Designing Security and Privacy Requirements in Internet of Things: A Survey

NADA ALHIRABI, Cardiff University, UK
OMER RANA, Cardiff University, UK
CHARITH PERERA, Cardiff University, UK

The design and development process for the Internet of Things (IoT) applications is more complicated than that for desktop, mobile, or web applications. First, IoT applications require both software and hardware to work together across many different types of nodes with different capabilities under different conditions. Secondly, IoT application development involves different types of software engineers such as desktop, web, embedded and mobile to work together. Further, non-software engineering personal such as business analysts are also involved in the design process. In the addition to the complexity of having multiple software engineering specialists cooperating to merge different hardware and software component together, the development process required different software and hardware stacks to integrated together (e.g., different stacks from different companies such as Microsoft Azure, IBM Bluemix). Due to above complexities, more often non-functional requirements (such as security and privacy which are highly important in the context of IoT) tend to get ignored or treated as second class citizens in IoT application development process. In this paper, we have reviewed techniques, methods and tools that are being developed to support incorporating security and privacy requirements into traditional application designs. By doing so, we aim to explore how those techniques could be applicable to the IoT domain. During our review, we realised that by far the majority of the requirement engineering efforts are focused on security. In this paper, we primarily focused on two different aspects: (1) design notations, models, and languages that facilitate capturing non-functional requirements (i.e., security and privacy), and (2) proactive and reactive interaction techniques that can be used to support and augment the IoT application design process. Our goal is not only to analyse, compare and consolidate past research work but also to appreciate their findings and discuss their applicability towards the IoT.

CCS Concepts: • **Human-centered computing → Ubiquitous and mobile computing systems and tools**; *Visualization systems and tools*; • **Software and its engineering** → *Software design engineering*; • **Security and privacy** → Security requirements;

Additional Key Words and Phrases: Internet of Things, Software Engineering, Software Design Tools, Non Functional Requirements, Notation, Design Principles

ACM Reference Format:
Nada Alhirabi, Omer Rana, and Charith Perera. 9999. Designing Security and Privacy Requirements in Internet of Things: A Survey. 9, 9, Article 99 (September 9999), 36 pages. https://doi.org/0000001.0000001

Authors' addresses: Nada Alhirabi, Cardiff University, School of Computer Science and Informatics, Cardiff, CF24 3AA, UK, alhirabin@cardiff.ac.uk; Omer Rana, Cardiff University, School of Computer Science and Informatics, Cardiff, CF24 3AA, UK, ranaof@cardiff.ac.uk; Charith Perera, Cardiff University, School of Computer Science and Informatics, Cardiff, CF24 3AA, UK, pererac@cardiff.ac.uk.

1 INTRODUCTION

Until 2003, Cisco IBSG's [22] did not recognise IoT due to the few numbers of connected devices where there was only 0.08 device per person. With the exponential growth of smartphones, tablets, smart devices and their applications over the years, connected things significantly increase for each person. In 2010, there were 6.8 billion people using 12.5 billion devices which are 1.84 devices per person. Due to technological enhancements in distributed affordable sensors, contactless data exchange such as RFID, short-range wireless such as Bluetooth and ZigBee, and internet mobile access, a global network of connected things has increased [7]. It is predicted to reach 20-24 billion connected devices by 2020 [89] [34].

Designing and developing IoT applications is complicated compared to the one created for desktop, web, or mobile applications. First, IoT applications need to support different technologies working together such as hardware/firmware, software, sensor, semantic, cloud, data storing, data modelling, processing, and communication technologies [70]. All of these components are working throughout many different types of nodes under different conditions and challenges, and all of them could be vulnerable to attacks [34]. Moreover, applications should consider many features that IoT needs to support such as devices heterogeneity, scalability, ubiquitous data exchange, semantic interoperability and data management, etc. [61] [13]. In order to the Internet of Things to be successful, the development process needs to consider the whole connecting devices, and improving the scenarios used in traditional mobile computing [34].

In addition to IoT heterogeneity nature, IoT applications involve the cooperation of multiple software engineers whose having different expertise. Those engineers are working together on different components with different application domains such as home automation, smart cities, smart driving...etc. Due to the lacking of full-stack developers, the development process needs to have a common programming framework to support developers needs [29]. Moreover, end-users preferences are divers and they need to create these preferences using graphical user interface (GUI) at run-time. Consequently, applications developers and devices manufacturers should focus on the end-user requirements [38].

As a result of the stated complexities, non-functional requirements (NFRs), such as security and privacy, have not received sufficient attention [86] especially in the traditional Software Development Lifecycle (SDLC). It has been stated that the main source of software vulnerabilities is discovered to be in the early stages of the SDLC and the majority of them could be eliminated at this step [27]. Consequently, adopting security and privacy in SDLC is becoming critical, and embedding them in the early stage of the SDLC become an insistent [85] [27]. Microsoft took a step forward and introduced Microsoft's Security Development Lifecycle (SDL) that consists of practices for supporting security [59].

With the raising connectivity towards the IoT and having massive of new devices and applications that are connected to the internet every year, threats keep changing [43]. The burden of the vulnerabilities in the traditional SDLC become bigger with the IoT heterogeneity. Security by Design (SbD) has been approved that it is an effective way to create a secure system by emerging security at an early stage of the software development lifecycle (SDLC). As well as SbD, Privacy by Design (PbD) is recently recognized to be an important and useful approach for preserving privacy for software-based systems [78]. This importance is confirmed by REGULATION (EU) 2016/679 General Data Protection Regulation (GDPR) [21] which is applied to all systems that deal with personal data processing which is common in IoT applications. The contribution of this paper is as follows:

- Review the evolution of design notations, models, and languages that facilitates capturing non-functional requirements (i.e., security and privacy).

- Propose and use a taxonomy to compare and contrast past approaches.
- Review a few selected tools that have been built to facilitate capturing non-functional requirements from two different perspectives: (i) proactive and (ii) reactive interaction techniques.

Paper structure. The paper is structured into sections as follows: section 2 presents background information about the traditional software development life cycle and its phases. Section 3 gives a brief IoT overview and how IoT differ from embedded systems. This section includes a case scenario that will be used in a later section. Section 4 explains the difference between the distributed nature of IoT application development and traditional SDLC. In section 5, we briefly introduce functional and non-functional requirements. Literature search and selection methodology are introduced in section 6, with some details about search queries and steps of data selection and extraction. Section 7 and 8 gives an exploratory analysis of some of the available design notations and design principle respectively. Section 9 discusses the survey limitations. Section 10 concludes the survey.

2 TRADITIONAL SOFTWARE DEVELOPMENT LIFE CYCLE

Let us now briefly introduce software development life cycle (SDLC) in general as it helps to understand the remaining sections of the paper. In later sections, we will discuss specific challenges in IoT and their impact on SDLC.

In software engineering, SDLC is the most significant element. SDLC is a traditional methodology (or process) used for building and maintaining software systems where some essential phases should be followed. In general, software developments models have three primary goals which are improving the system quality, providing management controls and maximizing productivity. There are several different software development life cycle models. Each one is developed for specific purposes. According to Hoffer [40] and Valacich [87], SDLC consists of five phases which are planning, analysis, design, implementation and maintenance as shown in Figure 1.

The planning phase is used for identifying and analysing information system needs. Then, these needs are prioritized and translated into a development schedule. After that, the proposed problem is investigated to determine the proposed system scope. In analysis phase, requirements are determined, and some alternative designs are suggested and compared. After the requirements are analysed, they become well-defined and documented. The outcome of analysis phase is software requirement specification (SRS) document. SRS document lists all the essential requirements that needed to be designed and developed. At the end of this stage, SRS document is handled to the end-user to approve it [54].

In the design phase, all the provided description will be converted into logical and then physical system specifications. The logical design is independent of any software, hardware or platform.

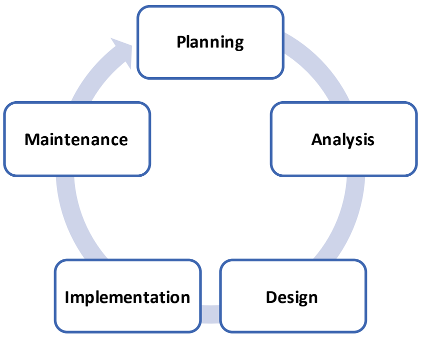


Fig. 1. Software Development Life Cycle (SDLC) based on [40] and [87]

It focusses on business features of the system. While in physical design, the specifications are transformed into technical specifications. In the implementation phase, six major activities are done which are coding, testing, installation, documentation, training and support. The goal of this phase is to translate the physical system specifications into reliable working software. It also record all the work that has been done. This phase also is considering the support for all users current and future once [40]. The last phase is maintenance phase, where fixing any issues founded by the customer/end-user is performed to keep the system working well. Although the previous described SDLC gives a general overview of the systems development process, there are very specific methods that use the idea of the SDLC with some additions. For example, Microsoft's Security Development Lifecycle (SDL) is a specialized SDLC that consists of practices for supporting security [59].

As mentioned earlier, system development has many models such as waterfall model, spiral, v model, prototype, agile model etc. Each model has its own features, drawback and usage. Initially traditional development was introduced with some extends such as in waterfall, v model, incremental and spiral models. In any model of them, each phase of SDLC has deliverables and outcomes that used for the next phase in the life cycle. They differ about the way that life cycle is arranged and executed as seen in Table 1. There are some weaknesses related to these types of models, such as high effort, time and cost to change after a milestone is delivered.

As known, all the models, regardless of their names or types, have somewhat of requirement management and design phase. Either way it is only done at the beginning of the model or repeated as an iteration through the system development. There are many consequences when managing requirements and design are only done once at early stage of the system development. The major effect is the high-cost of restarting or remodelling the system once major mistakes happened. The other impact that gradually affect the cost happens when end-user requirements are keep changing while developing the system where changes in this stage are expensive. As stated by Geer [27], the main source of software vulnerabilities is exposed in the early stage of the SDLC and the most of them could be excluded at this step. Consequently, it is important factor for cost reduction to highlight any vulnerabilities/mistakes at design stage which is much cheaper than fixing them in later stages.

As a result, other types of modelling such as agile modelling, as illustrated in Table 1, and object-oriented analysis and design (OOAD) are introduced to overcome the traditional SDLC limitations where remodelling/restarting is not expensive. A short comparison between traditional development, agile development and object-oriented analysis and design are given in Table 2. This comparison is based on some criterias such as primary objectives, requirement, cost of restarting and remodeling, etc. This table can give to some extent an overview about gathering and fulfilling user requirements and the cost of change in each model. It could help system engineers to clearly decide which model to follow before developing any system.

3 OVERVIEW OF INTERNET OF THINGS

Let us now briefly introduce IoT applications and their characteristics in general. In subsequent sections, we discuss SDLC from IoT perspective highlighting unique aspects and challenges of IoT. According to Gartner, the Internet of Things is the network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment [89]. These physical objects or things can be buildings, devices, automobiles, and other objects that embedded with sensors, software, electronics and network connection. DNA analysis devices, implanted heart monitoring and biochip transponders on animals are some examples of things that support the IoT concept. These devices gather suitable data using a variety of existing technologies and then move these data between other devices.

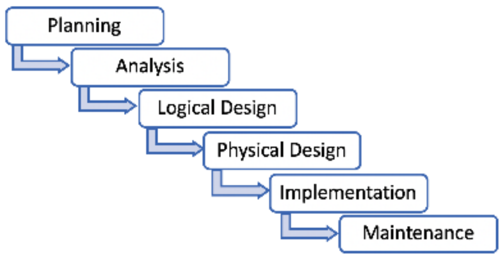
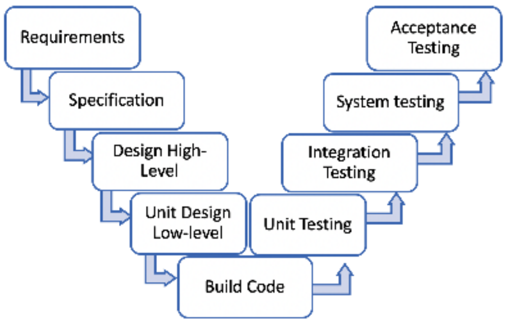
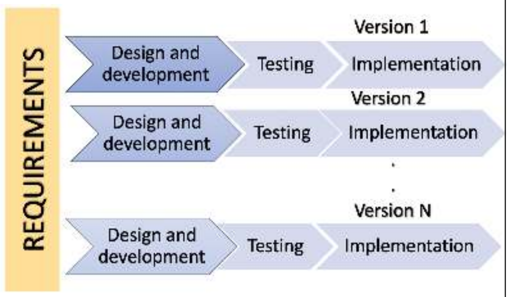
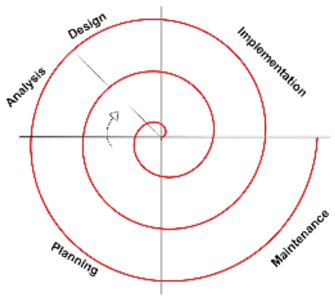
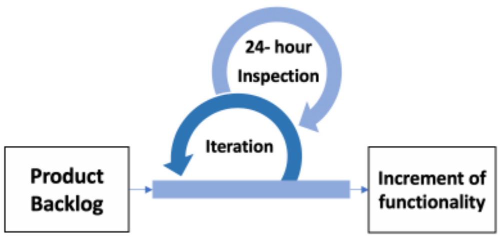
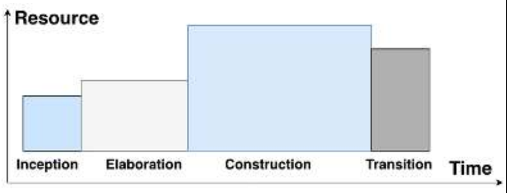
List of graphical illustration for some Software Development Models	
 Planning Analysis Logical Design Physical Design Implementation Maintenance Waterfall model. *Based on* [87].	 Requirements Specification Design High-Level Unit Design Low-level Build Code Unit Testing Integration Testing System testing Acceptance Testing V model. *Based on* [8].
 REQUIREMENTS Design and development Testing Version 1 Implementation Version 2 Version N Incremental model. *Based on* [54].	 Design Implementation Analysis Planning Maintenance Spiral model. *Based on* [87].
 24- hour Inspection Iteration Product Backlog Increment of functionality Scrum (Agile model). *Based on* [76].	 Rational Unified Process (RUP). *Based on* [87].

Table 1. List of graphical illustration for some Software Development Models: Waterfall model, V model, Incremental model, Spiral model, Scrum (Agile model) and Rational Unified Process (RUP) model. All of them vary on the way the life cycle is arranged and applied.

IoT history. Since the internet is founded in 1989, the idea of connecting things on the internet has been started. However, the term of Internet of Things (IoT) did not become popular till 1999 when it is created by Kevin Ashton, executive director of the Auto-ID Centre, MIT. At the same year, a global Radio-frequency identification (RFID) was invented which considered as a starting point for IoT. In 2000, LG company revealed its plan to produce a smart refrigerator that would determine by itself whether the food items are filled or not. The major trigger of IoT was in 2011 when IPv6 was lunched. Since then many famous IT companies such as Cisco, Ericson and IBM started to initiate many of IoT educational and commercial applications [81].

Software Development Models List			
	Traditional development	Agile development	object-oriented analysis and design (OOAD)
Development model	Life cycle model.	Evolutionary-delivery model.	Object-Oriented approach.
Primary objectives	Safety. Too many process and documentation is done for safety wich lead to slow development.	Quick results due to doing many iterations.	Quality and productivity by focussing in inheritance for refutability.
Organizational structure	Formal, targeting large organizations.	Flexible, targeting small\medium organizations.	Flexible, focussing on objects.
User requirements	Well-defined before implementation.	Co-operative input.	Co-operative.
Changes adaption	Poor changes adaption. Models such as waterfall does not allow changes of defined requirements as the project is progressing.	Changes and corrections are one each iteration.	Changes and corrections are one each iteration.
End-product fulfilling the requirements	Since changes are limited through the development, it is potential that the software could not fully meet the end-user requirement.	Since user is co-operating and changes are repeatedly applied, it is potential that the software would meet the end-user requirement.	Since user is co-operating and changes are repeatedly applied, it is potential that the software would meet the end-user requirement.
Cost of restarting	High.	Low.	cost is reduced. For each iteration, full assessment is done for needed correction [39].
Cost of remodeling	Expensive.	Not expensive due to user engagment and repeated assessment.	Not expensive due to continuous assessment.
Testing	Done after coding.	Done each iteration.	Done in iterations.
Developers	Organized with a plan.	Co-located and interactive.	Interactive.
Size of teams and projects	Large.	Small.	Large-scale projects. There are some extends to OOAD approaches, such as Rational Unified Process (RUP) [62], to support small\micro software teams\projects.

Table 2. Differences between traditional, agile and object-oriented analysis and design (OOAD) development approach. OOAD shares the iterative method from agile model, consequently they share some characteristics [54][87][39]

3.1 Difference Between IoT and Embedded Systems

Embedded systems, called sometimes embedded chipsets, are computing devices that are sitting on the edge of an IoT product. Connecting sensors to the internet is the main responsibility of the embedded system. It is also managing the communication between sensors and network by

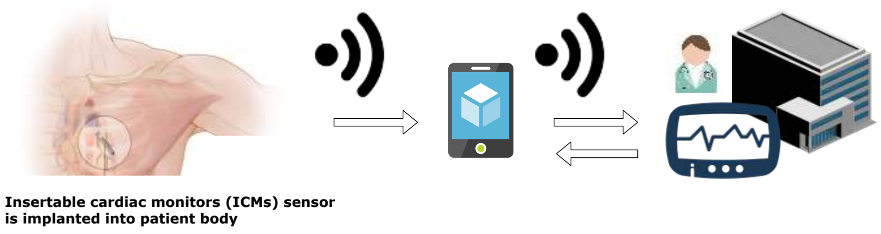

Insertable cardiac monitors (ICMs) sensor
is implanted into patient body

Fig. 2. Insertable cardiac monitors (ICMs)

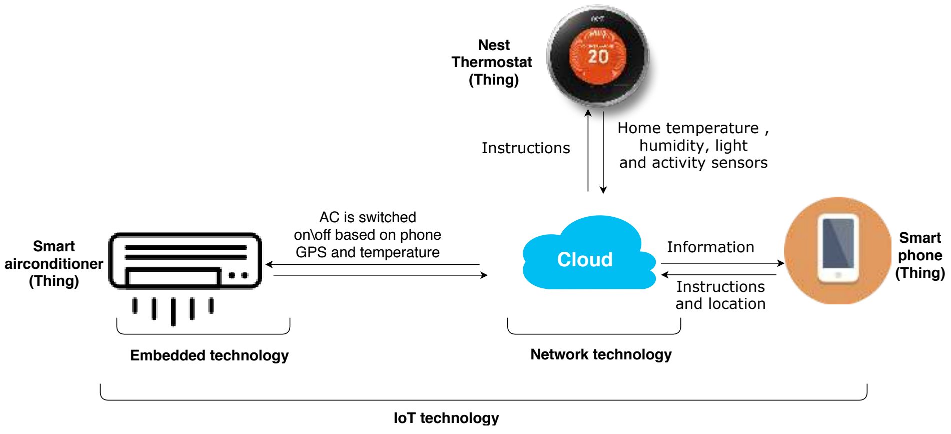

Nest Thermostat (Thing)
20
Instructions
Home temperature , humidity, light and activity sensors
AC is switched on\off based on phone GPS and temperature
Smart airconditioner (Thing)
Cloud
Information
Instructions and location
Smart phone (Thing)
Embedded technology
Network technology
IoT technology

Fig. 3. IoT Vs. Embedded systems

gathering, packaging and sending the data to an application. Furthermore, it may execute local application and security. Embedded system usually consists of a microcontroller that programmed to do a specific job. Once an embedded device is given access to the internet, it becomes an IoT device. For example, a lower power-consuming device such as implanted heart monitoring, showed in Figure 2, considered as an embedded device. It becomes an IoT device if it can transmit signal and pulses reading to a server's log via the Internet. These readings are checked then to see if there is an abnormality level. If there is, the server sends an alert to the concerned patient,s family mobile and doctors with some details such as the condition of the patient.

Another example that can illustrate the idea of embedded system and IoT is smart air conditioner (AC) and Nest learning thermostat as seen in Figure 3. Nest thermostat is a programmable smart thermostat with self-learning Wi-Fi developed by Nest Labs. It can conserve energy by adjusting the heating and cooling of homes and businesses. Nest has an embedded system that sends home temperature, humidity, light and activity sensors. The smart AC by itself has an embedded system that can switch the AC on\off based on the temperature sensor. It becomes an IoT system once AC is connected to the internet, the cloud on this example, and interacted with the Nest temperature device and app. By taking advantage of Nest and AC sensors and phones' locations, the energy can be turned to saving mode when nobody is at home.

3.2 Example IoT scenario

In this section, diabetes treatment and monitoring use case scenario is presented from a problem owner's perspective. Developing an IoT application can solve this problem. This scenario highlights some privacy challenges as we will explain later.

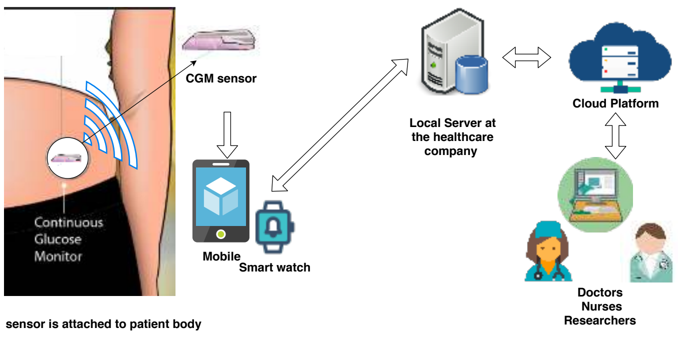

CGM sensor
Continuous Glucose Monitor
sensor is attached to patient body
Mobile
Smart watch
Local Server at the healthcare company
Cloud Platform
Doctors
Nurses
Researchers

Fig. 4. IoT application to support diabetes treatment monitoring

Use case: diabetes treatment and monitoring. Sara is a researcher in a healthcare company where patients with diabetes require treatment and continual monitoring. Sara is concerned about gathering and analysing data from a Continuous Glucose Monitor (CGM) device worn by patients where the sensor placed into the patient,s body, not into his bloodstream as seen in Figure 4. The sensor measures the glucose in the patient interstitial fluid by taking readings at regular intervals for several days. Sara has a monitoring application that can recognize any triggers or patterns for glucose abnormal levels. This application can analyse patient data and produce an alarm to notify the patient and the nurse. There is a speciality nurse that has a level of access to patient data for following-up and provide essential instructions when required. These instructions may include suitable insulin dosage, an exercise plan, daily meals\snacks, and types\dosage of medications.

4 IOT SOFTWARE DEVELOPMENT LIFE CYCLE

IoT has its own properties, requirements and challenges. Heterogeneous is one of the main challenges where managing and securing different objects, devices, sensors, protocols and applications are complicated [33]. Moreover, each object could be developed in a diverse way, by diverse manufacturers. As a result, adding some changes or enhancements to the traditional SDLC are required. Privacy as one of the NFRs could be the greatest trouble in a ubiquitous computing [42]. Pew Internet research [11] stated that 54 % of app users refuse to install a mobile app when they realized the amount of personal information the app gathered. Also, 30 % of the smartphone owners do not want to share personal information which causes uninstalling the app after knowing that it collects personal data.

We will apply some of the privacy as one of NFRs to the previous use case mentioned in section 3.2 Figure 4 to see how these requirements are important in IoT. When the sensor of Continuous Glucose Monitor (CGM) patient's device sends the glucose readings to the application, the data travelling is not straight forward. These data will go through different nodes until it reaches its final destination as seen in Figure 5. One of these nodes could be a third party who could have access to patient sensitive data such as location without having the patient permission or without the research company knowledge. This can happen as a result of the app developers lack of knowing what third parties libraries are collecting from the users [9] [3]. Data subjects actually lose control over their data when they are stored on a server operated by a third party. According to [42], half of the analyzed app by the study team via PrivacyGrade.org are using location, not because the app wants it but because the third party library uses it.

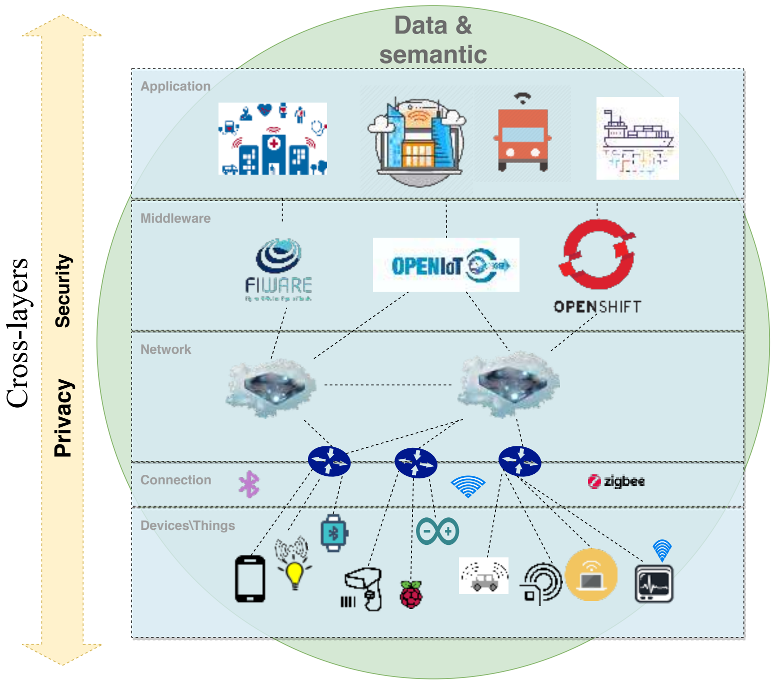


Fig. 5. Heterogeneous IoT Framework. Data and semantic travelling is not straight forward. Starting from the source, it goes through different nodes until it reaches its final destination.

In the case of the health field, such as the described use case, distributing patients, data such as location, phone number, patient file number and medical history is critical. In this use case anonymization as one of the privacy characteristics is violated. These problems can be solved by applying some of the privacy patterns during software development such as using *protection against tracking, onion routing, and anonymity set* privacy patterns.

Since smartphones are the most developed app in ubiquitous computing, one of the offered solutions is dividing the smartphone ecosystem for privacy into many entities [42]. These entities share the responsibility for privacy which are: Developers, Third-party developers, Service providers, App stores, OS providers, Hardware manufacturers, Government agencies and third parties interested in privacy, and Users. Therefore, it very clear IoT applications have a significant potential (much more than mobile applications) to collect personal information that could lead to significant privacy violation. Therefore, it is extremely important to integrate the non-functional requirement into IoT application designs.

5 FUNCTIONAL AND NON FUNCTIONAL REQUIREMENTS

Let us now briefly introduce non-functional requirements in general, before we focus on privacy and security in subsequent sections. Some studies have shown that the effectiveness of requirements engineering (RE), in software projects, is an important success factor [63]. There are many types of requirements and the three common types are: business requirements, the software value in business terms; functional requirements (FRs), known as qualitative requirements; and non-functional requirements (NFRs), known as quantitative requirements [74] [56]. Business requirements which define how the system improve the needs of the organization. FRs define the system services, functions or behaviour, that the system supposed to accomplish. On the other hand, NFRs can be defined as the quality attributes (e.g., security, integrity, reliability, usability) or the applied constraints on the application during the development process [32] [83]. Most of the efforts that have been done previously are focused on functional requirements (FRs). However, many efforts are focusing currently on non-functional requirements (NFRs) since many studies show that engaging NFRs in early design phases has improved significantly the end-user satisfaction [6] [32].

Functional requirements (FRs) defines the way to understand what is required to build a system correctly and deliver what the end-user expects. In API development, FRs define the expected functionality for the API which requires the end-user collaboration. Functional requirements gathering is located at in the analysis phase [74]. Holding interviews, meetings, or using questionnaires to ask end-users are some of the ways to capture FRs. These requirements are typically given a unique identifier for each one and a description in a requirements document. An example of these requirements is: *REQ1.1. The system shall authenticate that the entered PIN number by the user is correct.*

Table 3. high-Level Non-Functional Requirements (NFRs) Classification

Characteristics	Description	Sub-Characteristics
Intrinsic Qualities	characteristics of the product/solution itself	Functional suitability Performance efficiency Compatibility Usability Reliability Security Maintainability Portability
Usage Qualities	characteristics related to outcomes of user interaction with the product/solution	Effectiveness Efficiency Satisfaction Safety Usability scope
External Qualities	market-related characteristics associated with the product/solution	Service Cost Vendor Risk Mitigation Product Risk Mitigation

Non-functional requirements are used to support the functional one. While FRs describe how the system should behave, the NFRs describe the functioning constraints that guarantee end-user satisfaction. Examples of non-functional requirements include many characteristics such as: performance, flexibility, platform compatibility, security, scalability, usability and recovery [74]. ISO/IEC (the International Organization for Standardization) and (the International Electrotechnical Commission) proposes solution attributes, called 'Qualities', that can be divided into 3 categories in ISO/IEC 25010:2011 [45]. These Qualities serve as top-level non-functional requirements (NFRs), which are breakdown to detailed level NFRs as seen in Table 3 and 4. Privacy framework introduced in another standard called ISO/IEC 29100:2011(en) [1]. There are many standards from ISO/IEC that are divided based on the policy maker viewpoint such as 29151: Code of practice for personally identifiable information protection and 20889:Privacy enhancing data de-identification techniques. There are many standardization organization and not limited to ISO for defining NFRs, but the listed one here just to have an idea about some them and their meaning from ISO.

Characteristics	Sub-Characteristics	Description
	Functional completeness	degree to which the set of functions covers all the specified tasks and user objectives

Functional suitability	Functional correctness	degree to which a product or system provides the correct results with the needed degree of precision.
	Functional appropriateness	degree to which the functions facilitate the accomplishment of specified tasks and objectives.
Performance efficiency	Time behaviour	degree to which the response and processing times and throughput rates of a product or system, when performing its functions, meet requirements.
	Resource utilization	degree to which the amounts and types of resources used by a product or system, when performing its functions, meet requirements
	Capacity	degree to which the maximum limits of a product or system parameter meet requirements.
Compatibility	Co-existence	degree to which a product can perform its required functions efficiently while sharing a common environment and resources with other products, without detrimental impact on any other product.
	Interoperability	degree to which two or more systems, products or components can exchange information and use the information that has been exchanged.
Usability	Appropriateness recognizability	degree to which users can recognize whether a product or system is appropriate for their needs.
	Learnability	degree to which a product or system can be used by specified users to achieve specified goals of learning to use the product or system with effectiveness, efficiency, freedom from risk and satisfaction in a specified context of use.
	Operability	degree to which a product or system has attributes that make it easy to operate and control.
	User error protection	degree to which a system protects users against making errors.
	User interface aesthetics	degree to which a user interface enables pleasing and satisfying interaction for the user.
	Accessibility	degree to which a product or system can be used by people with the widest range of characteristics and capabilities to achieve a specified goal in a specified context of use.
Reliability	Maturity	degree to which a system, product or component meets needs for reliability under normal operation.
	Availability	degree to which a system, product or component is operational and accessible when required for use.
	Fault tolerance	degree to which a system, product or component operates as intended despite the presence of hardware or software faults.
	Recoverability	degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system.
Security	Confidentiality	degree to which a product or system ensures that data are accessible only to those authorized to have access.
	Integrity	degree to which a system, product or component prevents unauthorized access to, or modification of, computer programs or data.
	Non-repudiation	degree to which actions or events can be proven to have taken place, so that the events or actions cannot be repudiated later.
	Accountability	degree to which the actions of an entity can be traced uniquely to the entity.
	Authenticity	degree to which the identity of a subject or resource can be proved to be the one claimed.
	Modularity	degree to which a system or computer program is composed of discrete components such that a change to one component has minimal impact on other components.
	Reusability	degree to which an asset can be used in more than one system, or in building other assets.

Maintainability	Analysability	degree of effectiveness and efficiency with which it is possible to assess the impact on a product or system of an intended change to one or more of its parts, or to diagnose a product for deficiencies or causes of failures, or to identify parts to be modified.
	Modifiability	degree to which a product or system can be effectively and efficiently modified without introducing defects or degrading existing product quality.
	Testability	degree of effectiveness and efficiency with which test criteria can be established for a system, product or component and tests can be performed to determine whether those criteria have been met.
Portability	Adaptability	degree to which a product or system can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments.
	Installability	degree of effectiveness and efficiency with which a product or system can be successfully installed and/or uninstalled in a specified environment.
	Replaceability	degree to which a product can replace another specified software product for the same purpose in the same environment.
Privacy *	anonymization	process by which personally identifiable information (PII) is irreversibly altered in such a way that a PII principal can no longer be identified directly or indirectly, either by the PII controller alone or in collaboration with any other party
	pseudonymization	process applied to personally identifiable information (PII) which replaces identifying information with an alias
	consent	personally identifiable information (PII) principal,s freely given, specific and informed agreement to the processing of their PII

Table 4. Detailed-Level of Non-Functional Requirements (NFRs) Classification from ISO/IEC 25010 [45]. (*)Privacy characteristics are example and are not limited to the listed one here from ISO/IEC 29100 [1].

6 METHODOLOGY

In order to build this survey, we followed a search strategy. The first step was selecting papers in google scholar to avoid publisher basis. Then we search on specific libraries based on forward and backword snowballing. After that, it was data extraction step to extract some of the properties from each notation/representation. Finally, we analysed the data to find the review results.

6.1 Data Sources and Search Strategy

For data collection and extraction, Kitchenham method [50] is generally (not all the steps) used as a guideline to extract data from each paper. At the beginning, we create the initial search query by using Google Scholar, using some keywords that include the word notation or non-functional requirements (security and privacy in specific). This resulted in general queries with combination of AND and OR in between as seen in Table 5. These queries resulted in having many papers where some of them is not very related to visual notation or non-functional requirements. However, this step is used to give us an idea about which digital libraries and journals that interested in notation, visualization and non-functional requirements. After that, we tried hybrid search using more complex quires in specific journals and libraries which listed in Table 6 such as IEEE Xplore , Scopus, Springer, ACM, ScienceDirect etc.

For the notations study, papers from 1999 onward have been checked to ensure comparability and to cover wide varieties of notations in general and secure one in specific. The search process is done in many stages and several iterations that include three search processes: automatic, manual, and snowballing.

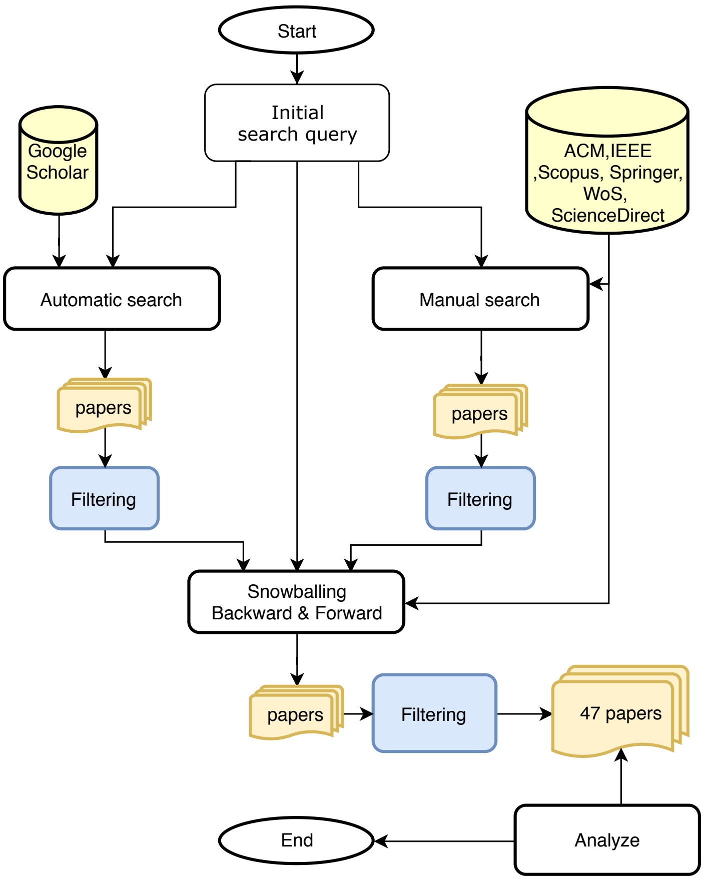


Fig. 6. Search Process

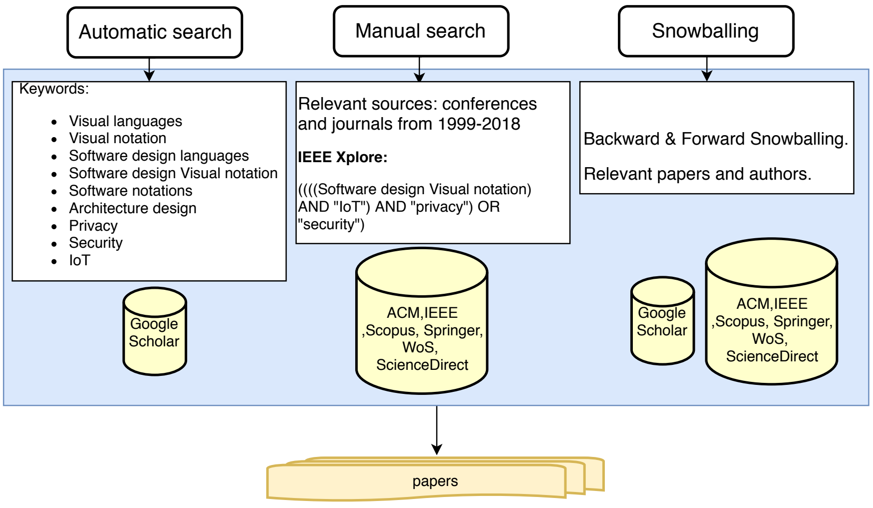

Automatic search
Manual search
Snowballing
Keywords:
Visual languages
Visual notation
Software design languages
Software design Visual notation
Software notations
Architecture design
Privacy
Security
IoT
Relevant sources: conferences and journals from 1999-2018
IEEE Xplore:
((((Software design Visual notation) AND "IoT") AND "privacy") OR "security")
Backward & Forward Snowballing.
Relevant papers and authors.
Google Scholar
ACM,IEEE ,Scopus, Springer, WoS, ScienceDirect
Google Scholar
ACM,IEEE ,Scopus, Springer, WoS, ScienceDirect
papers

Fig. 7. Initial search

Table 5. Some queries and terms for online libraries search.

Category	Queries and Terms
General	"Architecture design" "Visual" AND (" languages" OR "notation") "Software design languages" OR "Software design Visual notation" "Security" AND (combinations of the above) "Privacy" AND (combinations of the above)
More spastic	**IEEE:** ((((Software design Visual notation) AND "IoT") AND "privacy") OR "security") **ACM:** (+Visual +notation software requirements +security +privacy) **Scopus:** TITLE-ABS-KEY-AUTH("non functional" AND requirements AND IoT) AND (LIMIT-TO (SUBJAREA,"COMP") OR LIMIT-TO (SUBJAREA,"ENGI")) AND (LIMIT-TO (LANGUAGE,"English"))

- Automatic search (Figure 6 and 7): This stage was performed using search engine using keywords that combined any of the terms *'Visual languages', 'Visual notation', 'Software design languages', 'Software design Visual notation', 'Software notations', Architecture design, 'Privacy', 'use all above combinations', 'Security', 'IoT', 'Architecture', 'cyber physical systems' and 'ubiquitous Computing'*. Google Scholar is used as a start point for searching since it is considered as a good substitute to avoid bias in favour of any specific publisher [92]. Using this way, we took the advantage looking for the whole spectrum for all the available publications regardless of the publishers.
- Manual search (Figure 6 and 7): This stage was done using the proceedings of some conferences and journals as sources such as Ubiquitous Computing (UbiComp), Journal of Systems and Software (JSS), Transactions on Software Engineering and Methodology (TOSEM) and others listed in Table 6. For these sources, the studied time period is 2000-2019.
- Snowballing (Figure 6 and 7): This stage was performed on the set of papers that gathered previously in manual search and based on known papers from the same relevant authors and time period. Then, backward snowballing is performed by checking the references to select relevant papers based on relevant title, abstract, and general structure review.

6.2 Data Extraction

As stated previously, Kitchenham method is generally followed for data clollecting. For each paper many characteristics were extracted. All the collected and extracted data was structured in an Excel spread sheet. For this study, a repository has been built which contains the meta-data for the analyzed papers. Based on the built repository, we accomplished our analysis of the analyzed papers. The meta-data contains: notation ID, name, publication year, scope, visual design based, tool support, security/privacy support, IoT support, validation type, experiments' participants background, and the conference proceeding/journal. Finally, a list of 47 notations in different publications remained, and all of them were analyzed in this survey.

Analysis of the apers. The publications distribution per year from 1999 to 2019 is presented in Figure 8 (A). The distribution of all analyzed papers' publication types can be found in Figure 8 (B). It can be noticeable that the peak publications on secure notation was in 2009-2010 with 12 papers in total. In addition, most of the found papers come from journals and conference proceedings.

7 DESIGN NOTATION, LANGUAGES, AND REPRESENTATIONS

Many design notations have been created for security that intended to document security concepts and features into a software design model. These efforts about notations are distributed and not organized. This distribution makes it complex for researchers to assist existing notations to decide

Table 6. Sources of Selected conference proceedings and journals for Manual and Automatic search that the notations are selected from.

Venue	Abbr.	Source	Publisher
Conference	OOPSLA	ACM SIGPLAN conference companion on Object Oriented Programming Systems Languages and Applications	ACM
	ESORICS	European Symposium on Research in Computer Security	Springer
	DAC	IEEE Design Automation Conference	ACM\IEEE
	EuroS PW	IEEE European Symposium on Security and PrTransactions on Software Engineeringivacy Workshops	IEEE
	COMPSAC	IEEE International Computer Software and Applications Conference	IEEE
	ICECCS	IEEE International Conference on Engineering Complex Computer Systems Navigating Complexity in the e-Engineering Age	IEEE
	ICECCS	IEEE International Conference on Engineering of Complex Computer Systems	IEEE
	SCC	IEEE International Conference on Services Computing	IEEE
	ICSA-C	IEEE International Conference on Software Architecture Companion	IEEE
	ICWS	IEEE International Conference on Web Services	IEEE
	ISESS	IEEE International Software Engineering Standards Symposium and Forum	IEEE
	VL/HCC	IEEE Symposium on Visual Languages-Human Centric Computing	IEEE
	AVI	International Conference on Advanced Visual Interfaces	ACM
	ICSE	International Conference on Software Engineering	ACM\IEEE
	SEKE	International Conference on Software Engineering and Knowledge Engineering	Springer
	FME	International FME Workshop on Formal Methods in Software Engineering	ACM\IEEE
journal	IST	Information and Software Technology	Elsevier
	-	Computers & Security	Elsevier
	-	Computer Networks	Elsevier
	-	Decision Support Systems	Elsevier
	TSE	IEEE Transactions on Software Engineering	IEEE
	IEEE T SYST MAN CY C	IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)	IEEE
	IST	Information and Software Technology	Elsevier
	IJSE	International Journal of Software Engineering	CSC
	INTR	Internet Research	Emerald
	JSW	Journal of Software	-
	-	Journal of Visual Languages and Computing	Elsevier
	-	Science of Computer Programming	Elsevier
	SoSyM	Software and Systems Modeling	Springer
	-	The Journal of Systems and Software	Elsevier

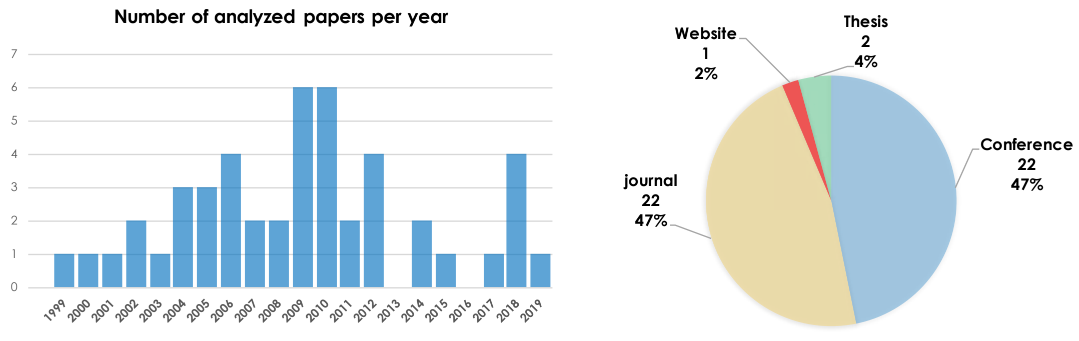


Fig. 8. Overview of the number of analysed papers. (a) distributed per year (1999-2019), and (b) by venue.

which technique they need to follow. There is an effort has been done in the field of reviewing security notations such as [90]; however, it does not have anything about IoT and it needs to be updated.

The following section presents a systematic literature review that investigates the available non-functional requirements (security and privacy) notations and produces a comprehensive analysis for each one of them. This will be done to understand which the techniques are they used to represent

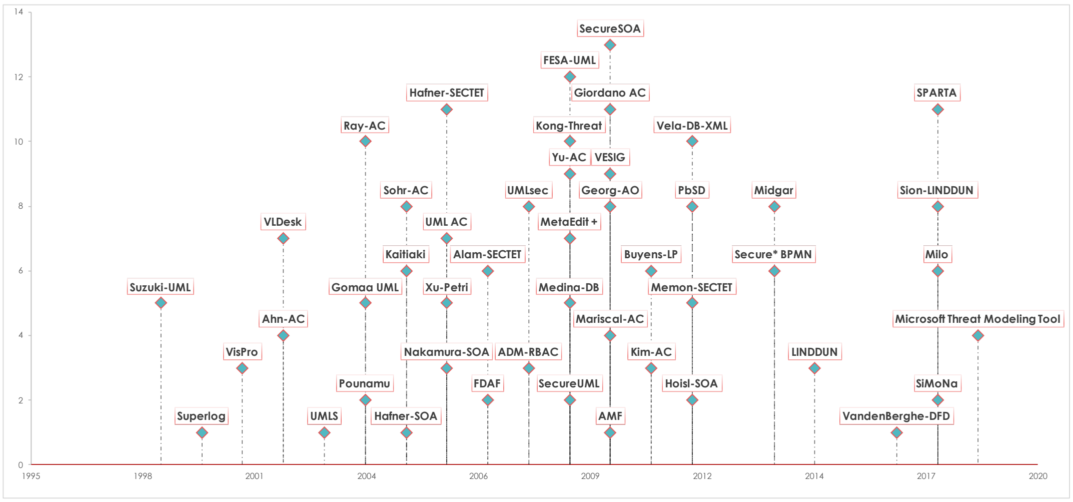


Fig. 9. Timeline for the analyzied notations, models, languages from 1999 to 2019.

security characteristics visually. After that, we will observe which privacy technique we can match to develop later a model for a privacy notation.

In the beginning, we analyze 47 notations, languages, and representations for the period (1999-2019) to have an overview of design representations over the last 20 years as seen in Figure 9. These notations could follow some of the known representation standards such as UML and DFD, which most of them do, as seen in Figure 10. In addition to the representation model, all the notations are investigated in term of scope, coverage and tool support. Then we assessed the way these notations have been evaluated and who are the participates in case the notations are experimentally validated.

7.1 Scope and Coverage

This study analyses 47 design notations for different coverage and purposes (see Table 7). This study is trying to look at a variety of notations' domains to assess how they are built, how they support NFRs and how IoT can be built. Since the study are related to IoT, three of these notations are covering IoT systems which are SiMoNa, Midgar and Microsoft Threat Modeling Tool. Since building IoT systems is expanding rapidly for the last decade, it is expected to see the three systems are recently produced in the last five years; 2014 for Midgar, 2018 for SiMoNa and 2019 for Microsoft Threat Modeling Tool. Regarding the Non-functional requirements (NFRS), security and privacy are the two assessed NFRs in this study. Security has been supported for more than the half of the analysed papers (32 notations out of 47), while privacy is only covered in SPARTA, LINDDUN and Sion-LINDDUN (3 notations out of 47).

It is noticeable that more than half of the analysed notations (25 out of 47) are generic and are not focused on a specific application domain. There are some domain-specific notations, and service-oriented architecture (SOA) is relatively common (7 notations). The other notations are vary between database (3 notations), DSML (6 notations), web system(2 notations), design pattern recovery (2 notations), authorization (1 notation), education (1 notation) and system-on-Ship (1 notation).

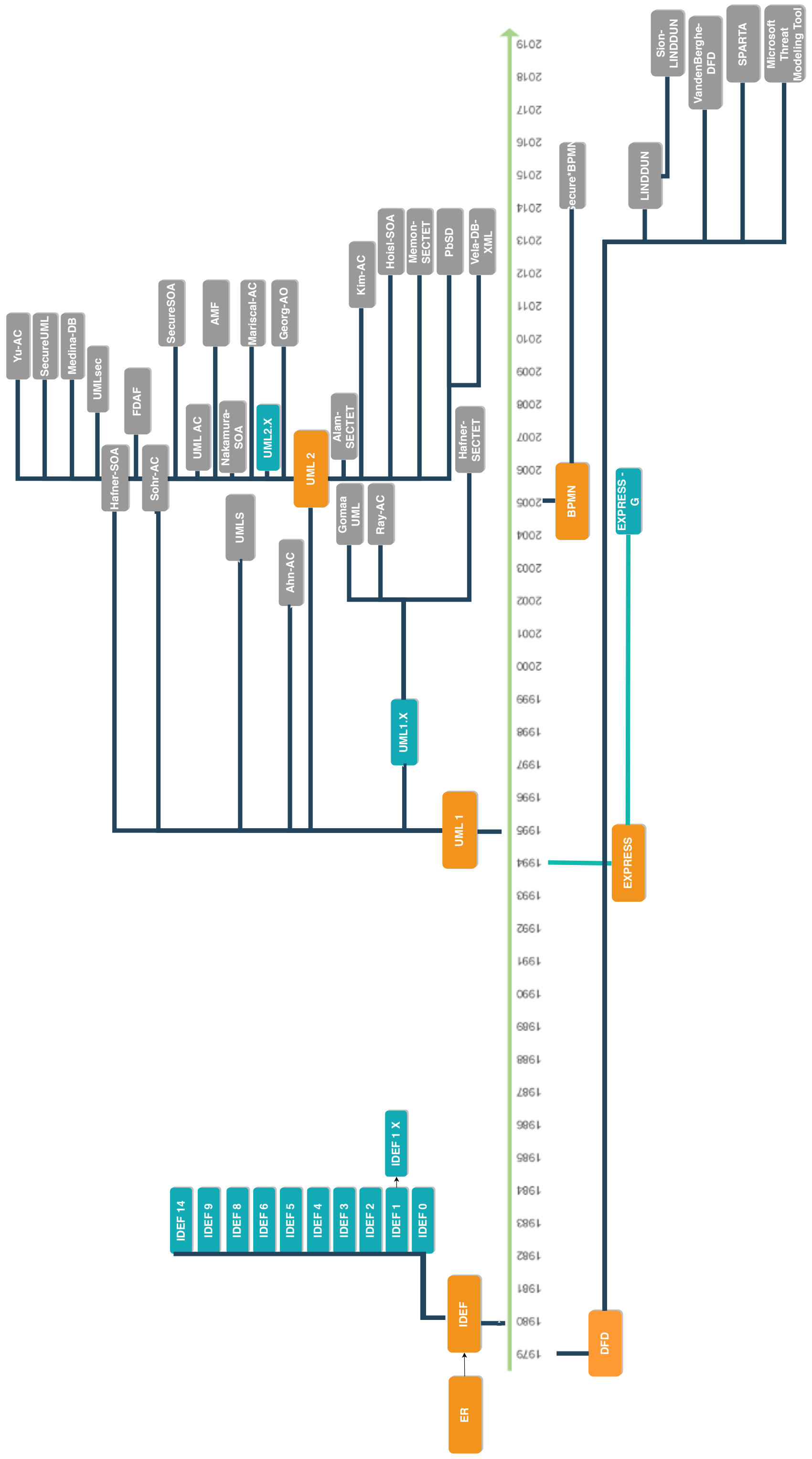


Fig. 10. The analyzied notations, models, languages over the time (in grey rectangles) with their associated standard notations such as UML, DFD and BPMN (orange rectangles).

Table 7. Analyses of 47 design notations based on the covered scope of notation.

List of notations														
			Scope									Coverage		
Name	Citation	Year	Generic	Database	Web system	SOA	Authorization	Patterns design	Education	DSML	System-on-Ship	IoT	Security	Privacy
ADM-RBAC	[20]	2008			✓								✓	
Ahn-AC	[4]	2002	✓										✓	
Alam-SECTET	[4]	2007				✓							✓	
AMF	[44]	2010					✓						✓	
Buyens-LP	[12]	2011	✓										✓	
FDAF	[16]	2007	✓										✓	
Georg-AO	[28]	2010	✓										✓	
Giordano AC	[30]	2010	✓										✓	
Gomaa UML	[31]	2004	✓										✓	
Hafner-SOA	[35]	2005				✓							✓	
Hoisl-SOA	[41]	2012				✓							✓	
Kim-AC	[49]	2011	✓										✓	
Kong-Threat	[52]	2009	✓										✓	
Mariscal-AC	[69]	2010	✓										✓	
Medina-DB	[86]	2009		✓									✓	
Memon-SECTET	[57]	2012				✓							✓	
Nakamura-SOA	[75]	2006				✓							✓	
PbSD	[2]	2012		✓									✓	
Ray-AC	[73]	2004	✓										✓	
SecureSOA	[58]	2010				✓							✓	
SecureUML	[10]	2009	✓										✓	
Sohr-AC	[79]	2005	✓										✓	
UML AC	[51]	2006	✓										✓	
UMLS	[37]	2003	✓										✓	
UMLsec	[48]	2008	✓										✓	
Vela-DB-XML	[91]	2012		✓									✓	
Xu-Petri	[95]	2006	✓										✓	
Yu-AC	[96]	2009	✓										✓	
Hafner-SECTET	[36]	2006				✓							✓	
SPARTA	[77]	2018	✓										✓	✓
VandenBerghe-DFD	[88]	2017	✓											
LINDDUN	[94]	2015	✓											✓
Sion-LINDDUN	[78]	2018	✓											✓
Kaitiaki	[53]	2005								✓				
Suzuki-UML	[82]	1999			✓									
VLDesk	[26]	2002						✓						
VESIG	[17]	2010						✓						
Superlog	[25]	2000									✓			
VisPro	[97]	2001	✓							✓				
FESA-UML	[23]	2009	✓											
Pounamu	[64]	2004	✓							✓				
Milo	[72]	2018							✓					
SiMoNa	[18]	2018								✓		✓		
Midgar	[33]	2014								✓		✓		
MetaEdit+ 5.5	[84]	2017								✓				
Microsoft Threat Modeling	[60]	2019	✓									✓	✓	
Secure*BPMN	[15]	2014											✓	
Total			25	3	2	7	1	2	1	6	1	3	32	3

7.2 Tool support

It can be seen from the Table 8 and Figure 11 that the majority of the analyzed notations that they did not produce final tool (34 notations:16 notations with kind of prototype and 18 without any tool support). Only around a quarter of the notations have a tool (12 out of 47 notations) and some of the notations that have a tool do not support security such as SiMoNa which is IoT

Table 8. Representation model and tool support for investigated notations.

		Final product			Supported Model		
Notation	Citation	Tool	Prototype	None	UML based	DFD based	Non UML/DFD
ADM-RBAC	[20]		✓				●
Ahn-AC	[4]			✓	●		
Alam-SECTET	[4]		✓		●		
AMF	[44]		✓		●		
Buyens-LP	[12]		✓				●
FDAF	[16]		✓		●		
Georg-AO	[28]			✓	●		
Giordano AC	[30]		✓				●
Gomaa UML	[31]			✓	●		
Hafner-SOA	[35]			✓	●		
Hoisl-SOA	[41]	✓			●		
Kim-AC	[49]		✓		●		
Kong-Threat	[52]			✓			●
Mariscal-AC	[69]		✓		●		
Medina-DB	[86]		✓		●		
Memon-SECTET	[57]	✓			●		
Nakamura-SOA	[75]			✓	●		
PbSD	[2]		✓		●		
Ray-AC	[73]			✓	●		
SecureSOA	[58]		✓		●		
SecureUML	[10]	✓			●		
Sohr-AC	[79]		✓		●		
UML AC	[51]			✓	●		
UMLS	[37]			✓	●		
UMLsec	[48]	✓			●		
Vela-DB-XML	[91]			✓	●		
Xu-Petri	[95]			✓			●
Yu-AC	[96]			✓	●		
Hafner-SECTET	[36]			✓	●		
SPARTA	[77]		✓			●	
VandenBerghe-DFD	[88]			✓		●	
LINDDUN	[94]			✓		●	
Sion-LINDDUN	[78]			✓		●	
Kaitiaki	[53]		✓				●
Suzuki-UML	[82]		✓		●		
VLDesk	[26]	✓			●		
VESIG	[17]	✓					●
Superlog	[25]			✓			●
VisPro	[97]	✓					●
FESA-UML	[23]		✓		●		
Pounamu	[64]	✓					●
Milo	[72]	✓					●
SiMoNa	[18]			✓			●
Midgar	[33]	✓					●
MetaEdit+ 5.5	[84]	✓					●
Microsoft Threat Modeling	[60]	✓				●	
Secure*BPMN	[15]						
Total		12	16	18	27	5	14

infographics domain-specific modelling language. As stated in [90], lacking tool support is due to the low maturity of the secure software design field and there is a gap in this area.

7.3 Representation support

UML-based notations . The majority of used notations to represent security concerns are founded to be a Unified Modeling Language (UML) based, as stated in [90]. Since the main focus, in this paper, is non-functional requirements (NFRs), most of the notations that have been collected are concern about security. Consequently, as seen in Table 8 most of them found to be UML based (27 notations out of 47). Some of the design notations are created to only detect new vulnerabilities while others such as Georg-AO [28] does not. In Georg-AO proposed notation [28], they assess if this attack cause a huge risk then some security mechanism will be applied to mitigate it.

DFD-based notations . As mentioned previously, most of the common security design notations are *UML-based.* Data Flow Diagram (DFD) is another notation based that followed by some, such as VandenBerghe-DFD and SPARTA modeling notations [88][77]. In VandenBerghe-DFD, they proposed a model which is inspired by DFD with some security elements and attached with well-defined semantics. In this model, the expert developer knowledge's that used for identifying and mitigating the potential threats will be supported to guarantee the correctness of applied security solution. This model has been illustrated using a banking system to show how it can be used as a strong foundation for security by design paradigm. In SPARTA, they represented a prototype that simplifies the embedding of security and privacy. That's done by supporting the process of capturing security and privacy patterns in a DFD-based design then provide threat elicitation based on constructed knowledge.

LINDDUN and Sion-LINDDUN both are another example that use DFD in their activities representation [93] [94] [78]. LINDDUN is threat modelling methodology that is focusing mainly on privacy where a systemic approach for producing the privacy requirements is proposed. Identifying all potential privacy threats is done by iterating over the model elements. After that, the threats are manually assessed based on their importance (likelihood and impact). Likewise, Microsoft Threat Modeling, which follows a STRIDE, is threat modelling methodology for security.

Non UML or DFD based notations . UML and DFD, as explained previously, are general-purpose modelling languages for many of the systems. On the other hand, some languages are Domain-Specific Modeling Languages (DSML) or very specialized ones (14 in total in this study) as seen in Table 8. In DSML, a domain-specific language is used to represent a system such as SiMoNa [18]. SiMoNa is an IoT Infographic Domain-Specific Modeling Language. It is built using MetaEdit+ 5.5 workbench software which uses a GOPPRR(Graph, Object, Port, Property, Relationship and Role) meta-modelling language.

Furthermore, there are very specialized visual language such as ADM-RBAC and Giordano-AC [20] [30]. ADM-RBAC (Ariadne Development Method with Role-Based Access Control) is used for modelling hypermedia and web systems area to specify RBAC access rules at integrated two abstraction levels. This notation is an extension of the Ariadne Development Method (ADM) which does not use UML for its representation. Instead, the visual model is designed for role-based access control (RBAC); where a role can be created with defined relations, assigned permissions and generated policies. In conceptual models, policies are specified using function specifications, authorization rules and the user diagrams. In detailed models, policies are specified using access tables in addition to some of the previously produced models from conceptual phase with more details [20]. Likewise ADM-RBAC, Giordano-AC model [30] is a visual model for role-based access control (RBAC) which does not use UML mainly for its representation. In Giordano AC system, there are many tools for enabling a different kind of users to edit the security policies visually. After that, the system will generate XACML (eXtensible Access Control Markup Language) code.

7.4 Validation of the Notations

As seen in Table 9, a summary of the performed validation type that has been done for each design notation along with the number and educational background of participants if there is any. It is noticeable that the majority of the notations lack case studies and experiments; in contrast, most of them have been illustrated in one or multiple examples. However, most of the illustration examples

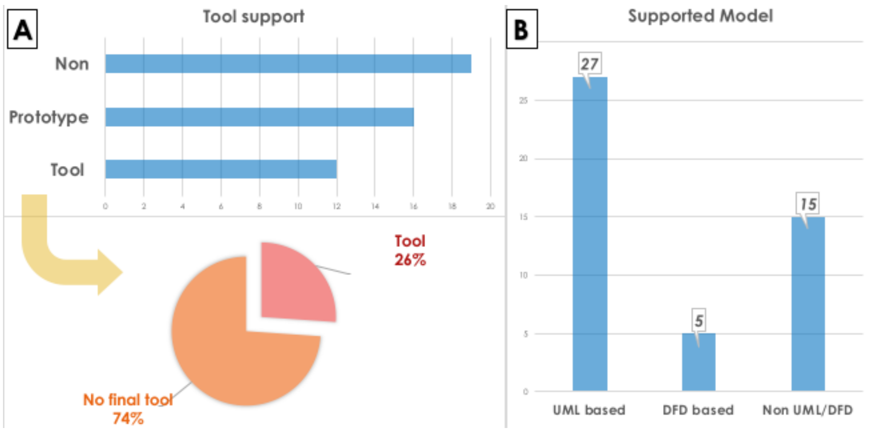


Fig. 11. chart (A) shows most of the notations did not produce any tool (13 out of 47). chart (B) shows that most of the analyzed papers are UML-based modelling.

are abstract and lack of deep details. Only a few of the notations used experiments (10 out of 47) and some have case studies (10 out of 47) as a way of validation. In the experiments, it is noticeable that most of the experiments‘ participants have master degree as seen in Figure 12. Besides that, some do not have any kind of illustration validation such as Sohr-AC and Ahn-AC.

While some notations such as Giordano AC uses a group of heterogeneous people such as from universities, industrial managers and technicians, others like ADM-RBAC and PbSD, uses a group master and bachelor students respectively. In ADM-RBAC [20], a comparison has been done on the experiment; where 18 evaluators are divided into two groups. One of the groups trained while the other is not. All the evaluators are first-year master students, and the evaluation is part of a Hypermedia Design course. These experiments are done on a small scope and lack of deep analysis. In Giordano AC [30], the authors validate their model using both the use case and experiment. The case study done by using 20 evaluators who are managers and technicians are from universities and industries, participated to test system usability using a technique called think. In the experiment, they compared their notation to the most related visual-based tool called XGrid. The evaluators participated for three days and follow the think-aloud technique using a one-to-one session. Each validator has an introductory course for 90 minutes about the two systems and their notations. After that, they were asked to use these systems for 20 minutes with tutor support. Then, they were asked to apply three different scenarios with using the systems tools in the definition of access policies. The scenarios include file hosting service, Massively multiplayer online role-playing game (MMORPG), and content management system. This experiment is done on a small scale (only 6 participants) and lacks an in-depth analysis. In the use case, they produce a Multimedia Content Management System (MCMS) as a collaboration with a software development company and embed access control policies using their approach. MCMS is based on open source content management system called OpenCMS, that provide extending some modules simply. By applying their approach, the authorizing and administrating users are simplified.

In PbSD [2], 148 third-year undergraduate students participated in the experiment from (Security of Computers and Communication Networks) course. The students are from different departments: Information Systems Engineering (ISE) and Software Engineering (SE) programs and were divided into two groups to apply some tasks to see the security requirements comparison between PbSD with SQL and OracleâĂŹs VPD.

Table 9. Validation techniques that used for each notation which can be Case studies, Experiments and illustrations. Novice, undergraduate, master, PhD and expert are the experimentsâĂŹ participants educational background. (*)For Experiments column: the number represents the total number of participants in the experiment for each notation. Empty cell indicates there is no experiment done. (-) in SPARTA means that they stated there is an experiment, but they did not give details about the participants number or background.

		Participants background					**Validation technique**		
Notation	**Citation**	Novice	Undergraduate	Master	PhD	Expert	Case studies	Experiments*	Illustrations
ADM-RBAC	[20]			✓				18	✓
Ahn-AC	[4]								
Alam-SECTET	[4]								✓
AMF	[44]								✓
Buyens-LP	[12]						✓		✓
FDAF	[16]								✓
Georg-AO	[28]								✓
Giordano AC	[30]			✓	✓	✓	✓	20	
Gomaa UML	[31]								✓
Hafner-SOA	[35]								✓
Hoisl-SOA	[41]								✓
Kim-AC	[49]						✓		✓
Kong-Threat	[52]								✓
Mariscal-AC	[69]								✓
Medina-DB	[86]						✓		✓
Memon-SECTET	[57]								✓
Nakamura-SOA	[75]								✓
PbSD	[2]			✓				148	✓
Ray-AC	[73]								✓
SecureSOA	[58]								✓
SecureUML	[10]								✓
Sohr-AC	[79]								
UML AC	[51]								✓
UMLS	[37]								✓
UMLsec	[48]						✓		✓
Vela-DB-XML	[91]						✓		✓
Xu-Petri	[95]						✓		✓
Yu-AC	[96]								✓
Hafner-SECTET	[36]								✓
SPARTA	[77]							-	
VandenBerghe-DFD	[88]								✓
LINDDUN	[94]								✓
Sion-LINDDUN	[78]								✓
Kaitiaki	[53]								✓
Suzuki-UML	[82]								✓
VLDesk	[26]						✓		✓
VESIG	[17]			✓				12	✓
Superlog	[25]								✓
VisPro	[97]						✓		✓
FESA-UML	[23]								✓
Pounamu	[64]								✓
Milo	[72]		✓					20	✓
SiMoNa	[18]								✓
Midgar	[33]	✓				✓		21	✓
MetaEdit+ 5.5	[84]								✓
Microsoft Threat Modeling	[60]								✓
Secure*BPMN	[15]			✓		✓	✓	31	✓
Total		1	1	5	1	3	10	270	43

Likewise PbSD, VESIG initial tool was tested in (Formal Methods for Web Design) course using 12 master students on Science and Technology in Computing [17]. The required design task is

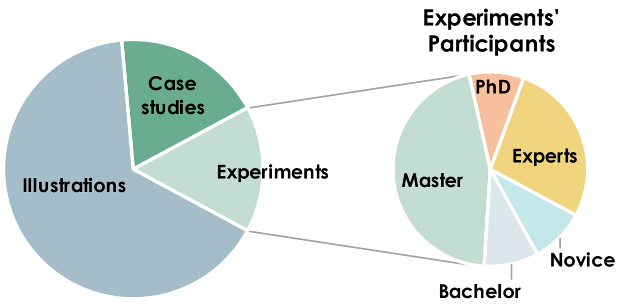


Fig. 12. Pie chart (left) shows validation type. Pie chart (Right) shows experiment participants educational background.

done by pairs for 90 minutes divided as: 10 minutes was an introduction to the tool from VESIG team, 45 minutes to draw a sketch on a paper, and the remaining time is used to apply the sketch using the tool and completing 6 open questions. In the end, 6 different design and 12 questioners are collected. Not only postgraduate students are the one used for evaluation, but Milo web-based visual programming experiment is also done on 20 computer science undergraduate students [72].

In Midgar [33], the experiment is done based on 21 participants who tested individually. 12 of the participants are software developers and 9 of them are people who interested in IoT. They have been asked to evaluate the applications' editor and generator from the developer and the user point of view. In SPARTA [77], they stated that the evaluation is done by running a risk analysis on the WebRTC, submitting SPARTA system to SecureDrop system, and comparing it with Microsoft Threat Analysis performance.

Some of the notation their case studies are similar to an example scenario. For example, Kim-AC case study is presented as an application for online shopping while Xu-Petri case study is about modelling real-world shopping cart application [95] [95]. Both of them is applying STRIDE technique to identify threats. There are no participants for real and extensive analysis. Medina-DB on the other hand, has a deep case study about the Secure Data Warehouse for several pharmacies done by the authors themselves in separate paper [80]. Likewise, VisPro and UMLsec have some examples and case studies some of them have some details [48][97].

8 DESIGN PRINCIPLES (TOOLING) FOR NON FUNCTIONAL REQUIREMENTS

So far reviewed number or different design notation and languages that are being developed to design application. The next most important step is to make these notations available for the software engineers to use within SDLC. For example, Visual Paradigm (visual-paradigm.com) is a Tool that has been developed to make design notations and languages (such as UML 2, SysML and Business Process Modeling) available for software engineers to use. These tools are designed to augment the software design capabilities of software engineers and to make the software development process efficient and effective (e.g., help to designs faster, help to reduce human errors/mistakes). In this section, we are going to review different design principles and how they are being applied in existing design tools (related to security / privacy / IoT).

Designing is considered as an easy thing to do when everything goes smoothly and right until a mistake or misunderstanding happened then complications occur rapidly. One of the solutions is adapting a human-centred design (HCD) approach [47]. In this approach human needs, behaviours and capabilities are taken into consideration in the first place. Therefore, understanding psychology should go with understanding the technology side by side. Additionally, this approach adapts building good communication to achieve good design especially if there is a machine required in the design. This is done for understanding the interaction between machines and humans. It is more important for designers to understand what will happen if things go wrong more than what will happen if it goes as planned.

8.1 Fundamental Principles of Interaction

There are many principles of good interaction design and one of them is Norman [65]. He divides the principles to five which are: affordances, signifiers, mapping, feedback, and conceptual model. Three tools will be evaluated based on these principles later to see who it can be applied. It can be seen in Table 10 which are: ViSiT, Microsoft threat modelling and ThreatDragon.

Affordance term means the relationship between the individual and object, in the way of how the person can understand the features of the object and how it can be used [65]. The physical object should express how people can interact with them and this should be visible in the design. For designers, affordance distinguishability is very critical because visible affordances deliver clear signs to the operations of objects. Users should not pay a lot of effort to understand the proper way of how to use the features of things.

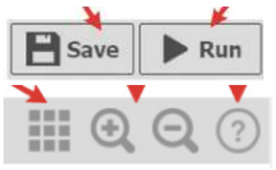
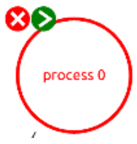
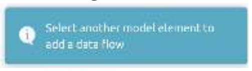
Signifiers are the guidance to the user of how to follow what the object can afford and do it in the right way. Designers have the responsibilities to design something understandable and quickly picked by the users. If there is a touch screen where the users need to touch something in order to go to other windows, there should be signifiers to tell them where exactly to touch. For example, there is a clinic profile as seen in Figure 13. This page has many icons and arrows which offer signals about the produced actions for the clinic. Swiping up left and right arrows move to new clinic suggestions. Swiping down is a sign to see people recommendations and reviews. Clicking the map pin icon will direct to the clinic location on the map. Affordance and signifiers are easy and common to mix in between [65], so in this case, touching is what the object affords, the arrow is the sign where the user should apply the touch. If we apply this principle to ViSiT [68], the tool UI has many signifiers such as: Having grid-lines in the canvas that guide the user to place the target collections and properties, having many buttons such as the one to save and run the transformation, having icons to zoom canvas or ask for help.

Mapping in simple words is the relationship between two things. In design, it is easy to use object if there is a mapping between the layout and device. For example, if there is an arrow pointing up at the touch screen, it supposed when it is pressed to move the object up not down, left or right. A good design put in mind, during planning, how people behave to develop mapping. IoT design application tool should consider this principle while implementing the UI. For Example, in Table 10 ViSiT [68] has the icon of a floppy disk which is known as mapping for saving the task.

Feedback: is away of allowing the users to know that the request is under process. It is a kind of communication with the users, so they do not feel ignored or lost. Good design must balance between ignoring the feedbacks or overusing them which both are annoying and lead to stop using the object (such as system software). So, while designing a designing tool this should be kept in mind to balance between asking and annoying users about every single privacy concerns, and keeping them notified about the critical one and keep some of them as default settings.

A conceptual model as it is defined in some sources as âĂIJa high-level description of how a system is organized and operates” [46]. In general, it is a simplified and easy way to explain to

Table 10. Comparison between three tools: ViSiT, Microsoft threat modeling and Threat Dragon in term of principles of good interaction design.(*) ViSiT is analyzed based on the Figures on paper. (**) Not mentioned on the paper.

	ViSiT*	Microsoft threat modeling	ThreatDragon
AFFORDANCES	It has visible affordance where user can understand the idea of dragging and dropping expressions then running the transformations.	It has visible affordance where user can understand the idea of drawing DFD as any other DFD drawing tool. Then it can do the threat analysis. It also supports creating custom threat templates.	It has visible affordance where user can understand the idea of drawing DFD. However, it does not support drag and drop which is used to be in most of designing tools.
SIGNIFIERS	It supports many signifiers such as: • icons (zoom in/out, help, delete expression) • buttons (save, run ..etc) • gridline 	It supports many signifiers such as: • icons (zoom in/out, help) • buttons (save, run ..etc) • drop-down menus • grid-line However, linking two objects is not clear due to lacking for signifiers to say how to do that. It takes many tries to understand that one of the ways is selecting the two object then right-click to choose the linking option.	It supports many signifiers such as: • icons (zoom in/out) • drop-down menus However, the design is simple and lack of extra buttons and more signifiers. The green arrow above the object: 
MAPPING	When an object is taken from the target and properties collection, the object automatically placed in the shown grid at the left-hand side. It supports drag and drop from the toolbox.	Mapping is supported.	Low mapping support. When the object from the toolbox is double-clicked, it appears in the canvas. Having toolbox located at the left contains many objects, sometimes indicates dragging the objects to the canvas; however, it does not support drag and drop which is used to be in most of designing tools.
FEEDBACK	N/A**	Feedback is supported. For example, when a developer is changing the name of any object from the stencils, it appears immediately at the object in the main canvas.	There is feedback for example when the arrow is pressed message appears to tell the user what to do. However, the response time is slow, and the user can doubt about the request will be done or not. The message:  Select another model element to add a data flow

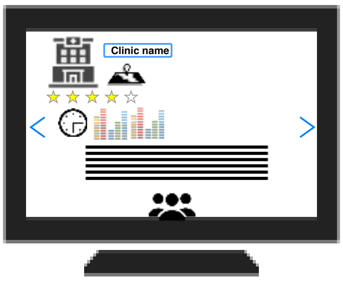


Fig. 13. Icons and arrows signifiers are used here on a touch screen. They offer signals about the produced actions for the clinic. Swiping left and right arrows move to new clinic suggestions. Swiping down, to see people recommendations and reviews. Clicking the map pin icon will direct to the clinic location on the map; clicking on the clock will show the opening hours.

people about how something going to work. For example, using folders and icons in a computer OS is a way of helping users to create the conceptual model; while in the reality this is not the way files are organized and saved inside the computer. There are many different conceptual models such as the one shown in the product technical support; however, the one that most concerned in designing is the mental model [47]. The mental model reflects how people understand how things work. The same item could have different mental models created by different people. In fact, a person may have a variety of mental models for the same item depending on his interpretation for each part's function. The good conceptual model should be clear and understandable by the support of using affordances, signifiers and constraints.

By adopting a good conceptual model in IoT application designing tool, it will be easier for the developer to predict the effects and what plan to follow if things go wrong and not as planned while designing a data flow for an IoT application. If the model is complex that means the user should read the manual, again and again, to understand how things work.

Parush [68] adds some details and divides the conceptual model for interactive systems it into five layers framework. These layers starting from bottom up are: function, configuration, navigation and policy, form and details layer. For illustration, all these layers will be mapped to Visual Simple Transformations tool called ViSiT [5] as seen in Figure 14. This tool is enabling the end-user to connect the Internet of Things (IoT) entities (services and things). It visualizes IoT solutions by using the metaphor of the jigsaw puzzle for specifying the transformation. This hierarchical conceptual model could be adapted while developing an interactive IoT designing tool.

In the bottom is the function layer where a group of objects and tasks and their properties are stated to fulfil the required work such as how thing and service in ViSiT tool interact with each other. Configuration layer represents the abstraction of the functions and its relationship between its elements. It can be defined as functional architecture where the configuration of the conceptual model elements is based on the elements' relationship. In websites design, functional and information architecture are equivalent but in conceptual design is not. Information here is one of the elements as well as the object and its parameters. The navigation and policy layer describes the interactive system where navigating takes the user from place to another place based on predefined navigation map for routing instructions'. For example, if the user wants to drag a puzzle piece into the canvas (physical place) will it be in the same window (place) or dragged into other windows (places). Form layer is the layer that transforms from a high abstraction level to more detailed one that can be designed later into a user interface (UI). In this layer, many decisions are made which could affect the performance and usability of the system later on. The details of function chunks are mapped into physical places and connected with some actions such as where to fill in the properties and values of one event of ViSiT in order then to link it with a method.

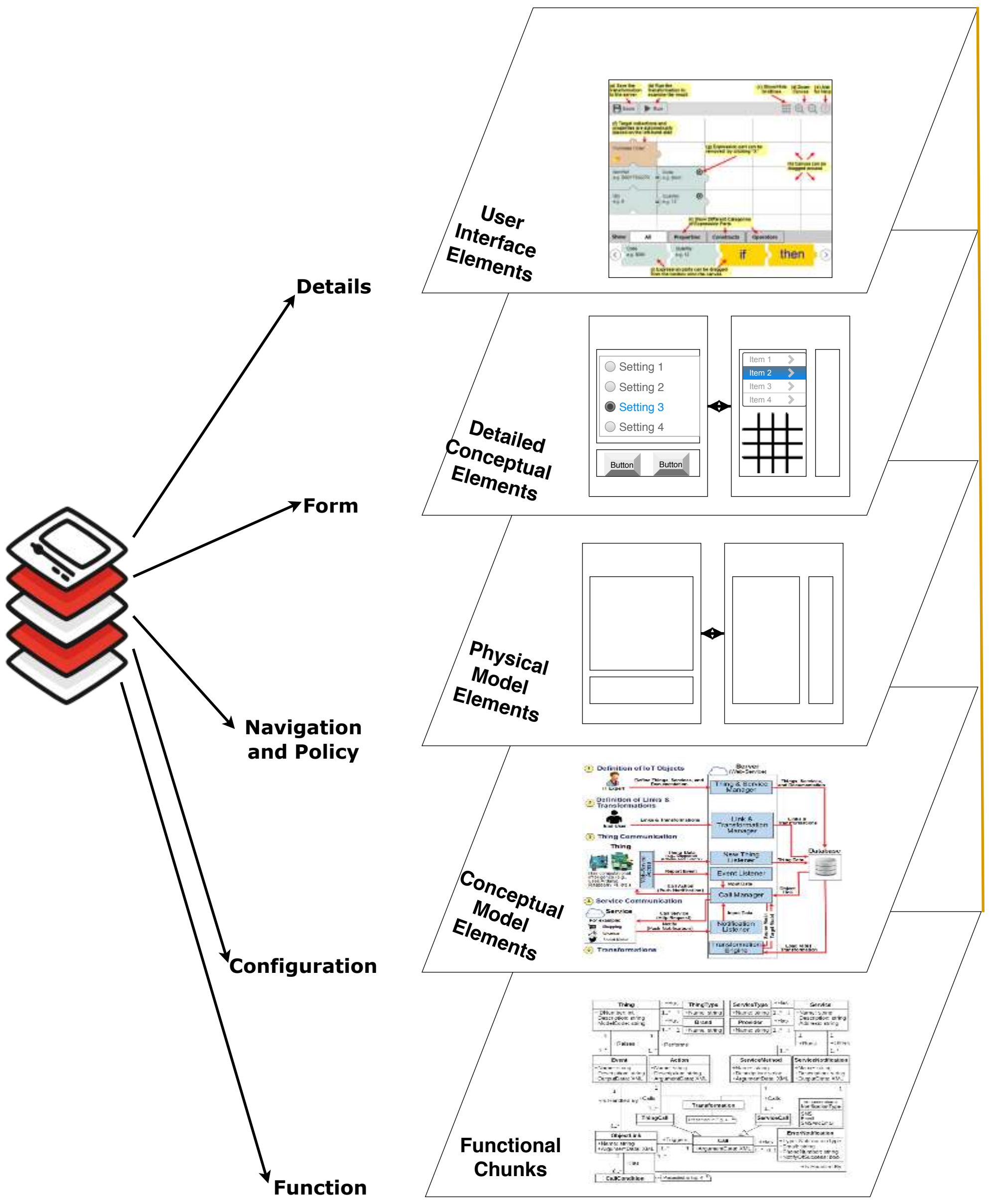


Fig. 14. Parush layered framework [68] that represents conceptual model for interactive systems is applied on ViSiT IoT tool [5] to explaine how it can adapted while developing an interactive IoT designing tool.

Finally, the user interface is the top layer of the conceptual model where all the details of graphical representation and visual layout are specified such as locations, colours, fonts and icons of the items.

As said ViSiT IoT tool supports somehow detailed model while other tools do not. Threat Dragon, which is threat modelling tool, is more basic without any clear support to IoT. The threat is represented simply without further details in the right side of the tool windows in the form of

Table 11. Threat Dragon compared to Microsoft threat modeling.

	Web-based	Desktop-based	Open source	Representation	IoT support
Threat Dragon	✔	✔	✔	DFD	✗
Microsoft threat modeling	✗	✔	✗	DFD	✔

chick boxes. On the other hand, the Microsoft threat modelling tool supports IoT security threats in the last version (2018). In Microsoft, after pressing analyze button the threats are presented in the bottom side of the same windows. All the threat is classified based on the term of threat title, category, description and priority...etc. Some of them even linked to the Microsoft website for further explanation about the threat and how it can be overcome. A short comparison between both of them (Microsoft threat modelling and Threat dragon) is illustrated in Table 11.

As will as ViSiT IoT tool, some analyzed notations such as ADM-RBAC [20] supports detailed models for access specification to provide further details from the previous phase. In contrast, both Microsoft Threat Modeling tool and Threat Dragon tools have a single level of view where everything is shown in the same window and they do not support any kind of multiple detailed views or moving into other windows. Both approaches, basic or detailed user interface can be mixed to have an IoT designing tool that affirms to what user needs.

9 RESEARCH CHALLENGES AND OPPORTUNITIES

As explained previously IoT application design has some challenges and adding some changes or enhancements to the traditional SDLC are required. Below are some of the presented solutions or enhancements to make IoT development more controllable such as integrating some threat modelling techniques during the design stage such as STRIDE, introducing a framework to support privacy, or enhancing the SDLC to include different views such as prosumerization.

9.1 Lack on Notations and Languages for Privacy

One of the solutions to make a system secure while designing is using threat modelling techniques such as OWASP [67] and Microsoft Threat Modeling Tool [60] during the design phase. There are two main methodologies, STRIDE and DREAD, that have been used for most of threat modelling techniques [24]. STRIDE, which Microsoft Threat Modeling Tool uses, is acronym made up for mapping the threat categories which are: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege. DREAD acronym for risk calculation metrics which are: Damage Potential, Reproducibility, Exploitability, Affected Users and Discoverability. The difference between the two is that STRIDE is used for threat identification while DREAD is used for risk calculation. Both OWASP [67] and Microsoft Threat Modeling Tool [60] are used for security threats and none of them used for privacy in particular.

9.2 Lack of Tools to Supplement Methods

As stated in Section 9.1, there are some tools that are used for security threats with absence for a framework for privacy. Since the main focus of STRIDE is security threat modelling, therefore LINDDUN was introduced to cover the privacy requirements. LINDDUN is a model-based technique/ framework that has been done to focus exclusively on modelling privacy threats in software-based systems [94]. It is an acronym for seven types of threats: Linkability, Identifiability, Non-repudiation, Detectability, information Disclosure, content Unawareness and policy, and consent Non-compliance. All of these threats are mapped to the application DFD parts after it is created as seen in Table 12.

Table 12. Security and privacy concerns with its corresponding threats mapped with DFD elements that vulnerable to threats. (DF: Data flow, DS: Data store, P: Process, E: External entity). STRIDE and LINDDUN are proposed by the Security Development Lifecycle (SDL) [43] and Privacy Threats in Software Architectures [94] respectively.

		Property	Threat	DF	DS	P	E
Security	STRIDE	Authentication	**S**poofing			•	•
		Integrity	**T**ampering	•	•	•	
		Non-repudiation	**R**epudiation		•	•	•
		Confidentiality	**I**nformation Disclosure	•	•	•	
		Availability	**D**enial of Service	•	•	•	
		Authorization	**El**evation of Privilege			•	
Privacy	LINDDUN	Unlinkability	**L**inkability	•	•	•	•
		Anonymity and pseudonymity	**I**dentifiability	•	•	•	•
		Plausible deniability	**N**on-repudiation		•	•	•
		Undetectability and unobservability	**D**etectability		•	•	•
		Confidentiality	**D**isclosure of information		•	•	•
		Content awareness	Content **U**nawareness	•			
		Policy and consent compliance	Policy and Consent **N**on-compliance		•	•	•

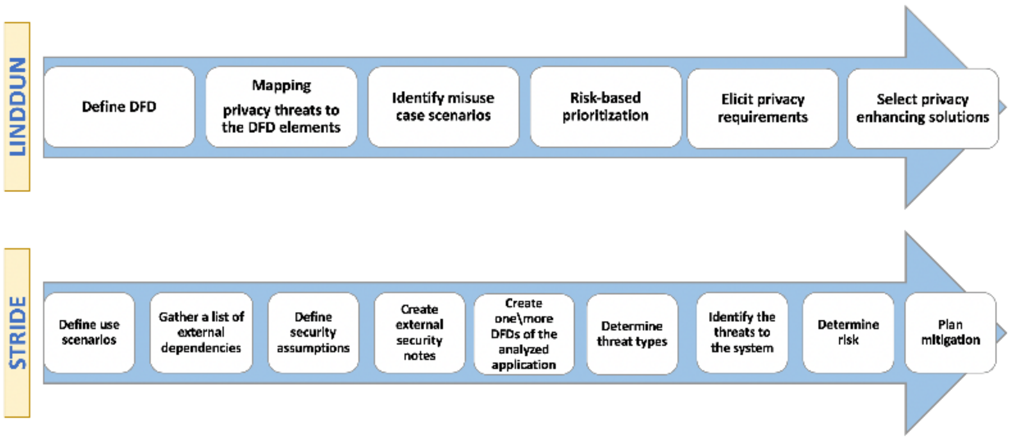

LINDDUN
Define DFD
Mapping privacy threats to the DFD elements
Identify misuse case scenarios
Risk-based prioritization
Elicit privacy requirements
Select privacy enhancing solutions
STRIDE
Define use scenarios
Gather a list of external dependencies
Define security assumptions
Create external security notes
Create one\more DFDs of the analyzed application
Determine threat types
Identify the threats to the system
Determine risk
Plan mitigation

Fig. 15. LINDDUN vs STRIDE Methodology

STRIDE and LINDUDUNN have a different methodology in the way of the order of the steps that deal with the threats as seen in Figure 15. In STRIDE, it is started with defining use case scenarios while it is the thirds step in LINDDUN which start with defining the DFD. As well as STRIDE, which adapted in Microsoft threat modelling tool to cover security concerns, we believe LINDDUN could be used to fulfil the privacy gap on other IoT tools. As stated, LINNDU was initiated as a framework but it does not have privacy pattern or tool yet.

9.3 Proactive Assistance

Proactive service means anticipating/predicting usersâĂŹ concerns and addressing them proactively. For instance, most of the users when they have any concerns while browsing a website or using a tool, they will go to help or contact us option. Then, it becomes a waiting issue to resolve these concerns or questions. While in proactive service, the user concerns are predicted and solutions are offered based on the behaviour and the action have been made.

By identifying and addressing any issue before it becomes a problem, availability of equipment and applications increases. Some places, such as hospitals, require high availability of critical hardware. For instance, Philips [71] has a solution called e-Alert which is used to ensure high performance of magnetic resonance imaging (MRI) systems. It is done by sensing and monitoring the system continuously and responding quickly to any possible issues with MRI by issuing mobile messaging alerts that sent to biomeds and Philips service engineers. Currently, many of the worldwide top-technical companies are tending to move from reactive to proactive engagement process. By supporting such a method, it will be possible to prevent any concerns before they arise. For example, Oracle [66] has proactive diagnostic support tools that predict system behaviour, prioritize actions, and prevent potential problems.

In IoT, the balance between the business's requirements and people's privacy is a big issue. Since IoT technology does not have completely standardize security and privacy requirements [14], there is a need for a proactive framework and assistant tool to help while developing IoT applications. There are tools with minimum proactive support or without proactive support especially the one that used during the designing phase of IoT applications. For example, if a developer wants to draw a data flow for a supermarket security system. There will be a smart camera in front of a supermarket which will be collecting Personally Identifiable Information (PII) such as human faces. The engineer does not know how long you keep the data, where it should be stored, and what is the compatible way to deal these data to agree with the Law. There is a need for a tool that waves the burden from the developer; whenever a camera is dragged and dropped, privacy concerns will pop up automatically somewhere to give a hint about IoT privacy concerns. We are focusing on the architecture/design part, where a data is flowing the privacy is preserved.

9.4 Integration of IoT Standards and Privacy

One of the studies that have been done is integrating OWASP standard Secure-SDLC with Microsoft Threat Modeling Tool in the IoT Application SDLC to produce a Secure-SDLC for IoT [24]. In Fernandes [24] study, privacy and security are demonstrated in healthcare application (IoT based Health Monitor). This application is used for tracking and monitoring the body temperature and pulse rate of hospitalsâĂŹ patients. Here, security has been integrated while designing and implementing the system. The main idea is embedding security requirements from first to the final software\system development stages, from requirements gathering to deployment.

At each stage, a number of known vulnerabilities are identified and mitigated. After that, testing for IoT based Health Monitor is made by using Open Web Application Security Project (OWASP) top 5 IoT vulnerabilities. The SDLC is divided into five phases: Requirements Gathering, Design, Development, Testing and Deployment [24]. Each phase has its level of own security review. In Requirement Gathering phase (planning and analysis), any security requirements and significant risks are identified and added at this stage. Security requirements can be associated with one of these: authentication, authorization, error handling, session management, input validations, logging, secure communications, storage... etc.

In the Design phase of the IoT solution, security threat modelling techniques are used for revising the design. At the development\implementation phase, code revision is done to identify all coding errors that can cause any security risks. In the testing phase, the system is tested to confirm and validate that it meets the FRs as well as the NFRs. For example, testing the system using tests cases with OWASP Top 10 vulnerabilities. This step can make sure the system has a level of security. In the final stage, deployment, secure configurations are added to the system configurations with retesting for further variabilities [24].

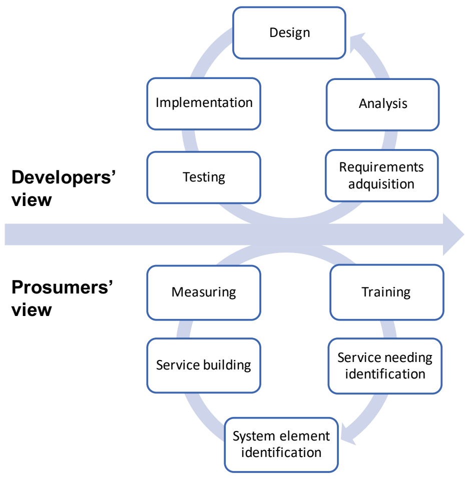


Fig. 16. Methodological approach in Developers' and Prosumers' views based on [55].

9.5 Tools and Methods for Stakeholder Engagement through Codesign

Another way to develop IoT system is to follow End-user development (EUD) approaches for design and development such as the one represented in Martin work [55]. End-user development (EUD) suggests that end-users could develop their own programs by giving them suitable tools that fulfil their requirements and save costs and efforts.

In this Martin [55], a framework was created for enabling the prosumer users to create services in IoT scenarios since these scenarios are ideal for adopting the prosumer role. Prosumer is a combination of producer and consumer where the individual consumes and produces a product. The main goal for this framework is to enable the end-user (who does not have a high level of expertise) to be involved in development stages. The framework structured based on two interacted roles which are developers and prosumers. Each one of them has its own view as seen in Figure 16.

In the developers' view, the defined methodology has the five traditional stages of software development which are: requirement acquisition, analysis, design, implementation and testing. Each of these developer stages in the service creation framework should take into account the prosumer value. There are five more stages that present the prosumers' view which are: training, service needing identification system element identification, service building, and measuring. This framework was tested for creating and personalizing templates of web technologies in hospital pharmacy drug management. The built prototype was tested by allowing prosumer users, who have no technical expertise, such as pharmacy workers. They did some actions such as creating and sharing stock notifications for drugs available in the pharmacy. This testing was made to make sure the presented approach has benefited from engaging the prosumer in the design phase to increase the usability of the system.

10 CONCLUSION

There is evidence that number of IoT applications being developed are growing. This increasing is facing many complications in the field of designing and developing these applications such as supporting nodes heterogeneity, multiple hardware/software, multiple technologies, and developers' divers working togetherâĂęetc. With all of these complexities, all of the components are vulnerable to attacks. In IoT, security/privacy as non-functional requirements (NFRs) are critical due to its complicated nature; however, they tend to be ignored.

This survey has stated the results of a literature review in the field of designing non-functional requirements in the internet of things. We examined, how such results can be applied to reduce IoT application design process complexity. In this survey, 47 notations, language, representation have been systematically scanned. Design principles, challenges and opportunities are also provided. The study has shown that most of the analyzed publications support security somehow, and rarely support privacy. It also highlights potential issues in supporting proactive design tools for IoT privacy. Finally, we identified and discussed six research gaps related to privacy in IoT that need to be addressed.

REFERENCES

[1] ISO/IEC JTC 1/SC 27. 2011. ISO/IEC 29100:2011(en): Information technology âĂŤ Security techniques âĂŤ Privacy framework. https://www.iso.org/obp/ui/{#}iso:std:iso-iec:29100:ed-1:v1:en

[2] Jenny Abramov, Arnon Sturm, and Peretz Shoval. 2012. Evaluation of the Pattern-based method for Secure Development (PbSD): A controlled experiment. *Information and Software Technology* 54, 9 (2012), 1029–1043. https://doi.org/10.1016/j.infsof.2012.04.001

[3] Yuvraj Agarwal and Malcolm Hall. 2013. ProtectMyPrivacy: detecting and mitigating privacy leaks on iOS devices using crowdsourcing. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. ACM, 97–110.

[4] Gail Joon Ahn, Seung Phil Hong, and Michael E. Shin. 2002. Reconstructing a formal security model. *Information and Software Technology* 44, 11 (2002), 649–657. https://doi.org/10.1016/S0950-5849(02)00092-7

[5] Pierre A Akiki, Arosha K Bandara, and Yijun Yu. 2017. Visual Simple Transformations: Empowering End-Users to Wire Internet of Things Objects. *ACM Trans. Comput.-Hum. Interact* 24, 10 (2017). https://doi.org/10.1145/3057857

[6] D Ameller, X Franch, and J Cabot. 2010. Dealing with Non-Functional Requirements in Model-Driven Development. In *2010 18th IEEE International Requirements Engineering Conference*. 189–198. https://doi.org/10.1109/RE.2010.32

[7] Luigi Atzori, Antonio Iera, and Giacomo Morabito. 2010. The internet of things: A survey. *Computer networks* 54, 15 (2010), 2787–2805.

[8] S. Balaji. 2012. Waterfall vs V-Model vs. Agile: A Comparative Study on SDLC [Electronic version]. *International Journal of Information Technology and Business Management* 2, 1 (2012), 26–30. https://doi.org/10.1.1.695.9278

[9] Rebecca Balebako, Abigail Marsh, Jialiu Lin, Jason I Hong, and Lorrie Faith Cranor. 2014. The privacy and security behaviors of smartphone app developers. (2014).

[10] David Basin, Manuel Clavel, Jürgen Doser, and Marina Egea. 2009. Automated analysis of security-design models. *Information and Software Technology* 51, 5 (2009), 815–831. https://doi.org/10.1016/j.infsof.2008.05.011

[11] Jan Lauren Boyles, Aaron Smith, and Mary Madden. 2012. Privacy and Data Management on Mobile Devices. https://www.pewinternet.org/2012/09/05/privacy-and-data-management-on-mobile-devices/

[12] Koen Buyens, Riccardo Scandariato, and Wouter Joosen. 2013. Least privilege analysis in software architectures. *Software and Systems Modeling* 12, 2 (2013), 331–348. https://doi.org/10.1007/s10270-011-0218-8

[13] Ioannis Chatzigiannakis, Georgios Mylonas, and Sotiris Nikoletseas. 2007. 50 ways to build your application: A survey of middleware and systems for wireless sensor networks. In *2007 IEEE Conference on Emerging Technologies and Factory Automation (EFTA 2007)*. IEEE, 466–473.

[14] Abhik Chaudhuri and Ann Cavoukian. 2018. The Proactive and Preventive Privacy (3P) Framework for IoT Privacy by Design. *EDPACS* 57, 1 (2018), 1–16.

[15] Y. Cherdantseva. 2014. Secure * BPMN - a graphical extension for BPMN 2 . 0 based on a Reference Model of Information Assurance & Security Yulia Cherdantseva Cardi ff University. December (2014).

[16] Lirong Dai and Kendra Cooper. 2007. Using FDAF to bridge the gap between enterprise and software architectures for security. *Science of Computer Programming* 66, 1 (2007), 87–102. https://doi.org/10.1016/j.scico.2006.10.010

[17] Paloma D\'{i}az, Ignacio Aedo, Mary Beth Rosson, and John M Carroll. 2010. A visual tool for using design patterns as pattern languages. In *Proceedings of the International Conference on Advanced Visual Interfaces*. ACM, 67–74.

[18] Cleber Matos De Morais, Judith Kelner, Djamel Sadok, and Thea Lynn. 2018. SiMoNa: A proof-of-concept domain specific modeling language for IoT infographics. *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC* 2018-Octob, section IV (2018), 199–203. https://doi.org/10.1109/VLHCC.2018.8506502

[19] Mina Deng, Kim Wuyts, Riccardo Scandariato, Bart Preneel, and Wouter Joosen. 2011. A privacy threat analysis framework: Supporting the elicitation and fulfillment of privacy requirements. *Requirements Engineering* 16, 1 (2011), 3–32. https://doi.org/10.1007/s00766-010-0115-7

[20] Paloma Díaz, Ignacio Aedo, Daniel Sanz, and Alessio Malizia. 2008. A model-driven approach for the visual specification of Role-Based Access Control policies in web systems. *Proceedings - 2008 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2008* (2008), 203–210. https://doi.org/10.1109/VLHCC.2008.4639087

[21] European Union. 2016. Regulation 2016/679. *Official Journal of the European Communities* 59, L 119 (2016), 1–88. https://doi.org/pri/en/oj/dat/2003/l_285/l_28520031101en00330037.pdf arXiv:arXiv:1011.1669v3

[22] Dave Evans. 2011. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper* 1, 2011 (2011), 1–11.

[23] Tibor Farkas, Carsten Neumann, and Andreas Hinnerichs. 2009. An integrative approach for embedded software design with UML and simulink. *Proceedings - International Computer Software and Applications Conference* 2 (2009), 516–521. https://doi.org/10.1109/COMPSAC.2009.185

[24] Avelet Maria Fernandes, Anusha Pai, and Louella M.Mesquita Colaco. 2018. Secure SDLC for IoT Based Health Monitor. *Proceedings of the 2nd International Conference on Electronics, Communication and Aerospace Technology, ICECA 2018* Iceca 2018 (2018), 1236–1241. https://doi.org/10.1109/ICECA.2018.8474668

[25] Peter L Flake and Simon J Davidmann. 2000. Superlog, a unified design language for system-on-chip. *Proceedings 2000. Design Automation Conference.(IEEE Cat. No. 00CH37106)* (2000), 583–586. https://doi.org/10.1145/368434.368814

[26] R Francese, G Scanniello, G Costagliola, A De Lucia, and M Risi. 2002. A component-based visual environment development process. *Proceedings of the 14th international conference on Software engineering and knowledge engineering* (2002), 327–334. https://doi.org/10.1145/568813.568818

[27] David Geer. 2010. Are companies actually using secure development life cycles? *Computer* 43, 6 (2010), 12–16.

[28] Geri Georg, Indrakshi Ray, Kyriakos Anastasakis, Behzad Bordbar, Manachai Toahchoodee, and Siv Hilde Houmb. 2009. An aspect-oriented methodology for designing secure applications. *Information and Software Technology* 51, 5 (2009), 846–864. https://doi.org/10.1016/j.infsof.2008.05.004

[29] Nam Ky Giang, Michael Blackstock, Rodger Lea, and Victor C M Leung. 2015. Developing IoT applications in the fog: a distributed dataflow approach. In *Internet of Things (IOT), 2015 5th International Conference on the*. IEEE, 155–162.

[30] Massimiliano Giordano, Giuseppe Polese, Giuseppe Scanniello, and Genoveffa Tortora. 2010. A system for visual role-based policy modelling. *Journal of Visual Languages and Computing* 21, 1 (2010), 41–64. https://doi.org/10.1016/j.jvlc.2009.11.002

[31] M Eonsuk Gomaa, Hassan and Shin. 2004. Modeling Complex Systems by Separating Application and Security Concerns. In *Ninth IEEE International Conference on Engineering of Complex Computer Systems*. 19—-28.

[32] Francisco Gomariz-Castillo, Irene Garrigós, Jose-Alfonso Aguilar, Jose Zubcoff, Sven Casteleyn, and Jose-Norberto Mazón. 2018. Evaluating different i*-based approaches for selecting functional requirements while balancing and optimizing non-functional requirements: A controlled experiment. *Information and Software Technology* 106, January 2017 (2018), 68–84. https://doi.org/10.1016/j.infsof.2018.09.004

[33] Cristian González García, B Cristina Pelayo G-Bustelo, Jordán Pascual Espada, and Guillermo Cueva-Fernandez. 2014. Midgar: Generation of heterogeneous objects interconnecting applications. A Domain Specific Language proposal for Internet of Things scenarios. *Computer Networks* 64 (2014), 143–158. https://doi.org/10.1016/j.comnet.2014.02.010

[34] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems* 29, 7 (2013), 1645–1660.

[35] M Hafner, M Breu, R Breu, and A Nowak. 2005. Modelling inter-organizational workflow security in a peer-to-peer environment. In *IEEE International Conference on Web Services (ICWS'05)*. 540. https://doi.org/10.1109/ICWS.2005.83

[36] M. Hafner, R. Breu, B. Agreiter, and A. Nowak. 2006. SECTET - An extensible framework for the realization of secure inter-organizational workflows. *Internet Research: Electronic Networking Applications and Policy* 16, 5 (2006), 491—-506.

[37] Rogardt Heldal and Fredrik Hultin. 2003. Bridging Model-Based and Language-Based Security. *European Symposium on Research in Computer Security* LNCS 2808 (2003), 235–236.

[38] Sehyeon Heo, Sungpil Woo, Janggwan Im, and Daeyoung Kim. 2015. IoT-MAP: IoT mashup application platform for the flexible IoT ecosystem. In *2015 5th International Conference on the Internet of Things (IOT)*. IEEE, 163–170.

[39] Michael Hirsch. 2002. Making RUP Agile. (2002).

[40] Jeffrey A Hoffer. 2012. *Modern Systems Analysis and Design, 6/e*. Pearson Education India.

[41] Bernhard Hoisl, Stefan Sobernig, and Mark Strembeck. 2014. Modeling and enforcing secure object flows in process-driven SOAs: An integrated model-driven approach. *Software and Systems Modeling* 13, 2 (2014), 513–548. https://doi.org/10.1007/s10270-012-0263-y

[42] J Hong. 2017. The Privacy Landscape of Pervasive Computing. *IEEE Pervasive Computing* 16, 3 (2017), 40–48. https://doi.org/10.1109/MPRV.2017.2940957

[43] Michael Howard and Steve Lipner. 2006. *The security development lifecycle*. Vol. 8. Microsoft Press Redmond.

[44] Hongxin Hu and Gail-joon Ahn. 2010. Constructing Authorization Systems Using Assurance Management Framework. 40, 4 (2010), 396–405.

[45] ISO/IEC JTC 1/SC 7. 2011. ISO/IEC 25010:2011(en): Systems and software Quality Requirements and Evaluation (SQuaRE) âĂŤ System and software quality models. https://www.iso.org/obp/ui/{#}iso:std:iso-iec:25010:ed-1:v1:en

[46] Jeff Johnson and Austin Henderson. 2002. Conceptual models: begin by designing what to design. *interactions* 9, 1 (2002), 25–32.

[47] Esther Jun, Huafei Liao, April Savoy, Liang Zeng, and Gavriel Salvendy. 2008. *The design of future things, by D. A. Norman, basic books, New York, NY, USA*. Vol. 18. 480–481 pages. https://doi.org/10.1002/hfm.20127

[48] Jan Jürjens, Joerg Schreck, and Peter Bartmann. 2008. Model-based security analysis for mobile communications. In *Proceedings of the 13th international conference on Software engineering - ICSE '08*, Vol. 2. 683. https://doi.org/10.1145/1368088.1368186

[49] Sangsig Kim, Dae Kyoo Kim, Lunjin Lu, Suntae Kim, and Sooyong Park. 2011. A feature-based approach for modeling role-based access control systems. *Journal of Systems and Software* 84, 12 (2011), 2035–2052. https://doi.org/10.1016/j.jss.2011.03.084

[50] Barbara Kitchenham and Pearl Brereton. 2013. A systematic review of systematic review process research in software engineering. *Information and Software Technology* 55, 12 (2013), 2049–2075. https://doi.org/10.1016/j.infsof.2013.07.010

[51] Manuel Koch and Francesco Parisi-Presicce. 2006. UML specification of access control policies and their formal verification. *Software and Systems Modeling* 5, 4 (2006), 429–447. https://doi.org/10.1007/s10270-006-0030-z

[52] JUN KONG, DIANXIANG XU, and XIAOQIN ZENG. 2010. Uml-Based Modeling and Analysis of Security Threats. *International Journal of Software Engineering and Knowledge Engineering* 20, 06 (2010), 875–897. https://doi.org/10.1142/S0218194010004980

[53] Na Liu, John Hosking, and John Grundy. 2005. A visual language and environment for specifying user interface event handling in design tools. In *EEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*. 278–280.

[54] Bogdan GHILIC-MICU Marian STOICA, Marinela MIRCEA. 2013. Software Development: Agile vs. Traditional. *Informatica Economica* 17, 4 (2013), 64–76. https://doi.org/10.12948/issn14531305/17.4.2013.06

[55] Diego Martin, Ramon Alcarria, Tomas Robles, and Augusto Morales. 2013. A systematic approach for service prosumerization in IoT scenarios. *Proceedings - 7th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, IMIS 2013* (2013), 494–499. https://doi.org/10.1109/IMIS.2013.89

[56] Bruce R Maxim and Marouane Kessentini. 2016. Chapter 2 - An introduction to modern software quality assurance. In *Software Quality Assurance*, Ivan Mistrik, Richard Soley, Nour Ali, John Grundy, and Bedir Tekinerdogan (Eds.). Morgan Kaufmann, Boston, 19–46. https://doi.org/10.1016/B978-0-12-802301-3.00002-8

[57] Mukhtiar Memon, Gordhan D. Menghwar, Mansoor H. Depar, Akhtar A. Jalbani, and Waqar M. Mashwani. 2014. Security modeling for service-oriented systems using security pattern refinement approach. *Software and Systems Modeling* 13, 2 (2014), 549–572. https://doi.org/10.1007/s10270-012-0268-6

[58] Michael Menzel and Christoph Meinel. 2010. SecureSOA - Modelling security requirements for Service-oriented Architectures. In *Proceedings - 2010 IEEE 7th International Conference on Services Computing, SCC 2010*. IEEE, 146–153. https://doi.org/10.1109/SCC.2010.63

[59] Microsoft. 2004. Microsoft Security Development Lifecycle (SDL). https://www.microsoft.com/en-us/securityengineering/sdl/

[60] Microsoft. 2018. Microsoft Threat Modeling Tool. https://docs.microsoft.com/en-us/azure/security/azure-security-threat-modeling-tool

[61] D Miorandi. 2012. Internet of Things: vision, applications and research challenges. *Internet of Things: vision, applications and research challenges* 10, 7 (2012), 1497–1516.

[62] Paula Monteiro, Pedro Borges, Ricardo J Machado, and Pedro Ribeiro. 2012. A Reduced Set of RUP Roles to Small Software Development Teams. In *Proceedings of the International Conference on Software and System Process (ICSSP '12)*. IEEE Press, Piscataway, NJ, USA, 190–199. http://dl.acm.org/citation.cfm?id=2664360.2664387

[63] M H N Nasir and S Sahibuddin. 2011. Critical success factors for software projects: A comparative study. *Scientific Research and Essays* 6, 10 (2011), 2174–2186. https://www.scopus.com/inward/record.uri?eid=2-s2.0-79956218741{&}partnerID=40{&}md5=294f5a21a02adf8352a4728178c5f3eb

[64] Nianping Zhu, J Grundy, and J Hosking. 2004. Pounamu: A Meta-tool for Multi-View Visual Language Environment Construction. *2004 IEEE Symposium on Visual Languages-Human Centric Computing* (2004), 254–256. https://doi.org/10.1109/vlhcc.2004.41

[65] Don Norman. 2013. *The design of everyday things: Revised and expanded edition*. Basic books.

[66] Oracle. [n. d.]. proactive support tools diagnostics. http://www.oracle.com/us/support/library/http://www.oracle.com/us/support/library/proactive-support-tools-diagnostics-069181.pdf-069181.pdf

[67] OWASP. 2018. OWASP Secure Software Development Lifecycle Project (S-SDLC). https://www.owasp.org/index.php/OWASP{_}Secure{_}Software{_}Development{_}Lifecycle{_}Project

[68] Avi Parush. 2015. Conceptual Design for Interactive Systems Designing for Performance and User Experience. Morgan Kaufmann, Boston, 164. https://doi.org/10.1016/B978-0-12-419969-9.09992-7

[69] Jaime A. Pavlich-Mariscal, Steven A. Demurjian, and Laurent D. Michel. 2010. A framework of composable access control features: Preserving separation of access control concerns from models to code. *Computers and Security* 29, 3 (2010), 350–379. https://doi.org/10.1016/j.cose.2009.11.005

[70] Charith Perera, Arkady Zaslavsky, Peter Christen, and Dimitrios Georgakopoulos. 2014. Context aware computing for the internet of things: A survey. *IEEE communications surveys & tutorials* 16, 1 (2014), 414–454.

[71] Philips. [n. d.]. Philips e-Alert. https://www.philips.co.uk/healthcare/resources/feature-detail/e-alert-faq

[72] Arjun Rao, Ayush Bihani, and Mydhili Nair. 2018. Milo: A visual programming environment for data science education. *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC* 2018-Octob (2018), 211–215. https://doi.org/10.1109/VLHCC.2018.8506504

[73] Indrakshi Ray, Robert France, Na Li, and Geri Georg. 2004. An aspect-based approach to modeling access control concerns. *Information and Software Technology* 46, 9 (2004), 575–587. https://doi.org/10.1016/j.infsof.2003.10.007

[74] Martin Reddy. 2011. Chapter 4 - Design. In *API Design for C++*, Martin Reddy (Ed.). Morgan Kaufmann, Boston, 105–150. https://doi.org/10.1016/B978-0-12-385003-4.00004-X

[75] Fumiko Satoh, Yuichi Nakamura, and Koichi Ono. 2006. Adding authentication to model driven security. In *Proceedings - ICWS 2006: 2006 IEEE International Conference on Web Services*. 585–592. https://doi.org/10.1109/ICWS.2006.25

[76] Ken Schwaber. 2004. *Agile project management with Scrum*. Microsoft press.

[77] Laurens Sion, Dimitri Van Landuyt, Koen Yskout, and Wouter Joosen. 2018. SPARTA: Security & Privacy Architecture Through Risk-Driven Threat Assessment. In *Proceedings - 2018 IEEE 15th International Conference on Software Architecture Companion, ICSA-C 2018*. 89–92. https://doi.org/10.1109/ICSA-C.2018.00032

[78] Laurens Sion, Kim Wuyts, Koen Yskout, Dimitri Van Landuyt, and Wouter Joosen. 2018. Interaction-Based Privacy Threat Elicitation. In *Proceedings - 3rd IEEE European Symposium on Security and Privacy Workshops, EURO S and PW 2018*. 79–86. https://doi.org/10.1109/EuroSPW.2018.00017

[79] Karsten Sohr, Gail-Joon Ahn, Martin Gogolla, and Lars Migge. 2005. Specification and validation of authorisation constraints using UML and OCL. *European Symposium on Research in Computer Security* 3679 LNCS (2005), 64–79. https://doi.org/10.1007/11555827_5

[80] Emilio Soler, Juan Trujillo, Eduardo Fernandez-Medina, and Mario Piattini. 2007. Application of QVT for the Development of Secure Data Warehouses: A case study. In *The Second International Conference on Availability, Reliability and Security (ARES'07)*. IEEE, 829–836.

[81] P Suresh, J Vijay Daniel, V Parthasarathy, and R H Aswathy. 2014. A state of the art review on the Internet of Things (IoT) history, technology and fields of deployment. In *2014 International Conference on Science Engineering and Management Research (ICSEMR)*. IEEE, 1–8.

[82] Junichi Suzuki and Yoshikazu Yamamoto. 1999. Toward the interoperable software design models: Quartet of UML, XML, DOM and CORBA. *Proceedings - 4th IEEE International Symposium and Forum on Software Engineering Standards, ISESS 1999* (1999), 163–172. https://doi.org/10.1109/SESS.1999.766591

[83] Maurice H. ter Beek and Axel Legay. 2019. Quantitative Variability Modeling and Analysis. *Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems - VAMOS '19* (2019), 1–2. https://doi.org/10.1145/3302333.3302349

[84] Juha-pekka Tolvanen and Steven Kelly. 2009. MetaEdit+: Defining and Using Integrated Domain-Specific Modeling Languages. *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications* (2009), 819–820.

[85] Inger Anne Tondel, Martin Gilje Jaatun, and Per Hakon Meland. 2008. Security requirements for the rest of us: A survey. *IEEE software* 25, 1 (2008), 20–27.

[86] Juan Trujillo, Emilio Soler, Eduardo Fernández-Medina, and Mario Piattini. 2009. An engineering process for developing Secure Data Warehouses. *Information and Software Technology* 51, 6 (2009), 1033–1051. https://doi.org/10.1016/j.infsof.2008.12.003

[87] Joseph S Valacich and Joey F George. 2017. *Modern Systems Analysis and Design*. Pearson Education, Inc.

[88] A Van Den Berghe, K Yskout, W Joosen, and R Scandariato. 2017. A model for provably secure software design. In *Proceedings - 2017 IEEE/ACM 5th International FME Workshop on Formal Methods in Software Engineering, FormaliSE 2017*. Institute of Electrical and Electronics Engineers Inc., 3–9. https://doi.org/10.1109/FormaliSE.2017.6

[89] Rob van der Meulen and Gartner. 2017. Gartner Says 8.4 Billion Connected 'Things' Will Be in Use in 2017, Up 31 Percent From 2016. https://www.gartner.com/en/newsroom/press-releases/2017-02-07-gartner-says-8-billion-connected-things-will-be-in-use-in-2017-up-31-percent-from-2016

[90] Alexander vanÂădenÂăBerghe, Riccardo Scandariato, Koen Yskout, and Wouter Joosen. 2017. Design notations for secure software: a systematic literature review. *Software and Systems Modeling* 16, 3 (2017), 809–831. https://doi.org/10.1007/s10270-015-0486-9

[91] Belén Vela, Carlos Blanco, Eduardo Fernández-Medina, and Esperanza Marcos. 2012. A practical application of our MDD approach for modeling secure XML data warehouses. *Decision Support Systems* 52, 4 (2012), 899–925.

https://doi.org/10.1016/j.dss.2011.11.008
[92] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. Citeseer, 38.
[93] Kim Wuyts. 2014. LINDDUN : a privacy threat analysis framework. (2014).
[94] Kim Wuyts. 2015. *Privacy Threats in Software Architectures.* Number January. 192 pages. [freely{%}0Aavailable]
[95] D. Xu and K.E. Nygard. 2006. Threat-driven modeling and verification of secure software using aspect-oriented Petri nets. *IEEE Transactions on Software Engineering* 32, 4 (2006), 265–278. https://doi.org/10.1109/TSE.2006.40
[96] Lijun Yu, Robert France, Indrakshi Ray, and Sudipto Ghosh. 2009. A rigorous approach to uncovering security policy violations in UML designs. In *Proceedings of the IEEE International Conference on Engineering of Complex Computer Systems, ICECCS*. 126–135. https://doi.org/10.1109/ICECCS.2009.16
[97] Kang Zhang, Da Qian Zhang, and Jiannong Cao. 2001. Design, construction, and application of a generic visual language generation environment. *IEEE Transactions on Software Engineering* 27, 4 (2001), 289–307. https://doi.org/10.1109/32.917521