physical systems and critical infrastructures. Ubiquitous computing, ambient intelligence, smart environments, and the Internet of Things can be considered catchwords pointing to different aspects of intensified networking, and our respondents left no doubt that this has wide-ranging implications for software systems hitherto deemed to run in isolation. Finally, experts expected an increase in distributed systems, i.e. in distributed software as well as distributed architectures, functionality and services, as exemplified by Service Oriented Architectures (SOA).

c) Legacy: The Complexity of Evolved Software Ecosystems (10)

The third theme to be accounted for is the historically evolved complexity of the software ecosystem [E4, E6, E8, E11, E12, E13, E16, E19, E20, E23]. Some of the systems composing the latter possess considerable histories with countless developers having contributed to their actual form. While legacy problems coming with complexity and a rather long lifespan are well-known, a nice way to express this was provided by one of our experts, who characterized software systems as »emergent systems« in order to highlight their constant changeability. In this respect, many respondents also identified an acceleration in the change and danger of unmanageability.

d) Mobility & Apps (7)

According to our respondents [E1, E2, E4, E13, E15, E21, E22, E23], a trend that has already been underway for a while, and, moreover, most probably is going to persist, is the further intensification of mobility. Most of the experts agree that apps and rather small special purpose software predominantly in use on smart phones and tablets will further propagate; so will the use of mobile devices. In this respect, one of our interviewees identified a trend towards blurring the boundary between the internal and external use of mobile devices: such devices will increasingly be used in both ways. That is to say that devices will become mobile in general.

e) Compositional Software Systems and Modularization (6)

A feature gaining ever more foothold in the future according to our respondents [E3, E5, E7, E9, E11, E18] is the one of compositionality. Increasingly, software will be made up of more

or less standardized modules or components that will be pieced together, thus building a particular system for the purpose of a particular use case. Many respondents pointed to the automotive industry as providing the blueprint for this kind of breaking up a product into standardized components so as to knit standard elements together afterwards. Compositional systems will be produced not only by software vendors, but software code is and will also be distributed widely via the internet and be integrated into software products, thus being reused.

f) Domain Convergence (5)

An outcome that to a certain degree also, but not exclusively, follows from intensified networking is domain convergence [E4, E6, E10, E17, E19]. For example, cars may be equipped with Advanced Driver Assistance Systems (ADAS) one may connect with a smart phone in order to make use of online maps. In such a case, web-based information processing and safety relevant systems, such as the car's sensor based embedded systems, affect each other. Moreover, the car may be connected to some backbone business IT that sends information used to improve the car's performance. Whereas there are profound implications for security/safety, domain convergence will also go along with hardware convergence, as it is ever more difficult to distinguish hardware devices: laptops, phones and tablets will hardly be specifiable, and hybrid technologies, such as »phablets« (a combination of smart phone and tablet), will emerge.[6]

g) High Availability (4)

The last characteristic we will account for here is the high availability of software systems. This characteristic is, of course, predominantly relevant as it regards systems that are accessed via the internet or otherwise and provide a specific service to the user. A case in point exemplifying the relevance of high availability may be search engines: using search engines as a service is something that is taken for granted in everyday life. The systems are more or less accessible any time, and they have to be in order to be perceived as obvious, everyday life infrastructure. This is, of course, crucial. The more they are perceived in this way, the more users these respective services attract – which is in turn highly relevant, as the companies providing search engines usually rely heavily on targeted ads.

Thus, according to our experts, these are the seven main characteristics of current and future software systems. We will take up some of these characteristics explicitly again when dealing with our major trends in software development below; others will be treated implicitly in the analysis. Before indulging in the latter task, however, we will first provide a scenario of the general current IT security situation as portrayed by our study participants.

3.2. General IT Security Situation

6 *See Newman (2013).*

As regards the general IT security situation, we mainly summarize the experts' statements that referred either to the current situation (e.g. attack scenarios) or to the relevance of and awareness for IT security in the past, present and near future. Whereas we did include one explicit question concerning the relevance of security for different stakeholders in the SD process[7], relevance and awareness were also addressed repeatedly in the interviews by the experts themselves. As our coding and category scheme was developed from the empirical material itself, we clustered such statements into two categories, the content of which we will present here.

3.2.1. IT Security So Far: A 2nd Order Problem

According to our experts, security is by and large still treated as a 2nd order problem [E3, E7, E10, E11, E12, E15, E17, E18, E20, E22, E23]:

- **Customers:** According to our interviewees' experience, customers do not care too much about security [E1, E2, E3, E7, E10, E11, E15, E18, E23]. There are several reasons for this: first, customers and consumers oftentimes lack a proper understanding of security issues: they tend to simply take security for granted when purchasing software products or services without being able to specify their security requirements and without being willing to pay a premium [E10, E15]. While security awareness among customers is apparently highly domain specific [E16, E21], generally speaking, there is not (yet) sufficient demand in order to make security a competitive advantage [E21], for many customers still focus rather on functionality and positive use value of a given software, instead of risk [E1, E2, E10, E16].

- **Firms:** Hence, as taking care of security is expensive without visibly or directly paying off, many firms treat the matter likewise as a 2nd order problem: the functionality-security trade-off more often than not is lost by security [E1, E2, E6, E12] unless security breaches are made public [E12, E20]. At this point, it becomes obvious that the industry necessarily has an interest in increasing awareness, for if customers are not aware, they are only little willing to pay for security. However, if breaches occur, producers (not customers) will be held responsible (not necessarily in legal, but surely in public attention terms). In this sense, it is the producers who carry the risk of suffering from reputational damage. In contrast, if customers are security aware and willing to pay, producers acquire the resources to integrate security in an economically reasonable way, thus decreasing the risk regarding reputation.

- **Developers:** While firms necessarily have a »natural« interest in producing secure products, developers themselves, according to our experts, tend to ignore security if they are not made aware by management [E3, E11, E22]. Some respondents saw one of the reasons for the alleged lack of developers' awareness in a psychological problem: taking care of security in

7 *»I would like to ask specifically for the role IT security plays in the development process. What do you think is the role of IT security for clients of software developers and for software developers themselves?«*

SD for developers amounts to taking a destructive stance when actually being in a productive mood – it requires one to think about the system's deconstruction (or destruction) while still constructing it [E6, E10].

- **The Security-Innovation-Dilemma:** The problem here is that innovations in software oftentimes are produced without the developers having a precise usage context in mind. Systems are created and usage context is only established »in the wild« once the system is running (e.g. Facebook was introduced as a college networking platform before gradually evolving into the most successful online social network worldwide). Therefore, it is still harder to anticipate threat models and attack scenarios correctly. If firms do have a strong focus on security at the outset, they may develop secure applications – however, at the cost of quickly rolling out innovations [E12]. Thus, while awareness for security is a pre-condition for developing secure software [E1, E2, E6, E10, E15, E19], stakeholders' attitudes, nevertheless, tend to be characterized by a »security paradox«[8]: customers, producers and the general public may give security rhetorically greater relevance than when compared to their actual practices [E23].

3.2.2. IT Security From Now On: Increased Relevance and Awareness

However, according to our experts, as there are novel types of technologies as well as novel types of actors and adversaries involved, the IT security situation has been profoundly transformed in the last 10 to 15 years; also, there are increasing stakes. Consequently, more than half of the experts interviewed (12) agree that the relevance of and awareness for IT security has been heightened (albeit emanating from a rather low general awareness level, see above); and they expect security to continue to gain relevance in the future. As reasons for the increase in relevance in the future they indicate a range of things:

- **Embedding Software into All Kinds of Things:** The first reason is to be found in the networking of everything, including critical infrastructures and the physical world, with security and safety issues possibly converging [E1, E2, E4, E6, E12, E14, E15, E18, E19, E22]. With awareness in critical (say, financial or safety-related) areas traditionally being high [E7, E13, E19, E20], we may infer that as the whole world moves on into the digital realm, awareness increases in general. Participants expressed their belief that this might be considered as some kind of »natural« process towards more security/safety (similar to introducing seatbelts in car manufacture [E14]).

- **New Types of Actors/Adversaries:** There are new types of actors involved, such as large scale organizations that dispose of massive resources (organized crime; states involved in industrial espionage, cyberwar, terrorism or counter-terrorism) [E9, E14, E21, E22]. As

8 *The term »security paradox« is borrowed from S.B. Barnes (2006) who coined the term »privacy paradox« when doing IT-related privacy research. »Privacy paradox« describes the discrepancy between the relevance users ascribe to privacy when being asked about it and their actual privacy practices.*

observable in 2013, when US secret service NSA's far-reaching activities became public, these novel types of actors are able to pose Advanced Persistent Threats (APIs), i.e. to orchestrate long-lasting attacks that are meticulously planned and hardly detectable by the target with virtuosity. While our study was carried out before the NSA scandal was covered by the mass media, some participants indicated that awareness had already at that time, i.e. »pre-NSA«, reached the executive management level [E3, E16, E22].

- **The Increasing Complexity of Software:** The more complex the systems, the more difficult (or even impossible) it is to integrate security retroactively; in this sense, the increasing complexity of software systems heightens the relevance of engineering security into the systems right from the beginning [E4].
- **The Increase in Integrating Externally Produced Software Components:** As supply chains in software development become more complex, there are more and more open source or outsourced components that software producers integrate. This makes security more relevant [E18].

As security gains relevance, awareness also increases. Study participants mentioned two further reasons for such an increase:

- **More Attacks and Security Breaches That Gain the Attention of the Mass Media** [E1, E2, E3, E6, E14, E18, E22]: From the explanations above, it follows that such breaches concern state-run as well as criminal activities.
- **Nowadays Literally Everybody Faces Security Issues in His or Her Everyday Private Life** [E19]: For example, when browsing the web, people are made aware of the need to install firewalls, be careful with data in online social networks, take data protection measures, etc.

The bottom line of the experts' statements regarding the general IT security situation is that relevance and awareness have increased in the last 10 to 15 years, albeit emanating from a rather low level. While experts expect IT security to be of continuing or even increasing relevance in the future, whether awareness is going to keep pace, not only informing practitioners' (customers, firms, developers) ideas but also their practices remains an open question. This is the background situation against which we will next present trends in software development and their implications for IT security.

4. RESULTS II.: TRENDS IN SOFTWARE DEVELOPMENT & IT SECURITY IMPLICATIONS

As we stated above, the experts participating in our study raised a manifold of issues. We coded the statements and sorted them by building 14 categories, with one category being sub-divided into two sub-categories. From the categories we further condensed the material by building six clusters identified as the main trends in software development. These trends concern the following phenomena:

1. SD's Working Environment: Dynamic, Flexible, Distributed
2. Agile Software Development
3. Code Generation and Assembly of (Prefabricated) Code
4. Compositional Systems and Modularization
5. Distributed Systems and Intensified (Cross Domain) Networking
6. Legacy: The Complexity of Evolved Software Ecosystems

In what follows, we will treat these trends successively. For each trend, we will first flesh out the aspects our experts accounted for as well as the implications these trends may have for IT security. We will conclude each section by providing a lessons-to-be-learned sub-chapter summarizing what follows from the insights in terms of IT security.

4.1. SD's Working Environment: Dynamic, Flexible, Distributed

As regards the SD working environment, experts pointed out three general trends: acceleration; rendering organizational structures flexible; and distribution of production sites.

4.1.1. Acceleration of SD Processes

One section of the interviews directed the interviewees' attention to the general SD working environment. In this respect, 11 out of 23 respondents identified an extreme acceleration of SD processes as having begun to characterize SD in the last 10 to 15 years. Business pressure is significant [E5, E11, E20, E22], customers expect to be integrated into the development process and new features to be rolled out quickly [E8, E13]. Furthermore, with Cloud Computing, players without massive computing capacity may become competitors [E3, E13, E23]. As a result, there could be less time to focus on requirements gathering and design; developers begin to code early on in the development process [E5, E13], and development cycles get extremely short [E6, E15, E20, E21, E23]. To illustrate this point, we may refer to one of our expert's statement holding that in an app environment, idea to market phases may last one week or less [E23]. Also, enterprises must realize their ideas very quickly: as today the flow of information in general is hardly controllable anymore (due to online social networks etc.), business secrets are not able to be kept for a long time. Thus, companies have to deliver

fast if they want to beat their competitors, for the latter will duplicate product features quickly [E23].

4.1.2. Rendering Organizational Structures Flexible & Fluctuation of Staff

In the interviews we asked participants for an alleged increase in limited working contracts, i.e. whether there is more freelancing, more fluctuation of (and therefore less permanent) staff. The picture that emerged from analyzing the statements is somewhat ambiguous:

- There were eight respondents confirming the trend towards more fluctuation of staff, pointing to phenomena such as more freelancing, contract workers [E9, E11, E12] and intensified division of labor [E17]. However, some respondents were convinced that the trend pertains to large-scale projects in the first place [E15] and is restricted to specific domains, such as SD for the web [E16]. One interviewee saw freelancing as an empowerment of highly-skilled experts who may choose what enterprise to work for and for how long [E12]. Another stated his conviction that the trend is going to persist, at least in Germany; for due to demographic change and insufficient immigration, there will be a shortage of experts in certain areas [E17]. It is noteworthy, though, that of the eight respondents confirming the trend, only one worked in industry.

- In contrast, another eight respondents, seven of whom are working for global players in their respective SD sectors, reported that working conditions in their (Germany based) enterprise are rather stable due to demand for skilled software developers being extraordinarily strong. They held that fluctuation is highly dependent on the region and sector focused upon. For example, there is a much stronger »culture of rotation« in the US (Silicon Valley) and India [E1, E2, E6, E19], in start-up firms or stock corporations [E19], and in global outsourcing centers [E8].

To explain the differing views on personnel fluctuation, we refer to the distinction of »fluid organizations« and »caring companies« (Spath 2012): while the former tend to recruit so-called »cloud workers« on a short-term contractual basis, the latter strongly bind employees to them to »smooth cyclical developments by long-term 'internal' measures of flexibility« (ibid.: 18). From this assessment we derive that in »fluid organizations« there will be a lot of fluctuation in the sense that the staff base will be unstable; hence, »outsiders« will frequently join and leave such companies. We will call this external fluctuation. As regards »caring companies«, they are more strongly characterized by what we will call internal fluctuation. This latter form of flexibility is indeed what most of our interviewees addressed in the first place. According to them, what we will increasingly face is internal structures being made flexible, i.e. a transition towards

flat hierarchies, internal fluctuation of staff and non-long duration or »fluidity« of internal organizational structures [E13, E23]. Moreover, employees do or will expect not to be bound to a particular place and time when fulfilling their tasks [E3, E6, E19, E23].

While a certain degree of external fluctuation was deemed positive (breathing some fresh air into a company's SD processes [E9, E15, E18, E21, E23]), experts nevertheless agreed that there would be problems if external fluctuation passes a certain threshold, such as isolated freelancing narrowing the developers' perspective, thus keeping her/him from finding lean and possibly simple solutions that do not offer more attack surface than necessary [E16]. Likewise, respondents considered internal fluctuation beneficial for employees' creativity and innovativeness [E3, E6, E8, E10, E12, E18]. Further, they identified IT security problems this may bring about. There are two major problems associated with internal and external fluctuation:

- **Problems with External Fluctuation: Possible Leakage Of Internal Expertise**
  As regards the leakage problem, several respondents [E6, E9, E10, E15, E18,] mentioned two issues: first, the possibility of hired short-term freelancers to build back-doors into software products or harness their detailed knowledge of the code by attacking the vulnerable parts of the software systems once they do not work for the company any longer. Second, developers hired may feed expertise they have acquired in one company into a competitor's business. While this may come across as malicious intent, interviewees held that it may also happen unintentionally. External fluctuation may raise loyalty problems that can occur as a result of developers applying knowledge they have acquired in the course of the different projects without being necessarily able to tell what knowledge was developed in which project.

- **Problems with Internal Fluctuation: Loss of Expertise & Weakening of Know-How Transfer**
  The main issue in this regard is to build continuity into SD processes: seven participants mentioned the danger of losing expertise by developers moving on to a new position within the company or to a competitor [E3, E4, E12, E15, E16, E18, E20]. Yet even if developers stay within the same firm, internal fluctuation threatens to take away expertise from the sector where it was built. Similarly, an effort is to be made to make sure that the expertise established is transferred to »newcomers« so as to provide continuity to the SD process. While instructing new team members always involves frictional loss [E20], without taking measures that make continuity and know-how transfer possible, firms will be dependent on a small number of key developers and unable to cope with the complexity of security issues [E20]. Thus, code quality may suffer [E3].

4.1.3. Distribution of Production Sites

As regards the distribution of SD, 22 out of 23 interview partners confirmed the trend, with one implicitly (unintentionally) skipping the question. Distribution may take two distinguishable forms, with different problems associated with them: the distribution of production sites, and outsourcing the production & integrating 3rd party components. Before listing security issues associated with these two forms, we would like to clarify that many of our interviewees deemed distribution generally indispensable. They highlighted that for large companies it is mandatory to dispose of production sites at localities where specific types of expertise are available [E1, E2, E12]. In respect to outsourcing and integrating 3rd party components, respondents held that no business is able to produce every software component from scratch [E1, E2]; as software systems are getting ever more large and complex, it would be too costly to produce everything in-house [E20]. Therefore, enterprises reduce costs [E7], and increase efficiency and speed by distributing the SD process [E9, E16]. In what follows, we will indicate the security issues possibly arising from the different forms of distribution.

- **Distributed Production Sites of One and the Same Company:**
  As far as large-scale enterprises are concerned, they tend to maintain different production sites (or expand via Mergers & Acquisitions) to harness geographically bound expertise and the innovative capacities dispersed throughout the world [E1, E2, E3, E21, E23]. Such distribution threatens to bring about a range of well-known coordination problems, e.g. lack of control, frictional loss, misunderstandings, coordination problems resulting in redundancy or architectural issues. Those problems are due to increased difficulty in effectively organizing communication in geographically distributed settings [E1, E12, E14, E17, E19, E20]. If the scale of distribution is transnational or even global, cultural differences concerning hierarchies, norms or legal frameworks – while potentially inducing diversity and creativity and extending a company's knowledge base [E1, E19] – may also further aggravate the communication/coordination problem [E10, E19], thus threatening to generate security issues [E23]. Geographical distribution of cooperating production sites makes it harder to build teams and develop a shared vision of the product to be produced, i.e. to develop and share a common goal, which according to experts is one of the greatest sources of IT insecurity [E1, E2]. Accordingly, one expert stated categorically that the greater the number of developers involved and the more distributed and decentralized the SD process, the worse the quality of the resulting code [E14].

- **Outsourcing & Integration of 3rd Party Code (Including OS):**
  There is a global division of labor and a »modularization« of production via partnering, outsourcing, offshoring, subcontracted supply, etc. Whereas almost all respondents (22) agreed on this, there are also several problems. The first one concerns trustability in terms of supply

chain security [E21]: there is always a risk associated with collaboration if outsourcing partners are only trustable to a limited degree [E9]. Outsourcing may result in industrial espionage, IP leakage, draining of know-how, but also in the withering of one's own competencies [E4, E6, E7, E9, E16, E19]. Consequently, many experts argue that critical components (authentication, cryptography) should be produced in-house, if possible [E8, E14, E15]. Still, as regards the problem of loss of know-how, experts identified a risk of becoming dependent on external outsourcing partners [E1, E2, E7, E12], particularly if an ongoing modification of the product is necessary [E10]. Second, outsourcing complicates the precise specification of requirements [E4]. As requirements are, according to experts, impossibly specifiable unambiguously, outsourcing may lead to increased costs and coordination overhead, thus requiring more formalization, testing, QA and validation [E16, E22]. Requirements specification has become more difficult because of market dynamics and fast requirement changes [E20], so it is even more important to explicitly agree on the features of outsourced components, including security and architectural standards; and to attach importance to precisely laying these down in contracts [E13]. Third, there can be »compatibility problems«: if the code base of one's own and the integrated component are not known in detail, unutilized features may increase the attack surface [E6]. Also, using different types of Integrated Development Environments (IDEs) or tools may cause security issues [E22] (which is why some of the experts we interviewed provide outsourcing partners with complete IDEs [E13]). As a result, outsourced services threaten to deliver poor quality [E4, E22] and thus poor security. Fourth, and mainly related to integrating 3rd party components, the problem remains that the quality of, and the meeting of security standards by, externally produced (proprietary or OS) components must be properly reviewed [E6, E7, E8, E16] although so far there is no proper metric to measure security [E1, E2, E10, E15, E17, E18, E21], especially if it comes to OS components [E18]. Consequently, experts suggested that 3rd party components should never be integrated as black boxes [E16], especially in safety areas [E6].

4.1.4. Lessons to Be Learned

Highly dynamic, flexible and distributed working environments are characterized by rapid change in terms of the social networks of collaborating actors; this poses problems insofar as rapid change does not allow for the establishment of strong social ties. At the same time, however, what implicitly follows from our experts' statements is that the weaker the social relationships between collaborating actors, the greater the probability of security issues to emerge. This makes for the following lessons to be learned:

- **Acceleration & External/Internal Fluctuation of Staff:**
  In respect to acceleration and external/internal fluctuation of staff, the challenge is to build continuity into ever changing networks of cooperating partners. Companies are well advised to find ways to protect as well as capture their knowledge and expertise.

- **Distribution of Production Sites:**
  Mitigating communication and coordination problems requires companies to invest significantly in proper communication (e.g. making face-to-face meetings possible, making use of up-to-date communication media etc.) [E1, E2, E16, E19, E20] and to enable team members to build strong team relationships. At best, improved communication is accompanied by defining requirements as accurately as possible upfront [E3, E13] so as to make sure that security standards are indeed met.

- **Outsourcing & Integrating 3rd Party Components:**
  This is most probably the area that poses the greatest challenges, because social relationships tend to be the least stable ones. Weak binding of developers, for example, potentially raises loyalty problems, heightening the risk of IP leakage, etc. Our listing thus comprises a number of issues:
  - Smart outsourcing: outsourcing critical components of the overall system is problematic.
  - Specification: specifying requirements as precisely as possible upfront, including security standards, is required to mitigate security issues.
  - Establishing long-time and trustable relationships: over time, relationships to partners will strengthen, thus involving mutual control, dedication, and loyalty.
  - Housing partners: housing outsourcing partners within one's facilities may be considered. If doing so does not evoke the danger of IP leakage, working in spatial proximity will strengthen mutual dedication and facilitate communication. Outsourcing entails a certain loss of control – spatial proximity may recapture control to a certain extent.
  - Harmonization: any attempt to harmonize the tools, frameworks and IDEs used by the outsourcing partner with one's own tools will potentially decrease the number of vulnerabilities.
  - Measuring security: not only when discussing distributed production interviewees lamented the lack of metrics and tools to measure software components' security level.
  - Certifying security: if there were proper metrics, it would be easier to certify products. Until then, it is advisable to work with partners whose processes have been certified (ISO-certified, common criteria etc.).

4.2. Agile Software Development

According to our experts, whereas a certain professionalization has already taken place in SD [E5, E8, E9, E10], further process systematization is still required [E3, E8, E19].[9] When it comes to methodologies, our respondents identified the widespread adoption of agile (and lean) methods or values as a major trend in SD[10] [E1, E2, E3, E12, E14, E15, E17, E18, E20, E21, E22] – especially of Scrum and Continuous Integration [E1, E2, E14, E18]. Study participants explained that there are several factors that determine the appropriateness of a particular

9 *While there are various methodologies on offer, recent research indicated that only 43 percent of software development organizations have a defined development process in place, with only 30 percent in fact adhering to it (Ponemon 2013: 5).*

10 *Research revealed that of those companies that follow a defined process the bulk sticks to agile methodologies, followed by waterfall approaches (Errata 2010). In January 2010, an analysis of Forrester Research on the adoption of agile methods in the industry indicated that about one third of software developers rely on some agile approach; the rather traditional waterfall model, in contrast, is only used by thirteen percent. See the article »Evolving Agile« in: Information Age. Insight and Analysis for IT Leaders, URL: www.information-age.com/technology/applications-and-development/1596528/evolving-agile (19.2.13).*

methodology, such as project scale and number of stakeholders involved [E6, E8, E15, E16, E17, E19]; the stability of requirements defined upfront [E1, E2, E4, E17, E19, E20, E22], and business [E3] and safety criticality [E4, E6] of a given software. There is a certain tendency in our interviewees' statements: the bigger the project, the more stable the requirements, and the more critical a piece of software, the more likely the SD process is to be plan-based, sequential, extensively documented and so on – in other words, the more it tends to resemble the waterfall model [E4, E6, E19]. However, at the same time many of the interviewees harbor doubt regarding the appropriateness of waterfall-like approaches – either in general or in environments characterized by massive business pressure and acceleration [E1, E2, E11, E17, E20, E22]. There are two things following from these considerations: first, given that the SD business in general is characterized by pressure and acceleration (see above), it is most likely that the trend towards agility is going to persist; second, as security is gaining relevance, the question is how to integrate security into agile approaches. Clearly, about one third of our interviewees explicitly stated that agility, in general, is not at all an obstacle to integrating security into SD processes [E8, E13, E15, E18, E19, E21, E22]. The question remains, though, whether the widespread adoption of agility has implications for IT security. In this section, we will present an analysis of our experts' account of agility[11]: we will first briefly summarize some benefits of agility[12] and then discuss possible drawbacks and implications agility may have for IT security; next we will summarize the lessons to be learned before sandwiching a short interlude that discusses whether there is a novel approach on the horizon.

4.2.1. The Benefits of Agile Software Development

Agile software development is considered quite appropriate in highly dynamic market environments [E17, E20, E23] where a strong focus on end user requirements is mandatory [E4, E5, E15, E20], and there is no reason to expect that market dynamics are going to cease in the future [E17]. Also, agile was deemed by most interviewees to be the weapon of choice when projects are explorative, are about innovative development, offer leeway for developers and do not involve too complex software architecture [E4, E6, E16, E20], although there were also experts deeming agile SD appropriate for any type of project [E1, E2]. Regardless of the context in which agile SD is implemented, respondents clarified that agile SD must not serve as an empty label behind which developers hide chaotic SD practices [E20]; moreover, agile SD neither frees developers from a priori specification of the system to be developed nor renders systematic planning and procedures obsolete [E20]: However, it still presupposes developers keep track of the global system to be developed [E10] and is not tantamount to setting aside proper documentation [E1, E2, E4, E10, E12, E16, E18, E15]. Having said this, according to our experts, there is a range of benefits coming with proper agile implementation:

11 *Without defining »agile« explicitly, we stayed with the characteristics formulated in the »Agile Manifesto« (www.agile-manifesto.org/)). More generally, we used the term as an umbrella for non-sequential, non-linear approaches, with short development and strong feedback cycles (customer involvement), and fast output of features. Teams are flexible and roles are not rigidly fixed. Our interviewees tended to take up one or several or all of these features when discussing agile development.*

12 *Please note that our account here is in no way exhaustive. There may be a manifold of benefits that we do not put on record: as we are mainly interested in this report in IT security implications of the trends identified, our discussion of possible drawbacks and security implications will be much more detailed than our appraisal of agile SD's benefits (in addition, we take it that agile values are becoming widely accepted anyway).*

- **Speeding Up SD in Unstable Environments:**
  Generally, agile SD allows SD processes to speed up, account for the dynamics of emerging requirements, and integrate customers/users into the SD process [E8, E4, E6, E12, E14, E22].
- **Empowering Developers and Teams:**
  From the developers' point of view, agile SD strengthens entrepreneurial mentalities as well as heightens individual freedom and responsibility by equalizing and empowering team members [E1, E2, E8, E12, E13, E22].
- **Improving Coordination:**
  Agile SD decreases loss of time spent for meetings and coordination; it fosters transparency and collective learning, and improves predictability and productivity [E1, E2, E8, E13, E12, E22].

What we can further conclude from the statements is that the effectiveness of agile methodologies depends on the way they are implemented by a given company; in this sense, experts stressed that introducing agility does entail a change in business culture and organizational structures of the wider organization indeed [E8, E18].

4.2.2. IT Security Implications of Agile Software Development

Many respondents mentioned that agile values affect SD processes in certain ways. In the following sections we will discuss the implications this may have for IT security. From our analysis of experts' statements, it follows that these implications eventually revolve around the notion of expertise. In the next part, we will map out four areas where agile SD affects security: individual expertise; scalability of expertise; preservation of expertise; and the involvement of laymen expertise.

4.2.3. Changing Requirements and Developers' Expertise

One of the fundamental agile values is »welcome changing requirements, even late in development.«[13] However, altering or introducing new requirements may have security implications [E14]. Also, change in requirements, design and features threatens to bring about problems for organizations insofar as any change on the SD level may have repercussions for the management, QA, documentation and support level [E16]. Moreover, when outsourcing components, it may prove problematic to change requirements late in the process; hence, how well agile SD works with outsourcing strategies [E13, E8] is questionable. For these reasons, some of the experts generally advised against following agile approaches when dealing with security relevant system components [E6, E16]. In general, however, experts agreed that there still is the necessity to specify security requirements of the overall system upfront [E4, E8, E18, E20, E21, E22, E23]. If these requirements are treated as user stories, then in analogy to functional fea-

13 *www.agilemanifesto.org/principles.html (10.10.13)*

tures [E14, E22], it has to be considered that they are very special kinds of features: they can't be modularized nor integrated in the final stages or a posteriori; and they have to be regarded from the overall system's view [E4, E11, E14]. What follows from these considerations is that agile SD demands a great deal of individual developers in terms of security expertise. In fact, as roles of and within SD teams tend to be more fluid, every team member needs to have security awareness and expertise. This is because every team member must be able to specify security relevant requirements as well as judge whether and how some change in requirements affects security. In this sense, agile SD increases the security expertise that any individual developer is required to have. Whereas not every developer has to be an IT security expert [E15, E18], our interviewees were quite clear about their opinion that individual expertise is even more important than having institutionalized, formal security processes in place [E1, E2, E14, E15, E18, E19, E20, E21]: agile SD's emphasis on the individual in turn requires the individualization of security expertise (to a certain degree, that is). However, by and large developers still lack adequate security expertise due to insufficiencies in education [E1, E2, E6, E9, E10, E14, E15, E16, E19, E23].

4.2.4. Systematic Processes and Scalability of Expertise

Insofar as agile SD emphasizes the individual and downplays processes, it may evoke a weakening of the overall system's perspective [E1, E2]. This may prove problematic, since an overall system's view is necessary to take care of security[14] [E4, E10, E17]. Furthermore, while the formalization of processes may help to take up an overall system's position, a defining characteristic of agile SD is in fact the rejection of over-formalization.[15] Experts agreed that while formalization is necessary [E3], too much actually threatens to stifle the SD process – as does too much taking care of security [E12, E16, E22]. In this sense, agile SD challenges companies to implement systematic security processes [E14, E18], which allow security expertise to be distributed as widely as possible without overburdening individuals with security concerns [E21]. Any secure software development life cycle (SDLC) must be clearly defined and tailored specifically to the company adopting it [E3, E15, E18, E20, E21], so testing and planning in companies that have an agile SD process in place must be adapted likewise. In spite of the specifics of implementing an SDLC in relation to a company's organizational culture, experts mentioned three measures that will be considered:

- **Security Early On:**
  In agile SD it is even more crucial to integrate security into the SD process early on [E18]: threat models and solutions are to be specified early on and followed up over the whole SDLC; there must be early quality assurance, early (immediate) and systematic testing [E12] instructed by experts [E14, E21]; immediate feedback and fixing is necessary in the case of vulnerability occurences [E1, E2, E5, E18, E21]. This is even more

14 *For example, according to one expert, when agile SD is realized as test-driven development, it is not possible to provide universal system's specification, as tests only refer to particular courses of events; while universal specification is, of course, extraordinarily difficult to achieve the problem with test-driven development is that it suggests completeness with being able to provide for it [E17].*

15 *The manifesto reads: »Individuals and interactions over processes and tools (…) Responding to Change over following a plan.« www.agilemanifesto.org/ (4.10.13).*

important considering there is a clear trend towards continuous integration [E1, E2, E14, E18].

- **Scalability of Security Expertise:**
  Following an agile approach, companies may decide to introduce sprints that are dedicated to security relevant user stories. This is only feasible if there's sufficient time granted to those security sprints [E8] and if there are security experts keeping track of the overall system despite breaking SD down into sprints and feeding vulnerability detection back into the development process for fixing [E14, E21]. As it is not very easy to implement feedback mechanisms for immediate bug fixing in large-scale organizations [E14, E18], and as individual developers cannot be expected to have detailed knowledge of any testing tool (though such knowledge is required so as to not have too many false positives etc.), security specialists are supposed to manage testing tools centrally, with the developers able to choose correctly and use them if required [E10, E14, E21] so as to make security expertise scalable. One problem here is that while security awareness in companies has improved in recent years, the number of security experts advising development teams in industry still tends to be rather low [E14].

- **Strengthening Exchange:**
  Given that functional features and security issues tend to be treated by different people [E23], increased exchange between developers and security is desirable [E10]. Importantly, though, this not only requires individual developers to be equipped with the skills and tools to realize and fix security issues [E18], but also security experts to acquire in turn agile SD expertise so as to know how to integrate security into such processes [E23]. In this sense, there is not only a need to provide developers with security expertise, but also to provide security experts with agility expertise.

4.2.5. Preservation of Knowledge and Expertise

The biggest problem with agile SD, according to experts, is to develop a long-term perspective [E1, E2, E10] and to preserve expertise in spite of the dynamics of agile projects. Agile SD does not focus on lengthy conception, which may be unproblematic for developing many types of software [E5]; however, if it comes to complex long-living software systems that run for years or even decades, with staff varying repeatedly and considerable legacy involved [E10, E16] the need arises to capture the assumptions and knowledge that went into the coding – which is not what agile SD particularly supports in the first place [E1, E2, E3, E4, E22]. Having said this, the analysis of the interview material allows to identify two strategies to preserve knowledge and expertise:

- **Documentation:**
  Documentation has always been an issue regardless of the methodology [E15, E23]. Still, it is traditionally held to cater to a long-term perspective including security [E14, E16, E17, E21]. The agile manifesto cherishes »working software over comprehensive documentation«[16], which is why some of our interviewees at least implicitly held that agile SD may not be well suited to preserve knowledge via documentation [E4, E16, E17, E19]. While some of our respondents expressed their belief that quality of documentation is not at all affected by the particular approach to be followed [E12, E15, E18, E23], others were rather confident that agile SD does not neglect [E10, E21] – but quite on the contrary may improve – documentation [E6]: in Scrum, for instance, if backlogs are being cared for properly, the resulting archive would amount to much more fruitful system documentation compared to the voluminous project documentation in waterfall-like approaches [E1, E2, E8]. Whatever one's take on this matter, what we may learn from expert statements is that traceability of design and coding decisions will become even more important in the face of complexity, legacy [E10] and the permanent, rapid change that nowadays characterizes the SD business. Especially if agile SD is also meant to be adopted in safety or those areas where certification plays a role, in-depth knowledge of the system, transparency and traceability of the code base are indispensable [E5, E19, E22]. For this reason, it is currently quite difficult to adopt agile SD in safety areas [E5, E6, E16], or may have to be adapted to those areas [E19]. Hence, following our experts, we would like to plead for the improvement of explicit rules and techniques allowing for such traceability [E18]; examples mentioned by study participants include companies generating data bases capturing knowledge or experience in reaction to increased turnover of employees [E4] and backlogs [E1, E2]. Nevertheless, to date there is a lack of such techniques, no matter what SD methodology is in place [E5].

- **Organizational Learning:**
  Inventing or harnessing instruments that safeguard traceability will foster companies' organizational learning, an aspect that is in fact propagated by agile SD.[17] Accordingly, experts explained that »security« is at best to be considered a maturing process, aiming to improve processes and products. Further, companies can be considered as social systems that need to take measures (e.g. in terms of awareness, education, policies, communication, design etc. [E14]) to learn from past experiences, thus improving their security performance. One of the measures taken is to define explicit goals and to attempt systematically to reach them [E1, E2, E12, E14, E22] (what is still lacking, though, is metrics to measure progress consistently [E1, E2]). Yet, what has to be kept in mind is that due to fluctuation of staff, distributedness of SD, and extensiveness of SD collaborative networks learning processes have to stretch beyond small teams where members work in spatial proximity to each other.

16 *www.agilemanifesto.org/ 10.10.13*

17 *»At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.« www.agilemanifesto.org/principles.html*

4.2.6. Involving Laymen Expertise

The Agile Manifesto is quite explicit in valuing »customer collaboration over contract negotiation.«[18] In this respect, agile SD has a profound strength on offer that can be harnessed for IT security. Our study participants left no doubt that a good deal of IT security has to be managed not by those who have developed the software, but by those effectively running or using it, i.e. customers, employees, IT operation divisions, or end users [E10, E14, E18, E21]. Also, deployment is part of the complete SDLC, and in the deployment phase security issues may arise as well [E15]. Moreover, there is a trend towards separating SD from IT departments (both in-house) with misunderstandings or issues possibly occurring at the point of transition: operators may lack sufficient knowledge concerning security-relevant aspects of the software being operated, and they may not know how to monitor and check the software systems for security issues, even if tools are available [E14]. In this sense, human beings may create potential security gaps [E18, E19], and some experts argued that security eventually is only definable by customers/users [E1, E2], therefore software systems shall be developed in reference to them [E21]. In this sense, one of the ways to narrow the human security gap is to produce systems that are easily usable. Quite generally, usability gains relevance [E4, E8, E18, E20] as a security factor because of software becoming invisibly embedded into physical space thus invoking safety issues [E4], because of software governing ever more aspects of everyday life [E18], and because of the complexity of software systems [E16, E18, E20]. At the same time, however, if anything, customers and end users communicate security requirements insufficiently at best [E1, E2, E3, E7, E10, E11, E15, E18, E23]. The same applies the other way around: risk assessments [E10], for example, and security levels of a given software are often improperly communicated to end users and customers [E9]. By involving customers and end users more systematically, agile SD promises to improve usability, and by doing so, to harness the expertise of those who effectively use the systems. Achieving this requires to integrate not only security, but also usability as early as possible into the SD process [E4, E18].

4.2.7. Lessons to Be Learned

As we could see, our analysis reveals a number of implications agile SD has for IT security. There are the following lessons to be learned from the experts' statements:

- **Developers' Expertise:**
  The permanent change in requirements which is accounted for in agile SD, the changing roles of team members and the stress on the individual increases the relevance of individual developers' security expertise. Developers must be able to consider security implications of new features factored into the system, and they must have the skills to deal with security issues no matter what role they play in what project. From this there follows the challenge:
  - Improvement of developers' security education.

18 *www.agilemanifesto.org/ 10.10.13*

- **Scalability of Expertise:**
  Still, responsibility for security cannot be shifted entirely to individual developers. There must be systematic integration of security processes, including having security experts at central positions who distribute expertise as widely as possible throughout the organization; who coordinate testing, feed detected vulnerabilities back into the SD processes, and so on. From this there follow four lessons:
  - Integrate security early on in the SD process (specifying security requirements, threat models, solutions) is mandatory.
  - Installing systematic security processes, including immediate feedback and fixing is required.
  - There is a need to make sure to employ a sufficient number of security experts and enable scalability of expertise.
  - Enabling security experts to develop expertise also in agile SD makes them know how to integrate security in lightweight SD methodologies.

- **Preservation of Expertise:**
  While agile SD is about adapting rapidly to changing circumstances forced by market pressure, software systems tend to run for quite a while. This makes it mandatory to preserve expertise and knowledge and to trace decisions that went into constructing the system. To achieve this, there is a range of measures that can be taken:
  - Development of documentation strategies which are feasible in practice.
  - Ensuring that organizational learning is possible.
  - Development of techniques that allow to capture knowledge and expertise more easily.
  - Development of metrics to measure maturation of security processes and software products.

- **Involving Laymen Expertise:**
  Industry and research are called upon to take usability seriously, if it comes to security. Thus:
  - Usability must be integrated into the SD process early on by involving customers/ end users.
  - To this end, finding still better ways to involve customers/end users is desirable.

4.2.8. Interlude: A Hybrid Methodology on the Horizon?

The statements of our interviewees suggest that there is a need for the emergence of a hybrid SD methodology which combines the strengths of agile and waterfall approaches and is able to incorporate the benefits of agile SD without being based on its at times unrealistic assumptions concerning actual working environments (e.g. accounting for distributedness, s. below). In fact, what is striking is that many respondents indicated that in practice there are rarely pure waterfall or agile SD processes observable; instead, there is either co-existence of sequential

(waterfall) and agile SD methodologies in one and the same company, or there are hybrid methodologies already implemented [E3, E4, E6, E8, E13, E16, E17, E22]. Accordingly, some experts expect a new formal methodology to emerge [E1, E2, E3, E23] that integrates the light-footedness of the agile approach with the capacity to capture and preserve expertise and knowledge that plan-based techniques feature. A weakness of the agile SD process, experts held, is that it tends to increase the dependency of companies on individual developers [E22]; in some cases, such a dependency may threaten to ruin those companies if particular developers leave [E4]. Moreover, given the dynamics and the »distributedness« of SD processes, from the companies' point of view it becomes necessary to introduce techniques that allow for continuity in spite of permanent change. Thus, one participant criticized that agile SD is based on the assumption of a single production site, with all the developers involved being able to communicate face-to-face and in real-time [E3]. As far as the trends we identified so far are accepted as true, SD to a great extent is or will be rather characterized by independent teams that are geographically distributed on a global scale, perhaps working on project-based contracts, with individual developers writing or generating code pieces that will be assembled and integrated into one overall system [E3]. In addition, the systems which are to integrate those pieces may have already acquired complexity in the course of a considerable system's history. As systems evolve it is necessary to dispose of appropriate SD methodologies allowing for as it were channeling this evolution. However, existing methodologies generally tend to be based on the assumption of SD from scratch, with the SD process having a precise »point zero«; hence, what is required is a notion of »evolutionary development« [E5] that allows to consider software systems as emergent ones at the outset.

Against the background of these considerations, the hybrid methodology to emerge will be required to allow for the following things:

- to reconcile large-scale projects and management with flexibility on the actual SD level [E17];
- to allow for introducing agility also in the »safety-world« where accurate documentation is mandatory to get one's product certified [E19];
- to make sure at the outset that long-running systems being maintained and modified by varying staff can be cultivated – developers taking care of V10.0 need to be able to understand the coding of V1.0 [E22];
- to reconcile formality (to capture knowledge [E1, E2, E3, E22] and afford scalability [E1, E2]) and leeway (to get the job done [E22] without stifling creativity [E1]);
- to keep code comprehensive also under the condition of distributed SD, with developers possibly being mobile and contributing small code portions to the assembled overall system [E3];
- to account for the evolutionary or emergent character of many actual software systems.

4.3. Code Generation and Assembly of (Prefabricated) Code

According to about half (12) of our study participants, today's SD entails relying on a manifold of resources: instead of working in an isolated way, developers generate and sometimes also share code. Thus, there is a tendency to not write code directly anymore but to develop software on a more abstract level [E12], either by delegating code generation to tools or by assembling pre-fabricated code. Thus, many experts pointed to a massive increase in using frameworks [E12, E13, E15, E22] or to the possibility of model-based SD where domain experts define the system's requirements, the underlying code of which is then generated automatically [E4, E18, E19]. Another relevant trend is what we will call the »democratization« of SD. In this respect, many respondents stated that nowadays it is increasingly easy for non-software developers to assemble pre-fabricated, reusable code bits, say, to develop apps, for instance [E3, E5, E10, E11, E12, E18, E19, E22, E23]; this allows also end users to indulge in the task of programming within pre-defined limits (»end user programming«, [E4]). Another aspect to this trend is the emergence of crowd sourcing development. This not only threatens to render the jobs of moderately skilled software developers superfluous, but also transforms the task of developers working for companies: the latter are asked to be able to manage a crowd sourcing social network in the first place, which is why hiring companies will focus on project management skills rather than on excellent programming skills [E3, E12]. Thus, the role of software developers is to manage the production of software being produced externally (as Open Source, crowd sourced or outsourced components etc.), and to make sure these components are integrated into a »global« system [E3, E5, E10, E11]. To be sure, while more than half of the experts interviewed stressed the trend towards assembling and generating code, three respondents clarified that the automation of code generation has its limits, and that for the time being there won't be any total automation of programming [E1, E2, E4]. Nevertheless, there is a clear indication that the trend towards generating and assembling code is likely to increase in the future. In this section, we will map the security implications going along with the different aspects of this trend.

4.3.1. The »Democratization« of Software Development

By »democratization« we refer to the fact that ever more non-experts, i.e. people who are not formally (or not at all) trained as software engineers start developing applications [E3, E10, E12, E23]. These »laymen developers« may acquire coding skills on websites such as codeacademy.com or create apps without having any coding skills at all. For example, at http://www.maz-digital.com/ print publishers are provided with software to develop apps that allow for moving print content to smart phones. The website reads: »No Programming Required. Seriously. You don't need to be a developer to have an app. If you are able to navigate this website, then you are qualified to create a MAZ app.« In such environments the ability to write code quite

obviously has become less important. This also applies to companies' permanent app designer staff: it cannot be taken for granted that they have been trained as software engineers – they might be graphic designers etc. [E23]. Idea-to-market-cycles in app markets have become extremely short, in some cases ideas are realized and rolled out within one week [E23], and some of the apps being produced in this way meet with breathtaking success. The security issue arising from this, according to our experts, is that potentially vulnerable or unreliable pieces of code are introduced to the code-ecosystem [E3, E23]; self-learned developers tend to be rather unaware of security issues [E10, E12], and given the rapidness of idea-to-market, there is very little time to take care of security in any case [E12]. If we combine these findings with the insight that there is a convergence of domains – including security and safety domains (for example, a smart phone connected to a car's software system) – security implications are huge indeed [E12]. The question arising is how to integrate security into self-teaching and do-it-yourself code generating platforms [E12], for example by introducing easy to grasp guidelines; by providing robust, intrinsically secure code pieces [E23] (we will discuss this idea below again); or by creating components that are able to interact with an environment that is insecure in general [E5].

4.3.2. Generating Code in Industrial Software Development

Also, in more classic industrial software engineering there is a trend of developers increasingly relying upon tools which help them to generate code, such as Integrated Development Environments (IDEs) [E16], frameworks and libraries [E5, E15]. Experts pointed out that the use of frameworks and IDEs somewhat facilitates SD as they let developers do their job on a more abstract level, while the framework takes care of issues such as security. This may increase the latter [E12]. The same goes for Domain Specific Languages [E4]. Moreover, expertise, say as regards specific programming languages, may be incorporated into frameworks thus decreasing the need for individual developers to be experts in a particular language [E13]; this may come as a relief, for it seems impossible for individual developers to keep pace with rapid innovations in the manifold of programming languages – and tools are deemed to have the potential to counteract the security issues arising from this [E14]. Still, there are some problems:

- **Framework Acquaintance:**
  When delegating security to the framework, the latter's owner is the one who is in charge of making sure the code to be generated is secure [E12]. The problem that occurs here is that security issues arising on the lower levels of abstraction cannot be solved on the higher ones; in other words, if the code generator is robust and does not introduce vulnerabilities all the products that are generated are likewise robust [E11, E15]. In this sense, there is a trade-off: the uncertainty on the lower levels of abstraction goes along with an increase in the basic security concept [E11]. For these reasons developers are bound to know their

framework very well; if developers do not know a platform's weaknesses its usage may in fact decrease security; provided they have detailed knowledge the framework's presetting facilitates secure programming, though [E15].

- **Proliferation of Frameworks:**
  In this sense, the proliferation of frameworks themselves may pose a threat to security in that it may overburden developers with »framework proficiency.« Also related to the proliferation problem is the difficulty in harmonizing components that were produced by relying on different code generating tools – it may be hard to keep track of the overall system, or inconsistencies within the system may arise due to too many tools producing inconsistent »meshworks« [E15, E22].

- **Round-trip Engineering Required:**
  Another problem with code generation identified by one of our respondents is its one-way character. That is, if the generated code is modified ex-post, these modifications are not fed back on to the modeling level. If the system is modified on the modeling level, say for a 2.0 version, modifications at source code level are not preserved, and this may lead to inconsistencies [E16]. What is required therefore is round-trip engineering tools that allow for consistency on any level of abstraction.[19]

- **Improving Tools:**
  Quite in general, experts demanded further innovation in the framework and model-driven SD (MDSD) realm. As regards the latter, there is a need to render the source code building blocks generated by MDSD intrinsically secure so as to render the systems secure that domain experts produce [E19]. Another desideratum frequently mentioned was the integration of security techniques, cryptographic techniques, and (e.g. static) analysis tools into frameworks and IDEs [E11, E12, E13, E14, E21], and the integration of automatic repair functions in analysis tools [E21].

Although there is no valid empirical evidence so far that code generation improves security [E21], automation of security via code generation, according to our experts, promises to increase security. The improvements listed here may contribute to this.

4.3.3. Sharing Reusable Code (on the Web)

The possibilities for developers – be they app designers or concerned with huge long-running systems – to exchange expertise and reusable code have certainly increased in the last decade. Therefore we asked our experts about the risks associated with interactive cross-enterprise communication via fora such as stackoverflow.com; and with sourcing 3rd party code on the

19 *E.g. see the Roundtrip Engineering NG tool at www.uml-lab.com/de/uml-lab/features/roundtrip/ (11.10.13).*

web. As far as communication is concerned, experts considered a – very, very moderate – risk to overshare internal information, thus compromising business secrets or making vulnerabilities public [E6, E10, E12, E13, E14, E15, E20, E21, E23]. However, while many respondents saw individual developers responsible for not oversharing [E1, E2, E6, E10, E18, E19, E20], they were eventually very positive about exchange and communication, because of the »wisdom of the crowd effect«: rendering a solution robust by exposing it to so many peers [E4, E6, E7, E10, E11, E13, E14, E15, E18, E19, E23]. Still, experts mentioned the problem of unconsidered solution sharing or even the reuse of code in general [E1, E2, E11, E16, E22]. Yet, while some respondents restricted the »unconsidered sharing«-problem to developers who are not trained well-enough [E19, E6] or lack an analytic attitude [E11, E16], one expert mentioned the time pressure in the software business as a factor that drives developers to integrate solutions found on the web in a rather incautious way – there's no time to check for security [E22]. Yet, even if developers re-use »internal« code via »copy&paste programming«, issues may arise [E1, E2]. Problems aside, the bottom line was that experts ranked the positive effects of sharing and exchange definitely higher than the drawbacks. If companies implement internal fora [E19], establish explicit sharing policies [E21], or invest in developers expertise [E6, E19], the wisdom of the crowd effect can be harnessed while mitigating the risks.

4.3.4. Crowdsourcing Software Development

In close proximity to the latter phenomenon is another trend mentioned by our experts, namely crowdsourcing software development [E12]. In crowdsourcing SD, companies publish an open call to contribute to the development of a given product, including the contribution of portions of code. Software development thus is transformed towards managing a social network or infrastructure, such as mechanical turk [E12]. If the trend is going to prevail, companies, rather than searching for excellent coding skills, will accordingly watch out for developers with project management skills who are able to coordinate the assembling of code in distributed settings [E3]. In terms of security, there are some risks associated with crowdsourcing, such as adversaries designing vulnerabilities into systems, introducing bugs, or manipulating the sourced crowd [E12]. Chances are that the trend is significant considering that there are major players in the software industry that have already accomplished crowdsourcing projects, such as Microsoft and Oracle. Also, computer science research has already begun to address the phenomenon. In a recent paper, Wu/Tsai/Li allude to security issues associated with crowdsourcing when writing that »software crowdsourcing is different from general crowdsourcing (…) most code developed by the crowd carries no liability in case of damage« (Wu/Tsai/Li 2013: 59). As crowdsourcing is a rather novel phenomenon there is no detailed evidence yet as regards the security issues it elicits. Yet, it is easily comprehensible that crowdsourcing has the potential to evoke legal and technical issues. This calls for increased research in this area.

4.3.5. Lessons to Be Learned

Our trend analysis points to a number of lessons to be learned as regards the trend towards more generating and assembling (instead of writing) code. Generally, there have to be found ways that guarantee security is incorporated into the code that is produced or shared. This can be achieved by the following measures:

- **»Democratization« of SD:**
  In respect to the »democratization« of SD, security can be integrated into SD by either strengthening the expertise of »laymen developers«, or by automatizing secure SD:
  - Introducing security guidelines for career changers, i.e. developers originally not being trained as software engineers in companies and self-teaching platforms.
  - Providing self-learned developers with robust, possibly intrinsic secure code pieces.
  - Providing components that are able to interact with an insecure environment.

- **Generating Code in Industrial SD:**
  As regards code generation, the aim is to improve tools, to make them more trustable, and to coordinate their usage.
  - Incorporating security into IDEs, libraries and frameworks. This pertains to the code generated and to the integration of security techniques such as cryptographic and analysis tools (static analysis, copy&paste-coding analysis) as well.
  - Facilitating round-trip engineering so that there is always alignment between the abstract modeling level and the actual coding level.
  - Introducing framework security certification in order to guarantee the trustability and reliability of frameworks.
  - Harmonizing the repertoire of the tools being used, so that there are no inconsistencies arising from the diversity of tools.

- **Sharing Reusable Code:**
  The secure sharing of code can be achieved by taking measures relating to education and coordination.
  - Training developers – if anything, assembling and generating code makes individual security expertise more relevant, instead of less.
  - Considering establishing internal sharing platforms and explicit sharing policies.

- **Crowdsourcing SD:**
  Crowdsourcing SD is a relatively novel phenomenon, thus it has to be researched.
  - Researching the implications of crowdsourcing SD for IT security empirically.

As far as our experts are concerned, there was one more aspect to the assembling & generating code trend: the integration of open source repositories and libraries. As this phenomenon also concerns the trend towards modularized software and compositional systems, we will treat it in the next section.

4.4. Compositional Systems and Modularization

Experts explained that in industrial SD there is a strong focus on the production of standardized software modules that may be recomposed to software systems according to the context of any particular use case and the specific requirements a given customer demands [E3, E11, E17]. Quite obviously, if it comes to the modular design of compositional software systems there is a profound interplay between the social and technical organization of SD. That is to say, the geographical distribution of SD to a certain degree necessitates the compositionality or modular nature of software systems, whereas the compositionality of software systems facilitates the geographical distribution of SD. From our experts' account it follows that modular production can be expected to play an ever greater role in the future [E5, E7, E11, E17, E20, E23]: there is an economic imperative to integrate externally produced code pieces and software modules including open source components [E1, E2, E20]. The composite nature of the resulting software systems has multiple implications for IT security. Although being interconnected, in what follows we will first consider problems from the perspective of the components; second, we will treat issues from a system's point of view. We will again conclude by providing a listing of lessons to be learned.

4.4.1. Modular Components

Generally speaking, compositional systems render the reliable interplay of the components making the system up more important [E3]. At the same time, however, one of the major implications frequently mentioned by the experts was the problem of guaranteeing the security of an overall system which was produced by multiple stakeholders. There are non-technical as well as technical problems with guaranteeing such reliability:

- **Trustworthiness and Reliability of Partners:**
  First of all, software vendors who sub-license parts of their system oftentimes work with partners whom they, as a matter of principle, can trust only to a certain extent [E9, E17]. We treated this problem already above when discussing the social security aspects of distributed SD, yet we would like to reiterate the statement that supply chain security is of increasing importance [E3, E12, E18, E21].

- **Measuring Components' Security**
  However, even if collaborating partners have reasonable evidence of each other's trustworthiness there are security issues stemming from technical security aspects associated with the composite nature of software systems: even if a software component is delivered by a long-term collaborating partner, the entity being responsible for the overall system cannot be downright sure whether there are vulnerabilities contained in the component being delivered: there can't be absolute certainty about the integrated component [E4, E6, E10, E20], regardless of whether one does or does not presume malicious intent. This results in the difficulty of guaranteeing one's own overall system, as undetected vulnerabilities in external components may generate vulnerabilities in the resulting system [E6, E17, E18]. The main reason for this is that up to date it is still extremely difficult – perhaps impossible – to measure or guarantee, let alone reliably certify, the security of integrated OS or other components [E1, E2, E4, E17, E18, E21].

Although there are good reasons to assume that the external component – proprietary or open source – has been put to the scrutiny of a lot of experts [E6], the impossibility to measure, guarantee and certify the security level of a given software makes it even more important

- to explicitly specify security requirements upfront [E3, E13];
- to never integrate external components as black boxes [E16], especially in safety-areas [E6];
- to have a clear understanding of the integrated component's code [E7];
- to test security relevant components and raise security relevant parameters [E6];
- to continuously integrate new features or code portions [E1, E2];
- to check and safeguard the compatibility of the code base produced in-house with the code being integrated [E18, E22]; and
- to reduce the attack surface by deactivating the external component's functionalities that are not being used [E1, E2, E6], which requires developers to have an intimate knowledge of in-house and of externally produced code.

In addition, as it is not possible to have absolute certainty as regards the component's behavior, the latter may be encapsulated so to cause harm only locally if things go wrong [E4, E20].

4.4.2. Compositional Systems

Now, if we shift the perspective from the components to the overall system, the problem does not concern guaranteeing components' security, but the resulting system as such. According to the experts, the fundamental problem is that while two components may be secure when running separately, their combined interplay may be not [E1, E2, E10, E11, E17, E18]. In other words, two secure software components do not necessarily make for a secure compositional

system. The problem poses significant challenges, not least because of the context dependency of any security assessment. According to our interviewees, the security level of a given software system is in principle only determinable if one considers the system in context [E1, E2, E10, E12, E17, E20]. At the same time, however, compositionality invokes the production of components that are produced at the outset to be integrated into different products (contexts). This goes for large software companies that aim to produce standardized off-the-shelf components which are then reassembled for any specific use case, thus composing a system tailored to the particular needs of any one customer [E11]; but it also goes for a mode of production that is expected to gain momentum in the future: the production of standardized core components which are produced by outsourced entities or freelancers who at the outset will fabricate reusable components as potential resellers [E3]. Hence, modular components, whether company-internal or contract-based products, must quite obviously be adaptable to multiple contexts [E3] with the characteristics of the latter not being specifiable upfront [E17, E12]. Experts mentioned three strategies to cope with this:

- **New Testing Procedures:**
  Experts again brought up the dire need for the development of formal and automatable testing procedures, and for the specification of valid indicators concerning the static and dynamic qua-lity and security attributes of compositional systems [E1, E2, E9, E17, E18, E21]. Also, it will be necessary to find ways to test[20] [E21] and formally specify the security relevant characteristics of the components: in what kind of environment has a modular component been tested and proven to be robust and securely applicable [E11]? The interplay of multiple components, i.e. the collective security performance of the components may then be automatically assignable by setting their formal specification in relation [E11]. This may help developers to keep track of the overall system, which is quite difficult in compositional systems [E10], not least because of the complexity of these systems aggravating the detection of vulnerabilities [E11].

- **Setting Collective Interface Standards:**
  Another fruitful strategy, according to experts, will be to detach interfaces from proprietary products so as to effect improvement (more robustness) and establish interface standards collectively [E11].

- **Intrinsically Secure Components:**
  Experts expressed their hope that in the future ways will be found to produce intrinsically secure and reusable building blocks, or modular components, that can either be put together at will without compromising security [E19]; or that are able to interact within insecure environments without compromising security [E5]. Whereas one study participant was rather skeptical about the possibility of generating inherent system security [E17], the

20 *Related to the context issue there occurs a testing issue: the real test is always the system running in the wild, i.e. its real environment, thus pretesting has its limits [JE]. This follows logically from the »unspecifiable-context« problem. For, saying that the context is not completely specifiable amounts to say that neither is the tested system's environment – which is where the limitations of testing come in.*

respondent nevertheless expressed the importance of arranging for secure components thus agreeing with many experts on the need to invest in R&D in this area [E5, E10, E17, E19]. Along with the desideratum of technically securing modular components comes the one of certifying [E17] and legally guaranteeing component security [E10].

4.4.3. Lessons to Be Learned

Generally speaking, compositionality and modularization entail a profound paradigm change as regards security. As one interviewee pointed out, monolithic security solutions that cover whole areas are a thing of the past [E5]. In this sense, the challenge will be to introduce security to compositional systems, although security itself is not a modular or compositional property [E4, E17]. Here is a summary of what we learn from the experts' statements:

- **Component's Perspective:**
  Regarding components, the challenge is to determine the collaborating partners' trustworthiness and reliability as far as this is possible and to protect oneself as far as it is not; and to somehow measure a component's security, so far in absence of precise metrics.
  - No integration of external components as black box [E16], especially in safety-areas [E6].
  - Rigorous testing of security relevant components and raising of security relevant parameters [E6].
  - Continuous integration of new features or code portions [E1, E2].
  - Clear understanding of the integrated component's code [E7], checking and safeguarding the compatibility of the code base produced in-house with the code being integrated [E18, E22].
  - Reducing attack surface by deactivating the external component's functionalities that are not being used [E1, E2, E6].
  - Development of encapsulation techniques for integrated components [E4, E20].

- **System's Perspective:**
  From the point of the system's view, what is chiefly required is research and development testing, specification, standardization and certification procedures.
  - Development of formal and automatable testing procedures and specification of valid indicators concerning the static and dynamic quality and security attributes of compositional systems [E1, E2, E9, E17, E18, E21].
  - Formal specification development of the components' characteristics.
  - Collective establishment of interface standards.
  - R&D of intrinsically secure building blocks / components.
  - Development of modular instead of monolithic security strategies.
  - Certification and legal guarantee of components' security.

4.5. Distributed Systems and Intensified (Cross-Domain) Networking

According to our experts, the intensified networking of almost everything, including cross-domain systems, distributed functional features and distributed systems has already begun to play a greater role a while ago and can be considered a significant paradigm shift [E19]. Many interviewees assume that the intensification of networking is going to persist [E19], not least because of the economic incentive of network externalities [E12], which – roughly speaking – render systems the more valuable the more they are networked. Networking and distribution amounts to open up systems [E19], and security implications are huge [E12]. To state an obvious example, problems of authorization and authentication vis-à-vis other network components possibly shift if there is no central management anymore [E13]. There are numerous further aspects to the security issues that distributed systems raise. In what follows we will treat them successively before listing lessons to be learned.

4.5.1. Security Specification in Distributed Systems

Some of the challenges that distributed systems and intensified networking pose to IT security are quite similar to those that were identified in reference to compositional systems. First and foremost, also in distributed systems the problem of specifying, guaranteeing and certifying quality and security is thus far not solved; neither is it possible to provide universal specification of a distributed system's possible course of events [E17]. Moreover, distributed systems are highly dynamic. Unfortunately, while it is possible to guarantee security (and safety) at the point of design, it is not possible when systems run that are subject to dynamic change [E4, E17]. For this reason, it is required to develop indicators, techniques and (affordable) tools that allow for runtime-dynamic system testing in real-time [E10, E16, E17].

4.5.2. Cross-Domain Systems

Further problems arise when the fact that distributed systems involve the networking of various elements is taken into account. In this regard we come across the problematic again where two secure components do not necessarily make for a secure system. Likewise, the problem of how to secure the whole system reoccurs. There, raising the components' intrinsic security is used as a strategy to at least mitigate the problem [E17]. The problematic is aggravated by the existence of cross-domain networking, involving systems from different areas, as is the case when smart phones are connected to cars for example [E12, E17, E19]. As a result, sub-systems that use different types of protocols and programming languages together form networked

systems [E12]. Further, those sub-systems have been developed with different foci in mind, relying on diverse modes of modeling and tools [E19]. The inherent heterogeneity of such systems threatens to cause security problems and calls for ways and tools that allow for the secure integration of this heterogeneity [E12, E19]. Thus, requirements regarding the engineering of complex distributed systems will gain relevance, just as the provision of tools will that allow for understanding and controlling such system behavior as far as this is possible [E19].

4.5.3. Safety & Security in Distributed Systems

Yet, cross-domain networking still has further implications. As a fundamental issue, the amalgamating of IT has been mentioned by many experts on the one hand; on the other, embedded devices, physical systems, or critical infrastructures have received attention. This will make security and safety converge even more [E4, E6, E10, E19]. To take up the aforementioned example: when connecting a smart phone to a car in order to use a satnav app while driving, the integrity of the traffic data provided by the app may affect safety profoundly – if things go wrong, the driver might have an accident. Therefore, according to experts, whereas both of these worlds have, in fact, already merged in practice, so far engineering only knows how to cope with security and safety separately [E4]. Additionally, while awareness levels in safety areas are naturally quite high, awareness in other areas of SD still needs to be raised [E7]. However, if security does affect safety, awareness in (so far) non-safety areas will become just as important. For example, physical systems and their control systems were originally not meant to be connected to digital networks, such as the internet. Yet, when these systems became cyberphysical ones, those critical infrastructures became connected to the internet rather incautiously, which is why they have been proven vulnerable to attacks[21] in the recent past – and such attacks are increasing in the present.[22] Consequently, in the future their being connected requires awareness for integrating security right from the outset [E6],[23] and more R&D in regard to embedded software and cyberphysical systems [E7].

4.5.4. Cloud Computing

Distributed and networked systems may combine critical with uncritical and secure with insecure sub-systems. The challenge is enabling them to coexist in the same network without compromising security. In this regard experts stressed the need to develop techniques to encapsulate or separate critical/secure from uncritical/insecure sub-systems, e.g. by creating »virtual cages« that contain insecure components [E4, E5]. Separation was likewise considered a fruitful strategy to deal with data in cloud computing (understood as one particular form of distributed computing): non-sensitive data may be separated from sensitive data, the former being stored in low-security clouds while the latter is to be stored locally or in high security cloud environments [E10]. Obviously, such strategies presuppose that developers give thought to the nature

21 *www.heise.de/security/meldung/Kritische-Schwachstelle-in-hunderten-Industrieanlagen-1854385.html*

22 *www.heise.de/security/meldung/Attacken-auf-SCADA-Systeme-nehmen-zu-1910037.html*

23 *This is actually true regardless of whether those systems are connected or not: also, closed systems fell prey to attacks in the past as the case of stuxnet attacks on SCADA systems amply demonstrates [E18]. See also http://www.heise.de/security/meldung/Innenangreifer-half-bei-Stuxnet-Infektion-1520408.html*

of different types of data. Experts also held that there are new security requirements for software running in the cloud [E1, E2]. Central provider management of software was considered an advantage and disadvantage at the same time: on the one hand, central control amounts to providers having detailed knowledge of the use case and allows for immediate and very fast security response processes without retroactive patching by sending updates via the internet [E1, E2, E10, E12, E22]; on the other hand, the numerous entities who use cloud services can now be attacked simultaneously (there is a central point of attack); and they lose control of their data to a considerable degree [E1, E2, E10]. Therefore, and also because of cloud computing entailing a different kind of business model, there are plenty of legal issues to be solved: the interviewees stressed particularly a certain under-regulation of liability in cloud computing concerning data security and protection in terms of storage, access, control, notification duties etc. [E7, E9, E18, E21]. The questions to be posed are well-known.[24] While experts particularly emphasized the need to come to international agreements on these matters [E4, E9], others expressed their hope in the coming ISO/IEC 27017 norm which is bound to regulate »controls to protect personally identifiable information processed in public cloud computing services.«[25]

4.5.5. Lessons to Be Learned

Thus, in sum, our listing of lessons to be learned includes the following issues:

- **Security Specification in Distributed Systems:**
  Regarding security specification, there is a demand for the development of techniques that would allow the security level of distributed systems to be determined.
  - Finding ways to specify, guarantee, and certify quality and security in distributed systems.
  - Enabling universal specification of DS's possible course of events.
  - Developing indicators, techniques and (affordable) tools that allow for runtime-dynamic testing in real-time.
  - Raising DS components' intrinsic security levels.

- **Cross-Domain Systems:**
  Cross-domain systems call for an amalgamation of perspectives and techniques from different domains.
  - Developing ways and tools to integrate cross-domain (heterogeneous) DS components.
  - Improve strategies for requirements engineering and understanding complex cross-domain DS.

- **Safety & Security in Distributed Systems:**
  In respect to safety & security in DS, the challenge is fusing the analytical tools of the safety and the security world.

24 *E.g., what about the storage location? Who may have access, including government authorities? To what extent shall encryption be allowed? How to balance anonymity and law enforcement? How to issue, inspect, and verify security guarantees? Who's liable in case of data breaches, and how to call responsible entities to account? [E9]*

25 *www.iso27001security.com/html/27018.html*

- Development of ways to fuse security and safety concepts and strategies.
- Raising awareness for security in safety areas; and for safety in security areas.
- Integrating security in embedded software and cyberphysical systems early on.
- Fostering R&D in embedded software and cyberphysical systems.

- **Cloud Computing:**
  There are various issues related to cloud computing, our experts focused on two aspects.
  - Separating sensitive form non-sensitive data in Cloud Computing and treat it accordingly.
  - Solving legal and especially liability issues in Cloud Computing, at best via international agreements.

4.6. Legacy: The Complexity of Evolved Software Ecosystems

Today software development is – to a considerable degree – about developing software systems further, instead of developing software from scratch [E1, E2, E5, E6, E10, E21]. Many large software systems live much longer than initially expected in the days of early SD [E4]. Some software systems have a long-running history and are often-times the result of numerous, varying actors establishing and modifying the code base for years – and sometimes decades. Such code bases are the legacy contemporary developers have to deal with. Legacy threatens to provoke practically unmanageable complexity [E16, E20], and the problems it brings about are multifaceted. In the remainder of this chapter, we will present these issues as identified by study participants and draw the conclusions associated with these aspects.

4.6.1. Unknown Past Decisions and Fading Expertise

In the absence of appropriate documentation illustrating assumptions and design decisions that went into system production and due to their rising complexity, at some point these systems become largely incomprehensible [E6, E20, E22] and difficult to maintain [E3, E10]. Aggravating the problematic is the shortage of developers with profound expertise as regards elder systems and the programming languages these systems are based upon [E3, E6, E13, E14, E16, E22, E23], such as COBOL. Developing those systems further is thus tantamount to flying blind, since developers, not knowing underlying assumptions of the system's design, in principle cannot know the implications of their own decisions and code modifications [E12]. It is generally not possible to anticipate the behavior of complex systems with certainty [E20], and limited knowledge of the adopted system also threatens to concurrently adopt too many unused functionalities, which in turn enlarges attack surfaces [E6]. Furthermore, issues arise when new components that were produced by relying on novel coding techniques are to be integrated into legacy systems produced using obsolete techniques [E14]. Yet, even if the code base and the design decisions are reasonably well known, legacy induces a certain path dependency,

limiting the modifiability of systems [E10, E22]. Last but not least, legacy also involves problems with testing. It is not only generally difficult to conduct risk assessment of large complex systems [E1, E2], but also new features accessing old code are quite difficult to test for security [E18].

4.6.2. Dealing with Legacy Today

These issues quite obviously raise two questions: how are legacy systems to be handled today? And how can producing legacy problems be avoided in the future? Here we will deal with the first question; the following section will be devoted to the second one. Thus, as regards currently dealing with legacy inherited from the past, the final analysis of the experts' statements reveals three strategic options:

- **Isolation/Encapsulation:**
  Pragmatically isolating or systematically encapsulating untrusted legacy functional building blocks is one of the options to deal with past legacy [E4, E19, E22]. Due to its pragmatic nature, it may also be the most attractive one to industrial SD. One expert gave the example of partly isolating some legacy component by limiting communication between the latter and the overall system so as to mitigating risks [E22]. Due to the largeness of the systems in question, however, the problem may already be telling what components should fall under scrutiny. In this regard, stochastic techniques to tell developers what components are at risk need to be developed or applied [E20]. The reader will note that the encapsulation strategy was already highlighted as an option to deal with problems related to compositional systems and to DS (s. above). According to our experts, this requires a strong research focus on encapsulation solutions so as to be able to build virtual software »cages« easily [E4, E19].

- **Analysis and Secure Ex-Post Reengineering of Building Blocks:**
  The second option, the re-engineering of building blocks [E5, E19, E18] is costly. Integrating security into building blocks ex-post is difficult and expensive for two reasons: the shortage of experts for old systems and the lack of using up-to-date engineering tools. Instead of relying on frameworks or the like, the code base has to be modified manually, which makes reengineering time-consuming [E14].

- **Downright Rebuilding of Building Blocks:**
  The third option, downright rebuilding [E13, E19], requires one to disassemble functional building blocks in order to rebuild them. This is, of course, extremely laborious, time-consuming, and therefore comes at a very high price [E13, E19].

4.6.3. Avoiding Legacy in the Future

Considering the multitude of problems legacy systems bring about and the costs associated with further developing such systems [E23], avoiding legacy problems in the future is extraordinarily attractive. However, against the background of the extreme acceleration of SD processes – in terms of technology as well as of business, today's innovation quickly becomes tomorrow's legacy. Hence, respondents believed that SD processes need to be understood quite generally in an evolutionary rather than a »from scratch«-mode. Given the »unstableness« of the software industry (innovation at a high pace, fast turnover of staff, flexible working environments etc.), software tends to be modified by other developers than those who originally developed it in the first place [E5]. If software systems were continuously treated by a fixed set of people, legacy problems would not occur [E22]. However, as this is not the case, legacy also renders it increasingly important to find ways to introduce continuity into SD in spite of rapid change. In this respect, the experts mentioned two relevant dimensions:

- **Knowledge Capture:**
  To avoid legacy, it is important to capture expertise and knowledge as well as preserve assumptions and design decisions regardless of individual developers. Thus, documentation and transparency of SD processes become more important and need to be optimized in reference to legacy [E3, E5, E10, E16]. This includes harmonizing in-house code production by issuing programming guidelines, resulting in code that is not too condensed and thus still comprehensible for other developers at a later date [E16].

- **Methodologies:**
  Also SD methodologies should at best reflect the transition from »from scratch« to »evolution«. Whereas the bulk of SD is in practice about developing systems further instead of from scratch, methodologies still act on the assumption that SD is generally about new product development; to prepare developers for their task appropriately methodologies and education as well should be based on the real situation that developers are confronted with: facing a lot of legacy [E1, E2, E5, E6]. To give an example, methodologies could include a stronger focus on long-term maintainability [E5]; appropriate education could make developers ready for the »practice shock« of being confronted with all the legacy involved [E6].

4.6.4. Establishing Long-Term Perspectives

Aware that dealing with legacy is an integral part of SD may also somewhat alter the developers' or companies' mentality towards the establishing long-term perspectives. For example, our experts explained that one may develop and optimize software architecture based on long-term

and anticipative considerations in order to avoid legacy problems. In this sense, product line engineering may be considered a step towards such a long-term perspective, providing software systems with variability and adaptability [E4]. When producing software systems at the outset with multiple contexts and changing circumstances in mind, a long-term perspective should also guarantee the integration of security at the outset, for security specification early on is another pre-condition for further developing systems instead of SD from scratch [E5, E18]. For example, provided companies have a long-term perspective that prioritizes security from the beginning, the imperative of backwards compatibility that presently often results in vulnerabilities [E10] being carried over would not necessarily come at the cost of security.

4.6.5. Analyzing Complex Systems

Regardless of whether one deals with legacy systems produced in the past or strives to avoid producing legacy in the future, what is required to deal with the complexity that grown software structures possess are procedures and tools able to analyze security and safety in very large and complex systems [E1, E2, E4]. With legacy there is much more source code to be reviewed nowadays. This makes quality assurance much more difficult than in the past [E20]. Experts particularly mentioned the need to develop scalable techniques that allow complex systems to be broken down and analyzed in aggregates [E4]. Moreover, it is necessary to do the following: find ways of conducting risk analysis that begin at the top of complex systems and go all the way down [E1, E2]; stochastically identify components at risk [E20]; and make static analysis tools scalable for large systems [E4].

4.6.6. Lessons to Be Learned

Having presented our experts' account of legacy problems so far, we would like to present the lessons to be learned from the interviewees' expertise:

- **Dealing with Past Legacy:**
  Dealing with past legacy requires one to integrate security retroactively into systems.
  - Finding strategies to pragmatically isolate untrusted building blocks.
  - Encapsulation of untrusted building blocks by building virtual software cages.
  - Development of techniques to build security into legacy systems ex-post.

- **Avoiding Future Legacy:**
  To avoid future legacy, there is a need to build security proactively into systems.
  - Development of smart documentation techniques for expertise and knowledge capture, and for preserving assumptions and design decisions.
  - Harmonization of in-house code production to keep code comprehensible.

- Development of novel methodologies that account for SD's »evolutionary« character.
- Adapting education to the legacy situation.

- **Establishing Long-Term Perspectives:**
  Many software systems are indeed long-living entities; establishing long-term perspectives helps to mitigate path dependency that may limit undesirable non-modifiability due to wrong decisions.
  - Optimizing software architecture.
  - Introducing product line engineering.
  - Guaranteeing to integrate security early on.

- **Analyzing Complex Systems:**
  Before being able to manage them, one must be able to understand complex systems. Therefore, techniques that help to do so are required.
  - Development of tools and procedures to analyze security and safety in very large and complex systems.
  - Development of scalable techniques that allow for breaking down complex systems to analyze them in aggregates.
  - Development of risk assessment techniques that go from the top all the way down.
  - Development of techniques to stochastically identify components at risk.
  - Development of scalable static analysis tools.

5. SYNOPSIS AND CONCLUSION: TRENDS IN SOFTWARE DEVELOPMENT & FUTURE CHALLENGES FOR IT SECURITY

As our interview study reveals, there is a number of themes running across different trends. In our conclusion we will identify and cluster those themes by summarizing the major issues listed in the »lessons to be learned« sections above. The ultimate goal of this report is not to present solutions but to direct the attention of those dealing with software development and IT security to the (research) challenges brought about by the trends. The challenges that arise from experts' statements pertain to education, processes, methodologies, metrics, techniques, and tools. In what follows we will conclude by summarizing them.

- **Education: Raising Individual Expertise**
  It has become clear in the course of the analysis that as IT security gains relevance, individual developers' level of security expertise becomes much more important. The working environment of software development is characterized by rapid change, and agile values are gaining ever more foothold. Thus, there is frequent change in requirements, making it necessary for developers to have the skills to determine security implications of any change in requirements; also, in agile SD team roles are less rigid, so developers are likely to sooner or later face situations where they need to make security assessments. The increase in assembly and sharing of code demands a great deal of developers in terms of IT security expertise, too. Therefore, not only is it desirable that security expertise become more strongly integrated in developers' formal education, but it also seems fruitful to somewhat harmonize curricula, so as to guarantee that at least all computer science graduates have a comparable level of security expertise by training. That said, it is just as clear that IT security experts are required to have agility expertise. Last but not least, adapting education to the reality of software development would mean acquainting developers with agile SD methodologies as well as preparing them for being confronted with considerable legacy code.

- **Processes: Systematic and Scalable**
  The latter aspect requires companies to establish long-term perspectives even more in regards to their software systems and SD processes. The flexibility of SD's working environment and the heavy networking of all kinds of systems make having systematic security processes in place mandatory, guaranteeing security is integrated early on and requirements are specified upfront. Likewise, usability has to be accounted for early on. A great deal of proper SD processes, or so it seems, depends on whether or not effective communication and coordination exists among developers, and between developers and security experts. Consequently, companies need to have a sufficient number of security experts on board, and they need to have mechanisms that allows for making security expertise scalable. If software production as well as software products become modular, security must also do so by design. It is hardly possible to deliver a concluding security specification upfront by experts which

remains unaltered throughout the whole software development life-cycle so that developers may stick to it until some project ends. Instead, security expertise has to be conveyed where and when it is needed. This seems even more important given that many developers nowadays are career changers without formal engineering education. Thus, procedures and feedback mechanisms for vulnerability detection, bug tracking and fixing are necessary. To avoid redundancy and associated costs, companies need to develop mechanisms for organizational learning. As developers may share code and expertise on the web, companies might consider introducing sharing policies. Returning to the long-term perspective theme, we may conclude by stating that companies may also consider to optimize software architecture and harmonize in-house code production, so as to avoid running into unmanageable complexity.

- **Methodologies: Reconciling Flexibility with Continuity**
  We dealt quite a bit with methodology in this report. The main feature an up-to-date methodology must provide for is to make sure there is continuity in spite of constant rapid change of constellations. The key is to capture knowledge and to preserve expertise and design decisions that go into software systems' production. Code must be kept comprehensible not only over a long period for varying cohorts of developers, but also in respect to the geographical distribution of collaborating actors. At the same time, there must be reconciliation of formality and leeway. Novel approaches must be able to reconcile agility with long-term perspectives on software systems if the latter are bound to provide sustainable products. As spelled out above, methodologies – at best – account for the evolutionary or emergent character of software systems, thus adapting to the legacy situation of having no precise »point zero.«

- **Techniques: Sorting the Sheep from the Goats**
  One of the most frequently mentioned techniques that will gain relevance in the future is the one of encapsulating code or of shielding systems from non-trustable software components that interact with some overall software system. The frequent mentioning of the need to encapsulate networked components drives us to conclude that after years and years of unhesitating networking, the security issues that have emerged by now begin to induce a trend to sort out the sheep from the goats where required, of course without turning back to isolated systems. What research in industry and academia thus needs to focus upon are strategies to isolate networked components from each other partly, or to encapsulate code portions by building virtual cages around them. The need for this arises from untrusted legacy as well as from integrating untrusted 3rd party code and from networking with untrusted external systems. While legacy also brings up the need to make untrusted systems analyzable and retroactively render them secure, the trend towards assembling code and compositional systems evokes the need to develop secure software components and code portions intrinsically.

- **Metrics: Enabling Measurement of IT Security**
  Experts repeatedly brought up the topic of measuring security: there is a lack of metrics allowing one to measure software components', open source software's, compositional systems', and distributed systems' security levels consistently. Likewise, there are no proper metrics to measure security processes, and given that security and safety tend to fuse to a certain degree, it is problematic that there be no metrics to measure safety. The absence of security metrics makes it difficult to introduce widely accepted certification schemes that may be referred to when it comes to liability issues. The lack of legal guarantees regarding open source or cloud computing may abate the adoption and further evolution of these areas of software development. For this and other reasons the development of security metrics is extraordinarily desirable.

- **Tools: Automatizing IT Security**
  To a considerable degree, software development has become automatized. Hence, it is desirable to automatize security to the same degree. The integration of security into IDEs, frameworks, and libraries by recommending or generating secure code is what many experts put on their list of wishes. The same goes for integrating cryptographic techniques and static analysis tools with repair function. If it comes to analysis tools, there was repeated mentioning of scalable tools that are able to also analyze large complex systems by breaking them down into aggregates; and of formal and automatable testing procedures that allow for runtime-dynamic testing in real time, especially to test dynamic, distributed systems. Moreover, the feature of round-trip engineering may strengthen IT security by allowing one to keep modeling and coding consistent and transparent easily. Last but not least, it may be fruitful to certify frameworks for security in the future, so as to guarantee that a given frameworks generates secure code indeed.

Such are the future challenges of software development and IT security. The themes identified in this report flow in multiple directions. Our study is explorative in character, and our goal was to identify these very directions. While research addressing some of the issues identified here is already underway, other issues still wait to be approached. We hope to have inspired readers to take up some of these issues.

6. REFERENCES

Barnes, S. B. (2006): A privacy paradox: Social networking in the United States. In: First Monday 11, No. 9, URL: http://firstmonday.org/issues/issue11_9/barnes/index.html (8.11.13)

Errata Security (2010): Survey Results – Integrating Security into the Software Development LifeCycle. URL: http://www.docstoc.com/docs/37556537/Integrating-Security-into-the-Software-Development (8.11.13).

Kühl, S./Strodtholz, P./Taffertshofer, A. (2009): Qualitative und quantitative Methoden der Organisationsforschung – ein Überblick. In: Kühl, S./Strodtholz, P./Taffertshofer, A. (eds.): Handbuch Methoden der Organisationsforschung. Quantitative und Qualitative Methoden, Wiesbaden.

Liebold, R./Trinczek, R. (2009): Experteninterview. In: Kühl, S./Strodtholz, P./ Taffertshofer, A. (eds.): Handbuch Methoden der Organisationsforschung. Quantitative und Qualitative Methoden, Wiesbaden.

Newman, J. (April 2, 2013): Phablets Are a Niche, Not a Fad. In: Time, URL: http://techland.time.com/2013/04/02/phablets-are-a-niche-not-a-fad/ (8.11.13).

Ponemon Institute (2013): The State of Application Security. A Research Study by Ponemon Institute LLC and Security Innovation. URL: https://www.securityinnovation.com/security-lab/our-research/current-state-of-application-security.html (8.11.13).

Schulz-Schaeffer, I. (2012): Scenarios as Patterns of Orientation in Technology Development and Technology Assessment – Outline of a Research Program. Manuscript, URL: http://www.uni-due.de/imperia/md/content/skott/schulz-schaeffer2012scenariosaspatternsoforientation.pdf (18.2.13).

Spath, D. (Ed.) (2012): Arbeitswelten 4.0. Wie wir morgen leben und arbeiten, Stuttgart.

Steinmüller, K.-H./Schulz-Montag, B. (2004): Szenarien – Instrumente Innovationen und Strategiebildung. In: Wirtschaftspsychologie Aktuell 1/2004, pp. 63-66.

SwissQ (2013): Agile 2013. Trends and Benchmarks Report Switzerland. URL: http://www.swissq.it/wp-content/uploads/2013/08/Agile-Trends-and-Benchmarks-2013_Web_En.pdf (8.11.13).

Wu, W., Tsai, W-T. and Li, W. (2013): Creative software crowdsourcing: from components and algorithm development to project concept formations. In: International Journal for Creative Computing, Vol. 1, No. 1, pp.57–91.