



universität
wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Privacy Protection Framework for Agile Software
Development Methods in Distributed Teams“

verfasst von / submitted by

Arvid Till Enders Cartledge

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2020 / Vienna, 2020

Studienkennzahl lt. Studienblatt /
degree programme code as it appears on
the student record sheet:

A 066 926

Studienrichtung lt. Studienblatt /
degree programme as it appears on
the student record sheet:

Masterstudium Wirtschaftsinformatik

Betreut von / Supervisor:

Univ.-Prof. Dr. Dr. Gerald Quirchmayr

Abstract

The GDPR requires data processors and controllers handling personal data of EU citizens to perform adequate privacy protection. The framework designed in this thesis incorporates privacy protection in a lightweight manner to ensure agile software development projects comply with the GDPR when handling personal data by utilising the concept of Privacy by Design. At its core lie Privacy Impact Assessments as well as qualitative risk assessments to analyse data processes and allocate the respective risk level to each privacy user story. The identified privacy risks remain visible to the development team through the implementation of a Privacy Risk Board. The introduced components are designed for continuous integration into the agile development life cycle and, as an example, are deployed on the Scrum methodology. The framework is supported by a prototype with a focus on digitising the required artefacts for the work environment of distributed teams. This is enabled by implementing a data process assessment tool which communicates through RESTful APIs with a web service. The web service is used to visualise work-in-progress and create a distinct focus on identified privacy risks to the development team.

Abstract

Die DSGVO erfordert, seit ihrer Einführung im Mai 2018, von sämtlichen Datenverarbeitern die Implementierung von Datenschutzmaßnahmen wenn persönliche Daten von EU-Bürgern gehandhabt werden. Das in dieser Arbeit kreierte Model ermöglicht die effiziente Einbindung von Datenschutzmaßnahmen in die agile Softwareentwicklung um den Auflagen der DSGVO ohne zeitintensiven Mehraufwand nachzukommen. Diese Einbindung basiert auf dem Konzept *Privacy by Design*, Datenschutz durch Technikgestaltung. Im Kern wird dies durch systematische Analyse der Datenprozesse und Risikobewertung der einzelnen Datenprozesse durchgeführt. Die identifizierten Risiken werden auf einer Risikomatrix visualisiert. Die in dieser Arbeit eingeführten Komponenten für kontinuierliche Integration der Datenschutzmaßnahmen werden beispielhaft auf die Scrum-Methodologie angewandt. Das theoretische Model wird durch einen Prototyp unterstützt, welcher sich auf die Digitalisierung der Komponenten für den Einsatz in einem verteilten Projektteam fokussiert. Dies wurde durch die Implementierung von einem Programm für die Risikoanalyse von Datenprozessen, welches über RESTful APIs mit einem Webservice kommuniziert, gelöst. Der Webservice visualisiert die aktuellen Arbeitsprozesse mit einer graphischen Abhebung der identifizierten Risiken.

Contents

1	Introduction	1
1.1	Motivation and Background	3
1.2	Goals of the thesis	3
2	Problem Definition and Description	5
3	Literature Review	6
3.1	Privacy Challenges	6
3.1.1	Risk Assessment	8
3.2	Agile Methods	9
3.3	Secure Agile Software Development Methods	11
3.3.1	Open Web Application Security Project	11
3.3.2	Microsoft Security Development Life Cycle for Agile . . .	12
3.3.3	Stockholm Integrated Model	13
3.3.4	The Secure Agile Software Development Life Cycle	14
3.4	Privacy Impact Assessment	15
3.4.1	Privacy Impact Assessment by CNIL	15
3.4.2	Privacy Impact Assessment by the OAIC	17
4	Approach and Methodology	19
5	Development of a Framework for Privacy Protection	22
5.1	Privacy Protection Requirements	24
5.2	Roles	26
5.3	Sprint	29
5.3.1	Sprint-Zero	30
5.3.2	Artefacts	32
5.4	Ceremonies	32
5.5	Control Points	34
5.5.1	Scrum Control Points	35
5.5.2	Privacy Control Points	36
5.6	Privacy User Stories	39
5.6.1	Functional Features	44
5.6.2	Engineering Tasks	45
5.6.3	Ways of Working	46
5.7	Agile Privacy Impact Assessment	48
5.7.1	Initial Privacy Impact Assessment	49
5.7.2	Privacy Risk Assessment	53
5.8	Privacy Scrum Board	59
6	Proof of Concept	61
6.1	Web Service	62
6.1.1	Privacy Scrum Board	62
6.1.2	Repository	68
6.1.3	RESTful Application Programming Interfaces	69

6.1.4 Database	70
6.2 Data Process Assessment Tool	71
7 Discussion of Results	80
7.1 Limitations	81
7.2 Further Research	82
8 Conclusion	83
References	84
Appendices	88
A Web Service	88
B Data Process Analysis Tool	114
C Privacy Protection Sprint	126

List of Tables

5.1	Privacy Protection Checklist	38
5.2	User Story Template	39
5.3	Exemplary Epic	40
5.4	Privacy User Story Template	42
5.5	Semantic Groups of Privacy User Stories	43
5.6	Examples of Functional Feature User Stories	44
5.7	Exemplary Functional Feature "Withdraw Consent"	45
5.8	Examples of Engineering Task User Stories	46
5.9	Exemplary Engineering Task "Protect Audit Logs"	46
5.10	Example User Stories of Ways of Working	47
5.11	Exemplary Way of Working "Training Data Anonymisation"	47
5.12	Typology of personal data (CNIL, 2018a)	56
5.13	Template Privacy User Story Analysis	57

List of Figures

5.1	Privacy Protection Framework	22
5.2	Privacy Protection Sprint	29
5.3	Abstraction Level of Control Points	39
5.4	Iterative Privacy Impact Assessment	50
5.5	Severity-Likelihood-Matrix	55
5.6	Risk placement on Severity-Likelihood-Matrix	58
5.7	Risks mapped on Severity-Likelihood-Matrix	59
5.8	Standard Scrum Board	59
5.9	Privacy Scrum Board	60
6.1	Prototype components	61
6.2	Privacy Scrum Board	63
6.3	Add User Story	65
6.4	Repository	68
6.5	Database Model	70
6.6	GUI Login Screen	71
6.7	Graphical User Interface	73
6.8	Risk Matrix	74
6.9	Evaluated risks on risk matrix	77
6.10	Sensitive data process warning	77
6.11	Personal data process warning	77
6.12	Two risks evaluated	78
6.13	GUI Create Story	78

Listings

1	Login route	62
2	Index route	64
3	Visualise user stories	64
4	User story options	65
5	Add Task Form	66
6	Route for adding a Task	66
7	Code-snippet from addTask.HTML	67
8	Route for deleting a task	67
9	Create User Story via API	69
10	Handling user login	71
11	API call for login	72
12	Calculating risk probability	74
13	Calculating risk severity	75
14	Evaluate risk	76
15	Create user story	79
16	Flask initialisation code (webservice.py)	88
17	Python initialisation code (__init__.py)	88
18	Authentication code (auth.py)	88
19	Flask configuration file (config.py)	89
20	Routing code (routes.py)	89
21	Code for User forms (forms.py)	96
22	Code for database (models.py)	98
23	Code handling token creation (tokens.py)	101
24	Code for error handling (errors.py)	101
25	Base HTML-page (base.html)	102
26	Main page (index.html)	103
27	Login page (login.html)	107
28	Register page (register.html)	107
29	Add Task page (addTask.html)	108
30	Edit Task page (editTask.html)	109
31	Repository page (repo.html)	111
32	Edit repository page (editRepo.html)	112
33	Code of Data Process Analysis Tool (gui.py)	114
34	Code for API Requests of DPA Tool (request.py)	124

1 Introduction

The introduction of the General Data Protection Regulation (GDPR), which came in effect in May 2018, makes the implementation of adequate privacy protection a legal requirement for all controllers and processors of data by providing the individual with explicit rights over their data (Commission, 2018). Further, the concept of *Privacy-by-Design*, as stated in Article 25 of the legislation, determines that every organisation handling personal data has to include privacy protection at the very beginning of its endeavour. The GDPR also establishes regulators to oversee data controllers as well as enforces its implementation by imposing fines on violation of these regulations. One such example is the fine of €50 million imposed upon GOOGLE LLC (CNIL, 2019) by the French data protection authority (CNIL). The GDPR aims to give individuals the right to know exactly where and how the information gathered by organisations and governments is used. GOOGLE LLC was judged to have breached the Act in this respect.

The recent advances in technology have led to manifold gathering of information obtained from many sources. These data sources range from personal devices to public surveillance cameras. Mobile devices together with remotely controlled stationary devices can provide very significant information trails which, if used maliciously, may lead to critical, if not dangerous situations. With current technologies and data analysis techniques the amassed data is being increasingly used to evaluate and to characterise individuals (Krasnow Waterman & Bruening, 2014). Any unwanted access to this amassed data needs to be hindered through respective data protection measures (Liu & Kuhn, 2010). Since the introduction of the GDPR, data controllers and processors have to incorporate adequate privacy protection into their development life cycle to ensure data sets are handled in a way that honour each individual's personal rights. Article 35 of the GDPR (Commission, 2018) requires every organisation dealing with data of European citizens to evaluate how each and every one of their processes touches upon the personal data of individuals by performing a Privacy Impact Assessment (PIA). However, the PIA is only the first step towards including privacy protection into the software development life cycle, as the GDPR further defines key principles for every data controller or processor to abide by, which will be further analysed in this thesis.

This general requirement for every organisation handling data leads to the discussion on how to optimally transfer the requirements of the legislation into business processes. Especially in the environment of software development, where most services require data of the individual wanting to use the service, a method of incorporating privacy protection into the development life cycle needs to be determined. Therefore, the predominant software development techniques need to be addressed. Whereas the traditional waterfall methodology suits a slow and secure product development, the agile methodologies lack a focus on security (Beznosov & Kruchten, 2004) (Terpstra, Daneva & Wang, 2017). Gen-

erally, agile development methods have been a major topic of research since their introduction through the Agile Manifesto in 2001 (Beck et al., 2001a) and have since become one of the most used software development methods. The State of Agile Survey (Versionone, 2016) reaches the conclusion that approximately 43% of their respondents have more than 50% of their organisations' development teams using agile development methods. On the other hand, a mere 4% state that there are no development teams utilising agile methods in their organisation. The survey on such a large scale of four thousand respondents can, even though unscientific, still be taken as a measure of the current, very high status of agile development methods in organisations. The general philosophy of agile methodologies is to be more flexible by developing smaller functionalities in close discussion with the stakeholders (Abrahamsson, Salo, Ronkainen & Warsta, 2017).

Introducing secure software development into an agile development life cycle has been a major topic of research with various different approaches proposed (OWASP, 2011) (Ahola et al., 2014) (Nyfjord & Kajko-Mattsson, 2008a). Privacy-specific aspects have generally been neglected in this process because the context of privacy protection was not precisely determined (De Capitani Di Vimercati, Foresti, Livraga & Samarati, 2012). The GDPR changes the landscape by defining privacy as data protection and implementing a system to act on those in breach with the regulation (Voigt & Von dem Bussche, 2017). This forces organisations to include data protection into their development life cycle. At the same time the GDPR creates a business need, the core reason for any work in agile development methods, by introducing consequences to any unlawful action. Assessing the risk of unlawful actions by performing a thorough risk assessment of functionalities is very time consuming and, if no adequate procedures are established, greatly increases the time required until new functionalities can be integrated into a solution (Ghani, Azham & Jeong, 2014) (Keramati & Mirian-Hosseiniabadi, 2008). While agile software development methods aim to reduce the time needed until a new feature can be deployed, risk assessment is often the most time-consuming job in the development life cycle.

Therefore, the different types of risk assessment are reviewed and analysed regarding their usability in an agile software development environment. The two major groups of risk assessment, i.e. quantitative (Bisias, Flood, Lo & Valavanis, 2012) and qualitative (Munteanu, 2006) risk assessment, are addressed. The goal is to incorporate a lightweight risk assessment into the agile mindset.

The combination of ever faster development and deployment due to the increased usage of agile methodologies in industrial environments and the need to be highly adaptable to the feedback of stakeholders leads to the question of how to integrate a high level of privacy protection without hindering the positive effects of agile software development methodologies. Hence, there is a need of research on how to introduce privacy protection into the agile software development life cycle. Furthermore, the modern industrial environment increasingly supports remote working (Felstead & Henseke, 2017). This adds further cause

for research as traditional communication paths are entirely obliterated when the traditional work-space, i.e. the office, is removed from the work environment. Therefore, the author takes the work setting of distributed teams into consideration in the design of a privacy protection framework for agile software development methods.

1.1 Motivation and Background

The main inspiration for this thesis is the legal requirement for any data processor or controller to perform privacy protection as defined by the European Union in the General Data Protection Regulation (Commission, 2018) and how to integrate these requirements into the agile software development process.

Throughout recent years, information technology has increasingly become an integral part in everyday life. This integration has changed from people merely using a few gadgets such as computers or mobile phones to the adoption of controlling every possible aspect in private lives, towards devices constantly listening for input and will soon result in, for example, fully autonomous cars. Accompanying the enhancement of information technology is the constant requirement to analyse and optimise the delivered service. Meeting these requirements is only possible by gathering data and further analysing the gathered data. The following example of a voice-controlled device illustrates the extent of gathered data. If the device is activated via a spoken keyword, the device needs to constantly listen in order to pick up the chosen keyword. Within one year, this results in 6 360 hours of recordings. If more than 100 million such devices existed, as claimed by Amazon for the smart-assistant Alexa (Al-Heeti, 2019), this would result in 636 billion hours of recordings in one year for just one such device. This huge amount of data of individuals is then stored in a database for future analysis. While the data requires protection, the individual needs some form of assurance and security that their data is only being handled in the agreed way.

This is where the General Data Protection Regulation steps in. Previously, the sole problem companies faced when data was misused by a perpetrator was if the lost data related to trade secrets. If a company leaked data of individuals, there was no drawback such as, for example, a fine. The GDPR returns the right over handled data to the individual, forcing organisations to abide by concepts such as Privacy by Design (Cavoukian, 2009) and imposing clearly defined consequences in the case that an organisation is in breach with the regulation. The main issue of how to incorporate privacy protection in the quickly changing environment of information technology has not yet been addressed.

1.2 Goals of the thesis

This thesis focuses on incorporating privacy protection as defined by the GDPR into the software development life cycle of agile methodologies. This results

in a Framework that can be deployed by all software development teams who implement processes using personal data of European citizens and therefore need to comply with the General Data Protection Regulation.

The designed Framework is further enhanced to suit the modern work environment of distributed teams, which is especially relevant for those in the flexible field of information technology. However, this leads to additional requirements for the framework which must be included without compromising privacy protection. The resulting Privacy Protection Framework is then supported by a software prototype, which aims to provide a basis for adopting the framework in the context of a real-life distributed team.

2 Problem Definition and Description

A software project with adequate privacy protection is highly dependant on secure implementation practices as well as control points in the relevant positions throughout the development life cycle (Ahola et al., 2014) (Danezis et al., 2015). Besides many iterations over the performed work, it requires performing tests to ensure all flaws have been taken care of. On the one hand, these requirements are well suited to the waterfall methodology of software development (Balaji & Murugaiyan, 2012). On the other hand, the waterfall methodology neither allows for quick deployment of new features, nor the adaptation of existing work and it generally hinders the deployment of regular product increments required in the modern day environment. Yet, the modern world demands short feedback cycles to developed work, as requirements and desires of customers can change rapidly. Especially when creating a new, groundbreaking product, different features need to be adapted and changed depending on user reaction. If these changes take too long, the requirements are very likely to already be redundant. The serial nature of the general waterfall methodology demands that a stage is completed before the next one can begin. This leads to very slow release cycles (Munassar & Govardhan, 2010). Hence, the use of agile methodologies has increased significantly (Versionone, 2018). However, short deployment cycles with many new features, i.e. creating a product increment every few weeks, results in a lack of focus of ensuring a secure product.

Especially since the introduction of the GDPR (Commission, 2018), which enforces every organisation to perform privacy protection when handling personal data, it is necessary to create a framework which enables adequate privacy protection in a quick and lightweight way in every software development project. The regulation forces every single data controller or processor, who processes or controls personal data of individuals of the European Union, to adequately protect the privacy of each individual (Li, Yu & He, 2019). The EU enforces the GDPR by imposing fines of up to 4% of the annual turnover, or €20m, whatever is higher (Commission, 2018). As a consequence, there is a prevalent need to create a privacy protection framework for the modern day environment of agile software development.

Another key aspect to consider is the modern work environment, i.e. development teams working remotely from various places. This presents new challenges to software development as key communication channels and haptic work in progress boards become unsuitable when the standard work environment of the office ceases to exist.

Therefore, the author proposes a Privacy Protection Framework for Agile Software Development Methods for distributed teams.

3 Literature Review

3.1 Privacy Challenges

Firstly, it is necessary to discuss the nature of the term *Privacy*. The term itself is defined in the Cambridge English Dictionary as "*someone's right to keep their personal matters and relationships secret*" (*Cambridge English Dictionary*, 2020).

The definition of Privacy has, thanks to the European Union (EU) and the General Data Protection Regulation passed in 2016 (Commission, 2018), become, for the scope of this thesis, rather definite because the GDPR focuses on the concept of privacy as "*the protection of natural persons with regard to the processing of personal data*" (*Art. 1 GDPR – Subject-matter and objectives*, 2016).

The EU defines privacy as information specific to an individual and focuses on this data protection, moving from a theoretical discussion of Privacy to a practical viewpoint on how to handle data processors and controllers. GDPR defines the term 'personal data' as "*any information relating to an identified or identifiable natural person; an identifiable natural person is one who can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person*" (Commission, 2018). Thus, as soon as information can be related to an identified or identifiable natural person, this information is defined as personal data and all entities in contact with such personal data need to abide and comply with the GDPR. Furthermore, the person in charge of managing and processing data in an organisation holds criminal liability for any wrongdoings regarding data handling.

There are seven principles defined by the GDPR, which any data processor or controller has to abide by, in order to be compliant with the GDPR. These seven key Principles are:

1. Lawfulness, fairness and transparency
2. Purpose limitation
3. Data minimisation
4. Accuracy
5. Storage limitation
6. Integrity and confidentiality
7. Accountability

The first principle, *Lawfulness, fairness and transparency*, determines that all processing and controlling should be within the law and communicated transparently and fairly to the affected individual

The next few principles, the principles of *Purpose limitation*, *Data minimisation*, *Accuracy* and *Storage limitation*, can be grouped within the overarching concept of *Privacy by Design*. The concept of Privacy by Design is further defined in Article 25 as "*the controller shall implement appropriate technical and organisational measures for ensuring that, by default, only personal data which are necessary for each specific purpose of the processing are processed. That obligation applies to the amount of personal data collected, the extent of their processing, the period of their storage and their accessibility. In particular, such measures shall ensure that by default personal data are not made accessible without the individual's intervention to an indefinite number of natural persons.*" (Art. 25 GDPR - *Data protection by design and by default*, 2018).

The second principle, *Purpose limitation*, determines that an organisation should only gather data for a specific purpose and only be processed to accomplish that specific purpose. *Data minimisation* ensures that every data processor and controller gathers only the minimal amount of data required to perform their disclosed purpose. The principle of *Accuracy* dictates that all gathered data has to be kept accurate and up-to-date, resulting in required mechanisms that establish accuracy and rectify or erase inaccurate data. *Storage limitation* determines that gathered data is only kept for such a time period that is required to perform the disclosed purpose.

The remaining two principles, *Integrity and confidentiality* and *Accountability*, are more in the vicinity of *Information Security*. The goal of *Integrity and confidentiality* is to ensure that the handled data remains confidential and is not accessible by anybody else than those allowed. *Accountability* determines that all data controllers and processors have to be able to prove that they are performing all of the other six principles and thus complying with the GDPR, e.g. by maintaining the adequate artefacts documenting the compliance process.

The European Union passes two kinds of general laws for all nation states, *Regulations* and *Directives*. The difference between these two legislative acts is, in general, that a *Regulation* is a binding act which is applied in its entirety, whereas a *Directive* sets out a goal that all EU countries must achieve (*Regulations, Directives and other acts*, 2020). Because the GDPR is a *Regulation*, it is binding for all nation states and thus requires compliance by all EU member states. Furthermore, it also needs to be complied with by those companies who do business within the EU and handle personal data of European citizens.

In light of how the GDPR defines privacy as data protection and the associated fines imposed if a data controller or processor is in breach of the GDPR, one can view the holistic protection of data as a risk to one's project. Thus, the Challenge of Privacy is a challenge of adequate Risk Management.

The key to healthy Risk Management is to ensure each risk is managed to an adequate level, therefore deeming the risk as acceptable to the system (Herath, Herath & Bremser, 2010). Touhill defines Risk as *"the potential loss resulting from a given action. It is a function of the interaction of threats, vulnerabilities, and likelihood (or probability) of threats acting against you"* (Touhill & Touhill, 2014). This function of risk is rather difficult to calculate in the context of information technology as the different variables are hard to pinpoint. A common way of identifying threats to a system is *Threat Poker* (Rygge & Jøsang, 2018), an adaptation of *Protection Poker* (Williams, Meneely & Shipley, 2010). Threat poker utilises the knowledge of every team member to evaluate the risk level of identified risks in an informal group discussion. The National Cybersecurity Agency of France (ANSSI) proposes to identify the most critical risk scenarios to one's system as a starting point (ANSSI & AMRAE, 2019). The main scenarios of unlawful action on data and thus threats to an individual's privacy rights are defined as per GDPR:

- Access
- Destruction
- Loss
- Alteration
- Disclosure

Threats only turn into risks, if they can be matched to a vulnerability in the system. Then, a risk will have an impact as well as a probability of occurring (Touhill & Touhill, 2014). Therefore, there needs to be a vulnerability in the system matching the threats defined by the GDPR to pose a risk to the system.

3.1.1 Risk Assessment

The two most common techniques of risk assessment are quantitative and qualitative analysis. The quantitative analysis is based on determining values to all variables whereas the qualitative analysis is based on scenarios (Altenbach, 1995).

The *Quantitative Risk Assessment* tries to give value to each feared event by using statistical and mathematical methods after identifying threat sources, vulnerabilities, likelihood and impact (Embrechts, Frey & McNeil, 2005). The result of a quantitative assessment determines a monetary value to the risk. The GDPR helps with determining a monetary value to the impact, as a company in breach of individuals' privacy rights is *"subject to administrative fines up to €20.000.000, or in the case of an undertaking, up to 4 % of the total worldwide annual turnover of the preceding financial year, whichever is higher"* (Commission, 2018). Further, other costs such as e.g. the cost to acquire the information, damage to the organisations' reputation and the cost to replace

need to be evaluated (Touhill & Touhill, 2014). Once a monetary value is allocated to a risk, the Risk Manager can deem the risk level as adequate. If not, this risk needs to be readdressed with the goal of reducing the monetary value by e.g. reducing or even removing the vulnerability through adequate measures (ANNSI & AMRAE, 2019) (Ahola et al., 2014). The quantitative method is documentation and calculation heavy but delivers clear results of potential risks to the organisation (Munteanu, 2006).

In contrast, *Qualitative Risk Assessment* does not allocate a concrete value to each risk but rather ranks risks on a risk assessment matrix consisting of high, medium and low. Thus, a ranking of all risks is established. This is especially helpful for areas where it is difficult to determine exact measurements (Touhill & Touhill, 2014). Benefits of this more lightweight approach, e.g. not based on exact calculations, include a faster identification and assessment of which risks should be addressed as soon as possible (Landoll & Landoll, 2005). The process of risk assessment is identifying threats and vulnerabilities, then matching these to build a specific risk scenario. After estimating likelihood and potential impact of the identified risk, the risk is assessed on the previously mentioned risk assessment matrix.

The final step in risk assessment is deciding how to proceed with the identified risks by defining mitigation procedures which make the residual risk acceptable.

3.2 Agile Methods

Agile methods have been a major topic of research since their introduction through the Agile Manifesto in 2001 (Beck et al., 2001b). Multiple different Methodologies, such as *Scrum*, *Extreme Programming* and *Kanban*, have been developed since and will be addressed shortly in this chapter. In general, agile methods aim to reduce the amount of planning in turbulent environments in order to reduce the time required to introduce new features to a system and therefore increasing the ability to react to changes. This is accomplished by the core concepts of cooperativeness, self-organisation and reduction of task-size in order to develop software at regular intervals, with each iteration producing a working system (Martin, 2002).

Extreme Programming (XP) was first introduced as a Software Development Method in 1998 (Beck, 1998) and has since been further established. At the core of XP lie a set of best practices which a team should use for best possible outcome with no specifically defined area of deployment. These core principles are defined as (Beck, 2000):

- Small team size
- Multiple increments

- Best-case programming practices
- Continuous integration

Tasks are prepared in direct communication with the customer as user stories. One of the fundamental ideas of XP is that there is no process that fits every project perfectly, but rather practices should be tailored to suit the needs of individual projects (Lindstrom & Jeffries, 2004).

The *Scrum Methodology* was originally developed in the 1990s for managing and developing products but has since been adapted and used in many different areas such as software development, marketing and operations (Sutherland & Schwaber, 2017). These manifold use-cases of the Scrum Methodology are possible due to its rather generic design, more similar to a framework based upon empirical process control theory, rather than a strict Methodology. The Scrum Framework is used in approximately 72 % of the respondents to the State of Agile Survey (Versionone, 2018), albeit not a scientific paper, still highlights the importance of Scrum in modern software development. The Scrum Framework introduces the three key roles *Scrum Team*, *Product Owner* and *Scrum Master*, as well as highlights the special characteristics of the *development team* being self-organizing and cross-functioning. The key artefact for keeping track of features and tasks is the *Product Backlog*, which consists of all identified tasks for a project. The *Sprint* is coined as a time-frame of one to four weeks, where all tasks on the *Sprint Backlog* are completed. The development team determines which items are moved from the Product Backlog to the Sprint Backlog and thus handling the progress of the product whereas the Scrum Master ensures the team does not meet any hurdles during development. If a problem occurs nonetheless, the Scrum Master takes action to enable a best possible work environment for the team as well as ensuring that the Scrum principals are embodied (Stoica, Ghilic-Micu, Mircea & Uscatu, 2016). After each Sprint, the *Sprint Review* is used to analyse the completed Sprint.

The *Kanban* development method originated from the Japanese car manufacturer Toyota in the 1950s, where it was used to determine bottlenecks within a car factory. The Japanese word "Kanban" directly translates to "signboard" and describes the original use in manufacturing as a scheduling system. In 2004, David J. Anderson (Anderson, 2010) aided an IT team within Microsoft by using such a signboard in order to visualise the workflow currently being addressed. By visualising the workflow to the team, it became apparent that they were aiming to complete too many tasks at once, having started some tasks although they were dependant on the completion of other, not yet finalised, tasks. Thus, the Kanban Software Development Method was born. Anderson identifies five key principles (Anderson, 2010) which were observed in the successful deployment of Kanban.

- Visualise work in workflows
- Set a limit to Work-In-Progress

- Manage the workflow
- Model approach to collaboration
- Communicate management policies

The Work-In-Progress, once the five key principles are deployed by a development team, is much more focused, as well as clearly communicating to the Team which tasks currently have the highest priority, thus resulting in a more concise and productive work environment with a clear view on reducing or possibly eliminating bottlenecks in the development process (Ahmad, Markkula & Oivo, 2013).

3.3 Secure Agile Software Development Methods

The lack of privacy-minded research is clearly noticeable when researching previous publications in which privacy protection is included in an agile software development method. Due to the lack of previous research, the author turns to the similar field of security focused agile development methods. In the following, the most prolific Secure Agile Software Development Methods will be described with a focus on the features that distinguish each method from the next.

3.3.1 Open Web Application Security Project

The Open Web Application Security Project (OWASP) (Vähä-Sipilä, 2010) proposes splitting agile methods into respective phases, e.g. three phases for Scrum:

- Requirement Engineering
- Development phase
- Integration and release

Further, OWASP discusses how to address issues within each phase. These phases should not merely be interpreted as phases but rather as parts of a functioning machine, seamlessly working simultaneously.

During *Requirement Engineering*, OWASP proposes two different kinds of user stories. These two kinds are negative as well as positive security stories (OWASP, 2011). The positive security story deals with a specific task to enhance security such as ensuring a high level of cryptography when handling passwords. However, the negative, e.g. evil, user story envisions which persons might want to attack the system and by what means this could be made possible. Thus, by creating awareness as well as discussing possible attack vectors such as SQL injection, the evil user story mitigates each attack vector addressed by the security professionals. Obviously, this is a great way to ensure that common knowledge attack vectors are mitigated, but as to the extensive resources available to hackers and constant changes in environments, programs and applications one can only try to make the effort necessary to successfully attack the system higher than its worth.

The *Development Phase* is described as the phase during which backlog items are implemented. OWASP defines three key ideas to include in this phase. The first major idea is to add a security check to each feature on the backlog, e.g. whether the particular feature is actually completed from a security point of view and not just from a functional point of view. Secondly, OWASP suggests the use of threat modelling as a recurring task in every Sprint. Thus, threat analysis should be added as a separate task in the Sprint Backlog for every single potentially risky task (Vähä-Sipilä, 2010). The third main idea is to produce a checklist with all problems that have previously occurred as well as possible future risks. This checklist should be kept up-to-date in order to aide in correctly addressing recurring risks.

The *Integration and release* phase is highlighted by OWASP to ensure that the Product Owner, the person responsible for accepting a user story as finished, is aware that, as previously mentioned, there will always be some residual risk left. The Product Owner needs to accept these as adequate or, if necessary, to reschedule the task for another iteration if the remaining risk is too great.

3.3.2 Microsoft Security Development Life Cycle for Agile

The Microsoft Security Development Life Cycle for Agile (MSDL-Agile) is based upon the integration of a set of best security practices previously identified and established by Microsoft in the Secure Development life cycle (SDL). These security activities, in order to fit into an Agile Software Development Method, need to be analysed and made as lean as possible. One way to organise the security activities with the goal of reducing workload and required time, proposed by Sullivan (Sullivan, 2010), is classifying each activity within the SDL according to the frequency an activity needs to be completed. Thus, three groups of frequencies are identified:

- One-Time
- Every-Sprint
- Bucket

These frequency groups heavily indicate which kind of tasks are included. *One-Time* activities include cornerstone activities such as a Threat Impact Assessment and Threat Modelling, as well as identifying who will serve as a privacy expert on the development team. Although these activities are defined as being One-Time requirements, these activities are not completed once at the beginning of the project, stapled on the office walls and held as sacred because these activities will also need to be adapted and included in the iteration process throughout the lifespan of the entire project whenever changes and future learnings occur. On top of that, the development team will also need to complete Bucket as well as Every-Sprint requirements within the first Sprint, rendering the workload completely out of scope for Agile development teams of any size.

The MSDL-Agile therefore defines timelines by which each One-Time requirement should be initially completed and then updated in any Sprint whenever changes, updates or adjustments arise.

The opposite of One-Time activities are those in the group of *Every-Sprint* activities. No feature may be deployed without passing all Every-Sprint activities, as these are necessary in order to ensure each new feature meets the defined security requirements. These include running the implemented analysis tools, debuggers and tests on the new features. Each new feature must also be analysed if and to what degree it could infringe upon any security related topic. Thus, each new feature must be included in a previously conducted Threat Model.

The third group of activities are bundled as *Bucket* activities. MSDL-Agile defines these activities as to be completed periodically, with one activity out of the Bucket to be completed on every Sprint. Three Buckets are proposed, with Bucket A being *Security Verification*, Bucket B being *Design Review* and Bucket C being *Response Plans*. Bucket A includes tasks such as Fuzz-testing and Attack Surface Analysis. Bucket B includes Reviewing Privacy impacts as well as User Account Control. Bucket C includes tasks such as establishing support documents and preparing Plans for worst-case scenarios. However, MSDL-agile does not specify any pre-defined order in which these Bucket requirements should be fulfilled, it rather states that each needs to be completed multiple times during a product life cycle.

Taking this approach of categorizing security related tasks by how often they will need to be completed during a product life cycle seems to be a good starting point for classifying privacy related tasks within an Agile Software Development Method.

3.3.3 Stockholm Integrated Model

Nyfjord and Kajko-Mattsson from Stockholm University propose an integrated model, merging agile practices with risk management (Nyfjord & Kajko-Mattsson, 2008a). The integrated model described in the research study is deployed as an example upon the agile method Scrum. The framework consists of four major roles:

- Business Manager
- Product Owner
- Risk Management Forum
- development team

The introduced roles allow a direct matching to two roles defined within the Scrum Framework. Firstly, the *Product Owner* can be directly translated to the Product Owner in Scrum. The Product Owner is deemed responsible for determining risks in the Requirement Engineering Phase, but can delegate this to another person, usually from within the development team. Secondly, the role

of *development team* is directly translatable to the development team in Scrum. The other two introduced roles, the *Business Manager* and *Risk Management Forum*, are introduced as new positions. The Business Manager is responsible for an abstract, high-level planning and management of risks regarding business decisions. The Risk Management Forum is designed to be an organisation-wide communication platform for all discovered risks.

The Product Owner and development team are responsible for identification, assessment and management whereas the team is solely responsible for monitoring and final analysis as well as acceptance of risks. This allocation eliminates the role of *Scrum Master*, whose organisational responsibilities are distributed to the Team itself.

3.3.4 The Secure Agile Software Development Life Cycle

The Secure Agile Software Development Life Cycle (SASDLC) by a conglomerate of Finnish professionals (Ahola et al., 2014) as part of the *Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT and digital business* introduce security in an agile software development method by creating *Generic Security User Stories* to be included in the backlog. This leads to more visibility for Security Tasks as Activities to be completed. SASDLC bundles the Security Activities into three classes depending on the nature of the Activity. The three classes are *Functional Features*, *Engineering Tasks* and *Ways of working*. *Functional Features* describe all tasks that have a clear function that needs to be developed as a feature such as authentication, logging or prompting for user-consent. *Engineering Tasks* are security-related tasks that have a clear beginning and an end, and thus can fulfil the definition of done in backlogs. These include recurring as well as nonrecurring tasks such as risk analysis or testing. *Ways of Working* describe security-related concerns of how a task such as coding guidelines or encryption standards should be done.

These three different types of classes need to be included in the Product Backlog, but how this is performed varies greatly. Whereas *Functional Features* can be added directly, *Engineering Tasks* could be added as checks to each security-related Feature. While some *Ways of working* can merely be included as checklists, others need to be specially trained and agreed upon, leading to the suggestion of a *Sprint Zero* - a Sprint to be completed before any other Sprint - for these tasks. The SASDLC also creates vital control points for the example agile method Scrum. Ylimannela and Helenius (Ahola et al., 2014, p.22) highlight the fact that the Product Owner plays the most vital role within the Scrum Framework regarding the addition and acceptance of tasks as well as final responsibility towards the developed software. The authors establish a clear mismatch between the role of Product Owner with its knowledge of what's occurring within the development process and, in many cases, no adequate security training, deeming the Product Owner to be the least viable person to establish if a feature is secure.

Vähä-Sipilä argues that "*Each software security activity is an investment,*

and hence it needs to have a business reason to happen. A good way to find this business rationale is to base all security activities on risk analysis." (Ahola et al., 2014, p.17). As previously mentioned and also noted by Vähä-Sipilä, risk is the combination of severity and likelihood which are sometimes difficult to measure. Thus, the SASDLC takes a haptic Risk Board as a cornerstone to visualise all identified threats. Once a risk is identified, it is graded on a scale of low to high, described and allocated to a member of the team. This allocation makes the team member the sole *Risk Owner* of that specific risk. Hence, the responsibility of risks is distributed between the team members who need to ensure their risks are handled appropriately.

3.4 Privacy Impact Assessment

The General Data Protection Regulation (GDPR) that came in effect on May 25, 2018, is a *Regulation*, and therefore has to be complied with by all nation states of the European Union (*Regulations, Directives and other acts*, 2020). Article 35 of the GDPR states that every data controller is obligated to perform a Privacy Impact Assessment (PIA) to ensure every individual's data is protected (*Art. 35 GDPR – Data protection impact assessment*, 2018). PIA is the core process to establish if, how and to which degree a project will impact the privacy rights of individuals and assert controls to mitigate potential threats to these rights. Prior to and in light of the GDPR, the German Office for Information Security (BSI) (BSI, 2011) and the French Data Protection Authority (CNIL) (CNIL, 2018b) have published high-level guides on how to perform adequate PIAs. Especially the PIA as described by CNIL provides a good basis as well as additional information and templates on implementation (CNIL, 2018c) and is therefore analysed in the following. Another valuable PIA is the one published in 2014 by the Office of the Australian Information Commissioner (OAIC) (OAIC, 2014) regarding adherence to the Australian Privacy Act of 1988. Albeit being over 30 years old, the Australian Privacy Act had already determined that information does not always have to include specific details such as an individual's name to allow identification. This has proven increasingly true in recent years since the massively interconnected Internet of Things with concepts such as Machine Learning on Big Data and everyday interaction with the Internet is predominant for most individuals worldwide. Hence there rarely exists a case where a project has no single touchpoint with privacy rights of individuals, resulting in a high importance of PIAs. Both the GDPR and the Australian Privacy Principle (APP) require data processors and controllers to implement processes and controls to ensure compliance with their respective legislation.

3.4.1 Privacy Impact Assessment by CNIL

The PIA designed by CNIL (CNIL, 2018b) consists of four phases which are to be completed as iterations. It is intended for all stakeholders involved with a project that processes or controls personal data to clarify in what capacity the

personal data is handled. This PIA does not include an initial analysis whether a PIA is actually necessary for the project in question. Hence the necessity of a PIA needs to be addressed first.

The four phases are *Context*, *Fundamental Principles*, *Risks* and *Validation*. According to CNIL, these need to be completed multiple times before adequate privacy protection can be assured, as well as reviewed whenever changes or updates during the lifespan of a project occur.

Within the first phase, *Context*, the team is supposed to identify the "*nature, scope, context, purposes and stakes*" (CNIL, 2018b) of the data processing for the respective project. In this phase, the data controllers and processors are identified together with the legal statutes applicable to the project, e.g. whether the data is under jurisdiction of the European Union and therefore needs to comply with the GDPR. Another important task in this phase is identifying which personal data will be handled, who will be in contact with this data as well as how long it will need to be accessible or, in other words, stored. Furthermore, the processes of how this data will be gathered, distributed, stored, modified and deleted need to also be addressed from a high-level point of view. These tasks should be performed by the project owner together with an expert in privacy protection in order to gain a general view on the project in the context of privacy.

The second phase *Fundamental Principles* focuses on building "*the system that ensures compliance with Privacy Protection Principles*" (CNIL, 2018b). The first step in this phase is to decide and justify the purpose and basis for gathering data. The gathered data should be as minimal as possibly required whilst ensuring a quality as high as possible. Further, data should never be stored indefinitely. This stage can be used to determine which data is required for what length of time. The handling of data is defined according to the GDPR and would vary in other jurisdiction areas. The second stage of this phase focuses on controls and processes of gathered personal data. This includes obtaining consent by an individual and supplying the individual with all information on how the data will be used.

The third phase, *Risks*, studies all risks related to personal data. The first part of this phase is an assessment of any existing or planned controls on data security. The second part is the actual risk assessment. Four indicators are used to establish the risk level for each feared event:

- Impact
- Severity
- Threat
- Likelihood

These indicators determine a residual risk level for each identified risk. In light of the residual risk, a risk is either accepted or, if not, a proposal with possible changes to reduce the level of an indicator is to be made.

The fourth phase, *Validation*, deals with validating the decision of accepting the residual risk or alternatively re-iterating over previous steps to further reduce the risk level. The decision is made by highlighting firstly which controls ensure compliance to the fundamental principles of the GDPR and secondly which controls ensure data security and visual mapping of the risks on a matrix depicting severity and likelihood. The team further has to create a plan for every risk in case it becomes reality.

3.4.2 Privacy Impact Assessment by the OAIC

The PIA by the OAIC (OAIC, 2014), although not of legal use for organisations in the EU unless they are operating in Australia, is of broad nature and allows for good adaption to projects. It is the basis for Data Protection Guidelines in the United Kingdom (ICO, 2019). The PIA consists of ten general steps:

1. Threshold Assessment
2. Plan the PIA
3. Describe the project
4. Identify and consult with stakeholders
5. Map information flows
6. Privacy impact analysis and compliance check
7. Privacy management - addressing risks
8. Recommendation
9. Report
10. Respond and Review

The first step is identifying if a PIA is necessary or not. The Threshold Assessment (TA) includes general information on the project as well as processes within the project that touch upon privacy rights. If the TA result is that no PIA is necessary, the reasons as to why not need to be documented nonetheless. If the TA shows that a PIA is indeed necessary, the second step is planning the PIA by identifying who will conduct it, setting a timeframe, budget, resources and examining how detailed the PIA needs to be. Subsequently, the project is analysed in a broad overview in the third step. The fourth step focuses on communication with the stakeholders as they can be key assets in identifying privacy risks and concerns. The communication is especially valuable when proposing mitigation tactics that might have impact on the project. The fifth step, mapping information flows, aims to identify the path data takes within the particular system as well as the accessibility and security thresholds currently in place. Consequently, the impact of the information flow on privacy rights is

analysed in the sixth step. This includes for example risk analysis and compliance. The seventh step, privacy management, is to analyse previously identified risks and how to address these. The OAIC provides a collection of possible risks as well as suggested mitigation strategies (OAIC, 2014, p.28-29).

The last three steps deal with how to proceed after deciding on what to do with the identified risks. This includes step eight where a set of recommendations is formulated, step nine where a report concludes the information of the previous steps in a formal document. The final step, respond and review, defines the stage where the project manager takes final action on the report delivered. The OAIC recommends to update the PIA if necessary, describing it as an ongoing process rather than a one-time activity.

4 Approach and Methodology

The research methodology applied in this thesis will follow the methodological framework proposed in "Design Science in Information Systems Research" by Hevner et al (2004).

This methodology for design science introduces seven cornerstones for research in information systems:

- Design as an artefact
- Problem relevance
- Design evaluation
- Research contributions
- Research rigor
- Design as a search process
- Communication of research

In the following these cornerstones will be described and related to the design of this thesis.

Design as an artefact The first cornerstone describes that the conducted research must create a relevant artefact. This artefact can take the form of a model, a framework or a method. This thesis creates two such artefacts, a framework for an agile software development method as well as a software prototype to support the framework in the context of a distributed team.

Problem relevance The main goal of design science research must be to develop technology-based solutions for currently important issues. With the introduction of the GDPR, privacy protection has to be an integral part of all software development projects involving personal data of individuals. The relevance of incorporating privacy in agile software development is therefore high.

Design evaluation The created artefacts of design science research must be evaluated and discussed thoroughly. Both of the created artefacts, the framework and the prototype, of this thesis are critically examined in the respective chapter.

Research contributions The research must add clear value and, where possible, contribute towards further research. Complemented by additional research on the particular requirements, the framework of this thesis can be adapted for any legislation as well as other software development methods.

Research rigor Research must be conducted by rigorously utilising research and evaluation methods. This process begins with assessing the current State-of-the-Art, as done in section 3 and continues throughout the design of the framework in section 5. The models in this framework are primarily in the Unified Modelling Language (UML), a professionally and widely used modelling language in Information Systems. A prototype is developed using the programming languages Python, JavaScript, HTML and SQL.

Design as a search process The research artefacts must be the result of an evaluation of the problem space. This search process is conducted in section 3 in which the State-of-the-Art in secure agile software development methods is reviewed and analysed.

Communication of research The conducted research must be presented in a manner befitting of the specialised, technology-adept audience as well as a general-knowledge audience. The results of this design science research will be presented in the master thesis with the intention of being comprehensible to a wide audience.

The first step made in this thesis is identifying the motivation and the resulting scope of the problem. This first identification step results in the goal of the thesis as designing a model which allows the integration of privacy protection in an agile software development setting. In light of the design science research as proposed by Hevner et al (2004), the resulting artefact will be supported by a software prototype. The prototype aims to introduce the designed model into a real-life scenario of software development in distributed teams. To ensure that the model is of high relevance and actuality, the current State-of-the-Art is analysed. As the topic of this thesis lies in a novel area, the focus of the analysed research lies on an area with parallels to privacy protection, specifically the area of secure agile software development methods. Additionally, the problem scope of privacy protection is researched, with a specific focus on the General Data Protection Regulation, passed by the European Union (2018). Subsequently, these two areas are combined and deployed on an agile software development method.

The designed privacy protection framework is based upon the agile method *Scrum*, as it is the most commonly used agile software development method. The general design of agile methods and their nature to be applicable to many different scenarios, allows an adaptation of the designed framework to suit any other agile method as well. The author proposes key artefacts which act as cornerstones in the incorporation of privacy protection to the agile mindset of quick adaptability to changing requirements. These cornerstones, such as a *Privacy Scrum Board*, are designed to be directly adoptable for a software development team using Scrum, but remain adaptable to the deployed agile method and to the respective problem scope of individual projects.

Including the modern work environment of distributed teams changes the general layout of discussion to the direction that all communication and documentation needs to be accessible remotely. One such important artefact is the aforementioned *Privacy Scrum Board*, which visualises the current work-in-progress. The author proposes to digitalise these features in order to allow for remote access.

The goal of the designed software prototype is to embody these aspects. Further, it aims to support the distributed work environment by introducing a data process assessment tool and a privacy risk web service.

5 Development of a Framework for Privacy Protection

The key challenge of this framework is the protection of privacy in a lightweight manner to ensure agile software development projects adhere to the GDPR when controlling or processing personal data. As discussed in the Literature Review, the GDPR defines privacy as data protection, thus each project handling personal data undertaken in the European Union needs to adequately mitigate any possible risk to controlled or processed data. In the following, results of the conducted literature review are shortly highlighted regarding their usage within a Privacy Protection Framework for Agile Software Development Methods in Distributed Teams. Subsequently, these results are further analysed and modified to enable adequate privacy protection to suit the objective of the proposed framework. The identified requirements for incorporating privacy protection results in the privacy protection framework displayed in Figure 5.1.

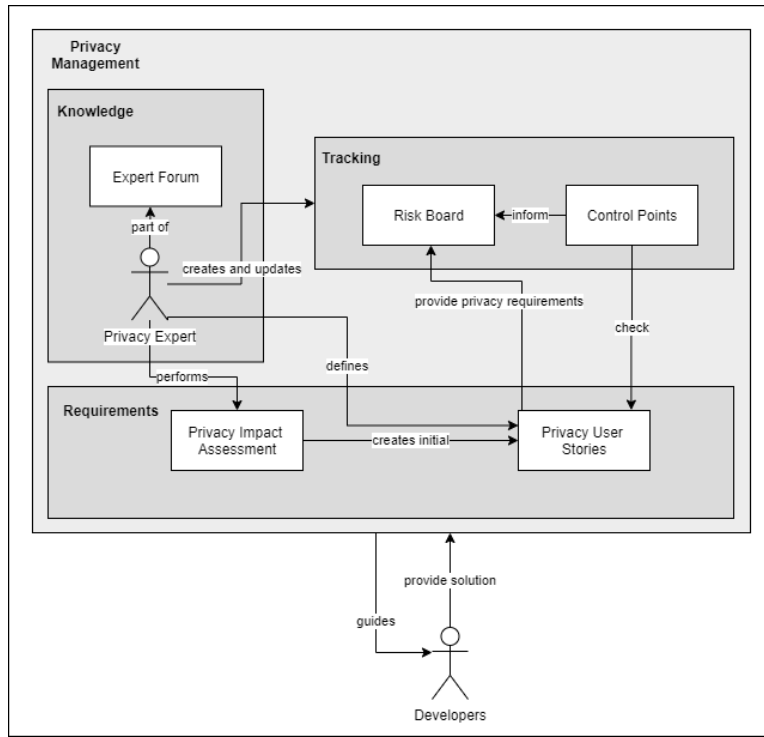


Figure 5.1: Privacy Protection Framework

Figure 5.1 displays the key components and their interactions to ensure adequate privacy protection. The encompassing field of *Privacy Management* guides developers who consequently provide solutions to managing privacy. Pri-

vacancy Management depends on the interaction of three components. These components are *Knowledge*, *Requirements* and *Tracking*. The Knowledge component consists of experts in the field of privacy. There has to be a *Privacy Expert* for each privacy project to enable the required privacy knowledge is available. The Privacy Expert is part of the *Expert Forum* where privacy protection is discussed on a peer-to-peer basis.

The second component is defined as *Privacy Requirements*. The combination of *Privacy Impact Assessment* and *Privacy User Stories* creates the necessary identification of Privacy Requirements for a project. The PIA is performed by the Privacy Officer, who also defines *User Stories*. Further, performing the PIA results in an initial group of identified *User Stories*. Privacy Requirements are provided to the third component, *Tracking*. The tracking component encompasses the visualisation of identified privacy requirements in the *Risk Board* and the examination of privacy user stories through *Control Points*. Control Points inform the Risk Board of the results of these checks. The tracking component is created by the Privacy Expert and updated throughout the development life cycle.

The law states that each project needs to incorporate *Privacy by Design* and perform *Privacy Impact Assessments* when data of individuals are controlled or processed (Commission, 2018). A cornerstone of Privacy Impact Assessments is identifying the requirements of a project and the risks to the data of individuals the project entails. In order to assess the impact of privacy threats, Risk Assessment of each threat is necessary. Although the GDPR has created monetary fines to anybody in breach with the data protection regulation, thus enabling risk assessors to allocate a monetary value to a potential privacy risk, the exact calculation methods of quantitative risk assessment remain difficult. The necessity of defining exact values to potential impact, likelihood or severity would require a time-consuming evaluation with a result based on guesswork. Therefore, the quantitative method of assessing risks is deemed inappropriate for this framework.

The Qualitative Method, described in subsubsection 3.1.1 suits the field of software development very well, although it is slightly more dependant on experience and knowledge of the assessor. With the goal of ensuring privacy protection and at the same time avoiding documentation and calculation heavy tasks, the Qualitative Risk Assessment Method with its lightweight procedure is selected by the author for this framework. It allows for a quick assessment of every risk and furthermore, establishes residual risk levels depending on the implemented privacy measures. The goal at the end of each assessment is to have an acceptable residual risk, e.g. a remaining risk level that is low enough to be deemed acceptable by the risk owner.

However, this cannot be accomplished by simply adding a set of privacy features to a project. Protecting privacy can only be accomplished by also ensuring an adequate level of security during the development life cycle. A project cannot meet the requirements of the legislation to privacy protection, if the project does not include relevant and up-to-date security measures. An

important factor in including security is to incorporate security best-practices as early as possible within the development life cycle, as adding security in hindsight is resource-intensive and hence very costly. Furthermore, as identified in subsection 3.1, the GDPR defines *Information Security* as a key tool for data controllers and processors in order to comply with the regulation. Without the use of security mechanisms, an individual's data is not protectable. Thus, security measures are imperative in the information technology environment to perform adequate privacy protection.

Therefore, privacy protection has to build upon security as a core tool to enable data protection as required by the GDPR. Consequently, the proposed Privacy Protection Framework will include important security aspects in light of privacy protection features. Hence, when discussing the term *privacy*, it always includes *security* as a tool to enable data protection in the information technology environment.

To enable the usage of the proposed Framework in the modern world, it is embedded into an agile software development method. The Scrum methodology is chosen as an exemplary method because it is by far the most used agile method within the software development environment (Versionone, 2018). Additionally, the three phases of Planning, Sprint and Sprint Review fit the Privacy Impact Assessment proposed by CNIL. Combined with the significance of Scrum as an agile framework lead the author to select this agile method for the proposal of a Privacy Protection Framework for Agile Software Development Methods in Distributed Teams. However, due to the nature of agile methods, this Framework can be used with every other agile software development method as well. Using this Framework with another method would require slight adaptation to honour the respective artefacts utilised in the various methods. The *Sprint-Zero*, a Sprint designed to establish a project's setting (Maria, Rodrigues Jr & Pinto, 2015), is also adapted to ensure the context and architecture of the project performs *Privacy by Design*. A key artefact for a project to adhere to the GDPR is the Privacy Impact Assessment. As previously outlined, the iterative methodology of the PIA designed by CNIL is based upon a waterfall model. Therefore, the process needs to be adapted and translated to suit the spiral model of an agile methodology such as Scrum.

5.1 Privacy Protection Requirements

Before focusing on the development of the privacy protection framework for Scrum, the key learnings identified in the Literature Review (section 3) for incorporating privacy protection into the general philosophy of agile methods are highlighted.

The author proposes to combine a mixture of the features used in the reviewed Secure Agile Software Development Methods for the proposed Privacy Protection Framework. At the core of the framework lies the definition of

privacy-related user stories. This concept stems from security user stories used in multiple of the reviewed secure agile software development methods. These identified privacy user stories are further visualised by introducing a *Privacy Risk Board*. The risk board enhances the currently identified privacy risks within the project by visualising them on an adapted Scrum Board for the Scrum Team. The Privacy Risk Board includes all current user stories with a visual distinction for privacy user stories.

The idea of an organisation-wide Risk Management Board in the nature of a forum (Nyfjord & Kajko-Mattsson, 2008b) is adopted as a knowledge exchange forum and escalation platform for severe privacy risks. The concept of a Sprint-Zero is adopted and designed to initialise the projects development life cycle by identifying the initial project's scope. This allows for early identification of data collection and process activities, resulting in a clear picture of the upcoming work and which development measures need to be incorporated into the project during the initialisation stage. The identified concepts to enable privacy protection are:

- Privacy User Stories
- Qualitative Risk Assessment
- Privacy Risk Board
- Iterative Privacy Impact Assessment
- Privacy Expert
- Privacy by Design

First of all, there needs to be a business need allocated to the required time focused on privacy protection. This is performed by utilising the concept of *Privacy User Stories*. Every privacy user story describes a specific privacy requirement to the product from the point of view of one user. This way of describing the required work translates the sometimes vague field of privacy to a specific task which can be completed. Describing privacy requirements as user stories creates a direct need of one user towards the product, whether this user is the end user or a developer is irrelevant. Especially in the context of agile software development methods, where all work is focused on providing additional value to the product and needs to be directly discussed with the respective stakeholders, a business need for the work to be done has to be established. However, the nature of every privacy user story poses a risk to the project as legislators have introduced legal fines for those in breach of their respective privacy laws. The risk level of every privacy risk has to be evaluated and presented to the developers. Evaluating the risk level needs to happen in a lightweight manner, therefore a *qualitative risk assessment method* is incorporated by the author. Creating a *Privacy Risk Board* to display the calculated risk levels and their current status ensures that the privacy risks remain at the forefront of the development team.

Risk assessment is also a core part of *Privacy Impact Assessments* (PIAs), a process required by law in different legislations such as the European Union and Australia. PIAs ensure that an organisation identifies all data processes and potential risks of such data handling. However, the researched PIAs in section 3 are not of an iterative design such as agile software development methods. Therefore, the process of performing a PIA needs to be adapted to suit the iterative nature of agile methods. To be able to perform a PIA or create privacy user stories, expert knowledge in the field of privacy is required. Therefore, a *Privacy Expert* is essential in the development team. The privacy expert plays a core role in performing the necessary planning and reviewing of privacy user stories as well as tracking the status of privacy risks.

The concept *Privacy by Design* (PbD) is identified as a legal requirement for data controllers and processors, at least for those dealing with personal data in the European Union. PbD entails that privacy is discussed at the very beginning of a project's life cycle. This discussion begins with determining which data is required for the project to perform its goal but furthermore signifies that privacy must be included as a major factor in decisions such as architectural design and choice of third-party services.

To summarize, the cornerstones of the proposed Privacy Protection Framework introduced in the following chapters are:

1. Privacy Officer
2. Privacy Management Forum
3. Privacy User Stories
4. Privacy Review
5. Sprint-Zero
6. Digital Risk Board
7. Agile Privacy Impact Assessment

In the following, these key features and their functionality are analysed and deployed upon the agile methodology *Scrum*.

5.2 Roles

A major part in the Scrum Framework lies in defining the roles of key actors and their responsibilities. This section addresses the necessary adaptations to the previously described standard Scrum roles to enable a privacy protected development life cycle.

The *Product Owner's* (PO) role in the Scrum Methodology is the person solely responsible for the product. The PO is the sole contact person for the

client and needs to translate the client's requirements to the product into the work to be done. By acting as intermediary between the Scrum team and the client, the PO protects the team from disruptive contact with the stakeholders. Therefore, the PO is responsible for maintaining the Product Backlog. This entails defining user stories which are to be developed by the team. The PO is the gatekeeper at the end of each Sprint cycle for every developed user story. The PO decides if a user story meets the defined *acceptance criteria*. The acceptance criteria should be answerable with a *yes* or *no*. However, the PO will generally lack the required skills and training to include privacy related tasks and adequately assess their importance, scope, and necessity during the different stages of the development life cycle.

The heart of every agile software development method is empowering the development team to be capable of performing any required work. Therefore, the new role of *Privacy Officer* (PrO) is introduced. The PrO has to work closely with the Product Owner on defining privacy-specific tasks. These tasks are described as user stories and can then be included in the Product Backlog. Each Privacy User Story needs to be flagged for a special review performed by the PrO after development. If a user story is identified by a developer as privacy-related during the Sprint, this user story needs to be flagged and added to the special privacy review. For the PrO to be fully incorporated on the responsibility and handling of risks, it is necessary for the PrO to be the risk owner of each identified privacy risk. This leads to the added benefit that the PrO is not merely a control mechanism, but actively assists the team members during the development of a privacy user story. This further ensures the inclusion of the PrO during the development phase and aims to reduce potential reiteration of privacy user stories. This gives the PrO the required insights into the actual skillset of the team members, enabling the PrO to judge which training is required by whom. The PrO will take over from the PO in gatekeeping on all PUS in a special privacy review at the end of each Sprint cycle. Further, the GDPR defines multiple responsibilities of a company-wide Privacy Protection Officer which need to be upheld by the PrO for their project. These include training the employees on GDPR compliance requirements as well as conducting regular assessments. Both tasks will be implemented through privacy user stories, highlighted at a later stage in this thesis. The GDPR further demands documentation on all data processing activities conducted by the company. The PrO is therefore responsible for the granular view of their project. This entails documenting the conducted data processing activities and maintaining the documentation. The PrO is also the main contact person for data subjects who inquire about how and in what capacity their personal data is being used, as well as what measures have been put in place to protect this data. However, legislation demands these responsibilities on an organisation-level, whereas a PrO can only be deemed responsible for their project.

Therefore, an organisation-wide board for overarching privacy matters is introduced. The *Privacy Management Forum* (PMF) consists of all Privacy

Officers of the company's project teams. The main goal of the forum is to discuss privacy risks to the entire organisation identified by each PrO. This also allows for a possible escalation-stage in case severe privacy risks are identified within any singular Scrum Team. Additionally, the PMF is used as a platform to enhance the knowledge-base of every single PrO through continuous communication and discussion. It is also important to note that this will keep the main goals of the organisation in relevant perspective to the privacy risks of each Project Team. The PMF should not meet on a weekly basis, but rather on a bi-sprintly basis, e.g. every two Sprints. The reason for this is to reduce the amount of overhead another discussion board can and will produce but, at the same time, to utilise the knowledge gained from multiple Scrum teams on privacy problems as best as possible. In an optimal case, the PMF will be an internal board of expertise on privacy protection in the field of the organisation with the core business goals in mind. The PMF requires a leader, in general this should be the company's Data Protection Officer. This leader serves as the main point of contact for relevant superior authority, e.g. law officials.

The responsibilities of the *Scrum Master* are extended to encompass potential organisational problems that may arise with the PrO. The Scrum Master needs to ensure that the development team can work according to the Scrum characteristics. The SM is required to assess the velocity in the Sprint. The velocity describes how many user stories have been completed successfully in a Sprint. If the velocity drops, the Scrum team is not working up to its potential, indicating that something needs to be addressed. One such issue could be that the team has overburdened itself with user stories in a Sprint or outside issues are influencing the team, thus leading to a drop in relative completed work. However, the SM will not receive additional, privacy-related competencies as the role of Scrum Master is more of organisational than technical nature.

Lastly, the *Development Team*, which is a self-organizing and cross-functioning team. This entails that the team itself defines how best to accomplish its work with no further direction from outside, as well as declaring that every team member is capable of performing any given task. Furthermore, the team needs to embody commitment, courage, focus, openness and respect towards each other. These attributes form the basis of a team capable of learning from each other while allowing each team member to grow individually. This explains the namesake of the agile method - the Scrum in Rugby - where each team member is capable of carrying their own weight but at the same time is co-dependent of every other team member. In the concept of privacy protection, the premise that every team member can develop any given user story as good as every other team member is very unlikely to be met. This utopian concept is something the development team should strive towards throughout the development life cycle by appropriate training procedures and cooperation within the team.

A major part of the development of high quality privacy user stories is cooperativeness between team members and the PrO, as these two roles will need

to work together closely on the implementation of privacy user stories. The better the communication and teamwork between these two entities, the better the resulting product will be. Their goal is to reduce the residual risk level of every identified risk as much as possible, without creating too much additional work. The close cooperation on designing the solutions for privacy user stories further enables the PrO to identify the necessary training a team member requires in addition to the general training performed by the entire development team.

5.3 Sprint

Figure 5.2 depicts the resulting model of the Privacy Protection Sprint, also shown in Appendix C for better analysis. It is an iterative spiral model beginning with the placement of user stories in the *Product Backlog*. The Product Owner and the *Privacy Officer*, introduced in subsection 5.2, prioritise the user stories depending on their relevance. The *Sprint Planning* is used by the Scrum team to choose the stories which will be developed in the next Sprint. The Privacy Officer is responsible for highlighting necessary privacy user stories, but the ultimate decision lies with the team. Subsequently, the selected stories are placed on the *Sprint Backlog*. The development period, the *Sprint*, generally consists of one to four weeks. At the end of the Sprint, each user story is reviewed in the *Sprint Review*. The privacy user stories need to be reviewed separately in the *Privacy Review*. Whenever a developed story does not meet the defined *acceptance criteria*, or for any reason could not be completed to the required level, and therefore fails the review, the story needs to be reformulated and placed back on the Product Backlog. If a story passes the review process it can be deployed and the next Sprint cycle begins.

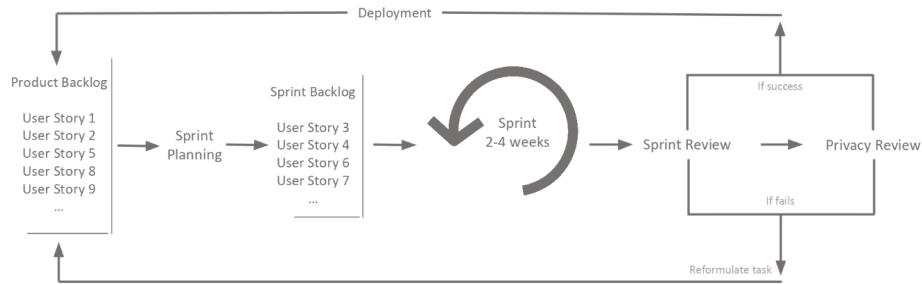


Figure 5.2: Privacy Protection Sprint

The Sprint describes a given time-frame for one development cycle. This time-frame is generally one to four weeks where all user stories on the Sprint Backlog are developed. At the beginning of each Sprint the user stories in the Sprint Backlog are distributed between the development team. The team identifies a common goal to be reached at the end of the Sprint to envision

the next increment of the working product. Consequently, each team member develops a solution to the user stories which have been allocated to them. If a user story is flagged as a Privacy User Story, the team member has to consult with the PrO on the design of the solution.

At the end of the Sprint, each user story is reviewed. If a user story meets the defined requirements it is accepted and can be deployed. If it is not accepted, the user story needs to be further refined and placed back on the Product Backlog for the upcoming Sprint. After each Sprint a working increment of the system including the newly accepted user stories is deployed. Hence, the Product Owner can discuss current solutions and progress with Stakeholders and the developed user stories can be evaluated in a quick and agile way. In addition to the general review, each PUS and each flagged user story need to be reviewed not just by the PO, but also by the PrO. If a PUS meets the defined requirements and its residual risk is deemed acceptable, the PUS can be deployed. If not, it needs to be reiterated upon in the next sprint.

There are multiple different options to include the focus on privacy protection during the development life cycle. One would be to dedicate an entire Sprint to privacy protection and scheduling it multiple times in the development life cycle. Another option would be to include continuous privacy protection through specific privacy user stories. In a privacy sensitive project, there is a clear need to include privacy protection as soon as possible because of the imminent risk of data manipulation from the beginning of a project's life cycle. If processes and skills such as identifying the scope of a project, secure coding best practices or adequate encryption methods are addressed too late in a project's development life cycle, an incredible amount of unnecessary reiteration over completed work is created. Hence, the Sprint-Zero is introduced to enable the PrO to adhere with and to perform the concept of Privacy by Design.

It is therefore of high importance to establish the scope of the project and the software product to be deduced as early as possible. Whilst the problem scope can be defined before the beginning of a project, the precise nature and range of data that needs to be handled during its lifetime might vary, but an initial scope can still be defined. Thus the author proposes one specific, privacy focused sprint ahead of all sprints, the Sprint-Zero, as well as continuous integration of privacy specific user stories into every sprint.

5.3.1 Sprint-Zero

The Sprint-Zero lies at the beginning of every project's development life cycle and is to be completed by the Scrum team as the very first Sprint. It enables the Product Owner and Privacy Officer to initialise the project with the development team and the Scrum Master. Together, the Scrum Team determines key aspects to ensure that every team member is aware of the privacy standards required for the project. This reduces unnecessary reiteration due to changing

specifications as well as creating the initial problem scope for the Scrum Team. The main goal of the Sprint-Zero is to provide the basis of *Privacy by Design* for the project. This ensures the project adheres by the key principles defined by the GDPR as identified in subsection 3.1.

To minimise the risk of completing the same work through changing specifications at multiple stages in a development life cycle, a variety of user stories are to be addressed in the Sprint-Zero. This entails answering top-level questions of the project scope such as what data types are required and how data is intended to be gathered. Additionally, potential security measures to ensure an adequate level of security need to be considered from a top-level view. At this early stage the Privacy Officer can specify security standards to be used during development. The proposed user stories to be completed in the Sprint-Zero are:

- Conduct initial Privacy Impact Assessment
- Determine coding standards
- Determine security standards
- Establish initial architecture design
- Determine necessary training programmes for developers

A key user story in the Sprint-Zero is the initial Privacy Impact Assessment. In comparison to the other stories in the Sprint-Zero, the initial PIA is a considerably large story. The completion of a PIA in the Sprint-Zero lets the PrO identify if and to what degree controlling or processing of personal data is intended by the project. This will determine the project's major privacy risks with a preliminary plan in order to reduce the residual risk level to an acceptable level. As part of the Sprint-Zero, another major process is to establish the standards to be followed by the developers during the development life cycle. This is in particular the case for the coding and security standards. Establishing the coding standards early on ensures that all developers follow one set of guidelines, resulting in a higher quality of developed code, better code reviews and an overall structure (Maruping, Zhang & Venkatesh, 2009). Establishing the security standards can vary depending on the project's grade of sensitivity, but establishing a core set of standards ensures that all developers abide to the same security mechanisms (Jones & Rastogi, 2004).

Establishing the initial architectural design depicting on how the project could be structured is of great benefit towards security, as well as distinguishing dependencies and delivering an overview on the project's technical design. The Sprint-Zero is also used to establish the skill-level of every developer and to perform an initial plan on the first set of training programmes each developer needs to undertake. This will vary greatly depending on the respective area of expertise of each developer as well as on the scope of the project. The initial training does not suffice to ensure expert developers, but rather ensures that

the basic required skills for the respective project are fulfilled. Further training programmes will be required at regular intervals throughout the development life cycle.

Completing the Sprint-Zero will create a clear picture for the Scrum Team of the project itself, whilst at the same time identifying the core necessities of data handling and personal data protection. This will empower the Scrum Team to understand what kind of work is expected and to prepare optimally for it and ensures the project performs *Privacy by Design*.

5.3.2 Artefacts

The Scrum Methodology defines three Scrum Artefacts: the *Product Backlog* (PB), the *Sprint Backlog* (SB), and the *Product Increment*. These three artefacts share the same goal of visualising the project's work. The *Product Backlog* consists of all user stories defined by the Product Owner and the Privacy Officer. They will need to be completed at some stage in the future by the development team. The *Sprint Backlog* consists of the selected user stories for the upcoming Sprint, e.g. within the following one to four weeks. These user stories have been selected from the PB by the development team. The *Product Increment* is situated at the end of the Sprint. It describes the combination of the completed work of the last Sprint as well as all previously deployed work, therefore describing the next version of the delivered Product. All three artefacts visualise the work and value that is brought to the product by the team.

The PB and SB need to be filled with privacy-specific user stories besides the usual user stories. The PO, who is generally in charge of filling the PB with user stories, is aided in this task by the PrO who prepares and adds privacy specific user stories. The responsibility of prioritising privacy specific user stories for the SB lies with the PrO. Therefore, it is absolutely necessary for the Privacy Officer and the Product Owner to work closely together.

5.4 Ceremonies

In Scrum there exists a handful of critical meetings grouped in the concept of *Scrum Ceremonies*. The critical meetings are labelled *Sprint Planning*, *Daily Scrum*, *Sprint Review* and *Sprint Retrospective*. These meetings cover the scope of planning, discussing and reviewing phases to enable a Scrum Team to plan ahead as well as look back on recent work to learn and improve. The inclusion of privacy protection in the agile software development environment requires the Privacy Officer to be included in most of the tasks that are generally performed solely by the Product Owner. In most cases the PO will not have the required specialised knowledge on privacy protection to incorporate the skills of the Privacy Officer.

A Sprint can only begin, if the Sprint Backlog has been filled with user stories. These user stories need to be allocated to team members and evaluated

regarding the amount of time required to develop each of them. These activities are the focus of the *Sprint Planning*. It requires attendance of the Product Owner, Privacy Officer, the Scrum Master and the Team. In this meeting user stories are selected for the next Sprint and thus moved from the Product Backlog to the Sprint Backlog. Depending on the velocity of the team, the amount of user stories for the Sprint Backlog is determined. Further, the user stories are estimated and additional specifications are discussed. The goal is to create a clear picture of the upcoming work for the team members. The Sprint Planning creates an open platform for the team members to ensure that each user story is interpreted correctly as all team members have the same understanding of the tasks. This supplies the PrO with an important setting to discuss the level of privacy protection required to meet the success criteria of the privacy user stories and that the requirements are clearly understood by the team members. The PO represents the client in this setting, as they are the intermediary between client and development team.

The Scrum Framework defines the development team as cross-functional. This means that the team has to be capable of solving any problem on its own without any help from outside. Furthermore, it describes the utopian view of every team member being able to handle any given problem. This is a goal to strive for by providing the required training to the development team. However, until that level of cross-functionality is acquired by the team, the team must decide which team member is most capable to provide the privacy user story with the required acceptable level of privacy. The atmosphere within the Sprint Planning needs to be open and honest enough for a team member to concede a lack of skill if they are not capable of performing a given privacy user story. The discussion on what the general goal of the upcoming Sprint is should lead to an overall consensus on the work to be done by the team. During the Sprint, the Scrum Team meets in the *Daily Scrum*, a stand-up meeting, to shortly discuss their tasks for the day. This can be used by the PrO to have a quick check-up on potential issues the team members are facing during development of privacy user stories. The PrO is the Risk Owner of every privacy user story and should therefore be involved closely in the technical development of the user stories. Team members can use this short meeting to call for additional guidance in a separate, one-on-one meeting with the PrO.

The *Sprint Review* takes place at the end of each Sprint. This meeting is required to inspect the developed solutions of each user story and evaluate if these solutions meet the previously defined acceptance criteria. The main purpose for the team is to present their developed user stories in front of the PO, PrO and the SM. Stakeholders may also join this meeting. This is where each user story is reviewed and its value to the product increment is analysed. This is also the meeting where the PO will realise if the workload set for the team in the Sprint Planning was correctly judged. On the one hand, there can be the case that too many user stories were placed on the Sprint Backlog. One indicator for an overburdened development team is that user stories are not

finished whereas another indicator is that developed user stories do not meet the defined acceptance criteria or quality requirements. On the other hand, there's the case of developing too few user stories in the Sprint. While this case is probably rare it is identifiable through open communication within the team.

The *Privacy Review*, as proposed by the author, is used specifically for reviewing privacy user stories. This meeting is used to focus on every developed privacy user story and, above all, to evaluate the residual risk level of the developed solution to the user story. With the PrO as Risk Owner, this review meeting is the key procedure to ensure that if the PrO accepts a privacy user story, it meets the required level of privacy protection. Additionally, a privacy user story could be flagged as *severe*. This entails that the story has an effect on previously developed solutions and risk evaluations. If a privacy user story is flagged as severe, the Privacy Review meeting is used to determine the necessary follow-up user stories.

The *Sprint Retrospective* is a meeting used to analyse the past Sprint, to adapt new learnings and overcome previous challenges. This is a very important meeting to create a continuous learning curve in the Scrum Team. It actually gives the Scrum Team the opportunity to identify possible issues of the last Sprint and create a plan on how to overcome them. Focusing on privacy protection, this is a suitable meeting to establish the actual skills of the team members and to create a training plan for each developer based on this evaluation. It is of high importance to use the Sprint Retrospective to define specific privacy user stories focused on the identified problems and to place them on the next Sprint Backlog. This ensures that there are specific learnings and measures taken from the Sprint Retrospective.

5.5 Control Points

To ensure a privacy protected development life cycle, there need to be control procedures set in place which can identify possible problems or shortcomings to the desired quality and state of the project. In general, a control point can be a person qualified to adequately assess the given situation, can take the form of documents or implemented automatic testing procedures. Such control points need to be placed at different stages of the development life cycle and from various point of views to enable an encompassing measurement and review system. Such Privacy Control Points during the development life cycle create a high level of trust towards the developed project for every stakeholder. The major issue of introducing privacy control points to an agile development method is to keep the additional work of these measures as little as possible. The goal is to find a balance between inducing control measures focused on ensuring a privacy protected project and enabling rapid production cycles. Therefore, each of the following control measures are set in perspective with this goal in mind.

The Product Owner plays the most crucial part in Scrum, as the PO has sole responsibility towards the delivered product. The PO defines the user stories,

prioritises the work to be done in the Planning phase of each Sprint as well as reviews every user story once completed at the end of the cycle. The PO is rarely involved with the actual implementation during the Sprint itself and often lacks any training regarding privacy. These characteristics deem the PO to not be, in most cases at least, an adequate assessor of privacy user stories. The general Scrum Control Points where the PO poses as a control mechanism are *Sprint Planning* and *Sprint Review*. However, the level of control through the PO can, due to a lack of knowledge, rarely meet a high standard for privacy sensitive topics.

Thus, the introduction of the Privacy Officer in subsection 5.2, who has the required knowledge and skills as well as close interaction with the development team on the implementation of privacy features, elevates the general Scrum Control points to a higher standard. Furthermore, the author proposes the following additional privacy control points for this Framework:

- Privacy Planning
- Daily Scrum
- Privacy Review
- Privacy Protection Checklist
- Privacy Management Forum

5.5.1 Scrum Control Points

In this section the existing Scrum Control Points are shortly described.

The *Planning* meeting, i.e selecting which user stories are placed on the Sprint Backlog, is the first control point in the standard Scrum Framework. Here, the Product Owner meets with the Scrum Master and the development team to discuss the problem scope of every user story. This meeting allows the team members to clarify any uncertainties regarding the story. The issue, that the PO lacks the required privacy knowledge, will often result in the wrong definition of user stories. Reflecting back on the purpose of a user story, as discussed in more detail in subsection 5.6, it should describe one specific feature required by a certain user as precise as possible. With the PO in charge of defining every user story, their definition in regards to privacy-specific stories will be too broad and the assessment of the required time for each user story will very likely be wrong. Furthermore, every user story includes the acceptance criteria which deem a user story completed at the end of the Sprint. The developers use the acceptance criteria as a goal which needs to be reached through their work. Therefore, the acceptance criteria needs to be declared in detail, with measurable or at least assessable metrics. The appropriate definition of acceptance criteria regarding privacy topics are extremely difficult for a PO without any training or knowledge in that field. Hence, the POs lack of privacy skills becomes most evident in the definition of privacy user stories.

The *Review* meeting is the second control point in the Scrum Framework. This control point is situated at the end of the Sprint where every user story, e.g. the completed work, is analysed. The analysis depends on the defined acceptance criteria of every user story. The team members present their developed solution to the PO and stakeholders. If the acceptance criteria are met the user story is accepted and can be deployed in the next Product Increment. If the acceptance criteria are not met the user story needs to be reassessed and placed back on the Product Backlog. Because the acceptance criteria play such a vital role in evaluating the work done by the team, they have to be unambiguous and preferably answerable with a yes or no.

In a general project scope, where the project does not touch upon personal data of individuals of the European Union, these control points are appropriate. However, in a privacy sensitive project, the PO does, in most cases, not have the required skill set to pose as an adequate control mechanism. The level of privacy would be too dependant on the skill set of the individual team member and their proposed solution without adding the additional level of security through an assessor. Therefore, it is necessary to create additional privacy control points. These take the form of privacy protection documents, as guidance during the Planning and Review phase as well as employing the knowledge of the Privacy Officer as a quality-assessor.

5.5.2 Privacy Control Points

Agile software development methods aim to reduce documentation-heavy procedures such as control points to the bare minimum. With the goal of creating a Privacy Protection Framework, there is a need to focus on lightweight procedures which introduce added privacy measures without hindering the quick release cycle of product increments. In order to remain as agile as possible, the author proposes to adapt the general Scrum control points, the Planning and Review stages, as well as to include a Privacy Protection Checklist with the relevant information of the respective project. The key role throughout these control measures is the Privacy Officer, not only because of their knowledge, but also because the Privacy Officer is the Risk Owner of every identified privacy risk, as further discussed in subsection 5.7.

Privacy Planning The standard Planning meeting of the Scrum Framework can be adapted towards incorporating the Privacy Officer in a *Privacy Planning* meeting. This process includes the required privacy knowledge and skills during the Sprint Planning to ensure that privacy user stories are defined accurately in a way that enables the team members to develop a solution of high quality. As the Privacy Officer creates the privacy user stories, these will subsequently be precisely defined with appropriate acceptance criteria. The PrO has to perform a privacy risk assessment for every identified privacy risk. The respective process is proposed in subsubsection 5.7.2. This is essential to identify the risk level of

the privacy user story and thus establish which level of protection measures are required.

Daily Scrum The daily meeting by the Scrum Team is called the *Daily Scrum*. This is an additional privacy control point as due to the inclusion of the Privacy Officer in this daily stand-up meeting, the actual Risk Owner of a privacy user story participates in the daily discussion. The meeting requires the team members to describe their current progress as short and precise as possible. Especially to aid the implementation of privacy user stories, this enables the team members to discuss privacy protection issues on the granular implementation level.

Privacy Review The standard Review meeting of the Scrum Framework at the end of each Sprint, i.e. where every user story developed in that Sprint is reviewed, is adopted for the reviewing of privacy user stories, e.g. user stories either initially classified as privacy user stories or flagged as privacy-related during the Sprint. The *Privacy Review* as proposed by the author is performed by the Privacy Officer in cooperation with the PO. This quality control measure is the most important one as it is an in-depth review of the implemented solution to the privacy user stories. During the review process, the PrO needs to evaluate the remaining residual risk of each privacy user story. The PrO also needs to evaluate the quality of the developed solution and decide if the risk is handled to an acceptable level or not. The Privacy Officer needs to evaluate the initially performed Risk Assessment with the developed solution and thus establish the residual risk level of the privacy user story. If the residual risk is deemed too great, the user story needs to be reassessed by placing it back onto the Product Backlog. Furthermore, the requirements of the user story will need to be addressed and adapted towards a higher level of security as well as redefining the acceptance criteria. If the residual risk is mitigated or deemed acceptable, the user story is accepted and can be deployed in the next Product Increment.

Privacy Protection Checklist The privacy review uses the acceptance criteria of the privacy user story as a target for the proposed solution, but including every single requirement would not be feasible. Therefore, the Privacy Officer requires a go-to document with important criteria that need to be implemented by each privacy user story. This control document is the Privacy Protection Checklist. The document is designed in the form of a checklist for the Privacy Officer to quickly assess if a developed privacy user story meets the defined criteria, based upon the data process analysis of CNIL (2018a). These criteria are focused on assuring compliance with the GDPR.

A generic Privacy Protection Checklist is proposed in Table 5.1 to aid the Privacy Officer in the assessment of a privacy user story in the *Privacy Review*. However, this checklist should be adapted and refined depending on the problem

Question	Compliance Check
Does the story collect the minimal data required?	
Is the gathered data sanitized?	
Is there a procedure in place to keep data accurate?	
Is the storage period limited?	
Are adequate security measures taken?	
Is the purpose of this data use communicated to the user?	

Table 5.1: Privacy Protection Checklist

scope of the particular project. Furthermore, the checklist should be extended to include topics such as coding practices determined by the Scrum Team during the Sprint-Zero as well as security-related questions that need to be implemented by every user story dependant on the respective project. Critical questions include whether the developed solution to a privacy user story only uses minimal data required to perform its purpose. Additionally, data sanitization describes the process of removing unnecessary data. In this context, the focus lies on processes ensuring individuals cannot be identified directly through analysis of the collected data. The respective grade of sanitization varies depending on the area of the project. Further, the data has to be kept as accurate as possible and there need to be adequate security measures in place such as the required encryption. Ultimately, the purpose of data gathering needs to be clearly communicated to the data subjects. Every statement needs to be deemed as completed for the privacy user story to be accepted. This process aids in complying with the principle of *Accountability* as determined by the GDPR towards all data processes and controlling throughout the development life cycle.

Privacy Management Forum The Privacy Management Forum, as introduced in subsection 5.2, is used as an escalation platform for severe cases. Such severe cases describe the situation where an identified privacy user story has an impact not just on the project itself, but has consequences for multiple projects or even the entire company. In these severe cases the Privacy Management Forum meets to discuss the currently developed solution of the user story and, more importantly, the risk it poses to the company. Especially in larger organisations where there can be multiple Scrum teams working towards the same product, the PMF is a vital discussion board. The proposed Privacy Control Points cover a great variety of detail, ranging from the granular focus of the Daily Scrum to the very high-level focus of the Privacy Management Forum.

Figure 5.3 shows the abstraction level of the different control points, taken and adapted from Ahola et al (2014). This showcases how granular each control measure assesses the risk. The closest inspection on development-level is performed in the *Daily Scrum*, where the Privacy Officer can be directly addressed by the developers. The functionality and how the developer implements the story is assessed in the *Privacy Planning and Review* meetings, where the Privacy Officer uses the aid of the *Privacy Protection Checklist* to ensure all

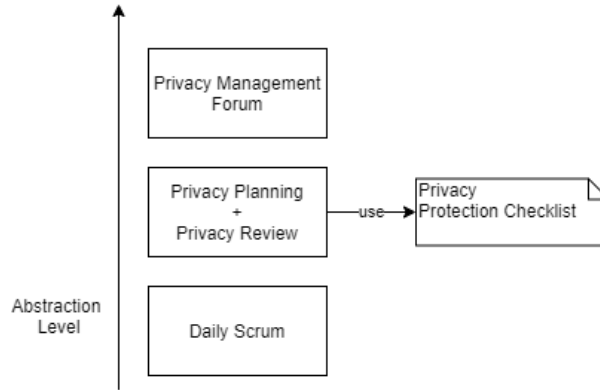


Figure 5.3: Abstraction Level of Control Points

relevant areas are covered. The top-level evaluation in severe cases is performed in *Privacy Management Forum*, albeit this evaluation will only happen rarely.

5.6 Privacy User Stories

The Scrum Framework uses the concept of a *user story* to describe the work that needs to be done. It is designed from the point of view of one specific user and describes the desired feature of a user as well as the value this feature provides to the product. This allows a variety of different features to be described, as the user's point of view does not necessarily have to be the end user, but could also be the Privacy Officer or e.g. a business analyst. Further, a user story needs to describe the goal of that feature and a brief description. Additionally, the acceptance criteria need to be defined. This results in the basic design of a user story as in Table 5.2.

As a	User
I want	a specific feature
So that	additional value is created
Acceptance criteria	Criteria 1 Criteria 2 Criteria ...

Table 5.2: User Story Template

The time required to develop a user story should be less than the Sprint cycle, as every user story should be completable within one Sprint. However, the concept of the user story can be used to describe features of varying detail and scope. A user story that is of general nature is described as an epic. An epic encompasses multiple smaller user stories that all need to be completed in

order to add the value described by the epic to the product. In order for a user story to be completed within the short time frame of a Sprint, it needs to be appropriately short and focused. The user story itself can then again be broken down into a few smaller development tasks, which is done by a developer during the Sprint. This makes the developer responsible and an active part in designing the problem solution, as they themselves decide how exactly to develop a user story in order to meet the acceptance criteria. Thus, the process of designing the solution to a user story creates a high level of responsibility and self-reliance for the developer.

The concept of epics which describes the different security requirements for a system as proposed in SDL-agile (Ahola et al., 2014) is a great addition in the context of privacy protection because the requirements by law are of a much broader nature than the standard user story should be. Defining privacy themes as epics, which subsequently will be broken down into user stories, can help the Privacy Officer to ensure that different areas of privacy protection are handled. An example of an epic in the format of a user story is shown in Table 5.3, where the theme of the epic is ensuring all data processes are compliant with the GDPR. This epic is obviously too great to be completed within one Sprint, but it can be divided into smaller user stories which can be completed within a Sprint.

As a	Privacy Officer
I want	to be accountable for all data processes
So that	the project can show all performed actions on personal data
Acceptance criteria	Audit logs are created Monitoring systems are implemented Privacy risk assessment is documented User life cycle model is created

Table 5.3: Exemplary Epic

The user story needs to be sufficiently defined for the developer to create the desired feature and, for an effective Sprint Review process, that acceptance criteria can be answered with a simple "yes" or "no". Further, designing acceptance criteria of a boolean nature ensures that there are no false interpretations of the task at hand. Imprecise defined acceptance criteria lead to additional workload and possibly to delays in product completion. Subsequently, the user story needs to be described to an extent, so that a developer can understand the requirements and create the desired feature. As pointed out, it is of high importance that the acceptance criteria are answerable with a simple "yes" or "no". This allows for concise assessment in the Sprint Review. If the acceptance criteria are not precise it will lead to discussions in the review phase and possible rescheduling of the user story back to the Product Backlog for another iteration, resulting in time wasted.

Shortly recapping the discussed roles and tasks: the Product Owner defines the general user stories based on the requirements of the customer. The PO does not have the required knowledge to create privacy-specific user stories which is why the Privacy Officer is responsible for this task. All user stories are placed on the Product Backlog and prioritised. Subsequently, the Scrum Team selects the user stories to be placed on the Sprint Backlog for development in the upcoming Sprint. In the Sprint planning phase each user story is allocated to a team member who becomes the story owner. The story owner implements a solution on their own with the goal of meeting the user story's acceptance criteria. Besides planning which team member is responsible for which user story, the planning phase is used to determine how much time a user story requires. This direct allocation to team members and assessment of user stories enables the Product Owner to see how the workload is distributed and what the current velocity is.

Creating privacy-specific user stories enables high visibility of privacy-related tasks and allocates the required development time and focus to them. The author proposes a few changes to the general layout of user stories to include the required information in privacy sensitive stories. For one part, a Privacy User Story needs to be identifiable as privacy-related. Therefore, it needs to include a privacy flag with a simple *true* or *false* identifier. The flag needs to be set when user stories are created by the Privacy Officer. If a developer realises a general user story touches upon privacy issues after beginning to develop the user story, this flag needs to be set to *true* straight away. Every user story that is flagged as a privacy user story needs to be reviewed separately in the *Privacy Review* introduced in subsection 5.5.2.

Another aspect is evaluating how big the impact of a privacy user story is on the previously developed product. This can also be indicated by adding a flag for severity, aimed at identifying privacy user stories with a severe impact on the existing product. Such is the case if the privacy user story has an effect on previously developed user stories, e.g. if an encryption standard changes. This could also be the case if the scope of the product changes in any way, e.g. if due to a change in nature of the product, it is now required to gather sensitive medical information. Such a user story entirely changes the risk evaluation and acceptance criteria of previously completed user stories. This is why the author proposes to add a severity flag to privacy user stories, in order to establish a way of identifying critical user stories which are not merely another feature but have a large impact on the existing product. The flag itself can be of boolean nature, but it requires a free form text field, as the impact of a user story can vary greatly. This requires the PrO to establish if a privacy user story is of severe nature and further describe what the consequences to the existing product are. At this stage, it is not required to define the potentially newly created epic, but rather to have a top-level description of what this privacy user story entails for the product. This additional information is included in the acceptance criteria of the severe privacy user story to ensure that the additional required work remains

visible and will not be pushed away. However, the overarching Privacy Impact Assessment, conducted for the first time in the Sprint-Zero, does not only need to be re-evaluated in case of a severe privacy user story, but also needs to be re-evaluated after a critical amount of privacy user stories have been completed. The resulting design of the proposed template of a privacy user story is shown in Table 5.4.

As a	User
I want	Feature
So that	Use Case
Acceptance criteria	Criteria 1 Criteria 2 Criteria X
Privacy Flag	Yes / No
If yes	Short description
Severe Impact	Yes / No
If yes	Short description

Table 5.4: Privacy User Story Template

The reviewed literature in section 3 identified that major Secure Agile Software Development Methods tend to group security user stories depending on their nature or, in some cases, more strictly by the frequency they need to be completed at. Strict grouping on the frequency of a story would result in many user stories to be completed in every Sprint. This directly contradicts the core concept of agile methods as it would create a high amount of work to be done all the time, therefore greatly hindering the short feedback cycles of the agile concept. However, there will be some privacy user stories, especially those focusing on testing, which are required in every sprint to ensure that the product increment will reach the necessary level of quality and security. Therefore, the author proposes to adopt the concept proposed by the Finnish conglomerate (Ahola et al., 2014), which groups the user stories regarding their functionality. This results in some similarities to the frequency concept but allows for a much clearer semantic understanding of which privacy aspects are being tackled as well as more fluidity in story selection. Each group of user stories has a tendency of when and how often they need to be completed, but there is no distinct value for each group. This tendency will be further discussed in the respective sections of privacy user story types. The proposed semantic groups of Privacy User Stories are:

- Functional Features
- Engineering Tasks
- Ways of Working

Functional Features are a crucial part of every product as this group describes all those privacy user stories which add a distinct function to the product. Another important part, and arguably a more essential one in the privacy context, is the group of *Engineering Tasks*. This group describes the privacy user stories that can be "done" but, rather than creating an additional feature, focus on testing and hardening the existing product and previously developed features. The third important group, *Ways of Working*, describes the group of privacy user stories that do not have a specific result such as a document or feature, but rather focus on establishing standards and ensuring the team members receive adequate training in the required fields, such as best coding practices in a privacy-sensitive environment.

Group Name	Group Description	Exemplary User Story
Functional Features	Adds feature to the product	Enable newsletter registration
Engineering Tasks	Testing and hardening the product	Test double opt-in process
Ways of Working	Create relevant documentation and train developers	Train team members on opt-in Requirements as per GDPR

Table 5.5: Semantic Groups of Privacy User Stories

Table 5.5 shows the different groups of privacy user stories with a short description and the title of an exemplary user story. In order to highlight the different kinds of value added to the product by the different types of user stories, Table 5.5 showcases a project that wants to add the feature of *Newsletter registration* to the product. Adding the functionality to the system is a privacy user story belonging to the *Functional Features*. It is privacy-related because it requires information about the user, in a minimal variant at least the e-mail-address. An *Engineering Task* would be to test all implemented processes that require a double opt-in process on a regular basis. For the team members to be able to develop any of these tasks, they need to possess the skills and knowledge on the double opt-in procedure as defined by the GDPR. Therefore, a *Ways of working* privacy user story, focusing on delivering the required training to the developers, is required. This scenario already highlights the importance of *Ways of working*, as without the required skills and knowledge of the team members, *Functional Features* and *Engineering Tasks* will rarely reach defined privacy requirements.

The semantic grouping of privacy user stories creates a simple overview on the included stories and thus enables the PrO to evenly select the stories from each semantic group for development. Furthermore, this semantic grouping creates awareness for the different kinds of user stories and their importance towards creating a holistic privacy protected product. Through the separation of stories based on their nature, the Privacy Officer also gains a direct overview of all developed stories and their semantic group, which facilitates prioritisation of stories from one semantic group. For example, if the development team consists of expert developers in the context of privacy, the Privacy Officer can place a focus on delivering more *Functional Features*. However, if the team

members do not have any knowledge regarding e.g. secure coding, the PrO can select a focus on user stories from *Ways of Working* to ensure that all team members reach the required skill level. This is especially necessary in an agile development method, as the team members receive a high level of trust towards performing the work. It is very important for the Privacy Officer to ensure a balance between the developed types of privacy user stories. With the general idea of agile development methods to be able to deliver additional value to the product as quickly as possible, the PrO will need to rely on adequate privacy risk assessment, see subsubsection 5.7.2, to further visualise and push the business need for *Engineering Tasks* and *Ways of Working*.

5.6.1 Functional Features

The user stories classified as *Functional Features* describe the group of user stories which provide an actual feature to the product. Their nature lies in providing the user with a certain usable function. This group of user stories is very obvious in every development life cycle as the stories provide a distinct new feature to the product.

This entails that an assessment, whether a *Functional Feature* is implemented or not, can be easily completed by testing the function itself. Thus, this section of functional user stories can be seen as developing a new privacy-related feature for the product such as prompting a user for their consent.

Functional Features
Notify users about data use
Accessible privacy policy
Double opt-in procedure
Intrusion detection
Monitoring
Access control implementation
Log data changes
Log data access

Table 5.6: Examples of Functional Feature User Stories

Table 5.6 shows an exemplary list of privacy user stories belonging to the group of Functional Features. These differ greatly depending on the type of project, as every Functional Feature dealing with personal data has to meet the designed privacy standards. The first three features are examples of direct features which the end user will notice for those projects with direct contact to the end user through, for example, a website. The end user needs to be actively notified about how their personal data is used. Further, this information needs to be easily accessible for the end user. In order to use personal data, e.g. for marketing purposes by sending regular e-mails to end users, there needs to be a *double opt-in* process set in place. This describes, additionally to accepting

to the usage through a check box on a website, that a user has to also respond positively to a second check regarding the data processing.

The next few Functional Features describe internal features such as designing an Intrusion detection system, which signals if there is an unwanted visitor in the network. Implementing access control is also a functional feature, albeit internally, as it is a task which needs to be completed once and provides a direct feature to the developers.

Table 5.7 shows an example for a privacy user story from the group of Functional Features. It describes that a user wants to be able to withdraw the consent of processing their personal data. This functionality is a requirement by the GDPR and therefore of high importance for any data processor or controller. The privacy flag is set to *yes* as it deals with personal data of individuals. The next field includes the short description on why this user story is a privacy user story. Furthermore, this privacy user story is of severe nature due to its effect on the database structure and thus, an effect on data handling for other data processes as well.

As a	User
I want	to be able to withdraw consent via e-mail to processing my personal data
So that	I can have control over the processing of my personal data
Acceptance criteria	Opt-out option in every user e-mail Activating opt-out stops project using user data
Privacy Flag	Yes
If yes	Handles personal data and gives the user the function to remove consent of data processing
Severe Impact	Yes
If yes	Changes to the database structure to allow removal of consent required

Table 5.7: Exemplary Functional Feature "Withdraw Consent"

There is a distinction to be made between implementing a functional feature and testing whether it is securely implemented. The latter type of work falls into the next category.

5.6.2 Engineering Tasks

Engineering Tasks describe the semantic group of privacy user stories that deal with privacy-related tasks which do not provide a specific functionality to the product, but can have a *definition of done* as per the Scrum framework. This includes a variety of hardening and testing processes. Some of these can and should be automated, especially of the testing domain, but some recurring tasks cannot.

Engineering Tasks
Architectural threat analysis
Interface robustness
Third party privacy assessment
Availability
Cryptographic key management
Account lifecycle
Protection of log events
Malware protection
Create test cases for user input
Protect system against DDoS attack

Table 5.8: Examples of Engineering Task User Stories

Table 5.8 showcases some examples of privacy user stories belonging to the semantic group of Engineering Tasks. Many user stories in this category deal with assessing and testing the developed product. This includes performing robustness tests of interfaces as well as performing architectural threat analysis. Although there is a high focus on testing the implemented features regarding functionality and security, other tasks also fall into this category. One example of these tasks is the assurance of availability of the product to its users.

As a	Privacy Officer
I want	to have the audit logs on personal data protected
So that	no unauthorized access is possible
Acceptance criteria	1) Audit logs are encrypted by using PKCS 12 2) Audit logs are signed to ensure changes are traceable
Privacy Flag	Yes
If yes	Audit log showcases changes to personal data which need to be protected
Severe Impact	No
If yes	n/a

Table 5.9: Exemplary Engineering Task "Protect Audit Logs"

Table 5.9 shows a privacy user story which focuses on the protection of audit logs. It is a story from the perspective of the Privacy Officer, who wants to ensure that the audit logs on personal data are protected. This requires the encryption and signature of the audit logs to only allow access to authorized users. The privacy flag is set to "Yes", but as this story does not entail other changes the severity flag is set to "No".

5.6.3 Ways of Working

This group describes privacy user stories which are more in the vicinity of recurring tasks but do not have an explicit definition of done, as they need to be updated or refreshed every once in a while throughout the development life cycle. A major part of this group is training team members to gain skills in

specific domains, especially in the privacy-sensitive domain. These activities are critical in agile development, as there lies an immense amount of trust on the team members to be able to perform tasks as defined in the acceptance criteria. Another big part is creating necessary documentation, such as a list of the current risks to the product, and frequently updating these. This also includes choices made on security design measures, such as the level of encryption used.

Ways of Working
Security Guidelines
Update Privacy Impact Assessment
Consent specific training
Coding guidelines
Update Top 5 Risk model
User data model
Access control policy
Privacy policy
Perform critical updates
Track third party changes/updates
Ensure secure coding training

Table 5.10: Example User Stories of Ways of Working

Table 5.10 shows exemplary privacy user stories of the semantic group *Ways of Working*. These include defining and updating important guidelines in the field of security and coding, providing specific training in a special field for the developers as well as regularly updating important models such as the top five privacy risks.

As a	developer
I want to	learn data anonymisation
So that	I can develop functionality to ensure individuals cannot be directly identified through the stored data
Acceptance criteria	I have successfully taken part in the training on data anonymisation
Privacy Flag	Yes
If yes	Training to be able to perform data anonymisation on personal data
Severity Flag	No
If yes	n/a

Table 5.11: Exemplary Way of Working "Training Data Anonymisation"

The User Story showcased in Table 5.11 describes the training for a team member in the topic of information sanitization. This describes cleaning the gathered data so that it cannot be re-translated to the original individual it belongs to.

5.7 Agile Privacy Impact Assessment

The goal of this section is to establish how agile projects can adhere to the GDPR by performing Privacy Impact Assessments during the development life cycle without creating too much overhead. The GDPR defines that a project needs to perform a PIA at the beginning of a project without a clear definition on when and how to reiterate or adapt the initially performed PIA during the project's life cycle. Completing the same PIA again at a later stage of the life cycle creates a large, documentation-heavy, task without properly including the previous completed work. The comprehensive PIA as proposed by CNIL (CNIL, 2018b) fits well into the traditional waterfall method of software development. The four stages of this method, i.e analysis, design, implementation, and testing, form a systematic and gated approach. It allows for thorough quality assurance before software is launched and is therefore a great match to the development stages of the waterfall method.

The PIA as proposed by the OAIC (OAIC, 2014) is, as identified in section 3, more in the vicinity of multiple checklists and especially heavy on documentation regarding the three final steps of reporting. A possible adaptation of the ten steps would be grouping them into three phases depending on the content they focus on:

1. Planning
2. Privacy Impact
3. Review

The author proposes to combine these two approaches into an agile PIA, ensuring a lightweight assessment of Privacy Risks. This seems to be very suitable for the Scrum Framework, where there is a Planning phase, the Sprint and a Sprint Review. Thus the templates and process guides of CNIL and OAIC are adopted to suit an agile method, in this case Scrum. The focus lies on reducing the heavy documentation whilst ensuring high visibility and co-operation within the team to enable a holistic view on all identified risks.

For the agile PIA, the author proposes an iterative model which needs to be initialised in the Sprint-Zero. This satisfies the legal requirement of performing a PIA whilst being in the unique position at the beginning of the project where clear and structured analysis reduces unnecessary workload. This allows the Privacy Officer to establish, if the project touches upon personal data, to identify the core processes and activities which are necessary for the desired, currently planned, version of the product. Consequently, this early identification of personal data to be handled in the future allows the early definition of necessary privacy risk mitigation. It needs to be performed by the Privacy Officer and the Product Owner together: the PO adds vital knowledge of the planned product to the assessment, whilst the PrO has the necessary privacy protection

and legal knowledge required to assess the information processes appropriately. The inclusion of the development team during selected stages of the initial PIA helps the assessment as the developers have the necessary implementation skills to assess possible solutions to proposed protection measures.

5.7.1 Initial Privacy Impact Assessment

The initial PIA is an assessment prior to all development work. By setting the stage through identifying, analysing and validating potential privacy risks that may occur during the development life cycle, the Privacy Officer and Product Owner prepare themselves in an optimal way for the privacy challenges to come. The author also proposes an initial PIA with the following three sections:

1. Identify
2. Analyse
3. Review

The first phase, *Identify*, concentrates on setting the focus on privacy processes and identifying preliminary plans of using personal data of individuals. The second phase, *Analyse*, discusses more in-depth the types of data and gathering processes and therefore already needs to handle possible safety measures. The third phase, *Review*, is the formal acceptance phase of the planned safety procedures for the identified handling of personal data.

The initial PIA is used as a document in the continuous improvement and re-evaluation environment and needs to be updated when necessary. This necessity occurs in two scenarios. The first scenario takes place when a privacy user story is flagged as a *severe* user story. This requires the immediate re-evaluation of the PIA, as the *severe* flag indicates a high impact on the existing system, entailing that changes have to be made to previous evaluations. The second scenario describes the ongoing development without the occurrence of severe privacy user stories. The author proposes to update the PIA at least once for every three deployed privacy user stories. This ensures a regular iteration on the initial PIA without enforcing too much added documentation and additional work.

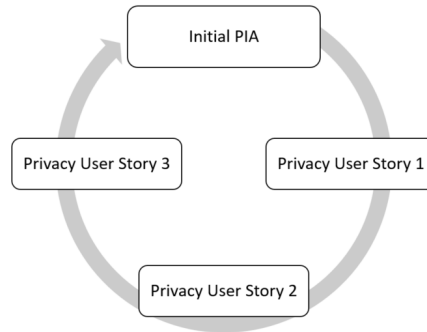


Figure 5.4: Iterative Privacy Impact Assessment

Figure 5.4 demonstrates the iterative nature of the PIA by highlighting the process when the initial Privacy Impact Assessment needs to be updated.

Identify The initial PIA begins with the *Identify* phase. This phase needs to assess the project’s context regarding high-level aspects of data. In short, the first phase is used to deal with the following questions:

1. Is a Privacy Impact Assessment necessary?
2. Which laws have to be complied with?
3. Which processing of data is planned?
4. Which entities collect and/or process personal data?

The first question of whether a PIA is even necessary needs to be answered as a starting point. If the project does not collect or process personal data in any way, this can be answered with a no. Albeit this may seem unnecessary, it is required by the GDPR, and thus by law, to document that a PIA was performed by a project and thus showcase, that there has been a proactive project assessment of whether personal data of individuals are handled. The case of a project not processing or controlling any personal data leads to the direct end of the Privacy Impact Assessment. The Privacy Officer, or the Product Owner as in this case there is no actual need for a PrO, needs to preserve the documentation of the result. However, if a Privacy Impact Assessment is necessary, the initial PIA needs to be continued. The next step is to identify the applicable laws. The main focus for this thesis lies on the European Union as legislator, but in times of internationalisation and globalisation it is very likely that a project touches upon personal data of individuals from multiple different legislations. For the scope of this thesis the adherence to the GDPR and the laws defined by the EU are the main focus, yet an enhancement to include other legislations remains possible. Necessary adherence to other legislations would be identified at this stage of the initial PIA, as they would have an effect on

privacy risk evaluation in later stages of the PIA. Returning to the focus of this thesis, the next question, after identifying the applicable laws, is to establish what kind of processing or controlling of data is planned.

The last question in this section deals with the entities who collect the data. These need to be identified in order to ensure that all entities with access to personal data of individuals are highlighted. For example the project team performing this PIA might not be the only team handling the gathered personal data. There might be the case that services of other companies, such as server capacities or data analysis, will be incorporated in the project. Identifying the necessities and dependencies on other companies' services helps ensuring that all entities adhere to the GDPR.

Analyse The second phase, the *Analysis* phase, goes in-depth into what kind of data is planned to be processed and/or controlled. It is the main part of the PIA and forces the Scrum Team to actively plan which data needs to be gathered and to discuss the reasoning as to why it is required. Further, the team should already perform preliminary plans on how to mitigate potential privacy risks. This entails answering the following set of questions:

1. Which data types need to be gathered?
2. How long does the data need to be stored?
3. How is the data gathered?
4. Is access to data limited?
5. How is data stored?
6. What risks do these impose?
7. How are these risks mitigated?

The first two questions of this phase deal with data types and the storage length. The GDPR determined that data may not be stored for an indefinite amount of time, but only for so long as required for the project, e.g. a length of time an individual has accepted. Distinguishing the type of data as well as minimising the time that data needs to be stored for is a key process to minimise overall privacy risk to the project. In this early stage, the team can ensure that only the absolute minimum of personal data required to perform the project's goal is gathered. The less information of a single individual is gathered, the smaller potential fines will be, e.g. through unwanted manipulation of data.

The next step is discussing how and during which stage individuals' data is gathered. Data collection or processing can vary from using a dataset gathered from somebody else, to gathering the information oneself through a form or website to funnelling the information to another project. Creating an initial

plan on how to gather the required personal data allows to actively ensure that the process is secured from the beginning. A key process in ensuring minimal privacy risk is the issue of how to store data and to limit access to storage. A best-practice here is to create a hierarchy of access control, e.g. someone only gains rights to perform specific actions when they need them, not by default. Further, the PrO needs to know where the servers storing the data are physically located, as their physical location changes the entire scope of privacy protection. This is due to the GDPR requirement that all data collected on EU citizens must be either stored within the EU or in a country within a legislation that has a similar level of privacy protection. The identified collecting or processing procedures need to be analysed on the topic of potential risks faced by using these methods. Further, the identified risks need to be prioritised by using the introduced qualitative risk method, which will be highlighted in the next section in which the method for analysing privacy is also addressed. Lastly, each identified privacy risk needs to be analysed and mitigation procedures need to be planned.

Review The final phase, the *Review* phase, deals with reviewing the identified privacy risks and the proposed solutions, resulting in a residual risk for each identified privacy risk. This residual risk needs to be analysed and reviewed. The result of the review is either accepting the remaining risk level or denying the proposed solution as not providing a high enough level of security. This would lead to a re-evaluation of the privacy risk to propose better mitigation procedures. Therefore, the following tasks need to be solved in this phase:

1. Mapping of residual risks
2. Validating residual risk

Firstly, a re-mapping of the identified privacy risks needs to be performed. This re-mapping focuses on ranking the identified privacy risks regarding the residual risk. It allows a clear and direct view of effects of the mitigation procedures on the privacy risks by visualising the planned change in the risk level of each privacy risk. Secondly, the *Review* phase deals with the validation of the residual risk level of each identified privacy risk. The PrO needs to assess if the residual risk level is either acceptable or, if the residual risk level remains too high, needs to re-evaluate the planned mitigation measures. This re-evaluation is to be performed immediately for identified privacy risks as the current planned measures result in a remaining risk level that is not acceptable. Once each privacy risk is validated, the initial PIA is completed.

As previously mentioned, the initial PIA needs to be performed within the Sprint-Zero. This ensures that nothing has been developed by the Scrum team yet and the project's goal of a working product is, at best, only planned. This is a key benefit for every identified privacy risk, as the procedures to reduce the privacy risk can be planned ahead of time and evaluated to their respective

degree of privacy protection. The initial PIA should give the team a high level overview of their attack surface, potential threats to the project as well as a general overview of the required privacy measures. This initialisation essentially provides the PrO with the first few necessary privacy user stories for the Product Backlog.

5.7.2 Privacy Risk Assessment

This section discusses the key process of evaluating the risk level of identified privacy risks. A key part of the initial Privacy Impact Assessment is the identification of the project's scope by visualising the required personal data and the handling of privacy risks. Thus, during the initial PIA a Privacy Risk Assessment is performed for each and every identified privacy risk as well as for each privacy user story. Because of the nature of privacy user stories, every single one has a certain risk level. This level allows for prioritisation, so that the most dangerous risks get focused on as soon as possible. Further, identifying the risk level of a privacy risk aids in the mitigation procedure, as the risk level generally correlates with the required mitigation procedures, e.g. the higher the risk level, the more sophisticated the protection measures have to be to mitigate the risk. *Privacy Risk Assessment* is the core process to evaluate every privacy user story and establish the required measures to ensure adequate protection towards each privacy risk.

The process of assessing identified risks is often a very time-consuming and documentation-heavy process with numerous, often difficult to determine, aspects involved. As discussed in the subsection 3.1 of this thesis, the qualitative method of assessing risks is chosen for this Privacy Protection Framework because of its lightweight manner. Furthermore it suits the complex and often indeterminate environment of information technology. The qualitative risk assessment method is based upon short discussions in a brainstorm-manner to establish the risk level for an identified risk. Subsequently, the risks are mapped according to their risk level, resulting in an overview of risks and their respective danger to the project. In light of the qualitative assessment method, the author proposes to define three levels of risk classification. These risk levels are defined as *low*, *medium* and *high*. This allows a lightweight assessment of each control question and therefore stays in line with the agile mindset as well as the complexity of the problem scope. The resulting assessment of every identified privacy risk enables the Privacy Officer to perform a prioritised list of privacy risks. Subsequently, the PrO can then formulate specific acceptance criteria for each privacy user story depending on the risk evaluation.

Privacy Risk Identification However, before the Privacy Officer can create a hierarchy of privacy risks, these risks need to be identified first. Generally, as researched in subsection 3.1, a risk combines the threat to a system with a matching vulnerability. Furthermore, there needs to be a feared event which occurs when a threat utilises a vulnerability. The risk level of a risk depends on two

calculations. The first evaluation is based on the likelihood that a threat utilises a vulnerability. The second evaluation is establishing the potential severity of the impact of the feared event on the assets at risk. Together, the likelihood and severity result in the risk level of the respective risk.

In order for a user story to be determined as a *privacy risk*, the user story has to include a potential effect on personal data. This potential effect does not only describe direct handling of an individual's dataset, but includes all processes which could provide access to any form of documentation on personal data. In the context of data protection as defined by the GDPR, the process of Risk identification varies from the general process, as the core risks are clarified by law. The explicit data protection risks as defined by the GDPR are:

- Illegitimate access to personal data
- Unwanted change of data
- Disappearance of data

Hence, the Privacy Officer has to identify potential threats as a risk source and vulnerabilities in the deployed assets to these defined risks. Risk sources to the defined risks can be divided into two semantic groups. These are based on the typology defined by CNIL (CNIL, 2018c) and adapted to focus solely on the information technology domain:

- Internal
- External

The first group, *Internal risk sources*, describes all risks that originate from sources within the organisation. These include all personnel such as employees, managers or trainees who abuse their access rights to a malicious effect, knowingly or not. The second group, *External risk sources*, describes all risk sources outside of the organisation. These include all customers, competitors, external firms that interact with the project in any extent as well as hackers, malicious code from unknown origin and so on.

A user story that could enable any of the above risk scenarios has to be flagged as a privacy user story and needs to be evaluated with the goal of setting adequate security measures in place to mitigate the risk.

Privacy Risk Evaluation The Privacy Officer has to perform a Risk Evaluation for every identified privacy risk. The main goal of the privacy risk evaluation is to determine the risk level of each privacy risk.

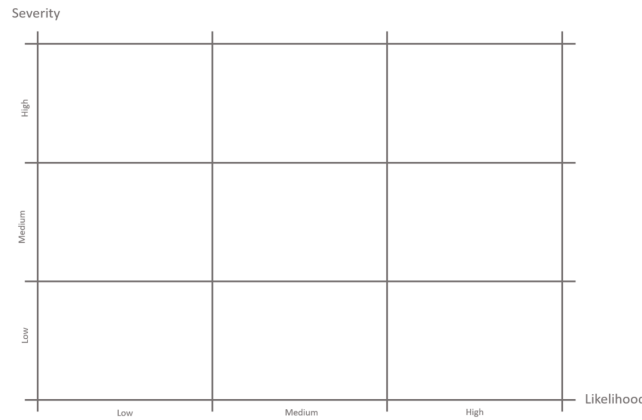


Figure 5.5: Severity-Likelihood-Matrix

The author proposes to evaluate each privacy risk on a Severity-Likelihood-matrix. This allows to visualise the risk level of every privacy risk depending on the two most important metrics. The matrix, as depicted in Figure 5.5, enables the Privacy Officer to place every privacy risk regarding their respective risk to the project. The Severity-Likelihood-Matrix is set up with the grade of severity on the y-axis and the grade of likelihood on the x-axis. Both grades range from low to high. To assess every privacy risk as accurately as possible, the Scrum team discusses each identified risk in a brainstorm discussion. An open discussion is necessary to showcase the different points of views of every team member. Every privacy risk needs to be understood by the team members in order to ensure a best-possible risk mitigation. To increase involvement and understanding, every team member is included in the discussion on placing the risks on the risk matrix. Whilst the Product Owner focuses on the business side and embodies the stakeholders' view, the development team adds important implementation knowledge to each risk. As a mediator and expert on the topic of privacy, the Privacy Officer needs to understand and set each view into perspective.

Establishing the severity of a privacy risk should not solely depend on the team member with the propose the best argument, it should heavily depend on multiple factors related to how the personal data is handled. Generally, the severity of a risk in the context of information technology is hard to determine, however, when focusing on personal data, there are three major categories which have a direct effect on the risk level. These critical categories form the basis of any privacy risk evaluation:

- Type of personal data
- Length of storage period
- Impact of feared event

Firstly, each privacy risk can touch upon different types of personal data with a varying degree of sensitivity. The second category describes the determined storage length of personal datasets. Lastly, it is of high importance to identify what possible events can occur on data by the potential abuse of this risk. This is established in the type of the feared event. Further, the GDPR defines different grades of personal data. These grades determine the level of sensitivity and can be used to establish a foundation for the risk level regarding severity. The three levels of data sensitivity are defined as:

- Common personal data
- Personal data perceived as sensitive
- Sensitive personal data in the meaning of DP-Act

Personal data types	Personal data categories
Common personal data	Civil status, identity, identification data
	Personal life (living habits, marital status, etc., excluding sensitive data)
	Professional life (résumé, education and professional training, awards, etc.)
	Economic and financial information (income, financial situation, tax situation, etc.)
	Connection data (IP addresses, event logs, etc.)
Personal data perceived as sensitive	Location data (travels, GPS data, GSM data, etc.)
	Social security number
	Biometric data
Sensitive personal data in the meaning of DP-Act	Bank data
	Philosophical, political, religious and trade-union views, sex life, health data
	Offences, convictions, security measures

Table 5.12: Typology of personal data (CNIL, 2018a)

Table 5.12, taken from the PIA guide of the French data protection authority (CNIL, 2018a), showcases the different types of data. *Common personal data* describes the lowest risk level of all data types. It describes generally known and accessible data of an individual such as information regarding their professional life, their marital status as well their location data and connection data. If a privacy user story touches upon common personal data, the risk level regarding the data type is set to **low**. *Personal data perceived as sensitive* represents the group of data that allows for direct identification of an individual. Often, this group is incorrectly perceived as the most sensitive of data, however pure identification of an individual does not entail negative action towards that individual. Such severe negative impact results from information types contained in the third group, *Sensitive personal data in the meaning of DP-Act*, as this information includes political or religious views, sexual inclinations, health information and so on. This set of information poses a much greater threat to an individual than raw identification. Therefore, the risk level in the case of *Sensitive personal data in the meaning of DP-Act* results in a **high** risk level, whereas data handling of *Personal data perceived as sensitive* results in a **medium** risk level.

Another indicator for the risk impact is the planned storage time for the respective dataset. It is required by law to determine a specific time frame

for storing data. However, the law remains fairly general with the phrasing of storing data only as long as that data is required (Commission, 2018). For each privacy risk this is a key question to pose as in general the shorter the storage time, the safer. This follows the simple logic of reducing the threat surface if the user story is implemented in a way that limits the storage time automatically. The author proposes to scale the risk level depending on the defined storage period. A low risk level is posed by every dataset that is stored no longer than three months, a medium risk is any dataset stored for a time period between three and nine months, whilst any dataset stored for longer than nine months results in a high risk level.

The third indicator for the risk level of a privacy user story is the feared event. There are three different types of feared events on personal data, similar to the general risk as defined by the GDPR:

- Illegitimate access to personal data
- Unwanted change of data
- Disappearance of data

Illegitimate access to personal data provides an entity with unauthorized access to an individual's data. In this case the entity is only able to read the stored data, which therefore results in a **low** risk level. *Unwanted change of data* enables the entity to read and modify an individual's data, resulting in the allocation of a **medium** risk level. The most dangerous consequences are possible in the section *Disappearance of data*, as in this case an entity can read, change and delete an individual's data. This enables an attacking entity to perform any operation on data, thus resulting in a **high** risk level. Combining all of the categories to an overview as shown in Table 5.13 provides the Scrum Team with a good basis for debate in evaluating the risk level of each privacy risk.

Privacy User Story	Data Type	Storage Time	Feared Outcome

Table 5.13: Template Privacy User Story Analysis

These categories can further be extended depending on the scope of the project. Such an extension for evaluating the risk level could be, for example, the amount of datasets potentially affected in relation to the total datasets of a project. Therefore, Table 5.13's design is easily adaptable to include more than just the most relevant categories.

After identifying an overview of each risk, the likelihood of the risk as well as the severity of the potential impact need to be determined. These can each take the form of *low*, *medium* and *high*. This is performed by placing each identified

privacy risk on the risk matrix previously shown in Figure 5.5. As a result, multiple privacy risks can be placed on the risk matrix. Figure 5.6 outlines the risk evaluation of the two privacy risks *Data Breach* and *Loss of Data*.

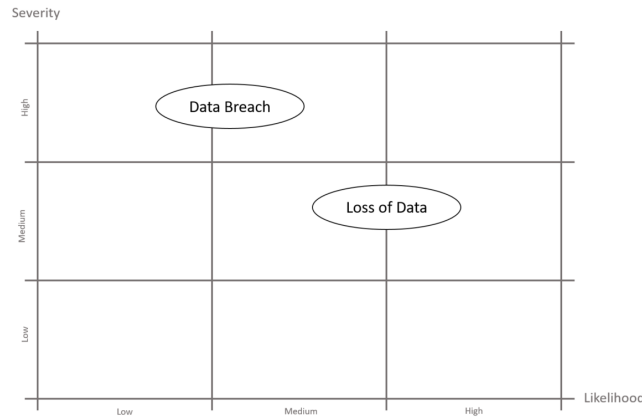


Figure 5.6: Risk placement on Severity-Likelihood-Matrix

Privacy Risk Mitigation After evaluating the risk level of every privacy risk, the Privacy Officer has to establish the required procedures to reduce the risk level to an acceptable one. The remaining risk level, i.e. after certain measures are planned against a risk, is described with the term *residual risk*. This residual risk level is set as an acceptance criteria for every privacy user story. The best case of mitigation measures would be to add such an amount of protection to the specific privacy risk that it would entirely mitigate the privacy risk. However, this is rarely possible and entails such a high amount of work, that it may not be worthwhile to pursue in an agile setting. The goal for mitigation procedures is to create such a level of privacy protection that it will not be worthwhile for a potential attacker to abuse the privacy risk. This results in a residual risk level to be deemed acceptable. Evaluating, when exactly the security measures reach a degree of security, that it is not worthwhile for an attacker to pursue the abuse of a privacy risk, lies in the responsibility of the Privacy Officer as Risk Owner and the development team as supporters.

In order to visualise the effect potential mitigation procedures have on the risk level of a privacy risk, the privacy risks are mapped on a risk matrix. The risk matrix has a similar layout as in the evaluation stage, but includes the change performed by the mitigation procedures. The mapping of privacy risks on a risk matrix provides the Privacy Officer with a valuable document to justify necessary actions on risks, as the effect of the planned work is showcased. Furthermore, the mapped risks enable the PrO to prioritise the risks relevant to their residual risk level. A risk map should be continued throughout the entire development life cycle, as general feared events on the misuse of data will be affected by many different privacy user stories.

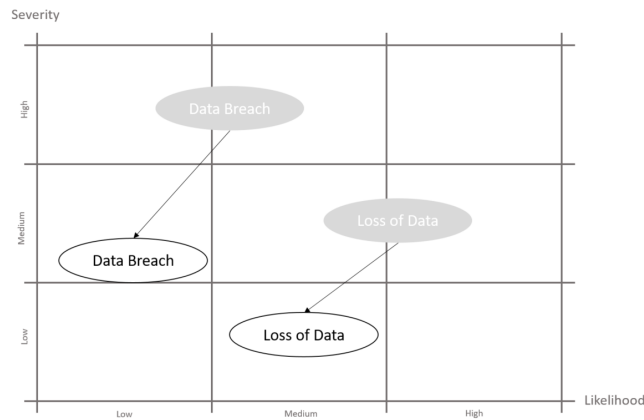


Figure 5.7: Risks mapped on Severity-Likelihood-Matrix

Figure 5.7 showcases risk mapping for the previously evaluated privacy risks. The change in the risk level of the respective risk - as a result of setting security measures in place - is highlighted by showing the initial risk level as well as the residual risk level. In this example, the residual risk level of *Data Breach* and *Loss of Data* has been reduced.

5.8 Privacy Scrum Board

Most agile methodologies utilise a haptic board to visualise the work in progress. This generally depicts all currently defined tasks as well as their current status in a tabular style. Figure 5.8 shows the classic Scrum Board. The goal of the Scrum Board is to enhance the knowledge of which value to the product is identified and currently worked upon.

Product Backlog	Sprint Backlog	In Progress	Review	Done
User Story 7	User Story 2	User Story 3	User Story 10	User Story 1
User Story 4	User Story 5	User Story 8		
User Story 6		User Story 9		
User Story 11				

Figure 5.8: Standard Scrum Board

The standard Scrum Board as shown in Figure 5.8 is designed with columns showing the different status a user story can take. The different status are *Product Backlog*, *Sprint Backlog*, *In Progress*, *Review* and *Done*. Every user

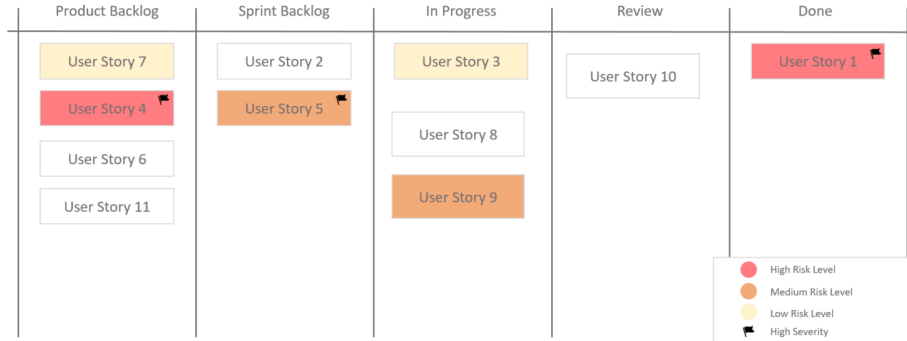


Figure 5.9: Privacy Scrum Board

story is placed on the Scrum Board in the column *Product Backlog* as soon as the story is defined. Subsequently, the story is moved along into the *Sprint Backlog* when it is selected for the next Sprint. When it is allocated to a team member, the story is moved to the *In Progress*-column. Once the developer is finished implementing the story and it is ready for review, the story is moved to the next column, i.e. *Review*. When a story successfully completed the Review, it is moved to *Done*. If not, it needs to be reformulated and placed back on the *Product Backlog*. This is a practical way to visualise which stories are currently allocated and in which stage of the development phase they are. However, a haptic Scrum Board is not suitable for distributed teams because the Scrum team will rarely be in the office to physically change the status of user stories by moving them along the board. Therefore, the Scrum Board needs to be digitised and made accessible to every team member.

The author proposes to adapt the Scrum Board to include a visual distinction between standard user stories and privacy user stories. The adapted Privacy Scrum Board is shown in Figure 5.9. The visual distinction is performed by adding a background colour to the story. The background colour depends on the respective risk level of the privacy user story. Furthermore, each privacy user story can be defined as "severe" as determined in subsection 5.6. This classification entails a severe change to the existing project and therefore requires a separate visual distinction. The visual distinction for severe privacy user stories is therefore a flag (*Free vector icons designed by Smashicons*, 2020) in the top right corner of the privacy user story.

The introduced components for *Privacy Management* are subsequently deployed in a prototype to establish a proof of concept for the proposed framework in distributed teams.

6 Proof of Concept

The Prototype to validate the designed Framework aims to assist a Privacy Officer and their distributed team at incorporating privacy protection in an agile software development environment. At the core of this prototype lie two programs, a web service and a data process risk assessment tool. The components and their communications are depicted in Figure 6.1.

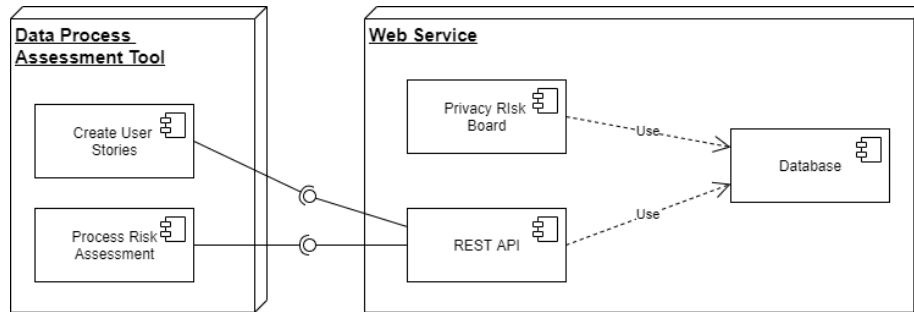


Figure 6.1: Prototype components

The web service visualises the digital risk board by displaying all active user stories for the registered user. This showcases the project's current work-in-progress and the status of every user story and focuses on highlighting privacy user stories. A web service is chosen because it allows access from any physical location as long as the user has access to the internet. Therefore, current privacy risks and work in progress remain visible for distributed teams as well. The web service allows full manipulation of privacy user stories as well as creating new stories once a user is logged in. The required information is stored in a relational Structured Query Language (SQL) database. The author chose this type of database as it is used predominantly in the real world which leads to multiple benefits such as scalability and transferability. Furthermore, the problem space at hand has a relational dependency between a user and their user stories which suits the relational database style of SQL very well.

Further, the web service is implemented with multiple interfaces in the form of RESTful Application Programming Interfaces (API), in theory allowing any tool or program to send information to the web service. These serve as information gateways for the data process risk assessment tool. The APIs enable communication between the tool and the web service. The author chose to implement the information gateways as RESTful APIs as it allows for easy adaptation and builds a solid foundation for future enhancement. It also enables other programs to send information to the web service in order to use the digital privacy risk board. For a better understanding when discussing the code, *variables* are displayed in *italic*, whereas **values** are shown in **bold**.

6.1 Web Service

The Prototype is programmed in Python 3.7.4 and makes use of the *flask* library to implement the web service. The front-end uses HTML pages for displaying in the web browser. Further the CSS-Framework *Twitter Bootstrap* is used for the front-end. The web service accesses a database in SQLite format.

6.1.1 Privacy Scrum Board

The first step for the user is logging in. The code for the routing is shown in Listing 1:

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).
            first()
        if user is None or not user.check_password(form.password.
            data):
            flash('Invalid username or password')
            return redirect(url_for('login'))
        login_user(user, remember=form.remember_me.data)
        next_page = request.args.get('next')
        if not next_page or url_parse(next_page).netloc != '':
            next_page = url_for('index')
        return redirect(next_page)
    return render_template('login.html', title='Sign In', form=
        form)
```

Listing 1: Login route

When the URL */login* is called, the first check that needs to be performed is whether the user is already authenticated. If the user is already logged in, they are redirected to the page */index*, which is the main page of the web service. When the user enters their login credentials into the **LoginForm**, the input is checked by querying the database for an existing match. If this check is successful, the user is redirected to the index page. If not, the user is prompted with an error message.

Product Backlog	Sprint Backlog	In Progress	Review	Done	Options
		Account lifecycle Max Mustermann			Edit Delete Save to Repository
		Communicate privacy policy Julia Musterfrau			Edit Delete Save to Repository
Monitoring					Edit Delete Save to Repository
	Review PIA				Edit Delete Save to Repository
		Process medical data Veronika Mustername			Edit Delete Save to Repository

Add Task

Figure 6.2: Privacy Scrum Board

The index page seen in Figure 6.2 shows the privacy scrum board of the logged in user, in this case the user with the username **arvicciiii**. The navigation bar at the top of the page enables the user to navigate through the web service. The options for the user are navigating to the main page, to their repository or logout of the web service. The main page of the web service begins below the navigation bar. First, the user is greeted by a short welcome message and the amount of privacy user stories to be completed until the Privacy Impact Assessment needs to be reiterated is displayed. Further, it shows all currently registered user stories and places them in the table depending on their status. The columns are equivalent to the status a user story can take in the Scrum development method. The user stories are depicted in this view with title and story owner. The tasks *Monitoring* and *Review PIA* are yet to be assigned to a developer, therefore the space for story owner remains empty, whereas the stories *Account lifecycle*, *Communicate privacy policy* and *Process medical data* are allocated to a developer.

Privacy user stories are highlighted with a red background colour. This depiction is chosen due to reasons of practicality. The core necessity is to create a visual distinction between normal user stories and privacy user stories. The actual implementation of this depiction depends on further research to ensure a large visual distance to create the necessary visual impact of privacy user stories. In this prototype, the colour varies depending on the privacy risk level

of the story, ranging from a *light red* for a low risk level to a *dark red* for a high risk level. Further, severe user stories are visually highlighted by a *red* border, whereas user stories that do not have an effect on the existing system have a *black* border. The user stories shown in the Privacy Scrum Board are queried from the database and handed to the rendering of the page. The routing for the main page is shown in Listing 2.

```
@app.route('/')
@app.route('/index')
@login_required
def index():
    #store the tasks of the current user in a variable
    db_tasks = current_user.tasks.all()
    #call return_template from flask to show index.html
    #pass tasks of current user to template
    return render_template('index.html', title='Home', tasks=
        db_tasks)
```

Listing 2: Index route

For each task, the attributes *sev_flag*, for severity, and *risk_level*, for risk level, are checked. If *sev_flag* is set to **true**, the border colour of the user story is set to **red**. If it is **false** the border colour is set to **black**. Further, the value of *risk_level* determines the background colour of the story. Finally, the owner of the story is displayed. An example of the code that handles the visualisation of user stories with the status **PB**, e.g. Product Backlog, is shown in Listing 3.

```
{% for task in tasks %}
    <tr>
        {% if task.status == "PB" %}
            {% if task.sev_flag == True %}
                <td style="border-color: rgb(252, 1, 1);">
            {% else %}
                <td style="border-color: rgb(0, 0, 0);">
            {% endif %}
            {% if task.risk_level == "high" %}
                <p style="background-color: crimson;"><b>{{ task.title
                    }}</b>
            {% elif task.risk_level == "medium" %}
                <p style="background-color: lightcoral;"><b>{{ task.
                    title }}</b>
            {% elif task.risk_level == "low" %}
                <p style="background-color: lightpink;"><b>{{ task.
                    title }}</b>
            {% else %}
                <p><b>{{ task.title }}</b>
            {% endif %}
            <p>{{ task.owner }}</p></td></tr></p>
```

Listing 3: Visualise user stories

The user has the possibility to edit, delete or save a task to their repository through the code in Listing 4. Each link leads to a specific route that requires the **task.id** to perform the desired action on the task.

```

<p> <a href="{{ url_for('editTask', id = task.id) }}">
    Edit</a>
    <a href="{{ url_for('deleteTask', id = task.id) }}">
        Delete</a></p>
<p><a href="{{ url_for('saveToRepo', id = task.id) }}">
    >Save to Repository</a></p>

```

Listing 4: User story options

The entire code for the visualisation of the index page can be seen in Listing 26.

Furthermore the user has the possibility to create a new user story through the link *Add Task*. The following code focuses on the logic of creating such a new user story. This is implemented in the Prototype through the combination of a *FlaskForm* for user input, visualising the form with a HTML-template and additional logic in the routing to add the user input in a new user story to the database. The result of the initial interface in the web service is shown in Figure 6.3.

The screenshot shows a web browser window with the address bar at `localhost:5000/addTask`. The page has a navigation bar with links for 'Privacy RB', 'Home', 'Repository', and a 'Logout' button. The main content area is titled 'Add Task' and contains the following form elements:

- Title**: A single-line text input field.
- Description**: A single-line text input field.
- Give more information**: A multi-line text area.
- Story Owner**: A single-line text input field.
- Acceptance Criteria**: A single-line text input field.
- Privacy Flag**: A checkbox.
- Privacy Issue Description**: A single-line text input field.
- Privacy Risk Level**: A group of four radio buttons labeled 'None', 'Low', 'Medium', and 'High'.
- Privacy Severity Flag**: A checkbox.
- Privacy Severity Description**: A single-line text input field.
- Add Task**: A button at the bottom of the form.

Figure 6.3: Add User Story

The library *flask_wtf* is used as a basis for user forms. Forms are defined in a class of type **FlaskForm** and called upon in the routing function. In the class form, each input type is stored in a variable. The input types require a label and, where necessary, can be defined as required through the function *DataRequired()*. The Listing 5 shows the definition of such a class.

```
class addTaskForm(FlaskForm):
    title = StringField('Title', validators=[DataRequired()])
    description = StringField('Description', validators=[
        DataRequired()])
    more_descr = TextAreaField('Give more information')
    owner = StringField('Story Owner', validators=[DataRequired()
    ])
    requirements = StringField('Acceptance Criteria', validators=[
        DataRequired()])
    priv_flag = BooleanField('Privacy Flag')
    risk_level = RadioField('Risk Level', choices = [('low', 'Low'
    ), ('medium', 'Medium'), ('high', 'High')])
    priv_descr = StringField('Privacy Description')
    sev_flag = BooleanField('Severity Flag')
    sev_descr = StringField('Severity Description')
    submit = SubmitField('Add Task')
```

Listing 5: Add Task Form

Each possible entry field for the user needs to be defined in the class. Allocating these fields to a variable is required to access the information the user enters into these fields at a later stage.

When defining the routing as shown in Listing 6, in this case for the URL */addTask*, the form is stored in a variable and passed along to the function *render_template*. This function is a function of the flask library which takes an HTML page and a title as input to display the HTML page. Furthermore one may pass variables, in this case a *form* along, which enables the CSS-framework *bootstrap* to use these variables when displaying the HTML page. The submitting of a form by the user can be checked through the function *validate_on_submit*, a function that exists for all *WTForms*. Because this function allocates the input of the user in the form to a task, a new task, where the information can be stored in, needs to be created first. Subsequently, the information entered in the fields is allocated to the respective fields of the new *Task()* in the database. Because this is a new user story, the status is fixed as **PB**, e.g. Product Backlog. Finally, the created task needs to be added to the database and stored. A security measure added to the route */addTask* is the decorator *@login_required*, which requires a user to be logged in to access this URL. If the URL for */addTask* is opened by a user who is not logged in, they will be redirected to the login page.

```
@app.route('/addTask', methods=['GET', 'POST'])
@login_required
def addTask():
    form = addTaskForm()
    if form.validate_on_submit():
        task = Task()
```

```

task.user_id = current_user.id
task.title = form.title.data
task.description = form.description.data
task.fileupload = form.more_descr.data
task.owner = form.owner.data
task.requirements = form.requirements.data
task.priv_flag = form.priv_flag.data
task.priv_descr = form.priv_descr.data
task.sev_flag = form.sev_flag.data
task.sev_descr = form.sev_descr.data
task.status = "PB"
task.risk_level = form.risk_level.data
db.session.add(task)
db.session.commit()
return redirect(url_for('addTask'))
return render_template('addTask.html', title='Add Task',
                      form=form)

```

Listing 6: Route for adding a Task

Because of the length of the HTML-page *addTask.html*, the complete page is added in the Addendum in Listing 29. The key aspects however are shown in the code snippet in Listing 7. This example shows how the information of the variable *title* defined in Listing 5 is extracted and displayed. Further, the size of the input field has to be defined. The final part is defining what should happen in case there is an error in the validation on submit. Here, the error message is then displayed in red.

```

<p>
  {{ form.title.label }}<br>
  {{ form.title(size=32) }}<br>
  {% for error in form.title.errors %}
    <span style="color: red;">[{{ error }}]</span>
  {% endfor %}
</p>

```

Listing 7: Code-snippet from addTask.HTML

When a user wants to delete a specific user story, the route shown in Listing 8 is called.

```

@app.route('/deleteTask/<int:id>')
@login_required
def deleteTask(id):
    #query the wanted task by id and store in variable
    task = Task.query.get(id)
    #check if the author of a task is the current user
    if task.author != current_user:
        return bad_request('Only the author may edit their tasks')
    #delete the desired task
    Task.query.filter_by(id=id).delete()
    #commit the change to the database
    db.session.commit()
    #redirect user to the index page
    return redirect(url_for('index'))

```

Listing 8: Route for deleting a task

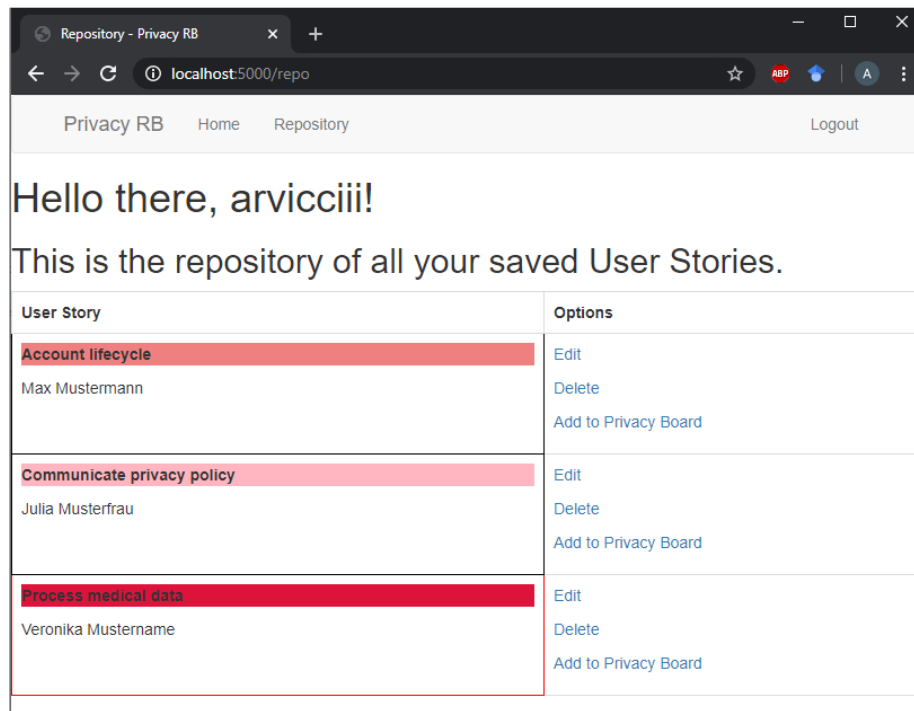


Figure 6.4: Repository

The user needs to call the URL `/deleteTask` with a specific task **id**. Then, that task is queried for and a check is made, if the current user is the author of the task. If not, they are not allowed to delete that specific task. This is a protection mechanism to ensure that one may only delete one's own tasks. If the author of the task is the current user, that task is deleted and the status change is committed to the database. If a user tries to access this URL, they will be redirected to their index page, unless they tried to delete a task that they are not the author of. In that case they would receive a bad request message.

6.1.2 Repository

The second part of the web service is the *Repository*, a library of all user stories saved for future reference by the user. This plays a key part in ensuring the knowledge of completed work is stored and reusable. Therefore, a function to store a user story to the users repository is included in this prototype. When the user chooses to store a user story to the repository, it is depicted in a table as shown in Figure 6.4. The stories are colour-coded in the same way as in the Privacy Scrum Board. This ensures coherent usability and easy understanding of the different levels of risk and impact of each user story.

As in the Privacy Risk Board, the user can perform multiple actions on each user story. If they want to see further information or edit the story, they can access the user story through the *Edit* link. However, the most important action on a user story is the possibility to add a user story to the Privacy Scrum Board. This enables the user to re-use the story and ensures that no knowledge is lost.

6.1.3 RESTful Application Programming Interfaces

For a program to send information to the web service and the web service subsequently performing an action based on that information, the author chose to implement Representational State Transfer Application Programming Interfaces (REST APIs). General APIs, such as a website, allow a service to implement a functionality at a certain endpoint such as a URL. To allow standardised communications between programs, a REST API uses HyperText Transfer Protocol (HTTP) to send data in standardised formats to an endpoint that focuses on performing one specific action. REST defines a set of criteria to standardise this process, such as being stateless and having a clear separation of client and server (Biehl, 2016). The author chose to use JavaScript Object Notation (JSON) as a data-format. In this prototype multiple RESTful APIs are programmed to perform following functionalities:

- User authentication
- Create privacy user stories
- Token generation
- Update user stories

As an example, the code in Listing 9 deals with the creation of a user story through a POST-request to the endpoint */tasks*. The decorator *@token_auth.login_required* enforces the requirement of a token. The information sent with the POST-request needs to be extracted and transformed from JSON-format to the respective fields in the database entity *Task*. However, before allocating information to a new task there needs to be a check if the sent information includes the correct data to protect the endpoint from malformed input. This is performed by checking for key values *title*, *description* and *requirements*. The newly created task can then be added and committed to the database. A key component of REST APIs is to send a response to the client which informs on how the request has been handled. The author chose to create a response that includes the information on the task as well as the HTTP status code **201** for *created*.

```
@app.route('/tasks', methods=['POST'])
@token_auth.login_required
def create_task():
    data = request.get_json() or {}
    if 'title' not in data or 'description' not in data or '
        requirements' not in data:
        return bad_request('must include title, description and
            requirements fields')
```

```

task = Task()
task.from_dict(data)
db.session.add(task)
db.session.commit()
response = jsonify(task.to_dict())
response.status_code = 201
return response

```

Listing 9: Create User Story via API

The code for the other APIs can be found in Listing 20.

6.1.4 Database

The flask library supports the integration of *SQLAlchemy* through the library *flask_sqlalchemy* and also includes a migration manager with the library *flask_migrate*. The migration library allows to make changes to the database and stores each change as a version.

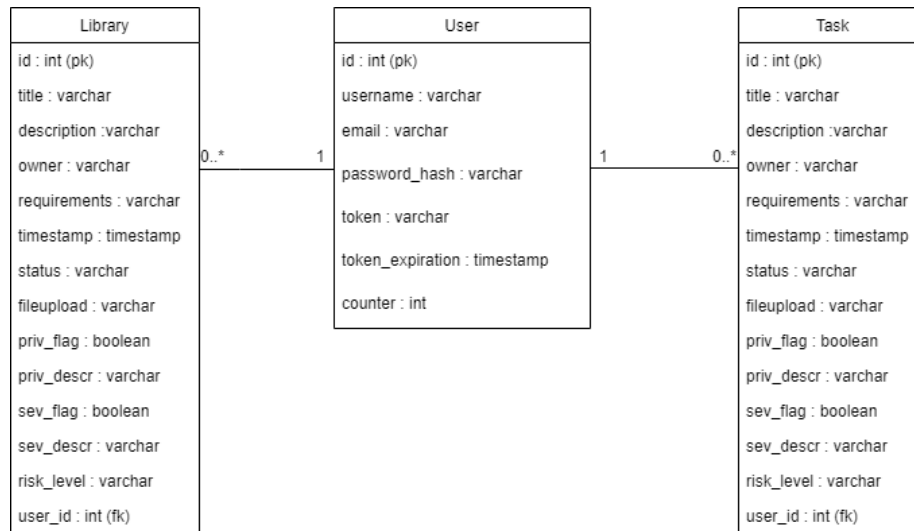


Figure 6.5: Database Model

The database is structured as depicted in Figure 6.5. The separation of the similar entities *Task* and *Library* is chosen to create the clear distinction of active user stories to user stories stored in the *Library* at the lowest possible level. This ensures that each entity is adaptable to any changes required to one part, but not the other. Such a potential addition to the *Library* could be the inclusion of the developed solution for a *Task*. The main entity in the database is the *User*. Every user can have multiple *Tasks*, but could also have **zero** tasks. The same is valid for the *Library*. However, every *Task* and every *Library* has

to be allocated to one specific *User*. The relationship between a *User* and their *Tasks* is implemented through the foreign key **user_id** in the entity *Task*.

6.2 Data Process Assessment Tool

The Data Process Assessment (DPA) tool allows the user to identify the risk level of a data process. It is written in Python 3.7.4 and uses the packages *tkinter* for the graphical user interface (GUI), *matplotlib* for visualising the risk matrix and *requests* for sending API calls to the web service. Its aim is to support the Privacy Officer in quickly assessing the risk level of a data process and enabling the straightforward creation of user stories which, in turn, are sent to the web service for visualisation.

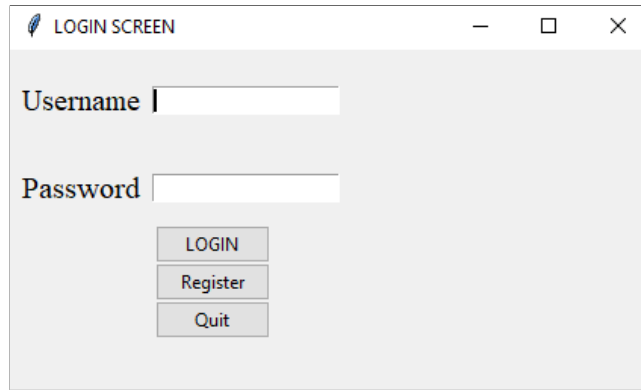


Figure 6.6: GUI Login Screen

The user is required to login to the service before they can assess a data process. The GUI for the login is shown in Figure 6.6. The core component of the *Login Screen* is the handling of the login procedure. For the mobility required in a distributed team, the database is connected to the web service. Therefore, the login for the DPA tool needs to access a REST API of the web service. Clicking on the login button leads to the execution of the function *login_user*, which triggers an API call to the web service with the user's input as login credentials. This API call is performed by the function *user_login*, highlighted in Listing 11. On a successful login, the web service issues a token to be used as a session token in further communication and the main GUI is executed. An unsuccessful login returns an error message and prompts the user to re-enter their login credentials. The associated code is shown in Listing 10.

```
def login_user(self):  
    #Call api request to check username and password are correct  
    response = request.user_login(self.username.get(), self.  
        password.get())
```

```

#check api reply
if "token" in response: #login was successful
    #save token and username to global variables for reuse
    global token
    global username
    username = self.username.get()
    token = response["token"]

    #Destroy the current window
    root.destroy()
    #Open new window based on pia_tool
    newroot = tk.Tk()
    pia_tool(newroot)
    newroot.geometry('1500x1200')
    newroot.mainloop()

else:
    print("Error:%s"%response)
    #Prompt user that either id or password is wrong
    self.message = tk.Label(text = 'Username or Password
        incorrect. Try again!',fg = 'Red')
    self.message.grid(row=6,column=2)

```

Listing 10: Handling user login

The previously mentioned API request, which sends the login information to the web service, requires the login information and an endpoint to send the information to. Also, it needs to handle the reply of the web service as the reply gives information on how the web service has handled the request. In the RESTful web service, these replies are in JSON format. The code is shown in Listing 11.

```

import requests

def user_login(user, password):
    # defining the api-endpoint
    API_ENDPOINT = "http://localhost:5000/tokens"

    # sending post request and saving response as response object
    r = requests.post(url = API_ENDPOINT, auth=(user, password))

    # extracting response text and return it
    json_reply = r.json()
    return json_reply

```

Listing 11: API call for login

The API of the web service at <http://localhost:5000/tokens> checks the credentials and, if they are valid, replies with a session token. The function `user_login` returns the token so that it can be used throughout the session for further API requests instead of having to resend the login credentials every time. The data process assessment depends on the input of the user in the GUI as depicted in Figure 6.7. The user can select the characteristics for their data

Figure 6.7: Graphical User Interface

process. The selected information is evaluated when the user clicks on *Evaluate Data Process*.

Firstly, the user needs to determine the type of data used by a process. These are grouped into the three data types, as defined per GDPR, with each type having an impact on the severity of a risk as discussed in subsubsection 5.7.2. Secondly, the user defines the storage time intended for the data. The longer the data is stored, the higher the probability of a risk for this data. Thirdly, the user has to determine the origins of the gathered data. The fourth question deals with the security measures taken by the user to protect the data process. All options have an effect on either probability or severity, depending on the respective security measure. Each security measure reduces the respective level by one third. This evaluation is only a preliminary choice to suit the scope of this thesis and would require further research in order to determine the actual effect of each security measure on the risk level.

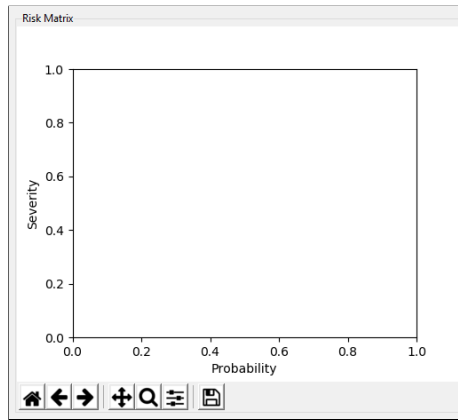


Figure 6.8: Risk Matrix

The data process is evaluated towards the risks *Unwanted Change*, *Illegitimate Access* and *Disappearance*. These risks were identified in subsubsection 5.7.2 to be the core risks defined by the GDPR. Therefore, the author chose to implement these as exemplary risks in the prototype. When the user clicks on the evaluation button, the probability and severity of each of these three risks is calculated. The risks are then visualised on a risk matrix as shown in Figure 6.8. The calculation for the probability is performed by the function *calc_prob* shown in Listing 12.

```
def calc_prob():
    prob_level = 0
    # determine a probability value by the selection on Storage
    # time and Gathering option
    for counter in [radStorage, radGather]:
        if counter.get() == 0:
            prob_level = prob_level + 10
        elif counter.get() == 1:
            prob_level = prob_level + 25
        elif counter.get() == 2:
            prob_level = prob_level + 50
        else:
            prob_level = prob_level
    # reduce the probability value depending on security
    # Measures
    for count in [radMin, radAuth, radLog]:
        if count.get() == 1:
            prob_level = prob_level / 3
        else:
            prob_level = prob_level
    # calculate probability value for the three core risks
    illAcc = prob_level / 10 * 3
    unwCha = prob_level / 10 * 2
    disApp = prob_level / 10
    # store in a list to plot on the risk matrix and return the
    # list
    prob = [illAcc, unwCha, disApp]
```

```
return prob
```

Listing 12: Calculating risk probability

The variable **radStorage** takes the value *0* for *up to three months*, *1* for *three to six months* and *2* for *longer than six months*. As the longer data is stored, the greater the probability the data process might be affected by a data breach is. The probability level increases with the duration of data storage and can be reduced by the implementation of security measures. For this prototype, each security measure reduces the probability level by one third. The security measures defined in the prototype are *minimal access*, *authentication* and *logging*. These three concepts have an effect on the probability of the core risks to the data process. To scale the probability level towards a value between zero and one, each potential risk is divided by ten and is multiplied by one, two or three, depending on the general probability level, identified in subsubsection 5.7.2, of each risk. The function then returns the probability levels to the function *evaluate*, which creates the risk matrix. The second coordinate for each risk, *severity*, is calculated in a similar way by the function **calc_sev**, see Listing 13.

```
def calc_sev():
    sev_level = 0
    for counter in [rad_com_data, rad_pers_data,
                    rad_sens_data]:
        if counter.get() == 0:
            sev_level = sev_level + 20
        elif counter.get() == 1:
            sev_level = sev_level + 25
        elif counter.get() == 2:
            sev_level = sev_level + 30
        else:
            sev_level = sev_level

    for count in [radEncrypt, radPseu, radAnon]:
        if count.get() == 1:
            sev_level = sev_level / 3
        else:
            sev_level = sev_level

    illAcc = sev_level / 10
    unWCha = sev_level / 10 * 2
    disApp = sev_level / 10 * 3
    sev = [illAcc, unWCha, disApp]
    return sev
```

Listing 13: Calculating risk severity

The function increases the variable **sev_level** depending on which type of data is used. The more sensitive the data is, the higher the severity of each risk. The function then checks for the users input on security measures set in place. The variable, when a security measure is checked by the user, takes the value of *1*. If the data process is protected by encryption, pseudonymisation

or anonymisation, the severity level is reduced by a third. This value can later be varied for each security measure, allowing for a more precise calculation of the severity level. Similar to the variable **prob_level**, the **sev_level** is then scaled to a value between zero and one. This is done by dividing the variable by 10 and multiplying it by the general severity level of the risk, as identified in subsubsection 5.7.2. The severity levels of each risk are then stored in a list and returned to the function *evaluate*, shown in Listing 14, which plots the data on the risk matrix.

```
def evaluate():
    # call function to calculate probability
    # values
    prob = calc_prob()

    # call function to calculate severity values
    sev = calc_sev()

    # store respective risk names to
    name = ['Illegitimate Access', 'Unwanted change', 'Disappearance']

    # place datapoints
    plot.scatter(prob, sev, color='r')
    # annotate datapoints with risk names
    for i, txt in enumerate(name):
        plot.annotate(txt, (prob[i], sev[i]), textcoords="offset points",
            xytext=(0,0.1), ha='center')
    # draw the graph
    matrix.draw()

    # if sensitive data is used, create a warning message
    # for the user
    if rad_sens_data.get() < 10:
        tk.messagebox.showwarning(title=None, message="This is a high risk data process")
    else:
        pass
```

Listing 14: Evaluate risk

Evaluate triggers the two functions which calculate probability and severity and takes the return values as input for the risk matrix. It plots the values and annotates each value with the respective risk name stored in the list **name**. The visualisation of the risks on the risk matrix is shown in Figure 6.9.

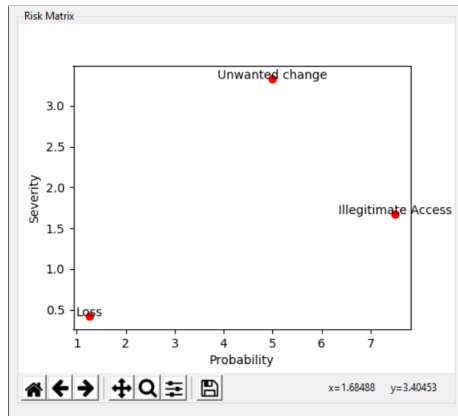


Figure 6.9: Evaluated risks on risk matrix

Furthermore it checks if the data process uses *Sensitive data* and, if yes, it triggers a warning message as displayed in Figure 6.10 to the user. If the process deals with *Personal data*, a medium level warning message as shown in Figure 6.11 is displayed. These warning messages aim to create a higher awareness for more sensitive data processes.

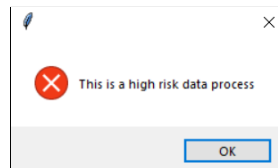


Figure 6.10: Sensitive data process warning

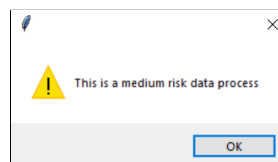


Figure 6.11: Personal data process warning

Every evaluated data process is displayed by a distinct colour, allowing the user to perform up to five assessments of data processes in one session and compare the resulting risk levels. The assessment of two data processes is shown in Figure 6.12.

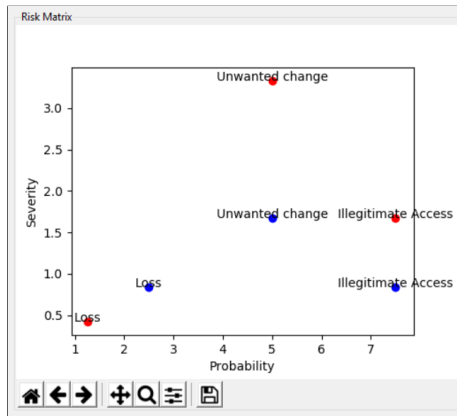


Figure 6.12: Two risks evaluated

The second tab, depicted in Figure 6.13, allows the user to create user stories which are subsequently sent to an API of the web service.

The GUI for creating a user story includes the following fields and controls:

- Title of the User Story:** Text input field.
- As a:** Text input field.
- I want:** Text input field.
- So that:** Text input field.
- Acceptance Criteria:** Text input field.
- ☐ **Is this a Privacy User Story?**
- Privacy Description:** Text input field.
- Risk Level:** Radio button group with options: Low, Medium, High, None.
- ☐ **Does this User Story affect previous work**
- Severity Description:** Text input field.
- Create:** Button to submit the form.

Figure 6.13: GUI Create Story

The input is read once the user submits their input by clicking on the button *Create Story*. The user story is then created by the function *createStory()*, which transforms the input into the dictionary format for the REST API. Sub-

sequently, the function creates a POST-request with the information stored in the dictionary **story** as well as the **token**, which is required for authenticating the user. Furthermore, the function checks whether the created user story is flagged as a *severe* story. If the story is flagged as severe, a privacy user story for reiterating the PIA is automatically triggered. The reiteration over the PIA is necessary because a severe story has an impact on the previous privacy risk evaluation of the project. Therefore, when a user story is flagged as a severe story, it automatically creates a privacy user story for reviewing the privacy impact assessment by calling the function *createPIA()*. The code that performs this is shown in Listing 15.

```
def createStory():
    # check if it's a privacy user story
    if var_privacy.get() == 1:
        priv_bool = True
    else:
        priv_bool = False
    # check if it's a severe user story
    if var_severity.get() == 1:
        sev_bool = True
        # call function to create PIA review story
        createPIA()
    else:
        sev_bool = False
    # set risk level as per users input
    if radRisk.get() == 0:
        risk = 'Low'
    elif radRisk.get() == 1:
        risk = 'Medium'
    elif radRisk.get() == 2:
        risk = 'High'
    else:
        risk = ''
    # create the dictionary with respective values
    story={
        'title' : title.get(),
        'description' : as_a.get() + " " + i_want.get() + " " +
            so_that.get(),
        'owner': '',
        'requirements': acc_crit.get(),
        'priv_flag' : priv_bool,
        'priv_descr' : priv_descr.get(),
        'sev_flag': sev_bool,
        'sev_descr': sev_descr.get(),
        'status': 'PB',
        'risk_level': risk,
    }
    # call function that sends API request with relevant
    # information
    r = request.create_story(token, story)
```

Listing 15: Create user story

This enables the user to directly create relevant user stories after analysing a data process.

7 Discussion of Results

The framework conceptualised in this master’s thesis provides distributed software development teams utilising agile methods with an applicable model to incorporate privacy protection. As the author identified a lack in previous scientific research in this area, the methods of including security into agile software development methods are analysed. The concepts introduced in the various approaches are reviewed in regards to their respective uses in the field of privacy. This results in a high-level view of components and structures to implement privacy protection into the agile world, but results in a possible lack of focus on the exact requirements, e.g. user stories, of protecting a product. However, as the exact requirements to protect a system vary greatly, the author chose to design the core components as required by the GDPR and to focus on the structural architecture required in an agile development method to include privacy protection. Further, this process enables adaptability of the components required to comply with the GDPR for other legislators. The main incentive to comply with the GDPR in agile methods is created by evaluating the respective risk level of each data process. Whilst this approach ensures that the necessity of protecting data processes is clearly visualised, it also requires an in-depth understanding of the project’s environment by the assessor.

The framework identifies three core components to *Privacy Management* (Knowledge, Requirements and Tracking), with an emphasis lying on the *Knowledge* component, which is also embodied by agile methods. The introduced Privacy Expert performs a vital role of defining, assessing and evaluating the developed work regarding their level of privacy protection. As part of the Privacy Management Forum (PMF), the Privacy Expert identifies necessary privacy requirements and tracks these through introduced control mechanisms. The introduction of an expert forum to discuss severe privacy matters should be handled as the final escalation platform. This could lead to an increase in overhead, however, the authors intention for the PMF is to create a knowledge platform in respect to privacy protection which only meets when a privacy risk with an effect on the entire organisation is identified. This trade-off should ensure the added overhead remains at an acceptable level for the agile methodology. However, the framework will not be able to provide the necessary level of privacy protection without the required skills and training of the involved persons. One addition to aid the knowledge component is the creation of a knowledge base, e.g. a repository, of all performed privacy user stories to ensure the experience of previous work is retained. This repository is developed in the proof of concept and requires further evaluation once applied.

The second component, *Requirements*, identifies the relevant data processes through the performance of Privacy Impact Assessments and the definition of privacy user stories. The iterative PIA designed in this framework is the vital component in evaluating data processes and privacy risks to ensure a holistic view of all privacy requirements of ones project is created. Privacy user stories are grouped semantically to ensure the various kinds of protection measures are

distinguished and included in the development life cycle. The author selected this semantic grouping because it has proven a vital addition in the parallel domain of agile security (Ahola et al., 2014). Therefore the author is content the benefits are translatable to the domain of privacy protection, but it remains to be evaluated once deployed.

The third component, *Tracking*, ensures all work is visualised in respect to the risk level and completed as per the defined requirements by performing regular checks in control points. The visualisation is performed in an adaptation of the haptic boards implemented by agile methods which creates a distinct additional value to the existing artefacts. This ensures the evaluated residual risk levels of each user story remains visible. Control points, the second key part of this component, are required in order to evaluate the developed solutions. The author proposes to enhance existing control points of agile methods with additions such as a privacy protection checklist to ensure adequate assessment whilst remaining as agile as possible. Subsequently, the core components are successfully deployed on the agile method Scrum.

The concept of *Privacy by Design*, a legal requirement as per GDPR, is utilised through the introduction of the *Sprint-Zero*, a timeframe of two to four weeks at the very beginning of a project which focuses on the privacy requirements of the respective project. One of the core tasks to complete in the Sprint-Zero is an initial Privacy Impact Assessment (PIA). This thesis develops an iterative model for PIAs to suit the agile methodology and to create a procedure with the purpose of identifying and assessing data processes regarding their privacy risk level. Whether the iteration cycle proposed in this thesis is adequate remains to be seen, but, as it is designed to be easily adaptable, it should be applicable to most privacy sensitive projects.

Finally, a prototype to prove the concept of the framework in the environment of distributed teams is implemented. The goal of the prototype is to assist the privacy expert in assessing data processes as per the GDPR and to create relevant privacy user stories. The visualisation of the data processes on a simplistic risk matrix is chosen to create an initial design of a prioritisation of risks, as a more extensive assessment is out of scope for this thesis. The additional goal of proving the defined components of the framework worthwhile to a distributed team is met through the general design of the prototype as it enables access to and use of the key functionalities from any location with access to the internet.

7.1 Limitations

The deployment of the framework on other agile development methods than Scrum will require adaptation and translation of the introduced roles, artefacts and control points. However, the initial design should provide a solid basis for deploying this framework in any agile software development method. The

designed framework focuses on the European Union as legislation, specifically regarding the implementation of the GDPR. Although the core concepts of the designed framework may be utilised in ensuring privacy protection as defined by other legislations, the framework will require adaptations to include the respective legislation. These adaptations will most likely include the areas of privacy impact and privacy risk assessment. Furthermore, the GDPR defines privacy as data protection, which is especially suitable in the context of software development, however this definition might vary in other legislations and would therefore also require re-evaluation.

The basis for this master's thesis lies mainly on the theoretical discussion of evaluating risks, incorporating security into agile software developments and assessing legal requirements. Combining the analysis of these three areas into the proposed framework is chosen due to a lack of existing research in the novel area of privacy protection. Therefore, further research, especially including assessment of field studies, is necessary.

7.2 Further Research

This thesis identified a lack of research in the domain of *Privacy Protection*, leading to the general necessity of further research on the incorporation of adequate privacy protection in software development. The framework designed in this thesis can be used as a basis for further research into different legislations other than the European Union. Furthermore there is a need to assess how software development teams have actually implemented privacy protection since the introduction of the GDPR. A study on the implementation of this framework would prove a vital contribution towards its legitimacy and goal of aiding every agile software development method with incorporating privacy protection. Such a field study should also focus on the analysis of the iterative model of the Privacy Impact Assessment. Another possibility for further research is the analysis of security measures on the risk level as well as further enhancement of the prototype in its functionalities. This could include a separate view for the developers or the visualisation of analysed risks on a risk matrix. Interestingly, the implementation of the concept of Privacy by Design (Cavoukian, 2009) has not been researched thoroughly yet. This implementation would be a vital addition to a privacy sensitive project, asking for research on whether the Sprint-Zero, as utilised in this thesis, is an adequate concept to fulfil the initial requirements. Therefore, it is of high importance that further research in the field of privacy and data protection is made to create the necessary training certifications. This ensures the required knowledge is transmitted and privacy sensitive topics can be developed securely.

8 Conclusion

Since the introduction of the GDPR in May 2018, data controllers and processors are required by law, in the European Union, to actively protect every individual's rights. The business need of deploying privacy protection in the real world has increased in recent years due to the legal fines imposed by the legislators on companies in breach with the GDPR. The framework in this thesis provides the basis for privacy protection with the European Union as legislator and helps to face the added challenges posed by the modern work environment of distributed teams. The problem of integrating privacy protection into an agile method lacks previous research, thus secure agile software development methods are analysed as a basis for including privacy protection into agile methods. Subsequently, the identified components are reviewed and adapted regarding their use in protecting privacy. Further, the process of Privacy Impact Assessments is analysed and modified to suit the iterative spiral model of agile development. After an initial design of the framework on an abstract level for agile software development methods in general, the framework is subsequently deployed on the method Scrum. The introduced framework incorporates the concept of Privacy by Design and utilises the iterative nature of agile methodologies for continuous integration of privacy protection in the development life cycle. This is performed by identifying relevant groups of privacy user stories, regular iteration of PIAs and analysing performed work through control points. Furthermore, the main incentive of privacy protection is identified as the imposed risk when neglecting adequate protection of a data process. Therefore, a lightweight risk assessment of data processes as per the data characteristics of the GDPR is designed and visualisation measures are introduced to ensure high visibility of identified risks. Finally, a proof of concept in the form of a prototype is developed to support the proposed framework. The mobile work environment of distributed teams is taken into perspective by digitising the previously designed artefacts, thus ensuring teams can continue working together without the necessity of working in the same office.

Because the GDPR has created a general awareness on protecting personal data of individuals in society, the landscape in other legislations will most likely change towards a higher level of privacy protection in the upcoming years. Therefore, the Privacy Protection Framework for Agile Software Development Methods in Distributed Teams such as developed in this thesis are deployed and will lead to further research in the domain of privacy protection.

References

- Abrahamsson, P., Salo, O., Ronkainen, J. & Warsta, J. (2017). Agile software development methods: Review and analysis. *arXiv preprint arXiv:1709.08439*.
- Ahmad, M. O., Markkula, J. & Oivo, M. (2013). Kanban in software development: A systematic literature review. In *2013 39th euromicro conference on software engineering and advanced applications* (pp. 9–16).
- Ahola, J., Frühwirth, C., Helenius, M., Kutvonen, L., Myllylahti, J., Nyberg, T., ... others (2014). Handbook of the secure agile software development life cycle. *University of Oulu*.
- Al-Heeti, A. (2019, Jan). *Amazon has sold more than 100 million alexa devices*. CNET. Retrieved from <https://www.cnet.com/news/amazon-has-sold-more-than-100-million-alexa-devices/>
- Altenbach, T. J. (1995). *A comparison of risk assessment techniques from qualitative to quantitative* (Tech. Rep.). Lawrence Livermore National Lab., CA (United States).
- Anderson, D. J. (2010). *Kanban: successful evolutionary change for your technology business*. Blue Hole Press.
- ANNSI & AMRAE. (2019). *Controlling the digital risk - the trust advantage*. *Art. 1 gdpr - subject-matter and objectives*. (2016, Aug). Retrieved from <https://gdpr-info.eu/art-1-gdpr/>
- Art. 25 gdpr - data protection by design and by default*. (2018, Nov). Retrieved from <https://gdpr.eu/article-25-data-protection-by-design/>
- Art. 35 gdpr - data protection impact assessment*. (2018, Mar). Retrieved from <https://gdpr-info.eu/art-35-gdpr/>
- Balaji, S. & Murugaiyan, M. S. (2012). Waterfall vs. v-model vs. agile: A comparative study on sdlc. *International Journal of Information Technology and Business Management*, 2(1), 26–30.
- Beck, K. (1998). Extreme programming: A humanistic discipline of software development. In *International conference on fundamental approaches to software engineering* (pp. 1–6).
- Beck, K. (2000). *Extreme programming explained: embrace change*. addison-wesley professional.
- Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... others (2001a). *The agile manifesto*.
- Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., ... others (2001b). Manifesto for agile software development.
- Beznosov, K. & Kruchten, P. (2004). Towards agile security assurance. In *Proceedings of the 2004 workshop on new security paradigms* (pp. 47–54).
- Biehl, M. (2016). *Restful api design* (Vol. 3). API-University Press.
- Bisias, D., Flood, M., Lo, A. W. & Valavanis, S. (2012). A survey of systemic risk analytics. *Annu. Rev. Financ. Econ.*, 4(1), 255–296.
- BSI. (2011). *Privacy impact assessment guideline*. Bundesamt für Sicherheit in der Informationstechnik.

- Cambridge english dictionary*. (2020). Retrieved from <https://dictionary.cambridge.org/dictionary/english/privacy>
- Cavoukian, A. (2009). Privacy by design: The 7 foundational principles. *Information and privacy commissioner of Ontario, Canada*, 5.
- CNIL. (2018a). *Privacy impact assessment knowledge bases*. Commission Nationale Informatique and Libertes.
- CNIL. (2018b). *Privacy impact assessment methodology*.
- CNIL. (2018c). *Privacy impact assessment templates*. Commission Nationale Informatique and Libertes.
- CNIL. (2019). *The cnil's restricted committee imposes a financial penalty of 50 million euros against google llc*. Retrieved 2019-05-10, from <https://www.cnil.fr/en/cnils-restricted-committee-imposes-financial-penalty-50-million-euros-against-google-llc>
- Commission, E. (2018). *Reform of eu data protection rules*. Retrieved from https://ec.europa.eu/commission/sites/beta-political/files/data-protection-factsheet-changes_en.pdf
- Danezis, G., Domingo-Ferrer, J., Hansen, M., Hoepman, J.-H., Metayer, D. L., Tirta, R. & Schiffner, S. (2015). Privacy and data protection by design: from policy to engineering. *arXiv preprint arXiv:1501.03726*.
- De Capitani Di Vimercati, S., Foresti, S., Livraga, G. & Samarati, P. (2012). Data privacy: definitions and techniques. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 20(06), 793–817.
- Embrechts, P., Frey, R. & McNeil, A. (2005). Quantitative risk management. *Princeton Series in Finance, Princeton*, 10(4).
- Felstead, A. & Henseke, G. (2017). Assessing the growth of remote working and its consequences for effort, well-being and work-life balance. *New Technology, Work and Employment*, 32(3), 195–212.
- Free vector icons designed by smashicons*. (2020). Retrieved from <https://www.flaticon.com/authors/smashicons>
- Ghani, I., Azham, Z. & Jeong, S. R. (2014). Integrating software security into agile-scrum method. *KSII Transactions on Internet & Information Systems*, 8(2).
- Herath, T., Herath, H. & Bremser, W. G. (2010). Balanced scorecard implementation of security strategies: a framework for it security performance management. *Information Systems Management*, 27(1), 72–81.
- Hevner, A., March, S. T., Park, J. & Ram, S. (2004). Design science research in information systems. *MIS quarterly*, 28(1), 75–105.
- ICO. (2019). *Guide to data protection*. Retrieved 2019-11-25, from <https://ico.org.uk/for-organisations/guide-to-data-protection/>
- Jones, R. L. & Rastogi, A. (2004). Secure coding: building security into the software development life cycle. *Information Systems Security*, 13(5), 29–39.
- Keramati, H. & Mirian-Hosseiniabadi, S.-H. (2008). Integrating software development security activities with agile methodologies. In *2008 ieee/acs international conference on computer systems and applications* (pp. 749–754).

- Krasnow Waterman, K. & Bruening, P. J. (2014). Big data analytics: risks and responsibilities. *International Data Privacy Law*, 4(2), 89–95.
- Landoll, D. J. & Landoll, D. (2005). *The security risk assessment handbook: A complete guide for performing security risk assessments*. CRC Press.
- Li, H., Yu, L. & He, W. (2019). *The impact of gdpr on global technology development*. Taylor & Francis.
- Lindstrom, L. & Jeffries, R. (2004). Extreme programming and agile software development methodologies. *Information systems management*, 21(3), 41–52.
- Liu, S. & Kuhn, R. (2010). Data loss prevention. *IT professional*, 12(2), 10–13.
- Maria, R. E., Rodrigues Jr, L. A. & Pinto, N. A. (2015). Scrums: a model for safe agile development. In *Proceedings of the 7th international conference on management of computational and collective intelligence in digital ecosystems* (pp. 43–47).
- Martin, R. C. (2002). *Agile software development: principles, patterns, and practices*. Prentice Hall.
- Maruping, L. M., Zhang, X. & Venkatesh, V. (2009). Role of collective ownership and coding standards in coordinating expertise in software project teams. *European Journal of Information Systems*, 18(4), 355–371.
- Munassar, N. M. A. & Govardhan, A. (2010). A comparison between five models of software engineering. *International Journal of Computer Science Issues (IJCSI)*, 7(5), 94.
- Munteanu, A. (2006). Information security risk assessment: The qualitative versus quantitative dilemma. In *Managing information in the digital economy: Issues & solutions-proceedings of the 6th international business information management association (ibima) conference* (pp. 227–232).
- Nyffjord, J. & Kajko-Mattsson, M. (2008a). Integrating risk management with software development: State of practice. In *Proceedings of the international multicongference of engineers and computer scientists* (Vol. 2008).
- Nyffjord, J. & Kajko-Mattsson, M. (2008b). Outlining a model integrating risk management and agile software development. In *2008 34th euromicro conference software engineering and advanced applications* (pp. 476–483).
- OAIC. (2014). *Guide to undertaking privacy impact assessments*. Office of the Australian Information Commissioner.
- OWASP. (2011). *Agile software development: Don't forget evil user stories*. Retrieved from https://www.owasp.org/index.php/Agile_Software_Development:_Don{t}_Forget_EVIL_User_Stories
- Regulations, directives and other acts*. (2020). Retrieved 2020-04-20, from https://europa.eu/european-union/eu-law/legal-acts_en
- Rygge, H. & Jøsang, A. (2018). Threat poker: Solving security and privacy threats in agile software development. In *Nordic conference on secure it systems* (pp. 468–483).
- Stoica, M., Ghilic-Micu, B., Mircea, M. & Uscatu, C. (2016). Analyzing agile development-from waterfall style to scrumban. *Informatica Economica*, 20(4).
- Sullivan. (2010). *Microsoft: Security development lifecycle for agile devel-*

- opment. Retrieved from https://www.blackhat.com/presentations/bh-dc-10/Sullivan_Bryan/BlackHat-DC-2010-Sullivan-SDL-Agile-wp.pdf
- Sutherland, J. & Schwaber, K. (2017). *The definitive guide to scrum: The rules of the game*.
- Terpstra, E., Daneva, M. & Wang, C. (2017). Agile practitioners' understanding of security requirements: insights from a grounded theory analysis. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)* (pp. 439–442).
- Touhill, G. J. & Touhill, C. J. (2014). *Cybersecurity for executives: A practical guide*. John Wiley & Sons.
- Vähä-Sipilä, A. (2010). Product security risk management in agile product management. *Stockholm, Sweden*.
- Versionone, C. (2016). *10th annual state of agile report*.
- Versionone, C. (2018). *The 13th annual state of agile report*.
- Voigt, P. & Von dem Bussche, A. (2017). The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing*.
- Williams, L., Meneely, A. & Shipley, G. (2010). Protection poker: The new software security "game". *IEEE Security & Privacy*, 8(3), 14–20.

Appendices

A Web Service

Listing 16: Flask initialisation code (webservice.py)

```
from app import app, db
from app.models import User, Task, Library

@app.shell_context_processor
def make_shell_context():
    return {'db': db, 'User': User, 'Task': Task, 'Library':
            Library}
```

Listing 17: Python initialisation code (__init__.py)

```
from flask import Flask
from config import Config
from flask_sqlalchemy import SQLAlchemy
from flask_migrate import Migrate
from flask_login import LoginManager
from flask_bootstrap import Bootstrap

app = Flask(__name__)
app.config.from_object(Config)
db = SQLAlchemy(app)
migrate = Migrate(app, db)
login = LoginManager(app)
login.login_view = 'login'
bootstrap = Bootstrap(app)

from app import routes, models
```

Listing 18: Authentication code (auth.py)

```
from flask import g, jsonify
from flask_httpauth import HTTPBasicAuth
from app.models import User
from werkzeug.http import HTTP_STATUS_CODES
from flask_httpauth import HTTPTokenAuth

basic_auth = HTTPBasicAuth()
token_auth = HTTPTokenAuth()

@basic_auth.verify_password
def verify_password(username, password):
    user = User.query.filter_by(username=username).first()
    if user is None:
        return False
    g.current_user = user
    return user.check_password(password)
```

```

@basic_auth.error_handler
def basic_auth_error():
    payload = {'error': HTTP_STATUS_CODES.get(401, 'Unknown error')}
    response = jsonify(payload)
    response.status_code = 401
    return response

@token_auth.verify_token
def verify_token(token):
    g.current_user = User.check_token(token) if token else None
    return g.current_user

@token_auth.error_handler
def token_auth_error():
    payload = {'error': HTTP_STATUS_CODES.get(401, 'Unknown error')}
    response = jsonify(payload)
    response.status_code = 401
    return response

```

Listing 19: Flask configuration file (config.py)

```

#configuration file for flask

import os
basedir = os.path.abspath(os.path.dirname(__file__))

class Config(object):

    SECRET_KEY = os.environ.get('SECRET_KEY') or 'you-will-never-guess'

    SQLALCHEMY_DATABASE_URI = os.environ.get('DATABASE_URL') or \
        'sqlite:/// ' + os.path.join(basedir, 'app.db')
    SQLALCHEMY_TRACK_MODIFICATIONS = False

```

Listing 20: Routing code (routes.py)

```

from flask import render_template, flash, redirect, request, url_for, jsonify, g, abort
from app import app, db
from app.forms import LoginForm, RegistrationForm, addTaskForm, editTaskForm, editRepoForm
from flask_login import current_user, login_user, logout_user, login_required
from app.models import User, Task, Library
from werkzeug.urls import url_parse
from app.api import tokens
from werkzeug.http import HTTP_STATUS_CODES
from app.auth import token_auth
from werkzeug.utils import secure_filename

def error_response(status_code, message=None):
    payload = {'error': HTTP_STATUS_CODES.get(status_code, 'Unknown error')}
    if message:

```

```

        payload['message'] = message
    response = jsonify(payload)
    response.status_code = status_code
    return response

def bad_request(message):
    return error_response(400, message)

@app.route('/')
@app.route('/index')
@login_required
def index():

    db_tasks = current_user.tasks.all()
    #call return_template from flask to show index.html
    #pass tasks of current user to template
    return render_template('index.html', title='Home', tasks=
        db_tasks)

@app.route('/login', methods=['GET', 'POST'])
def login():
    if current_user.is_authenticated:
        return redirect(url_for('index'))
    form = LoginForm()
    if form.validate_on_submit():
        user = User.query.filter_by(username=form.username.data).
            first()
        if user is None or not user.check_password(form.password.
            data):
            flash('Invalid username or password')
            return redirect(url_for('login'))
        login_user(user, remember=form.remember_me.data)
        next_page = request.args.get('next')
        if not next_page or url_parse(next_page).netloc != '':
            next_page = url_for('index')
        return redirect(next_page)
    return render_template('login.html', title='Sign In', form=
        form)

@app.route('/logout')
def logout():
    logout_user()
    return redirect(url_for('index'))

#render addTask.html. take data entered into form on addTask.html
and create a task out of the information
@app.route('/addTask', methods=['GET', 'POST'])
@login_required
def addTask():
    form = addTaskForm()
    if form.validate_on_submit():
        task = Task()
        task.user_id = current_user.id
        task.title = form.title.data
        task.description = form.description.data
        task.fileupload = form.more_descr.data

```

```

        task.owner = form.owner.data
        task.requirements = form.requirements.data
        task.priv_flag = form.priv_flag.data
        task.priv_descr = form.priv_descr.data
        task.sev_flag = form.sev_flag.data
        task.sev_descr = form.sev_descr.data
        task.status = "PB"
        task.risk_level = form.risk_level.data
        db.session.add(task)
        db.session.commit()
        flash('Your task has been created.')
        return redirect(url_for('addTask'))
    elif request.method == 'GET':
        #falls etwas beim anzeigen der task angezeigt werden soll,
        f r edit wichtig
        # form.title.data = current_user.username
        return render_template('addTask.html', title='Add Task',
                               form=form)

#render editTask.html. take data entered into form on editTask.
html and create a task out of the information
@app.route('/editTask/<int:id>', methods=['GET', 'POST'])
@login_required
def editTask(id):
    form = editTaskForm()
    task = Task.query.get(id)
    #check if current user is the author of task and therefore
    authorised to edit
    if task.user_id != current_user.id:
        return bad_request('Only the author may edit their tasks')
    #on form submit take input data and add to task and commit to
    db
    if form.validate_on_submit():
        #filename = secure_filename(form.fileupload.data.filename)

        task.user_id = current_user.id
        task.title = form.title.data
        task.description = form.description.data
        task.fileupload = form.more_descr.data
        task.owner = form.owner.data
        task.requirements = form.requirements.data
        task.priv_flag = form.priv_flag.data
        task.priv_descr = form.priv_descr.data
        task.sev_flag = form.sev_flag.data
        task.sev_descr = form.sev_descr.data
        task.status = form.status.data
        task.risk_level = form.risk_level.data
        db.session.add(task)
        db.session.commit()
        flash('Your task has been edited.')
        return redirect(url_for('index'))
    #show information of selected task to edit in the respective
    form fields
    elif request.method == 'GET':
        form.title.data = task.title
        form.description.data = task.description
        form.more_descr.data = task.fileupload

```

```

        form.owner.data = task.owner
        form.requirements.data = task.requirements
        form.priv_flag.data = task.priv_flag
        form.priv_descr.data = task.priv_descr
        form.sev_flag.data = task.sev_flag
        form.sev_descr.data = task.sev_descr
        form.status.data = task.status
        form.risk_level.data = task.risk_level
    return render_template('editTask.html', title='Edit Task',
                           form=form)

#render editRepo.html. take data entered into form on editRepo.
html and edit the existing task and save to db
@app.route('/editRepo/<int:id>', methods=['GET', 'POST'])
@login_required
def editRepo(id):
    form = editRepoForm()
    repo = Library.query.get(id)
    #check if current user is the author of task and therefore
    authorised to edit
    if repo.user_id != current_user.id:
        return bad_request('Only the author may edit their tasks')
    #on form submit take input data and add to task and commit to
    db
    if form.validate_on_submit():
        repo.user_id = current_user.id
        repo.title = form.title.data
        repo.description = form.description.data
        repo.fileupload = form.more_descr.data
        repo.owner = form.owner.data
        repo.requirements = form.requirements.data
        repo.priv_flag = form.priv_flag.data
        repo.priv_descr = form.priv_descr.data
        repo.sev_flag = form.sev_flag.data
        repo.sev_descr = form.sev_descr.data
        repo.risk_level = form.risk_level.data
        db.session.add(repo)
        db.session.commit()
        flash('Your task has been edited.')
        return redirect(url_for('repo'))
    #show information of selected task to edit in the respective
    form fields
    elif request.method == 'GET':
        form.title.data = repo.title
        form.description.data = repo.description
        form.more_descr.data = repo.fileupload
        form.owner.data = repo.owner
        form.requirements.data = repo.requirements
        form.priv_flag.data = repo.priv_flag
        form.priv_descr.data = repo.priv_descr
        form.sev_flag.data = repo.sev_flag
        form.sev_descr.data = repo.sev_descr
        form.risk_level.data = repo.risk_level
    return render_template('editRepo.html', title='Edit Repository
        Task',
                           form=form)

```

```

#display all tasks saved to repository for a user on repo.html
@app.route('/repo')
@login_required
def repo():
    #save all tasks linked to user from repo in db_library
    db_library = current_user.repo.all()
    #call return_template from flask to show repo.html
    #pass tasks of current user stored in db_library to template
    return render_template('repo.html', title='Repository', tasks=
        db_library)

#save the task to library for future reference
@app.route('/saveToRepo/<int:id>', methods=['GET', 'POST'])
@login_required
def saveToRepo(id):
    task = Task.query.get(id)
    repoTask = Library()
    #check if current user is the author of task and therefore
    authorised to save to Library
    if task.user_id != current_user.id:
        return bad_request('Only the author may save their tasks
            to their Library')
    else:
        repoTask.title = task.title
        repoTask.description = task.description
        repoTask.fileupload = task.fileupload
        repoTask.owner = task.owner
        repoTask.requirements = task.requirements
        repoTask.priv_flag = task.priv_flag
        repoTask.priv_descr = task.priv_descr
        repoTask.sev_flag = task.sev_flag
        repoTask.sev_descr = task.sev_descr
        repoTask.status = task.status
        repoTask.risk_level = task.risk_level
        repoTask.user_id = task.user_id
        db.session.add(repoTask)
        db.session.commit()
    return redirect(url_for('index'))

#save the task to library for future reference
@app.route('/saveToBoard/<int:id>', methods=['GET', 'POST'])
@login_required
def saveToBoard(id):
    repoTask = Library.query.get(id)
    task = Task()
    #check if current user is the author of task and therefore
    authorised to save to Library
    if repoTask.user_id != current_user.id:
        return bad_request('Only the author may add their tasks to
            the Privacy Board')
    else:
        task.title = repoTask.title
        task.description = repoTask.description
        task.fileupload = repoTask.fileupload
        task.owner = repoTask.owner
        task.requirements = repoTask.requirements

```

```

        task.priv_flag = repoTask.priv_flag
        task.priv_descr = repoTask.priv_descr
        task.sev_flag = repoTask.sev_flag
        task.sev_descr = repoTask.sev_descr
        task.status = "PB"
        task.risk_level = repoTask.risk_level
        task.user_id = repoTask.user_id
        db.session.add(task)
        db.session.commit()
    return redirect(url_for('repo'))

#delete a task
@app.route('/deleteTask/<int:id>')
@login_required
def deleteTask(id):
    task = Task.query.get(id)
    if task.author != current_user:
        return bad_request('Only the author may edit their tasks')
    Task.query.filter_by(id=id).delete()
    db.session.commit()
    return redirect(url_for('index'))

#delete a repo entry
@app.route('/deleteRepo/<int:id>')
@login_required
def deleteRepo(id):
    repo = Library.query.get(id)
    if repo.author != current_user:
        return bad_request('Only the author may edit their tasks')
    Library.query.filter_by(id=id).delete()
    db.session.commit()
    return redirect(url_for('repo'))

@app.route('/register', methods=['GET', 'POST'])
def register():
    if current_user.is_authenticated:
        return redirect(url_for('index'))
    form = RegistrationForm()
    if form.validate_on_submit():
        user = User(username=form.username.data, email=form.email.
            data)
        user.set_password(form.password.data)
        db.session.add(user)
        db.session.commit()
        flash('Congratulations, you are now a registered user!')
        return redirect(url_for('login'))
    return render_template('register.html', title='Register', form
        =form)

#now follow restful Apis for the pia tool
@app.route('/users/<int:id>', methods=['GET'])
@token_auth.login_required
def get_user(id):
    return jsonify(User.query.get(id).to_dict())

@app.route('/users', methods=['POST'])
@token_auth.login_required

```

```

def create_user():
    data = request.get_json() or {}
    if 'username' not in data or 'email' not in data or 'password'
        not in data:
        return bad_request('must include username, email and
            password fields')
    if User.query.filter_by(username=data['username']).first():
        return bad_request('please use a different username')
    if User.query.filter_by(email=data['email']).first():
        return bad_request('please use a different email address')
    user = User()
    user.from_dict(data, new_user=True)
    db.session.add(user)
    db.session.commit()
    response = jsonify(user.to_dict())
    response.status_code = 201
    response.headers['Location'] = url_for('get_user', id=user.id)
    return response

@app.route('/users/<int:id>', methods=['PUT'])
@token_auth.login_required
def update_user(id):
    user = User.query.get_or_404(id)
    data = request.get_json() or {}
    if 'username' in data and data['username'] != user.username
        and \
            User.query.filter_by(username=data['username']).first
                ():
        return bad_request('please use a different username')
    if 'email' in data and data['email'] != user.email and \
        User.query.filter_by(email=data['email']).first():
        return bad_request('please use a different email address')
    user.from_dict(data, new_user=False)
    db.session.commit()
    return jsonify(user.to_dict())

#restful api to get tasks for current user
@app.route('/tasks', methods=['GET'])
@token_auth.login_required
def get_task():
    tasks = Task.query.all()
    result = {}
    for task in tasks:
        if task.author == g.current_user:
            result[task.id] = task.title, task.description, task.
                owner, task.requirements
    return jsonify(result)

"""
#restful api to get a specific task for current user
@app.route('/tasks/<int:id>', methods=['GET'])
@token_auth.login_required
def get__specific_task(id):
    task = Task.query.get(id)
    if task is None:
        return bad_request('Invalid task')
    if task.author != g.current_user:
        return bad_request('Only the author may view their tasks')

```

```

        return jsonify(task.to_dict())
    """
    #restful api to create task for current user
    @app.route('/tasks', methods=['POST'])
    @token_auth.login_required
    def create_task():
        data = request.get_json() or {}
        if 'title' not in data or 'description' not in data or '
            requirements' not in data:
            return bad_request('must include title, description and
                requirements fields')
        task = Task()
        task.from_dict(data)
        db.session.add(task)
        db.session.commit()
        response = jsonify(task.to_dict())
        response.status_code = 201
        return response

    #restful api to update task for current user todo!
    @app.route('/tasks/<int:id>', methods=['PUT'])
    @token_auth.login_required
    def update_task(id):
        task = Task.query.get(id)
        data = request.get_json() or {}
        if task is None:
            return bad_request('Invalid task')
        if 'id' in data:
            return bad_request('Invalid operation')
        if task.author != g.current_user:
            return bad_request('Only the author may edit their tasks')
        task.from_dict(data)
        db.session.commit()
        return jsonify(task.to_dict())

    #restful api to delete a task of the current user
    @app.route('/tasks/<int:id>', methods=['DELETE'])
    @token_auth.login_required
    def delete_task(id):
        task = Task.query.get(id)
        if task is None:
            return bad_request('This task does not exist')
        if task.author != g.current_user:
            return bad_request('Only the author may edit their tasks')
        Task.query.filter_by(id=id).delete()
        db.session.commit()
        response = {'Delete': 'success'}
        return jsonify(response)

```

Listing 21: Code for User forms (forms.py)

```

from flask_wtf import FlaskForm
from flask_wtf.file import FileField
from wtforms import StringField, PasswordField, BooleanField,
    SubmitField, RadioField, TextAreaField
from wtforms.validators import DataRequired, ValidationError,

```

```

        DataRequired, Email, EqualTo
from app.models import User

class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    password = PasswordField('Password', validators=[DataRequired()])
    remember_me = BooleanField('Remember Me')
    submit = SubmitField('Sign In')

class RegistrationForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired()])
    email = StringField('Email', validators=[DataRequired(), Email()])
    password = PasswordField('Password', validators=[DataRequired()])
    password2 = PasswordField(
        'Repeat Password', validators=[DataRequired(), EqualTo('password')])
    submit = SubmitField('Register')

    def validate_username(self, username):
        user = User.query.filter_by(username=username.data).first()
        if user is not None:
            raise ValidationError('Please use a different username.')

    def validate_email(self, email):
        user = User.query.filter_by(email=email.data).first()
        if user is not None:
            raise ValidationError('Please use a different email address.')

class addTaskForm(FlaskForm):
    title = StringField('Title', validators=[DataRequired()])
    description = StringField('Description', validators=[DataRequired()])
    more_descr = TextAreaField('Give more information')
    owner = StringField('Story Owner')
    requirements = StringField('Acceptance Criteria', validators=[DataRequired()])
    priv_flag = BooleanField('Privacy Flag')
    risk_level = RadioField('Privacy Risk Level', choices = [('none', 'None'), ('low', 'Low'), ('medium', 'Medium'), ('high', 'High')])
    priv_descr = StringField('Privacy Issue Description')
    sev_flag = BooleanField('Privacy Severity Flag')
    sev_descr = StringField('Privacy Severity Description')
    submit = SubmitField('Add Task')

class editTaskForm(FlaskForm):
    title = StringField('Title', validators=[DataRequired()])
    description = StringField('Short Description', validators=[DataRequired()])

```

```

more_descr = TextAreaField('Give more information')
owner = StringField('Story Owner')
status = RadioField('Status', choices = [('PB', 'Product
      Backlog'), ('SB', 'Sprint Backlog'), ('IP', 'In Progress')
      , ('RV', 'Review'), ('DN', 'Done')])
requirements = StringField('Acceptance Criteria', validators=[
      DataRequired()])
priv_flag = BooleanField('Privacy Flag')
risk_level = RadioField('Privacy Risk Level', choices = [('
      none', 'None'), ('low', 'Low'), ('medium', 'Medium'), ('
      high', 'High')])
priv_descr = StringField('Privacy Issue Description')
sev_flag = BooleanField('Privacy Severity Flag')
sev_descr = StringField('Privacy Severity Description')
submit = SubmitField('Submit')

class editRepoForm(FlaskForm):
    title = StringField('Title', validators=[DataRequired()])
    description = StringField('Description', validators=[
      DataRequired()])
    more_descr = TextAreaField('Give more information')
    owner = StringField('Story Owner')
    requirements = StringField('Acceptance Criteria', validators=[
      DataRequired()])
    priv_flag = BooleanField('Privacy Flag')
    risk_level = RadioField('Risk Level', choices = [('none', '
      None'), ('low', 'Low'), ('medium', 'Medium'), ('high', '
      High')])
    priv_descr = StringField('Privacy Description')
    sev_flag = BooleanField('Severity Flag')
    sev_descr = StringField('Severity Description')
    submit = SubmitField('Submit')

```

Listing 22: Code for database (models.py)

```

from app import db, login
from datetime import datetime
from werkzeug.security import generate_password_hash,
    check_password_hash
from flask_login import UserMixin
import base64
from datetime import datetime, timedelta
import os
from flask import g

@login.user_loader
def load_user(id):
    return User.query.get(int(id))

#create the Class User
class User(UserMixin, db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(64), index=True, unique=True)
    email = db.Column(db.String(120), index=True, unique=True)
    password_hash = db.Column(db.String(128))
    token = db.Column(db.String(32), index=True, unique=True)
    token_expiration = db.Column(db.DateTime)

```

```

counter = db.Column(db.Integer)
tasks = db.relationship('Task', backref='author', lazy='dynamic')
repo = db.relationship('Library', backref='author', lazy='dynamic')

def __repr__(self):
    return '<User {}>'.format(self.username)

def set_password(self, password):
    self.password_hash = generate_password_hash(password)

def check_password(self, password):
    return check_password_hash(self.password_hash, password)

#transform to dictionary format for json
def to_dict(self):
    data = {
        'id': self.id,
        'username': self.username,
        'email' : self.email
    }
    return data

#transform from dictionary format (json-request)
def from_dict(self, data, new_user=False):
    for field in ['username', 'email']:
        if field in data:
            setattr(self, field, data[field])
    if new_user and 'password' in data:
        self.set_password(data['password'])

#generate authentication token
def get_token(self, expires_in=3600):
    now = datetime.utcnow()
    if self.token and self.token_expiration > now + timedelta(
        seconds=60):
        return self.token
    self.token = base64.b64encode(os.urandom(24)).decode('utf-8')
    self.token_expiration = now + timedelta(seconds=expires_in)
    db.session.add(self)
    return self.token

#cancel token
def revoke_token(self):
    self.token_expiration = datetime.utcnow() - timedelta(
        seconds=1)

@staticmethod
def check_token(token):
    user = User.query.filter_by(token=token).first()
    if user is None or user.token_expiration < datetime.utcnow():
        return None
    return user

```

```

#create class Task with attributes
class Task(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(140))
    description = db.Column(db.String(250))
    fileupload = db.Column(db.String(350))
    owner = db.Column(db.String(140))
    requirements = db.Column(db.String(250))
    priv_flag = db.Column(db.BOOLEAN)
    priv_descr = db.Column(db.String(250))
    sev_flag = db.Column(db.BOOLEAN)
    sev_descr = db.Column(db.String(250))
    status = db.Column(db.String(140))
    risk_level = db.Column(db.String(140))
    timestamp = db.Column(db.DateTime, index=True, default=
        datetime.utcnow)
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'))

    #return current task
    def __repr__(self):
        return '<Task {}>'.format(self.title)

    #transform from type dictionary(json request) to db
    def from_dict(self, data):
        #iterate through data to read values for each attribute
        and store in self
        for field in ['id', 'title', 'description', 'owner', '
            requirements', 'priv_flag', 'priv_descr', 'sev_flag',
            'sev_descr', 'status', 'risk_level']:
            if field in data:
                setattr(self, field, data[field])
        #creates relation to current User
        if g.current_user is None:
        user = User.query.get(2)
        setattr(self, 'author', g.current_user)
        else:
        setattr(self, 'author', g.current_user)

    #return Task in dictionary format (for json)
    def to_dict(self):
        data = {
            'id': self.id,
            'title': self.title,
            'description': self.description,
            'owner': self.owner,
            'requirements': self.requirements,
        }
        return data

class Library(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(140))
    description = db.Column(db.String(250))
    fileupload = db.Column(db.String(350))
    owner = db.Column(db.String(140))

```

```

requirements = db.Column(db.String(250))
priv_flag = db.Column(db.BOOLEAN)
priv_descr = db.Column(db.String(250))
sev_flag = db.Column(db.BOOLEAN)
sev_descr = db.Column(db.String(250))
status = db.Column(db.String(140))
risk_level = db.Column(db.String(140))
timestamp = db.Column(db.DateTime, index=True, default=
    datetime.utcnow)
user_id = db.Column(db.Integer, db.ForeignKey('user.id'))

#return current task
def __repr__(self):
    return '<Task {}>'.format(self.title)

#transform from type dictionary (json request) to db
def from_dict(self, data):
    #iterate through data to read values for each attribute
    #and store in self
    for field in ['id', 'title', 'description', 'owner', '
        requirements', 'priv_flag', 'priv_descr', 'sev_flag',
        'sev_descr']:
        if field in data:
            setattr(self, field, data[field])
    #creates new task with handed data
    setattr(self, 'author', g.current_user)

#return Task in dictionary format for json
def to_dict(self):
    data = {
        'id': self.id,
        'title': self.title,
        'description' : self.description,
        'owner': self.owner,
        'requirements': self.requirements,
        'priv_flag' : self.priv_flag,
        'priv_descr' : self.priv_descr,
        'sev_flag': self.sev_flag,
        'sev_descr': self.sev_descr,
    }
    return data

```

Listing 23: Code handling token creation (tokens.py)

```

from flask import jsonify, g
from app import db, app
from app.auth import basic_auth

@app.route('/tokens', methods=['POST'])
@basic_auth.login_required
def get_token():
    token = g.current_user.get_token()
    db.session.commit()
    return jsonify({'token': token})

```

Listing 24: Code for error handling (errors.py)

```

from flask import jsonify
from werkzeug.http import HTTP_STATUS_CODES

def error_response(status_code, message=None):
    payload = {'error': HTTP_STATUS_CODES.get(status_code, '
        Unknown error')}
    if message:
        payload['message'] = message
    response = jsonify(payload)
    response.status_code = status_code
    return response

```

Listing 25: Base HTML-page (base.html)

```

{% extends 'bootstrap/base.html' %}

{% block title %}
    {% if title %}{{ title }} - Privacy RB{% else %}Welcome to
        your Privacy Risk Board{% endif %}
{% endblock %}

{% block navbar %}
<script src="http://ajax.aspnetcdn.com/ajax/knockout/knockout
-2.2.1.js"></script>
<script src="http://ajax.aspnetcdn.com/ajax/jquery/jquery-1.9.0.js
"></script>
<script src="http://netdna.bootstrapcdn.com/twitter-bootstrap
/2.3.2/js/bootstrap.min.js"></script>
<nav class="navbar navbar-default">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle
                collapsed" data-toggle="collapse" data-target=
                "#bs-example-navbar-collapse-1" aria-expanded=
                "false">
                <span class="sr-only">Toggle navigation</span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="{{ url_for('index')
                }}">Privacy RB</a>
        </div>
        <div class="collapse navbar-collapse" id="bs-example-
            navbar-collapse-1">
            <ul class="nav navbar-nav">
                <li><a href="{{ url_for('index') }}">Home</a><
                    /li>
            </ul>
            <ul class="nav navbar-nav">
                <li><a href="{{ url_for('repo') }}">Repository
                    </a></li>
            </ul>
            <ul class="nav navbar-nav navbar-right">
                {% if current_user.is_anonymous %}
                <li><a href="{{ url_for('login') }}">Login</a>
                </li>

```

```

        {% else %}
        <li><a href="{ { url_for('logout') } }">Logout</a></li>
        {% endif %}
    </ul>
</div>
</div>
</nav>
{% endblock %}

{% block content %}
<div class="container">
    {% with messages = get_flashed_messages() %}
    {% if messages %}
        {% for message in messages %}
        <div class="alert alert-info" role="alert">{{ message }}</div>
        {% endfor %}
    {% endif %}
    {% endwith %}

    {# application content needs to be provided in the
    app_content block #}
    {% block app_content %}{% endblock %}
</div>
{% endblock %}

```

Listing 26: Main page (index.html)

```

{% extends "base.html" %}

{% block content %}
<h1>Hello there, {{ current_user.username }}!</h1>
<h2>You have {{ current_user.counter }} Privacy User Stories to
    complete until you need to reiterate over the PIA!</h2>
<table class="table table-bordered">
    <tr>
        <td><b>Product Backlog</b></td>
        <td><b>Sprint Backlog</b></td>
        <td><b>In Progress</b></td>
        <td><b>Review</b></td>
        <td><b>Done</b></td>
        <td><b>Options</b></td>
    </tr>
    {% for task in tasks %}
    <tr>
        {% if task.status == "PB" %}
        {% if task.sev_flag == True %}
        <td style="border-color: rgb(252, 1, 1);">
        {% else %}
        <td style="border-color: rgb(0, 0, 0);">
        {% endif %}
        {% if task.risk_level == "high" %}
        <p style="background-color: crimson;"><b>{{ task.title }}</b>
        {% elif task.risk_level == "medium" %}
        <p style="background-color: lightcoral;"><b>{{ task.

```

```

        title }}</b>
        {% elif task.risk_level == "low" %}
        <p style="background-color: lightpink;"><b>{{ task.
            title }}</b>
        {% else %}
        <p><b>{{ task.title }}</b>
        {% endif %}
        <p>{{ task.owner }}</p></p></p>
    </td>
</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>
    <p> <a href="{{ url_for('editTask', id = task.id) }}">
        Edit</a>
        <a href="{{ url_for('deleteTask', id = task.id) }}">
            Delete</a></p>
    <p><a href="{{ url_for('saveToRepo', id = task.id) }}">
        >Save to Repository</a></p>
</td>
{% elif task.status == "SB" %}
<td></td>
{% if task.sev_flag == True %}
<td style="border-color: rgb(252, 1, 1);">
{% else %}
<td style="border-color: rgb(0, 0, 0);">
{% endif %}
{% if task.risk_level == "high" %}

    <p style="background-color: crimson;"><b>{{ task.title
        }}</b></p>
    {% elif task.risk_level == "medium" %}

    <p style="background-color: lightcoral;"><b>{{ task.
        title }}</b></p>
    {% elif task.risk_level == "low" %}

    <p style="background-color: lightpink;"><b>{{ task.
        title }}</b></p>
    {% else %}

    <p><b>{{ task.title }}</b></p>
    {% endif %}
    <p>{{ task.owner }}</p>
</td>
<td></td>
<td></td>
<td></td>
<td>
    <p> <a href="{{ url_for('editTask', id = task.id) }}">
        Edit</a>
        <a href="{{ url_for('deleteTask', id = task.id) }}">
            >Delete</a></p>
    <p><a href="{{ url_for('saveToRepo', id = task.id) }}">
        >Save to Repository</a></p>
</td>

```

```

{% elif task.status == "IP" %}
<td></td>
<td></td>
{% if task.sev_flag == True %}
<td style="border-color: rgb(252, 1, 1);">
{% else %}
<td style="border-color: rgb(0, 0, 0);">
{% endif %}
{% if task.risk_level == "high" %}

    <p style="background-color: crimson;"><b>{{ task.title
    }}</b></p>
    {% elif task.risk_level == "medium" %}

    <p style="background-color: lightcoral;"><b>{{ task.
    title }}</b></p>
    {% elif task.risk_level == "low" %}

    <p style="background-color: lightpink;"><b>{{ task.
    title }}</b></p>
    {% else %}

    <p><b>{{ task.title }}</b></p>
    {% endif %}
    <p>{{ task.owner }}</p>
</td>
<td></td>
<td></td>
<td>
    <p> <a href="{{ url_for('editTask', id = task.id) }}">
    Edit</a>
    <a href="{{ url_for('deleteTask', id = task.id) }}">
    Delete</a></p>
    <p><a href="{{ url_for('saveToRepo', id = task.id) }}">
    Save to Repository</a></p>
</td>
{% elif task.status == "RV" %}
<td></td>
<td></td>
<td></td>
{% if task.sev_flag == True %}
<td style="border-color: rgb(252, 1, 1);">
{% else %}
<td style="border-color: rgb(0, 0, 0);">
{% endif %}
{% if task.risk_level == "high" %}

    <p style="background-color: crimson;"><b>{{ task.title
    }}</b></p>
    {% elif task.risk_level == "medium" %}

    <p style="background-color: lightcoral;"><b>{{ task.
    title }}</b></p>
    {% elif task.risk_level == "low" %}

    <p style="background-color: lightpink;"><b>{{ task.
    title }}</b></p>

```

```

        {% else %}

        <p><b>{{ task.title }}</b></p>
        {% endif %}
        <p>{{ task.owner }}</p>
    </td>
</td></td>
<td>
    <p> <a href="{{ url_for('editTask', id = task.id) }}">
        Edit</a>
        <a href="{{ url_for('deleteTask', id = task.id) }}">
            Delete</a></p>
    <p><a href="{{ url_for('saveToRepo', id = task.id) }}">
        Save to Repository</a></p>
</td>
{% else %}
<td></td>
<td></td>
<td></td>
<td></td>
{% if task.sev_flag == True %}
<td style="border-color: rgb(252, 1, 1);">
{% else %}
<td style="border-color: rgb(0, 0, 0);">
{% endif %}
{% if task.risk_level == "high" %}

    <p style="background-color: crimson;"><b>{{ task.title }}</b></p>
    {% elif task.risk_level == "medium" %}

    <p style="background-color: lightcoral;"><b>{{ task.title }}</b></p>
    {% elif task.risk_level == "low" %}

    <p style="background-color: lightpink;"><b>{{ task.title }}</b></p>
    {% else %}

    <p><b>{{ task.title }}</b></p>
    {% endif %}
    <p>{{ task.owner }}</p>
</td>
<td>
    <p> <a href="{{ url_for('editTask', id = task.id) }}">
        Edit</a>
        <a href="{{ url_for('deleteTask', id = task.id) }}">
            Delete</a></p>
    <p><a href="{{ url_for('saveToRepo', id = task.id) }}">
        Save to Repository</a></p>
</td>
{% endif %}
</tr>
{% endfor %}
</table>
<p><a href="{{ url_for('addTask') }}">Add Task</a></p>

```

```
{% endblock %}
```

Listing 27: Login page (login.html)

```
{% extends "base.html" %}

{% block content %}
    <h1>Sign In</h1>
    <form action="" method="post" novalidate>
        {{ form.hidden_tag() }}
        <p>
            {{ form.username.label }}<br>
            {{ form.username(size=32) }}<br>
            {% for error in form.username.errors %}
            <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.password.label }}<br>
            {{ form.password(size=32) }}<br>
            {% for error in form.password.errors %}
            <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>{{ form.remember_me() }} {{ form.remember_me.label }}</p>
        <p>{{ form.submit() }}</p>
    </form>

    <p>New User? <a href="{{ url_for('register') }}">Click to
        Register!</a></p>
{% endblock %}
```

Listing 28: Register page (register.html)

```
{% extends "base.html" %}

{% block content %}
    <h1>Register</h1>
    <form action="" method="post">
        {{ form.hidden_tag() }}
        <p>
            {{ form.username.label }}<br>
            {{ form.username(size=32) }}<br>
            {% for error in form.username.errors %}
            <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.email.label }}<br>
            {{ form.email(size=64) }}<br>
            {% for error in form.email.errors %}
            <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.password.label }}<br>
```

```

        {{ form.password(size=32) }}<br>
        {% for error in form.password.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.password2.label }}<br>
        {{ form.password2(size=32) }}<br>
        {% for error in form.password2.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>{{ form.submit() }}</p>
</form>
{% endblock %}

```

Listing 29: Add Task page (addTask.html)

```

{% extends "base.html" %}

{% block content %}
    <h1>Add Task</h1>
    <form action="" method="post">
        {{ form.hidden_tag() }}
        <p>
            {{ form.title.label }}<br>
            {{ form.title(size=32) }}<br>
            {% for error in form.title.errors %}
            <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.description.label }}<br>
            {{ form.description(size=32) }}<br>
            {% for error in form.description.errors %}
            <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.more_descr.label }}<br>
            {{ form.more_descr }}<br>
            {% for error in form.more_descr.errors %}
            <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.owner.label }}<br>
            {{ form.owner(size=32) }}<br>
            {% for error in form.owner.errors %}
            <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.requirements.label }}<br>
            {{ form.requirements(size=32) }}<br>
            {% for error in form.requirements.errors %}
            <span style="color: red;">[{{ error }}]</span>

```

```

        {% endfor %}
    </p>
    <p>
        {{ form.priv_flag.label }}<br>
        {{ form.priv_flag }}<br>
        {% for error in form.priv_flag.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.priv_descr.label }}<br>
        {{ form.priv_descr(size=32) }}<br>
        {% for error in form.priv_descr.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.risk_level.label }}<br>
        {{ form.risk_level }}<br>
        {% for error in form.risk_level.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.sev_flag.label }}<br>
        {{ form.sev_flag }}<br>
        {% for error in form.sev_flag.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.sev_descr.label }}<br>
        {{ form.sev_descr(size=32) }}<br>
        {% for error in form.sev_descr.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>{{ form.submit() }}</p>
</form>
{% endblock %}

```

Listing 30: Edit Task page (editTask.html)

```

{% extends "base.html" %}

{% block content %}
    <h1>Edit Task</h1>
    <form action="" method="post">
        {{ form.hidden_tag() }}
        <p>
            {{ form.title.label }}<br>
            {{ form.title(size=32) }}<br>
            {% for error in form.title.errors %}
            <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
    <p>

```

```

        {{ form.description.label }}<br>
        {{ form.description(size=32) }}<br>
        {% for error in form.description.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.more_descr.label }}<br>
        {{ form.more_descr }}<br>
        {% for error in form.more_descr.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.owner.label }}<br>
        {{ form.owner(size=32) }}<br>
        {% for error in form.owner.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.status.label }}<br>
        {{ form.status(size=32) }}<br>
        {% for error in form.status.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.requirements.label }}<br>
        {{ form.requirements(size=32) }}<br>
        {% for error in form.requirements.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.priv_flag.label }}<br>
        {{ form.priv_flag }}<br>
        {% for error in form.priv_flag.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.priv_descr.label }}<br>
        {{ form.priv_descr(size=32) }}<br>
        {% for error in form.priv_descr.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.risk_level.label }}<br>
        {{ form.risk_level }}<br>
        {% for error in form.risk_level.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.sev_flag.label }}<br>

```

```

        {{ form.sev_flag }}<br>
        {% for error in form.sev_flag.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>
        {{ form.sev_descr.label }}<br>
        {{ form.sev_descr(size=32) }}<br>
        {% for error in form.sev_descr.errors %}
        <span style="color: red;">[{{ error }}]</span>
        {% endfor %}
    </p>
    <p>{{ form.submit() }}</p>
</form>
{% endblock %}

```

Listing 31: Repository page (repo.html)

```

{% extends "base.html" %}

{% block content %}
<h1>Hello there, {{ current_user.username }}!</h1>
<h2>This is the repository of all your saved User Stories.</h2>
<table class="table table-bordered">
    <tr>
        <td><b>User Story</b></td>
        <td><b>Options</b></td>
    </tr>
    {% for task in tasks %}
    <tr>

        {% if task.sev_flag == True %}
        <td style="border-color: rgb(252, 1, 1);">
        {% else %}
        <td style="border-color: rgb(0, 0, 0);">
        {% endif %}
        {% if task.risk_level == "high" %}

            <p style="background-color: crimson;"><b>{{ task.title }}</b>
            {% elif task.risk_level == "medium" %}

            <p style="background-color: lightcoral;"><b>{{ task.title }}</b>
            {% elif task.risk_level == "low" %}

            <p style="background-color: lightpink;"><b>{{ task.title }}</b>
            {% else %}

            <p><b>{{ task.title }}</b>
            {% endif %}
            <p>{{ task.owner }}</p></td></td>

        <td>
            <p> <a href="{{ url_for('editRepo', id = task.id) }}">
                Edit</a></p>
        </td>
    </tr>
    {% endfor %}
</table>

```

```

        <p><a href="{{ url_for('deleteRepo', id = task.id) }}"
        >Delete</a></p>
        <p><a href="{{ url_for('saveToBoard', id = task.id) }}"
        ">Add to Privacy Board</a></p>
    </td>
    {% endfor %}
</table>

{% endblock %}

```

Listing 32: Edit repository page (editRepo.html)

```

{% extends "base.html" %}

{% block content %}
    <h1>Edit Repository Task</h1>
    <form action="" method="post">
        {{ form.hidden_tag() }}
        <p>
            {{ form.title.label }}<br>
            {{ form.title(size=32) }}<br>
            {% for error in form.title.errors %}
            <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.description.label }}<br>
            {{ form.description(size=32) }}<br>
            {% for error in form.description.errors %}
            <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.more_descr.label }}<br>
            {{ form.more_descr }}<br>
            {% for error in form.more_descr.errors %}
            <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.owner.label }}<br>
            {{ form.owner(size=32) }}<br>
            {% for error in form.owner.errors %}
            <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.requirements.label }}<br>
            {{ form.requirements(size=32) }}<br>
            {% for error in form.requirements.errors %}
            <span style="color: red;">[{{ error }}]</span>
            {% endfor %}
        </p>
        <p>
            {{ form.priv_flag.label }}<br>
            {{ form.priv_flag }}<br>
            {% for error in form.priv_flag.errors %}

```

```

        <span style="color: red;">[{{ error }}]</span>
    {% endfor %}
</p>
<p>
    {{ form.priv_descr.label }}<br>
    {{ form.priv_descr(size=32) }}<br>
    {% for error in form.priv_descr.errors %}
    <span style="color: red;">[{{ error }}]</span>
    {% endfor %}
</p>
<p>
    {{ form.risk_level.label }}<br>
    {{ form.risk_level }}<br>
    {% for error in form.risk_level.errors %}
    <span style="color: red;">[{{ error }}]</span>
    {% endfor %}
</p>
<p>
    {{ form.sev_flag.label }}<br>
    {{ form.sev_flag }}<br>
    {% for error in form.sev_flag.errors %}
    <span style="color: red;">[{{ error }}]</span>
    {% endfor %}
</p>
<p>
    {{ form.sev_descr.label }}<br>
    {{ form.sev_descr(size=32) }}<br>
    {% for error in form.sev_descr.errors %}
    <span style="color: red;">[{{ error }}]</span>
    {% endfor %}
</p>
<p>{{ form.submit() }}</p>
</form>
{% endblock %}

```

B Data Process Analysis Tool

Listing 33: Code of Data Process Analysis Tool (gui.py)

```
import tkinter as tk
from tkinter import ttk
import request
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,
        NavigationToolbar2Tk)
"""
from matplotlib.figure import Figure
import numpy as np
import pandas as pd
"""
import matplotlib.pyplot as plt

class login:

    def __init__(self,root):

        self.root = root
        self.root.title('LOGIN SCREEN')

        #labels with textforms for username and password
        tk.Label(text = ' Username ',font='Times 15').grid(row=1,
            column=1,pady=20)
        self.username = tk.Entry()
        self.username.grid(row=1,column=2,columnspan=10)
        self.username.focus() #sets focus of cursor into this
            entry

        tk.Label(text = ' Password ',font='Times 15').grid(row=2,
            column=1,pady=10)
        self.password = tk.Entry(show='*')
        self.password.grid(row=2,column=2,columnspan=10)

        ttk.Button(text='LOGIN', command=self.login_user).grid(row
            =3,column=2)

        # create a register button and call register_user
        ttk.Button(text="Register", command=self.register_user).
            grid(row=5, column=2)

        #create quit button which exits the loop
        ttk.Button(text='Quit', command=root.quit).grid(row=7,
            column=2)

    def login_user(self):
        '''Call api request to check username and password entered
            are correctly'''
        response = request.user_login(self.username.get(), self.
            password.get())
        #antwort auf api anfrage auf token checken, bedeutet
            anmeldung war erfolgreich
        if "token" in response:
            print(response)
```

```

        #token und usernamen global abspeichern f r
        weiterverarbeitung
        global token
        global username
        username = self.username.get()
        token = response["token"]
        print(token)
        #Destroy the current window
        root.destroy()

        #Open new window
        newroot = tk.Tk()
        pia_tool(newroot)
        newroot.geometry('1300x700')
        newroot.mainloop()

    else:
        print("Fehler:%s"%response)
        #Prompt user that either id or password is wrong
        self.message = tk.Label(text = 'Username or Password
            incorrect. Try again!',fg = 'Red')
        self.message.grid(row=6,column=2)

def register_user(self): #todo call register user api
    pass

class pia_tool:
    def __init__(self,root):
        risk_level = ["Low", "Medium", "High", "None"]
        data_type = ["Common personal data", "Personal data
            perceived as sensitive", "Sensitive personal data"]
        storage_time = ["up to 3 months", "3 to 6 months", "longer
            than 6 months"]
        colors = ["red", "gold", "blue"]
        gather_options = ["Newly collected", "Retrieved from
            existing", "Taken from 3rd party"]
        com_data = ["Civil Status", "Professional life", "
            Connection data"]
        per_data = ["Social security number", "Biometric data", "
            Bank data"]
        sens_data = ["Political views", "Religious views", "
            Criminal records"]
        color = ['red', 'blue', 'black', 'yellow', 'brown']

        self.root = root
        self.root.title('DPA Tool')
        #create tabs
        tabControl = ttk.Notebook(root)
        tab1 = ttk.Frame(tabControl)
        tab3 = ttk.Frame(tabControl)
        #add a tab
        tabControl.add(tab1, text='DPA')
        tabControl.add(tab3, text='User Story')
        #make the tabs visible
        tabControl.pack(expand=1, fill="both")

```

```

# create a frame for displaying in tab1
frame1 = ttk.LabelFrame(tab1, text='Data Process Analysis'
)
frame1.grid(column=0, row=0, padx=8, pady=4)

# tk.Label(frame1, text = ' Hallo ', font='Times 15').grid(
row=1, column=1, pady=20) #row 1
# Label(text = , font='Times 15').grid(row=2, column=1, pady
=20)
# print('test:' + token)

tk.Label(frame1, text = ' Which type of GDPR-specific data
are you dealing with? ', font='Times 12').grid(row=3,
column=1, pady=20) #row 3

# create frame for all common data types
frame_com_data = ttk.LabelFrame(frame1, text='Common Data'
)
frame_com_data.grid(column=0, row=4, padx=8, pady=4) #
common data frame col 0 row 5
rad_com_data1 = tk.IntVar()
rad_com_data2 = tk.IntVar()
rad_com_data3 = tk.IntVar()
# set a value to make sure the radio buttons are unselected
rad_com_data1.set(99)
rad_com_data2.set(99)
rad_com_data3.set(99)
rad_com = [rad_com_data1, rad_com_data2, rad_com_data3 ]
for col in range(len(com_data)):
    curRad = tk.Radiobutton(frame_com_data, text=com_data[
col], variable=rad_com[col], value=col)
    curRad.grid(column=1, row=col, sticky=tk.W)
# create free text entry
tk.Label(frame_com_data, text="Other:", font='Times 10').
grid(row=len(com_data), column=0,)
com_data_text = tk.Entry(frame_com_data)
com_data_text.grid(row=(len(com_data)), column=1, columnspan
=10)

# create frame for all personal data types
frame_pers_data = ttk.LabelFrame(frame1, text='Personal
Data')
frame_pers_data.grid(column=1, row=4, padx=8, pady=4) #
common data frame col 0 row 5
rad_pers_data1 = tk.IntVar()
rad_pers_data2 = tk.IntVar()
rad_pers_data3 = tk.IntVar()
# set a value to make sure the radio buttons are unselected
rad_pers_data1.set(99)
rad_pers_data2.set(99)
rad_pers_data3.set(99)
rad_pers = [rad_pers_data1, rad_pers_data2, rad_pers_data3]
# create radiobuttons for each value in per_data
for col in range(len(per_data)):

```

```

        curRad = tk.Radiobutton(frame_pers_data, text=per_data
                                [col], variable=rad_pers[col], value=col)
        curRad.grid(column=1, row=col, sticky=tk.W)          #
        row 5
tk.Label(frame_pers_data, text="Other:", font='Times 10').
    grid(row=len(per_data), column=0,)
com_data_text = tk.Entry(frame_pers_data)
com_data_text.grid(row=(len(per_data)),column=1,columnspan
=10)

#create frame for all sensitive data types
frame_sens_data = ttk.LabelFrame(frame1, text='Sensitive
Data')
frame_sens_data.grid(column=2, row=4, padx=8, pady=4)      #
    common data frame col 0 row 5
rad_sens_data1 = tk.IntVar()
rad_sens_data2 = tk.IntVar()
rad_sens_data3 = tk.IntVar()
#set a value to make sure the radio buttons are unselected
rad_sens_data1.set(99)
rad_sens_data2.set(99)
rad_sens_data3.set(99)
rad_sens = [rad_sens_data1,rad_sens_data2,rad_sens_data3]
#create radiobuttons for each value in sens_data
for col in range(len(sens_data)):
    curRad = tk.Radiobutton(frame_sens_data, text=
        sens_data[col], variable=rad_sens[col], value=col)
    curRad.grid(column=1, row=col, sticky=tk.W)          #
    row 5
tk.Label(frame_sens_data, text="Other:", font='Times 10').
    grid(row=len(sens_data), column=0,)
com_data_text = tk.Entry(frame_sens_data)
com_data_text.grid(row=(len(sens_data)),column=1,
    columnspan=10)

tk.Label(frame1, text = ' How long are you storing data
for? ',font='Times 12').grid(row=7,column=1,pady=10) #
    row 7

radStorage = tk.IntVar()
#set a value to make sure the radio buttons are unselected
radStorage.set(99)
#create radiobuttons with the text defined in the list
    storage_time
for i in range(len(storage_time)):
    curRad = tk.Radiobutton(frame1, text=storage_time[i],
        variable=radStorage, value=i)
    curRad.grid(column=i, row=8, sticky=tk.W)          #row
    9

tk.Label(frame1, text = ' How are you gathering data? ',
    font='Times 12').grid(row=11,column=1,pady=10) #row 11

radGather = tk.IntVar()
#set a value to make sure the radio buttons are unselected
radGather.set(99)

```

```

#create radiobuttons with the text defined in the list
gather_options
for j in range(len(gather_options)):
    curRad = tk.Radiobutton(frame1, text=gather_options[j], variable=radGather, value=j)
    curRad.grid(column=j, row=12, sticky=tk.W) #
    row 13

tk.Label(frame1, text = 'Other:', font='Times 10').grid(row=12, column=(len(gather_options)))
gather_options_text = tk.Entry(frame1)
gather_options_text.grid(row=12, column=(len(gather_options)+1), columnspan=10)

tk.Label(frame1, text = ' How are you securing this data?', font='Times 12').grid(row=13, column=1, pady=10) #row 13
radEncrypt = tk.IntVar()
radEncrypt.set(99)
#create radiobuttons with the text defined in the list
gather_options
tk.Radiobutton(frame1, text="Encryption", variable=radEncrypt, value=1).grid(column=0, row=14, sticky=tk.W)
radMin = tk.IntVar()
radMin.set(99)
tk.Radiobutton(frame1, text="Minimal Access", variable=radMin, value=1).grid(column=1, row=14, sticky=tk.W)
radAuth = tk.IntVar()
radAuth.set(99)
tk.Radiobutton(frame1, text="Authentication", variable=radAuth, value=1).grid(column=2, row=14, sticky=tk.W)

radLog = tk.IntVar()
radLog.set(99)
#create radiobuttons with the text defined in the list
gather_options
tk.Radiobutton(frame1, text="Access Logging", variable=radLog, value=1).grid(column=0, row=15, sticky=tk.W)
radAnon = tk.IntVar()
radAnon.set(99)
tk.Radiobutton(frame1, text="Anonymisation", variable=radAnon, value=1).grid(column=1, row=15, sticky=tk.W)
radPseu = tk.IntVar()
radPseu.set(99)
tk.Radiobutton(frame1, text="Pseudomysation", variable=radPseu, value=1).grid(column=2, row=15, sticky=tk.W)
radBack = tk.IntVar()
radBack.set(99)
tk.Radiobutton(frame1, text="Backup", variable=radBack, value=1).grid(column=3, row=14, sticky=tk.W)

#create frame for matplotlib on tab1 normal plot
frame_matplot = ttk.LabelFrame(tab1, text='Risk Matrix')
#place frame on grid
frame_matplot.grid(column=2, row=0, padx=8, pady=4)

```

```

fig = plt.Figure(figsize=(5, 4), dpi=100)
plot = fig.add_subplot(111)
matrix = FigureCanvasTkAgg(fig, master=frame_matplot)
toolbar = NavigationToolbar2Tk(matrix, frame_matplot)
toolbar.update()
matrix.get_tk_widget().pack(side=tk.LEFT, fill=tk.BOTH)
plot.set_xlabel('Probability')
plot.set_ylabel('Severity')

#
#
    for i, txt in enumerate(name):
        plot.annotate(txt, (prob[i], sev[i]))
def calc_prob():
    prob_level = 0
    for counter in [radStorage, radGather]:
        if counter.get() == 0:
            prob_level = prob_level + 10
        elif counter.get() == 1:
            prob_level = prob_level + 25
        elif counter.get() == 2:
            prob_level = prob_level + 50
        else:
            prob_level = prob_level
    print(prob_level)
    for count in [radMin, radAuth, radLog]:
        if count.get() == 1:
            prob_level = prob_level / 3
        else:
            prob_level = prob_level
    print(prob_level)
    illAcc = prob_level / 10 * 3
    unWCha = prob_level / 10 * 2
    if radBack.get() == 1:
        disApp = prob_level / 20
    else:
        disApp = prob_level / 10
    print("RadBack:")
    print(radBack.get())
    prob = [illAcc, unWCha, disApp]
    return prob

def calc_sev():
    sev_level = 0
    for counter in [rad_com_data1, rad_com_data2,
                    rad_com_data3,
                    rad_pers_data1, rad_pers_data2,
                    rad_pers_data3,
                    rad_sens_data1, rad_sens_data2,
                    rad_sens_data3]:
        if counter.get() == 0:
            sev_level = sev_level + 20
        elif counter.get() == 1:
            sev_level = sev_level + 25
        elif counter.get() == 2:
            sev_level = sev_level + 30
        else:

```

```

        sev_level = sev_level

for count in [radEncrypt, radPseu, radAnon]:
    if count.get() == 1:
        sev_level = sev_level / 3
    else:
        sev_level = sev_level

illAcc = sev_level / 10
unwCha = sev_level / 10 * 2
if radBack.get() == 1:
    disApp = sev_level / 40
else:
    disApp = sev_level / 10
sev = [illAcc, unwCha, disApp]
return sev

# reset checkbox selections
def reset():
    rad_com_data1.set(99)
    rad_com_data2.set(99)
    rad_com_data3.set(99)
    rad_pers_data1.set(99)
    rad_pers_data2.set(99)
    rad_pers_data3.set(99)
    rad_sens_data1.set(99)
    rad_sens_data2.set(99)
    rad_sens_data3.set(99)
    radStorage.set(99)
    radGather.set(99)
    radEncrypt.set(99)
    radMin.set(99)
    radAuth.set(99)
    radLog.set(99)
    radAnon.set(99)
    radPseu.set(99)
    radBack.set(99)

def evaluate():
    prob = calc_prob()
    sev = calc_sev()

    name = ['Illegitimate Access', 'Unwanted change', 'Loss',
            ]
    plot.scatter(prob, sev, color = color[0])
    for i, txt in enumerate(name):
        plot.annotate(txt, (prob[i], sev[i]), textcoords="
            offset points", xytext=(0,0.1), ha='center')
    matrix.draw()
    color.pop(0)
    if rad_sens_data1.get() == 1 or rad_sens_data2.get()
        == 1 or rad_sens_data3.get() == 1:
        tk.messagebox.showerror(title=None, message="This
            is a high risk data process")
    elif rad_pers_data1.get() == 1 or rad_pers_data2.get()
        == 1 or rad_pers_data3.get() == 1:

```

```

        tk.messagebox.showwarning(title=None, message="
            This is a medium risk data process")
    else:
        pass
    reset()

def _quit():
    root.quit()
    root.destroy()
    exit()

ttk.Button(frame1, text='Evaluate Data Process', command=
    evaluate).grid(row=17, column=1)
ttk.Button(frame1, text='Quit', command=_quit).grid(row
    =17, column=2)

def link_folder():
    f = tk.filedialog.askopenfilename(
        parent = tab1, initialdir='C:/',
        title='Choose file')
    print(f)
    tk.Label(frame1, text=f, font='Times 10').grid(column
        =3, row=18, pady = 10)

ttk.Button(frame1, text='Link relevant documents', command
    =link_folder).grid(row=17, column=3)

#         matrix.get_tk_widget().pack(side=tk.TOP, fill=tk.BOTH,
expand=1)

##### Tab3 #####

us_labels = ["Title", "As a", "I want", "So that", "
    Acceptance Criteria"]

#create a frame for displaying in tab3
frame3 = ttk.LabelFrame(tab3, text='Create a Privacy User
    Story')
frame3.grid(column=0, row=0, padx=8, pady=4)

#create label and textfield for each entry in us_labels
tk.Label(frame3, text = "Title of the User Story", font='
    Times 10').grid(row=0, column=0, pady=10)
title = tk.Entry(frame3)
title.grid(row=0, column=1, columnspan=10)

tk.Label(frame3, text = "As a", font='Times 10').grid(row=1,
    column=0, pady=10)
as_a = tk.Entry(frame3)
as_a.grid(row=1, column=1, columnspan=10)

tk.Label(frame3, text = "I want", font='Times 10').grid(row
    =2, column=0, pady=10)
i_want = tk.Entry(frame3)
i_want.grid(row=2, column=1, columnspan=10)

```

```

tk.Label(frame3,text = "So that",font='Times 10').grid(row
=3,column=0,pady=10)
so_that = tk.Entry(frame3)
so_that.grid(row=3,column=1,columnspan=10)

tk.Label(frame3,text = "Acceptance Criteria",font='Times
10').grid(row=4,column=0,pady=10)
acc_crit = tk.Entry(frame3)
acc_crit.grid(row=4,column=1,columnspan=10)

var_privacy = tk.IntVar()
priv_descr = ttk.Entry(frame3, state="disabled")

def privacyCheck():
    if var_privacy.get() == 1:          #whenever checked
        priv_descr.config(state="normal")
    elif var_privacy.get() == 0:        #whenever
        unchecked
        priv_descr.config(state= "disabled")

tk.Checkbutton(frame3, text="Is this a Privacy User Story?
", variable=var_privacy, command=privacyCheck).grid(
    row=(len(us_labels)+1), column=0)

tk.Label(frame3,text = 'Privacy Description:',font='Times
10').grid(row=(len(us_labels)+2),column=0,pady=20)
priv_descr.grid(row=(len(us_labels)+2),column=1,columnspan
=10)


frame_risk = ttk.LabelFrame(frame3, text='Risk Level')
#place frame on grid
frame_risk.grid(column=1, row=(len(us_labels)+3), padx=8,
    pady=4)
radRisk = tk.IntVar()
#set a value to make sure the radio buttons are unselected
radRisk.set(99)
#create radiobuttons with the text defined in the list
gather-options
for l in range(len(risk_level)):
    curRad = tk.Radiobutton(frame_risk, text=risk_level[l
    ], variable=radRisk, value=l)
    curRad.grid(column=0, row=l, sticky=tk.W)          #row
13

var_severity = tk.IntVar()
sev_descr = ttk.Entry(frame3, state="disabled")

def severityCheck():
    if var_severity.get() == 1:          #whenever checked
        sev_descr.config(state="normal")
    elif var_severity.get() == 0:        #whenever
        unchecked
        sev_descr.config(state= "disabled")

```

```

tk.Checkbutton(frame3, text="Does this User Story affect
previous work", variable=var_severity, command=
severityCheck).grid(row=(len(us_labels)+4), column=0)

tk.Label(frame3, text = 'Severity Description:', font='Times
10').grid(row=(len(us_labels)+5), column=0, pady=20)
sev_descr.grid(row=(len(us_labels)+5), column=1, columnspan
=10)

#create user story for reviewing pia, necessary whenever
changes to existing eval happen
def createPIA():
    story = {}
    story={
        'title' : 'Review PIA',
        'description' : 'As a PrO I need to reassess the
PIA',
        'owner': '',
        'requirements': 'Are all previously identified
risks reevaluated?',
        'priv_flag' : True,
        'priv_descr' : 'Assess all Privacy Risks',
        'sev_flag': True,
        'sev_descr': 'potential impact on previous work',
        'status': 'PB',
        'risk_level': 'medium',
    }
    r = request.create_story(token, story)
    print(r)

def createStory():
    if var_privacy.get() == 1:
        priv_bool = True
    else:
        priv_bool = False

    if var_severity.get() == 1:
        sev_bool = True
        createPIA()
    else:
        sev_bool = False

    if radRisk.get() == 0:
        risk = 'low'
    elif radRisk.get() == 1:
        risk = 'medium'
    elif radRisk.get() == 2:
        risk = 'high'
    elif radRisk.get() == 3:
        risk = 'none'
    else:
        risk = ''

    story={
        'title' : title.get(),
        'description' : "As a " + as_a.get() + " I want "
+ i_want.get() + " So that " + so_that.get(),

```

```

        'owner': '',
        'requirements': acc_crit.get(),
        'priv_flag' : priv_bool,
        'priv_descr' : priv_descr.get(),
        'sev_flag': sev_bool,
        'sev_descr': sev_descr.get(),
        'status': 'PB',
        'risk_level': risk,
    }
    r = request.create_story(token, story)
    print(r)

    ttk.Button(frame3, text='Create', command=createStory).grid(
        row=16, column=1)

if __name__ == '__main__':
    root = tk.Tk()
    root.geometry('425x225')
    login(root)

    root.mainloop()

```

Listing 34: Code for API Requests of DPA Tool (request.py)

```

'''
Post request to get token:
curl -i -X POST -u arviccii:vero http://localhost:5000/tokens
'''
# importing the requests library
import requests
import json

def user_login(user, password):

    # defining the api-endpoint
    API_ENDPOINT = "http://localhost:5000/tokens"

    # data to be sent to api
    # data = {}

    # sending post request and saving response as response object
    r = requests.post(url = API_ENDPOINT, auth=(user, password))

    # extracting response text
    json_reply = r.json()

    # print(json_reply)
    return json_reply

def create_story(token, story):

```

```

# defining the api-endpoint
API_ENDPOINT = "http://localhost:5000/tasks"

#header with access token
header = {'Authorization' : 'Bearer ' + token,
'Accept': 'application/json',
'Content-Type': 'application/json',
}

# data to be sent to api
# print(header)
# print(story)

# sending post request and saving response as response object
# print(requests.post(url = API_ENDPOINT, headers = header, data
= story))
r = requests.post(url = API_ENDPOINT, headers = header, data =
    json.dumps(story))
# extracting response text
json_reply = r.json()
# print(r.status_code)
# print(json_reply)
return json_reply

```

C Privacy Protection Sprint

