**UNIVERSITY OF PIRAEUS**
**SCHOOL OF INFORMATION & COMMUNICATION TECHNOLOGIES**
**DEPARTMENT OF DIGITAL SYSTEMS**
**SYSTEMS SECURITY**

# Master Thesis

## REQUIREMENTS CAPTURING AND SOFTWARE DEVELOPMENT METHODOLOGIES FOR TRUSTWORTHY SYSTEMS

**Supervisor:**
**Prof. Sokratis K. Katsikas**
**University Of Piraeus**

**Ioulia Tsitsa**

**Piraeus, November 2014**

# ABSTRACT

In this Master's thesis, the concept of information systems trustworthiness will be covered, in terms of describing existing methodologies for collecting and documenting security requirements as well as describing how existing methodologies support the delivery of trustworthy systems. Moreover, this essay will employ a case study, in order to enforce the essay's outcomes on how to achieve trustworthy software.

Trustworthiness is a characteristic that can be applied to any system that satisfies the desired level of trust by not failing. The systems that should possess such a property are mainly systems that manage sensitive records, critical infrastructure, etc.

The capturing of a system's requirements is the process of discovering and identifying the system's stakeholders and their needs. A system's requirements are the features and qualities that a system should possess, and are extracted from the system's stakeholders (i.e. owners, users). Therefore, the identification of security requirements is of crucial importance for the achievement of the desired security goals, namely trustworthiness.

With respect to security requirements, in order for a system to ensure that its security specifications are satisfied, security concerns must be taken into consideration in every phase of the software engineering lifecycle; namely, from requirements engineering to design, implementation, testing, and deployment.

In order to increase users' trust in the systems they use, software defects must be reduced through. Following a systematic development methodology, during the software development process, the risk of not achieving the acceptable result, is reduced, if not eliminated, since software development methodologies impose a disciplined process upon software development.

# TABLE OF CONTENTS

ABSTRACT ..... 1

TABLE OF CONTENTS ..... 2

1. Introduction ..... 5
   - **1.1. Structure ..... 5**
   - **1.2. Target ..... 5**

2. Software & trustworthiness ..... 6
   - **2.1. Introduction ..... 6**
   - **2.2. Trust and trustworthiness ..... 6**
   - **2.3. Trustworthy software ..... 8**

3. Software Design & Development ..... 10
   - **3.1. Introduction ..... 10**
   - **3.2. Software Processes ..... 10**
   - **3.3. Software Process Models ..... 11**
   - **3.4. Agile Software development methods ..... 14**
   - **3.5. Rational Unified Process ..... 15**
   - **3.6. Software methodologies / practices and trustworthiness ..... 16**

4. Requirements Engineering for Trustworthy Software ..... 19
   - **4.1. Introduction ..... 19**
   - **4.2. Requirements engineering activities ..... 19**
   - **4.3. Classification of requirements ..... 22**
   - **4.4. Security Requirements Analysis ..... 23**
   - **4.5. Requirements Engineering for Trustworthy Software ..... 24**

5. Software Evaluation ..... 25
   - **5.1. Introduction ..... 25**
   - **5.2. Software Evaluation activities ..... 25**
   - **5.3. Testing approaches ..... 27**

6. Analysis of "Legislation Compliance" IT system – A Case Study ..... 28
   - **6.1. Definition of the envisaged system (the WHAT) ..... 28**
   - **6.2. Project Approach (the HOW) ..... 29**
   - **6.3. Identification of stakeholders ..... 30**
   - **6.4. Analysis of requirements ..... 31**
   - **6.5. Verification, Validation and Testing ..... 39**

7. Conclusions ..... 42

**7.1. Introduction ..... 42**
**7.2. Conclusions ..... 42**

8. Bibliography ..... 44

# TABLE OF TABLES

Table 1: Functional Security tests for “Legislation Compliance” system ..... 41

# TABLE OF FIGURES

Figure 1: Users of “Legislation Compliance” System ..... 31

# 1. INTRODUCTION

Trustworthy systems are those systems whose design and implementation are aimed at gaining its users' trust, mainly by satisfying the expected security level. Such systems are vital systems that manage sensitive records, critical infrastructure, etc. Software system development mainly focuses on cost, performance and functional/technical requirements, paying limited attention on how software will perform in terms of system failures; nevertheless, trustworthiness is currently becoming an important requirement.

## 1.1. Structure

The essay consists of seven chapters. The following section will give a brief overview of the essay project outline:

- Chapter one introduces the area of study and problem definition. In addition it describes the research objective, research question and delimitation
- The second chapter covers the notion of software trustworthiness
- The third chapter's objective is the detailed description and comparison of current software process models and methodologies in terms of contributing to trustworthiness
- In chapter four, the processes of software requirements engineering are outlined
- Chapter five's intention is to describe current software evaluation methods
- Chapter six involves a case study, related to the analysis of a software system whose main objective is to support trustworthiness
- In chapter seven, main conclusions of the essay are developed. The seventh chapter is followed by the references used for the development of the essay

## 1.2. Target

This essay aims to provide a clear view of how current software process models and methodologies as well as how requirements capturing methods support the building of trustworthy software systems.

# 2. SOFTWARE & TRUSTWORTHINESS

## 2.1. Introduction

Software trustworthiness is the most important component for a system to establish its trusted usage. The National Institute of Standards and Technology (NIST) [29] defines trustworthiness as software that can and must be trusted to work dependably in some critical function, and failure to do so may have catastrophic results, such as serious injury, loss of life or property, business failure or breach of security. Users should be able to rely on these systems in the terms of systems' availability and security of their personal information. Therefore, trustworthiness refers to available, reliable and secure systems.

## 2.2. Trust and trustworthiness

In order to define trustworthy software, we can draw upon the concept of trust. Trust is a matter of perception, a relationship that involves the actual and the expected behaviour of software and it is always conditional. In order to disambiguate the notion of "trust", it is useful to define the terms "trusted" and "trustworthy":

- A "trusted" system is one which, if compromised, will breach the security policy.
- A "trustworthy" system is one that will not be compromised (i.e. is completely secure).

A system that has access to sensitive information but is not secure can be described as "trusted but not trustworthy"; on the other hand a system that is completely secure, but does not have any sensitive material on it, can be considered "trustworthy but not trusted". The ideal is to have a "trusted and trustworthy" system, notably, a system that can store sensitive data that is totally secure [2].

### *2.2.1 Trust*

Trust is the end users' prior estimation that a system will provide an appropriate outcome [17]. Trust in a system can be considered as a users' property which reflects their estimation that the system will provide the expected outcome. In general, trust describes a state where the outcome is unknown based on users' requirements

### *2.2.2 Trustworthiness*

Trustworthiness is a quality that cannot be claimed without being proved, since it adheres to the probability that a system will meet all of its stakeholders' requirements. Trustworthiness suggests that there are certain assurance criteria with which a user's confidence can be justified for a system. Therefore, trustworthiness is a property highly dependent on a number of attributes, encompassing security, safety and reliability. It is not sufficient to address only one or two of these diverse dimensions, nor is it adequate to just assemble components that are themselves trustworthy. Integrating the components and understanding how the trustworthiness attributes interact is a challenge, as long as complexity of software increases.

### *2.2.3 Trustworthiness attributes*

Trustworthiness involves a set of attributes by which the users verify that a system achieves the desired security level. Trustworthiness' attributes can be considered as system properties which suggest a system's capability to prevent unexpected outcome due to any potential threats. An early standard for secure systems, TCSEC [30] restricted trustworthiness to dealing solely with security. Microsoft and Sun Microsystems [28] focused on developing high-quality code that is available, reliable and secure, in order achieve trustworthiness. Hasselbring and Reussner [7] attempted to provide a holistic view of trustworthiness with respect to software building. This means the methods for building trustworthy systems must satisfy multiple properties. They identified that certain attributes that trustworthiness consist of are the following:

- correctness: the absence of the improper system states
- safety: the absence of catastrophic environmental consequences
- quality of service: QoS includes the attributes of availability, reliability, performance
- security: the absence of unauthorized access to a system
- privacy: the absence of unauthorized disclosure of information

## 2.3. Trustworthy software

Trustworthiness of software means its worthy of being trusted to fulfil requirements which may be needed for a particular software component, application, system, or network. It involves attributes of stability, data security, quality, privacy, safety, etc. By virtue of the increased complexity and scope of software, fault tolerance is at the heart of the building trustworthy software. Trustworthy software is stable and fault-tolerant software that it does not crash at minor flaws Finally, the National Institute of Standards and Technology [29] defines trustworthiness as "software that can and must be trusted to work dependably in some critical function, and failure to do so may have catastrophic results, such as serious injury, loss of life or property, business failure or breach of security". In order to achieve trustworthiness a lot of effort is required at all phases of a system's development lifecycle, mainly at increasing fault tolerance [13]. Notably, at design time, reliability of a system can be increased through fault avoidance techniques, whilst at implementation time, fault removal techniques can be utilised for the same purposes.

### *2.3.1 Fault avoidance*

Fault avoidance methods prevent the existence of faults in the final system by avoiding introducing them in the first place (i.e. preventing errors). Fault avoidance may include effective software architecture, formal specification, as well as utilisation of appropriate design methodologies.

### *2.3.2 Fault Removal*

Fault removal methods aim to detect the presence of faults, and subsequently locate and remove them, after the completion of development stage. Common fault removal comprises of verification (i.e. analysis and testing), fault identification, debugging and corrections.

### *2.3.3 Fault Tolerance*

Since some faults cannot be removed from design, unavoidably, errors will exist in software and they are very likely to take place in general, despite the efforts put in order to avoid or remove them. Therefore, even if a system seems to be fault-free, it must also be fault tolerant as there may be specification errors or the validation/verification may be

incorrect, and thus, through the adoption of fault-tolerant architectures problems are detected and fault recovery can be achieved.

Hardware and software system architectures, are complementary fault-tolerant architectures that support hardware and software redundancy.

- Adoption of techniques that lead to fault tolerant hardware is a common approach to achieve high availability in stand-alone systems. With fault tolerant hardware, redundant hardware is built into the hardware platform, and active hardware is constantly monitored for failures.
- Fault-tolerant software monitors how 'healthy' the individual software elements are, transferring the affected functions to a different or new process upon the detection of a problem. Moreover, fault-tolerant software techniques can be used to execute different versions of the software (e.g. backup processes), providing the ability to upgrade the system's without interrupting normal operation.

#### 2.3.3.1 Fault Detection

Fault detection involves detecting an incorrect system state and throwing an exception to manage the detected fault. Fault detection mechanism can be initiated before the state change is committed (Preventative fault detection) or after the system state has been changed (Retrospective fault detection).

#### 2.3.3.2 Fault Recovery

Fault recovery involves the correction of damage state and restoration of the system to a safe state. In order to achieve fault recovery, backward or forward error recovery techniques can be used. Backward recovery is used to restore the system state to a safe state using recovery blocks, whilst forward recovery is used to correct damage state.

#### 2.3.3.3 Fault Repair

An erroneous system should be modified in order to prevent recurrence of the fault. Identification of the fault location as well as system repair, are the two phases that are involved. To identify the fault location, either diagnostic checking is used to locate faulty component or input is provided so as to check the outputs. System repair can be done by reconfiguration in order to keep the system operational.

# 3. SOFTWARE DESIGN & DEVELOPMENT

## 3.1. Introduction

Software development is the process of planning, creating, testing, and deploying an information system. This section will focus on the major software engineering models and software development methodologies, and specifically on how the contribute to achieving trustworthiness of the final product. A software process model is an abstract representation of a process methodology, intended to specify the various stages of the process and the order in which they should be carried out.

Security is a very important component for the successful development of trustworthy software; however, it is not the only one. Therefore, security should not be taken into consideration as an isolated quality that a system must possess in order for it to be considered trustworthy. In this chapter, a description of existing software development models will be cited, along with arguing on how these models serve in the development of trustworthy systems.

## 3.2. Software Processes

Software development methodologies can be categorised into two main categories, Plan-Driven Development and Agile Development [11]. Both Plan-driven and Agile methodologies include processes, procedures and documentation. Therefore, in order to successfully complete a software development project, the followed methodology provides a different approach in utilising the aforementioned components.

“Plan-Driven” development constitutes a better approach to be followed the requirements can be stated before beginning the design, and thus the complete design should be stated before starting coding. “Agile” development is considered better approach when the properties of the final product cannot be determined in detail in advance. In the first case, it is assumed that it is possible to plan and design the whole project to be developed from the start; on the other hand, in the latter case, the investigation of the project’s requirements is required for every separate component of the product to be developed. Plan-driven development may be broken up into waterfall development and plan-driven incremental development.

Many development models have been developed in order to achieve different objectives. Those models define the appropriate methodologies for the complete software development lifecycle depending on the specific software system's goals. A software process methodology is a specific way of conducting a software project (e.g. RUP etc.). They define exactly what, when, and/or how various artefacts should be produced. They may not be entirely explicit but they define the actions that each member of a project team must deliver.

## 3.3. Software Process Models

Choosing right model for developing of the software product or application is very important. Based on the model the development and testing processes are carried out. There are various Software development models. They are as follows:

- Waterfall
- Incremental
- Rapid Prototyping
- Spiral
- Reuse Oriented
- Common Criteria
- OWASP CLASP
- etc.

In software engineering, a software development methodology is a framework that is used with the intent of better planning and management of the process of developing an information system, by dividing the whole software development process into distinct phases [8].

### *3.3.1 Waterfall*

Waterfall model describes a linear development method that is often considered the classic approach to the software development life cycle, firstly formally introduced by W. Royce in [18]. The main idea of the waterfall model is that software development proceeds from one phase to the next in a purely sequential way. It is called "Waterfall" as it can be

represented or graphically modelled as an actual waterfall, where project development is perceived as processes that are flowing steadily downwards through the phases of establishing requirements, to design creation, to program implementation, to system test, to release to customer. Therefore, the waterfall model suggests that a programming team should move to a next phase only when the previous phase is completed and everything is approved. The Waterfall model includes the following phases: Analysis of requirements, System design, Implementation, Testing, Deployment and Maintenance. The most significant weak point of the waterfall software development model is the possibility that a poor or wrong design choice will not be discovered until the final stages of implementation or testing.

### *3.3.2 Rapid Prototyping Model*

Rapid prototyping [19] produces a program that performs some essential or perhaps typical set of functions for the final product. A prototype approach is often used when the goal is to test the implementation method, language, or end-user acceptability. The prototype is the basis of the final product development. From that point Rapid prototyping is very similar to the waterfall model. The major difference between them is the creation of the operational prototype or functional subset very quickly.

### *3.3.3 Spiral Model*

The spiral model [5], is an enhancement of the waterfall/rapid prototype model, with risk analysis preceding each of the phases. This model is considered appropriate for the internal development of large systems and is especially useful when software reuse is a goal and when specific quality objectives can be incorporated.

### *3.3.4 Incremental Development*

The incremental model recognises that software development steps are not discrete. Instead, initial implementations are presented to the user at regular intervals until the software product completes the stakeholders' expectations [20]. The requirements of the solution to be developed are not completely known prior to development, and thus they are evolving during the software is being developed. One of the biggest advantages of the incremental model is that it is flexible enough to respond to critical specification changes

as development progresses. Another clear advantage is that analysts and developers can encounter complexity broken down in smaller pieces. The downside risk is, that time and budget may be exceeded.

### *3.3.5 Reuse Oriented Development*

Nowadays, in most systems software reuse can happen either form internal or external resources. Existing code can be used untouched, modified as required, or just complemented with graphical user interface. Reuse-oriented approaches [20] rely on a large base of reusable software components and an integrating framework for the composition of these components. Reuse-oriented software engineering has the advantage of reducing the amount of software to be developed and subsequently reducing cost and risks, however, requirements compromises are inevitable leading potentially to a system that does not meet the real needs of users.

### *3.3.6 Common Criteria (ISO 15408)*

The Common Criteria (CC) [26] is an approach to evaluate security properties of software systems. A "Target of Evaluation" is tested against the "Security Targets" that are composed of given functional security requirements and security assurance requirements and are selected based on a protection requirement evaluation. Furthermore, the evaluation can be performed at different strengths called "Evaluation Assurance Level". On the downside, there are some disadvantages: the development model does not easily allow adjustment to specific environments. Moreover, it is rather difficult to express the overall security of a system with metrics related to Common Criteria.

### *3.3.7 OWASP Clasp*

OWASP Clasp [25] is a project that targets to the enhancement of software security produced by the Open Web Application Security Project. OWASP is a worldwide not-for-profit charitable organisation focused on improving the security of software. Clasp tries to move security concerns early into the development process. It is an activity driven, role-based set of process components whose core contains formalized best practices for building security into your existing or new-start software development life cycles in a

structured, repeatable, and measurable way. Clasp is intended to be a complete solution that organisations can read and subsequently implement iteratively.

## 3.4. Agile Software development methods

In 2001 the Manifesto for Agile Software Development [27] was planned by a group of individuals. Agile is a methodology for software development that focuses of the delivering of software quickly, incrementally and with great stakeholder involvement. It places a strong emphasis on team engagement in planning, goal setting, and joint execution. Agile defines core values in the form of the "Agile" manifesto, time-boxed iterations, and continuous response to change, but it does not dictate how long the iterations should be.

The primary focus of Agile is to enable the development of quality software that provides a functioning feature within a specific period of time, the so-called "iteration". After each iteration a product developed is ready to be delivered. Each iteration must be fully developed, tested, and complete with documentation. Moreover, in 'Agile Model' after every sprint a demo feature is presented to the customer. Therefore, the customer can observe the implemented features and decide whether they are fulfilling their scope.

One of the biggest problems with software development is changing requirements. Agile processes accept the reality of change versus the hunt for complete, rigid specifications. There are domains where requirements can't change, but most projects have changing requirements. For most projects readily accepting changes can actually cost less than ensuring requirements will never change. We can learn so much more about the project requirements in the context of a working system. Agile processes like Extreme Programming accept that requirements will change and create opportunities for improvement and competitive advantage.

### *3.4.1 Extreme Programming*

Extreme Programming [3] is a highly disciplined method, which aims at improving software projects in five essential ways; communication, simplicity, feedback, respect, and courage. It involves considerable emphasis on disciplined planning by scheduling of relatively frequent small releases, extensive iteration planning, designing and redesigning throughout with continual iteration, coding and recoding as needed and testing repeatedly

throughout (unit tests before release). Extreme Programming seeks to have something running at the end of each defined period (e.g., each week) by deferring less important concepts until later.

### *3.4.2 Defensive Programming*

Defensive programming is an approach to fault tolerance that relies on the inclusion of redundant checks in a program. Programmers assume that there may be faults in the code of the system and incorporate redundant code to check the state after modifications to ensure that it is consistent. Defensive programming cannot cope with faults that involve interactions between the hardware and the software. Misunderstandings of the requirements may mean that checks and the associated code are incorrect. Even though software has a reliable design, when the software is developed and used in the field, its reliability may be unsatisfactory. The reason for this low reliability may be that the software was poorly developed. So, even though the software has a reliable design, it is effectively unreliable when fielded which is actually the result of a substandard manufacturing process. Just like a chain is only as strong as its weakest link, a highly reliable product is only as good as the inherent reliability of the product and the quality of the manufacturing process. Reliability is desirable in software construction regardless of the approach.

## 3.5. Rational Unified Process

The Rational Unified Process (RUP) [23], [9], is a process model that provides a disciplined approach to assigning tasks and responsibilities within a development organization. Its goal is to ensure the production of qualitative software that meets the needs of its end-users, within a predictable schedule and budget. The RUP is a phased model that identifies four discrete phases in the software process, which are closely related to business rather than technical concerns:

- Inception: Inception phase targets to the establishment of a business case for the system. All external entities (people and systems) that will interact with the system should be identified as well as the interactions between them. This information is subsequently utilised for the assessment of the contribution that the system makes

to the business needs. In case this contribution is minor, after this phase, the project may be cancelled.

- Elaboration: The objective of this phase is to develop an understanding of the problem domain, establish an architectural framework for the system, develop the project plan, and identify key project risks. Upon the completion of this phase a requirements model for the system has been developed, as well as an architectural description, and a development plan for the software.
- Construction: The construction phase includes the system design, programming, and testing. Parts of the system are developed in parallel and integrated during this phase. After the completion of this phase, a working software system and associated documentation that is ready for delivery to users are completed.
- Transition: The final phase of the RUP is concerned with moving the system from the development community to the user community and making it work in a real environment. Upon the completion of this phase, theoretically, a documented software system that is working correctly in its operational environment exists.

Iteration within the RUP is supported in two ways. Each phase may be enacted in an iterative way with the results developed incrementally. The most important innovations in the RUP are the separation of phases and workflows, and the recognition that deploying software in a user's environment is part of the process. Phases are dynamic and have goals. Workflows are static and are technical activities that are not associated with a single phase but may be used throughout the development to achieve the goals of each phase.

## 3.6. Software methodologies / practices and trustworthiness

A number of development practices have been proposed in order to address security during the lifecycle of software development [17], [14]. In this section, the way the major software engineering models and processes contribute to trustworthy software will be presented.

- Plan – driven processes (Waterfall, Rapid Prototyping Model , Spiral Model)
    - Pros:
        - In case the trustworthiness requirements are correctly documented and implemented, formal system variants are well suited to the development process

  - Cons:
    - In case of vague or incorrect security requirements, in terms of trustworthiness, the software solution is considered vulnerable in the first place
    - Limited flexibility is offered in case vulnerabilities are identified in later stages of the development process

- Incremental Development Practices
  - Pros:
    - Evolving requirements that potentially contribute to the trustworthiness of the end product can be incorporated as part of an iterative process
    - Stakeholders are part of the development process and have good understanding of the product to be delivered
  - Cons:
    - Security and consequently trustworthiness cannot be tested and evaluated by the end users, and thus, lose focus on the development
- Reuse Oriented engineering
  - Pros:
    - Utilisation of existing parts that are known/ proved trustworthy
    - The utilisation of a proven trustworthy part may increase users' trust on a system
  - Cons:
    - The composition of trustworthy parts does not certainly imply that the sum of those parts is also trustworthy
    - The utilisation of existing components in different contexts than the one for which they were initially created, may compromise an existing trustworthiness property
    - The assembly of different components increases complexity which may lead to uncertainty
- ISO 15408 Common Criteria
  - Pros:
    - Common Criteria Evaluation are performed by trusted 3rd parties
    - Evaluations that are related to security indicate the level up to which the software solution can be trusted
  - Cons:
    - A Common Criteria certification does not provide evidence for specific properties of small portion of the system

- OWASP Clasp Process
    - Pros:
        - The adoption of a proven set of best practices sets probable the fact that security will be addressed properly, leading to trustworthiness
    - Cons:
        - No evidence exists that software is actually trustworthy
        - Clasp methodology may reflect outdated knowledge, in case it is not further developed

# 4. REQUIREMENTS ENGINEERING FOR TRUSTWORTHY SOFTWARE

## 4.1. Introduction

The requirements are the descriptions of what a system should do, which services it provides and which are the constraints that are related to the software system's operation. As I. Sommerville defines in [20], [10], requirements engineering is t*he process of finding out, analysing, documenting and checking these services and constraints.* Requirements engineering is a set of activities for identifying and communicating the purpose of a system, including the contexts in which this specific system will be used. Hence, requirements engineering can be considered as a link between the users' needs who are affected by a software system, and the capabilities and opportunities that the specific software system provides.

Software requirements are characterized as either functional or non-functional. Functional requirements specify an action that a system must be able to perform, while non-functional requirements specify system properties, such as reliability and safety.

Software systems requirements engineering is the process of discovering the purpose of a software system along with the contexts in which it will be used. This process comprises the identification of the system's stakeholders and their needs, as well as the documenting of those users' needs in a form that they can be analysed, communicated, and subsequently implemented [15]. Requirements engineering is also referred to as requirements gathering, requirements capturing, or requirements specification.

## 4.2. Requirements engineering activities

As R.R. Young states in [21], several requirements-related activities exist that need to be addressed during the life cycle of a software system:

- Identification of the stakeholders: This is an activity that includes anyone who has an interest in the system or in its possessing qualities that meet particular needs.

- Understanding of the users' needs and expectations: This part concerns the elicitation of the several types of requirements.
- Identification of the actual requirements: Business requirements are derived from the business goals of an organisation. This part involves the understanding of the business requirements as well as their recording in simple sentences.
- Clarification of the elicited requirements: This activity aims to ensure that the already elicited requirements describe the real needs of the system's stakeholders. At this stage, the requirements are listed in such a form that can be understood and used by developers of the system.
- Analysis of the recorded requirements: This activity aims at ensuring that the requirements are well defined.
- Setting common understanding: the requirements should be defined in such a way that they mean the same thing to all of the stakeholders. Since each stakeholder group may have a different perspective of the system and the system's requirements, the definition of requirements may require considerable time and effort, so that they mean the same thing to all stakeholders.
- Specification of the requirements: This activity includes the precise detail of each requirement so that it can be included in a specification document.
- Prioritisation of requirements: Since not all requirements are of equal importance to the system's users, this activity provides the ability to address the highest priority first and possibly release a version of a product later that addresses lower-priority needs. By prioritising the requirements, critical requirements may be developed and delivered with top priority, whereas those with average priority/ low priority may be developed and delivered at later stages, sequentially.
- Record deriving requirements: This activity involves the recording of those requirements that come up to surface only during the development process. These requirements fall out because of the design of a system, but do not provide a direct benefit to the end user.
- Allocation of requirements: We allocate requirements to different subsystems and components of the system.

- Tracking requirements: This activity includes the process of tracing where each requirement is satisfied in the system, in order to verify that each requirement is being addressed.
- Requirements management: Requirements may be added, deleted, and modified during all of the phases of system lifecycle. Therefore, the need of updating the requirements is covered in this part.
- Testing and verification of requirements: This part includes the process of checking requirements, designs, code, test plans, and system products to ensure that the stakeholder's requirements are met.
- Validation of requirements: This part involves the process for confirming that the real requirements are implemented in the delivered system.

Having the fact that the context in which requirements engineering is conducted is always different, as well as the systems to which it is applied, there is not a single way to follow in order to perform successful requirements engineering. Therefore, there is a series of questions that should be addressed as a starting point in order to perform the aforementioned activities.

- Which problem needs to be solved - This question aims at setting an appropriate scope for the project by identifying its boundaries.
- Where is the problem - This question is related to the investigation of both the physical and organisational context, which aims to verify whether the right problem is being tackled.
- Whose problem is it - This question aims to the identification of all relevant stakeholders.
- Why does it need solving - This question's objective is the identification of the stakeholders' goals.
- How would a software system help – The purpose of this question is to identify a set of scenarios that capture the stakeholders' expectations related to the way a software solution would cover their business needs.
- When does the problem need solving – The purpose of this question is related to identification of all the relevant development constraints.

- What might prevent solving the problem - Requirements engineering is crucial for risk management related to the feasibility of implementing a solution within the given constraints. Therefore this question is intended to help the analysis of feasibility and the identification of potential risks.

In the process of making sense of the answers, requirements engineers may build models, and test them in various ways. They will help to bridge the gap between the problem and the solution, and manage the evolution that takes place as the problem changes.

## 4.3. Classification of requirements

This section provides the descriptions of the different types of requirements.

### *4.3.1 Business Requirements*

Business requirements are high-level requirements that capture the vision for the to-be-developed system. They provide the basis for creating the functional requirements, since they describe the business activities and workflows that should be supported by the system. Since business requirements act as the basis for the development of the real requirements for the software system, it would be best to document the business rules correctly and early.

### *4.3.2 Functional Requirements*

Functional requirements describe what the system or software must do. A functionality is a useful capability that is provided by one or more components of a system. Functional requirements are also called operational requirements, since they describe the inputs and the respective outputs of a system, as well as the relationships between them. The document used to communicate the requirements to customers, system, and software engineers is referred to as a functional a comprehensive collection of the characteristics of a system and the capabilities it will make available to the users. It provides a detailed analysis of the data the system will be expected to manipulate. It may include a detailed definition of the user interfaces of the system.

### *4.3.3 Non-Functional Requirements*

Non-functional requirement are used to define and describe the specify system properties and constraints of an envisaged software system, to drive the design of the system

architecture and to assess the viability of the proposed solution. Furthermore, non-functional requirements form the ground on which the system architecture and the system design will be based.

## 4.4. Security Requirements Analysis

In TCSEC [30] security policy is defined as "*set of laws, rules, and practices that regulate how an organization manages, protects, and distributes sensitive information*". Security requirements are strictly related to organisational policies, and thus they can be described as the translation of the high-level policies into detailed requirements of a system.

Security requirements are those qualities that a system must possess in order to increase its end-users' trust. In general, security requirements are complementary to functional requirements of a system and can be considered as a non-functional requirements, along with such aspects as performance etc., as described in the above section. In existing methods of requirements capturing, security concerns are addressed in early stages, however, security requirements are shown in details only after the completion of functional requirements.

- Confidentiality: Confidentiality refers to preventing the exposure of information to unauthorised users or systems maintaining also, as necessary, the privacy of the information that is stored into the system.
- Integrity: The system shall not put at risk the integrity of the business data and ensure their integrity.
- Availability: The availability a system relates to the time that the system is expected to be available for the end-users or other interconnected external systems.
- Accountability: Recording of the end-users actions performed
- Authentication: Authentication is the ability of the system to uniquely identify and confirm the identity of a person or a system, as well as to determine if access shall be allowed or not
- Authorisation: Authorisation is the functionality of the system to control the specific actions that the user is authorised to perform inside the system.

## 4.5. Requirements Engineering for Trustworthy Software

A requirement is a statement that identifies a capability, characteristic, or quality factor of a system in order for it to have value and utility for a user. A requirement is well defined and more specific than a need, which is a capability desired by a user or customer to solve a problem or achieve an objective [21]. In order to achieve trustworthiness while building a system, the first step to follow is to properly capture the stakeholder's requirements. A system's requirements are extracted from its stakeholders, including end-users, developers, and owners, and they should be specified before attempting to construct a system.

According to OWASP CLASP [25], the primary step is to identify at a high level the core security model for the application (i.e. the resources that may potentially be at risk, the roles and responsibilities of users that may access those resources and the potential consequences if these resources are compromised). These activities provide a context for making choices about how to deal with particular security issues throughout the development lifecycle as well as they define a framework for accountability that will be useful in case security problems are found in system design.

# 5. SOFTWARE EVALUATION

## 5.1. Introduction

Errors in software are unavoidable and very likely to take place in general. Therefore, in order to ensure software quality and performance, the software development process includes testing, verification, validation [1]. Those methods evaluate the products against system requirements. Verification is the process of ensuring that programs meet their design specifications. Validation is the process of ensuring that a program meets its functional specification. Software testing verifies and validates software.

## 5.2. Software Evaluation activities

### *5.2.1 Validation*

Validation refers to activities that ensure that the right product is built by determining whether it meets customer expectations and fulfils specific user defined intended purposes. Ensuring that the program meets the functional specification. Notably, no functions have been omitted and unintended side effects are not present. It's not possible to test all the combinations the program will encounter. Therefore, you must consider which ones to test, how to test them, and what degree of confidence these tests can produce in the program's validity.

Validation process should be proactive and continuous prior to development and verification. Validation is required whenever a requirements derivation process (that is, a translation of requirements from one domain to another) occurs. Validation ensures the requirements correctly capture the users' and stakeholders' expectations and should be performed whenever a translation of requirements from one domain to another occurs

### *5.2.2 Verification*

Verification is the process of ensuring that programs meet their design specifications. It refers to activities that ensure the product is built correctly by assessing whether it meets its specifications. When dealing with the verification of trustworthy systems, there are quality-related factors that should be verified. Measuring trustworthiness is possible only if there are specific attributes that can be measured.

### *5.2.3 Testing*

Testing and evaluation usually begin after the coding is done. This phase is intended to ensure that the coded application actually does the things it purports to accomplish, and to the users' satisfaction. Testing and evaluating test results are the means employed to validate and verify the final software application. A high level description of the involved types of tests is the following:

- Unit testing: Unit testing focuses on verifying the smallest testable elements of the software, by single programs or modules. It is applied to components represented in the implementation model to verify that control flows and data flows are correct and that they function as expected. Unit tests are carried out during development. Unit test execution is performed automatically by the integration server on each system build. System builds are automatically triggered on each developer's commit to source code repository.
- Functional testing: The goal of these tests is to verify proper data entry, processing, retrieval and appropriate implementation of the business rules. All system functionalities are tested using valid or invalid data, in order to verify that business rules are properly applied and the expected results occur, when valid data are used, and appropriate error or warning messages are displayed, when invalid data are used. Functional tests are based on well-defined Test Cases, mapped to all main and alternative flows documented in the Requirements Specifications documents.
- Security Testing: These tests aim at assessing security aspects of the envisaged solution. Security testing is automated and is based on test scripts for rapid deployment during all iterations. The OWASP (Open Web Application Security Project) testing methodology's objective is to identify the vulnerabilities of the system, describe them, classify them according to different criteria (e.g. severity), report, and eventually address them. These security testing techniques include Injection testing, Cross site scripting, broken authentication and session management, Web application penetration testing, etc.
- Performance testing: Performance tests are carried out in order to ensure that the system meets the specified performance requirements. Time-sensitive characteristics, such as response and transaction times will be measured with

varying workloads in order to ensure the stability of the system and validate its performance.

- Integration testing: Integration testing ensures that all system parts of a system co-exist, interoperate and exchange data successfully with each other and other systems in cases of integration.
- Regression Testing: Regression tests validate the system's previous functionality, ensuring that improvements and corrections have not resulted in the loss of previous functionality.

## 5.3. Testing approaches

Common practice of plan driven software testing is that testing is performed by an independent group of testers after the functionality is developed, before it is delivered. On the other hand, in extreme programming and the agile software development movement, software testing begins at the same moment the project starts and it is a continuous process until the project finishes. The test scenarios are continuously updated as new failure conditions and corner cases are discovered, and they are integrated with any regression tests that are developed. Unit tests are maintained along with the rest of the software source code and generally integrated into the build process. The ultimate goal of this test process is to achieve continuous deployment where software updates can be published to the public frequently.

Moreover, as stated in [6], a challenging approach is to follow implementation-based verification methods. These, involve automated tools that find security-related defects in the implementations practical systems. One approach is to use model checkers on abstractions that are derived automatically from source code in order to identify states in which potential security vulnerabilities may exist, whilst another approach, dictates to create versions of popular APIs with the purpose of simulating attacks and attempt to expose vulnerabilities.

# 6. ANALYSIS OF "LEGISLATION COMPLIANCE" IT SYSTEM – A CASE STUDY

## 6.1. Definition of the envisaged system (the WHAT)

Legislation Supervisory organisation acts as an intermediary organisation among other organisations whose target is to support the supervisory activities of the aforementioned organisation in terms of compliance to relevant established legislations.

The envisaged IT system, "Legislation Compliance", aims to facilitate the collaboration and collection of data from the involved stakeholders related to voting purposes, their processing and storage. The "Legislation Compliance" system, is a web application which should provide all necessary mechanisms to collect, validate and store data into the envisaged system, making them also available for searching activities.

The "Legislation Compliance" system will support the creation of isolated spaces within it, called Workgroups which will be utilised by groups of stakeholders with common tasks. Those groups are envisaged as restricted groups where only authorised members can exchange confidential information in efficient and secure manner.

One of the main activities of the "Legislation Compliance" system is the collection of its members' replies regarding their compliance on specific guidelines and legislations. Through a voting process, each Workgroup member should state (vote) whether the organisation that s/he represents complies or not to certain legislations, along with providing the relevant evidence for their statement. Dedicated business flows exist for this voting process.

### *6.1.1 Information System Objectives*

The main objectives of the envisaged IT system are the following:

- Facilitate the collection of data from the external stakeholders
- Perform data quality checks (e.g. data cleansing, integrity, enriching)
- Support the business processes
- Facilitate the storage of data

- Facilitate the availability and the restricted access of data

The objectives of the envisaged IT system will be transposed in functional/non-functional requirements.

#### 6.1.1.1 Assumptions and Dependencies

Technical assumptions:

- Complying with the client's policies (i.e. IT Strategy, Enterprise Strategy, etc.)
- The envisaged IT system will be hosted at client's data centre.

Business processes assumptions:

- The following business users are involved in the processes:
  - Internal business users
  - External business users
- Data provided by the users will be maintained at the organisation's database, without any modification

## 6.2. Project Approach (the HOW)

The methodology that will be followed towards delivering the information system will be based on an Agile iterative process. This will involve analysis, proposals, drafts and conclusions passing repeatedly between the implementation team members and the client's project members for discussion. More specifically, a number of practices and methodologies will be followed in order to ensure that the project will be able to accommodate changes in requirements, even at a late stage of the implementation. Such practices and methodologies include:

- Active stakeholder participation and tight integration of the implementation team with the business stakeholders: To this end, the implementation team will work in tight collaboration with the project owner and the various stakeholders. During the implementation phase, this tight collaboration, will ensure the optimum communication and clarification on the domain model.
- Inclusive modelling techniques: These techniques aim at enhancing the communication of the implementation team members with the stakeholders. This is

achieved through the use of user-centred design, based on simple tools that stakeholders can easily learn and utilise, thus becoming active participants in the modelling procedure and allowing timely decisions to be taken.

- Prioritised requirements: In order to address the issue of changing requirements, a procedure for prioritising them will be established. Prioritisation will take into account criteria such as clarity of requirements (less clear requirements get a lower priority), level of modelling (requirements modelled in greater detail get a higher priority), stakeholder needs, etc. During each iteration period, the highest priority requirements will be implemented.

This implementation approach is intended to ensure timely deliveries, early visibility of the implementation work and minimisation of the risks of deviations from the stakeholders' expectations.

## 6.3. Identification of stakeholders

The "Legislation Compliance" system will be accessible by both internal and external Users. The following user roles have been identified

- Supervisor (Administrator): This User performs administrative activities and tasks including the assignment of permission to the rest of the users.
- External member (External User): This user may perform actions on the system, according to the permissions that s/he is given by the Supervisor (i.e. Create, Red, Update, and Delete)
- Internal member (Internal User): This user is member of the  who is authorised to access information mainly for organisational purposes

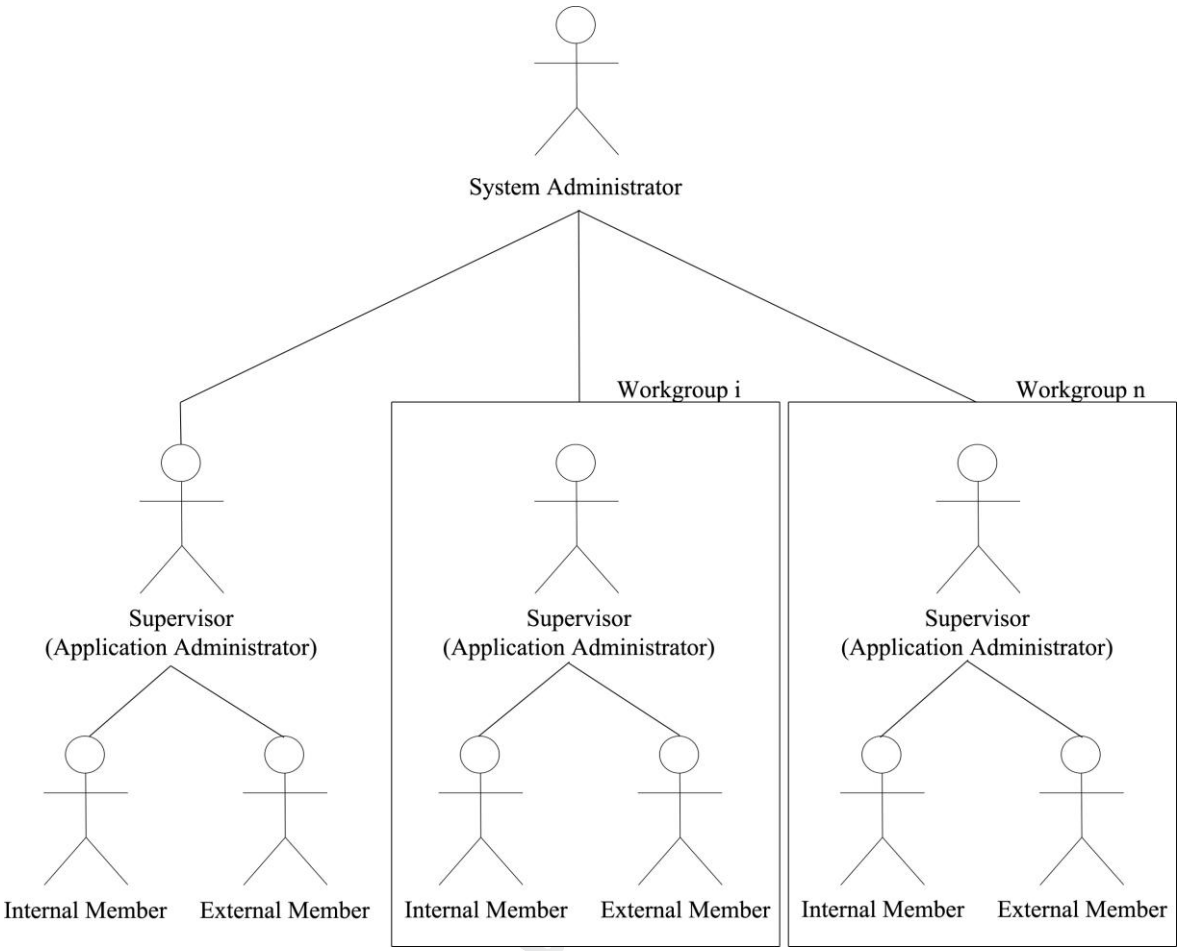


**Figure 1: Users of "Legislation Compliance" System**

## 6.4. Analysis of requirements

### *6.4.1 Business Requirements*

a. Authentication: Since the exchanged information is classified as strictly confidential, a strong authentication system is a must. Therefore, two-factor authentication shall be utilised including a unique combination of user credentials (need to know), as well as an authentication code (need to have).

b. User Management: There will be two levels of user management

   a. In case of a need for setting up a new Workgroup within the system, the System Administrator should create the respective new space and subsequently assign administration access rights to the respective users

   b. The members of the system should be managed by the appointed responsible system users.

c. File Management: Within the access rights agreed among system members, end users should be able to perform CRUD operations (i.e. create/ read/ update/ delete) on files.

d. Voting tool: Collection of members replies regarding their compliance on specific guidelines/ legislations.

e. Auditing: Audit logs recording user activities should be produced and kept for an agreed period in order to assist in future investigations and access control monitoring.

### *6.4.2 Functional Requirements*

Functional requirements form the basis for the implementation project and they can be further refined and elaborated at a more granular level, during the implementation of the IT system.

a. User Registration: A registration page will be available which will require the provision of the respective Information by the User in order to register her/himself to the Restricted Area

b. User Management (Workgroups)

c. Document Management

d. Votes Management

#### 6.4.2.1 User (internal/ external member)

- Register to system: Users register to system through simple registration form which includes only necessary personal information
- Login to system: Users log in the system utilising a combination of unique credentials and authentication code (see section 6.4.3.1.2 *Authentication*)
- Submit Votes: Users submit votes on respective voting areas
- Submit voting justification: Users are obliged to submit justification relevant to their voting submission in order for the system to accept the initial voting submission
- Import documents: Users are able to import document in order to justify their votes related to the submitted votes

- Export documents: Users are allowed to download search results in several data formats (see section 6.4.3.4.2 *Data Formats*)
- Perform search: Users are allowed to search on content and documents

#### 6.4.2.2 Administrator User (internal user)

- Approve user registration: The application administrator is responsible to approve the creation of a user account after a request through registration form
- Assign permissions to users: The application administrator is responsible to assign permissions to the approved users. The application administrator is also responsible to assign permissions to the respective Workgroups to which an account belongs (see section 6.4.3.1.3 *Authorisation*)
- Create Voting Areas: The application administrator is responsible to create voting areas relates to legislation compliance according to business needs

#### 6.4.2.3 System (user trigger)

- Retrieve audit records: The system retrieves existing audit records from the database according to application administrator's request
- Retrieve search results: The system retrieves results from the database according to users' requests
- Create users: The system saves the accounts of the users that are approved by the application administrator
- Delete users: The system physically deletes accounts that are deleted by the application administrator
- Process user input: According to certain business rules, the system processes users' voting submissions

### *6.4.3 Non Functional Requirements*

#### 6.4.3.1 Security

Due to the nature of the exchanged data, which is classified as strictly confidential, the "Legislation Compliance" system shall comply with the security standards. The "Legislation Compliance" system shall be protected at the physical (e.g. firewalls) and

software level and shall ensure that the information stored will be held and maintained securely. The "Legislation Compliance" system shall ensure that data is protected from unauthorised access, whereas all access shall be logged. Only authenticated users shall be allowed to log-in to the "Legislation Compliance" system, while only authorised users shall be allowed to access the stored information. Moreover, all data actions shall be audited.

Communication interfaces used by the "Legislation Compliance" system must be secured and the transmitted data shall be encrypted, where applicable.

#### *6.4.3.1.1 Confidentiality*

Confidentiality refers to preventing the exposure of information to unauthorised users or systems maintaining also, as necessary, the privacy of the information that is stored into the system. The "Legislation Compliance" system shall maintain the confidentiality and privacy of the sensitive information stored into the system and shall protect the confidentiality of the transmitted data. The system shall utilise methods to ensure confidentiality as encryption for the necessary data, by limiting the places where the data are stored, as well as by restricting access to these places.

#### *6.4.3.1.2 Authentication*

Authentication is the ability of the system to uniquely identify and confirm the identity of a person or a system, as well as to determine if access shall be allowed or not. Two-factor authentication (2FA) should be utilised for the external end-users of "Legislation Compliance" system, whilst, single sign on facility should be utilised for the internal users who access the system via the internal network. Authentication code for 2FA should be received either through token or independent message (i.e. SMS, email).

#### *6.4.3.1.3 Authorisation*

The "Legislation Compliance" system shall provide role-based access control. Access inside the system shall be limited to the functionality defined by the user's role. Access Control Lists (ACL) shall be used in order to define the access rights. ACL specifies which users are granted access, as well as what actions are allowed to perform. Actions requiring authorisation shall be supported for all standard data actions inside the system as the actions to create, update, read or delete a data element. The access rights for specific users or groups inside Workgroups shall be configurable only by authorised users.

#### *6.4.3.1.4 Logs Monitoring/Auditing*

Auditing is the capability of the system to track the changes and activities performed by the users in the system. The "Legislation Compliance" system shall provide auditing functionalities regarding the users' access and activities. All data changes shall be traceable to the user or system that performed the action. The auditing should include:

- What was changed
- When it was changed
- Who changed it

#### *6.4.3.1.5 Database Accessibility*

Database accessibility refers to the ability of the system to serve the data to external systems. Specifically database accessibility will be strengthened with the following actions:

- The default ports of the database server(s) shall be blocked
- Non-standard ports shall be used
- The communications with the database server shall be encrypted

#### *6.4.3.1.6 Secure Transport Layer*

Communication between servers, shall be encrypted using a Secure Transport Protocol (i.e. IPSec). Securing the transport layer will ensure that if a malicious party gains access to the network the information and data exchanged between the servers are not stolen.

#### *6.4.3.1.7 Encrypted Storage*

Passwords will be hashed, utilizing SHA-2 hash algorithm, in order to avoid malicious usage by anyone that has access to the database. It is also envisaged to encrypt the Database. In this manner, if a malicious party manages to remove physical media (such as drives or backup tapes), it will be prevented from restoring or attaching them and browse through the data.

#### *6.4.3.1.8 Physical protection*

The physical security of the hardware (servers, databases, etc.) will be aligned with the standard measures and procedures of the hosting agency/contractor.

#### 6.4.3.2 Reliability

The reliability of the "Legislation Compliance" system refers to the system and the end-user availability, as well as the integrity of the system and the data. The "Legislation Compliance" system shall prevent complete or partial failure of critical functions. The system shall be capable to continuously perform the critical functions and any interruption to the normal operations should be resolved quickly to restore the system back to normal conditions. To ensure the data reliability, the system shall prevent data loss or data corruption in case of failure. Furthermore, the system shall be able to operate normally after an unpredictable input from end users or external systems.

Reliability indicates the ability of the software not to fail. The reliability of the considered application is defined by the following sub-qualities:

- Fault tolerance: It indicates the ability of the software to maintain a specified level of performance in cases of software faults or of infringement of its specified interface.
- Recoverability: It indicates the capability of the software to re-establish its level of performance and recover the data directly affected in case of a failure and on the time and effort needed for it.

##### *6.4.3.2.1 Integrity*

The system shall not put at risk the integrity of the business data and ensure their integrity. The "Legislation Compliance" system shall protect the integrity of collected and stored data, as well as the integrity of the shared information. The system shall ensure the integrity of the system and the data also in cases of a failure in a communication channel with external systems.

##### *6.4.3.2.2 Availability*

The availability of the "Legislation Compliance" system relates to the time that the system is expected to be available for the end-users or other interconnected external systems. The "Legislation Compliance" system shall allow users to provide data with no interruption in the availability of the system and the system should not fail under conditions of maximum load. The "Legislation Compliance" system shall provide safeguards to prevent any downtime from simultaneous updates from its end-users.

The "Legislation Compliance" system shall be available during the business hours 365 days per year for the end-users. Redundancy shall be available for the whole architecture (servers, network, firewalls, etc.). The "Legislation Compliance" system shall provide data backup and recovery capabilities.

#### 6.4.3.3 Performance

The performance requirements address mainly the latency and the utilisation of the system. Latency is the time required by the system to complete a task, while the utilisation refers to the maximum load of the components of the system. Latency should be minimal for the users who are accessing the "Legislation Compliance" system.

##### *6.4.3.3.1 Response Times*

Response time refers to the time that is required from the moment the client requests a web page, until the client receives the response, including any depended requests/responses. For the "Legislation Compliance" system, all the servers operate at maximum of hardware utilisation during off-peak as well as peak hours

##### *6.4.3.3.2 Data Processing*

The end users are expected to provide data into the system. The system shall be able to support daily updates and to process data updates without failures during a day.

#### 6.4.3.4 Flexibility

"Legislation Compliance" system should adapt or respond to different user requirements and to support possible future changes in business processes or data.

##### *6.4.3.4.1 Communication Channels*

Communication channels refer to all available paths necessary for the exchanging of data between the end-users and the "Legislation Compliance" system. The "Legislation Compliance" system shall support multiple technologies for communication channels, such as web services and file imports, communication with different operating systems or applications implemented by different programming languages.

##### *6.4.3.4.2 Data Formats*

The "Legislation Compliance" system shall provide flexible data export/import support. In this context, the system shall be able to support different data formats (e.g. CSV, XML,

etc.) and shall be flexible enough to introduce, if necessary, in the future new formats and data types.

#### *6.4.3.4.3 Reporting*

The "Legislation Compliance" system should support flexible reporting functionality. The system shall ensure that all data are available for reporting or extraction. Furthermore, the data shall be available so as to support and serve any additional functionalities and services. Access to these data shall strictly follow the system's security model.

### 6.4.3.5 Extensibility

Extensibility refers to the ability of the "Legislation Compliance" system to be extended, ensuring that the system is designed taking into consideration that system growth/evolution might be achieved in the future. Extensions refer to new functionalities or to modifications and updates on the initial functionality, as well as to the implementation of additional features and the integration with other external systems. The "Legislation Compliance" system shall be designed based on an extensible architecture in order to ensure that future changes will require minimum implementation effort with minimum disruption on the existing functionality. Extensions should be introduced without major changes to the system architecture and infrastructure, if feasible.

### 6.4.3.6 Maintainability

Maintainability is the ability of the system to allow changes in its services, features or interfaces and also to be easily maintainable from architectural or functional level with minimal disruption of the operations. The "Legislation Compliance" system shall be designed with great maintainability in order to support the current and future needs. It shall be able to correct defects and identify their causes, as well as to repair or replace components of the system without affecting other working components. The "Legislation Compliance" system shall provide also the administrators of the system with the ability to perform common/every day maintenance tasks, such as to add, delete or update a user account or to change the security rights for a user or a role.

### 6.4.3.7 Scalability

Scalability is the ability of the system to perform under an increased or expanding workload and the ability to enlarge its capacities without needing to change the underlying

architecture. Scalability methods usually belong to the following categories: horizontal and vertical scaling.

Horizontal scaling (scale out) is the ability to expand the system by adding more nodes, as e.g. adding a new computer to a distributed system. Vertical scaling (scale up) is the ability to expand the system by adding resources to a single node, such as adding more CPUs or memory to a computer.

The "Legislation Compliance" system should ensure that it allows future. The system shall support the long term increase of resource usage (e.g. new users and transaction volumes), without affecting the overall performance of the system. The system should ensure that additional load requires only additional resources rather than design or architectural modifications. The system should support both vertically and horizontally scaling, in terms of hosting at more powerful servers or to be distributed to more than one server in order to increase the demand or the load.

## 6.5. Verification, Validation and Testing

Through a detailed Test Management Plan the following will be achieved:

- Provide a central artefact to govern the planning and control of the test effort. It will define the general approach that will be followed to test the software and to evaluate the testing results, as well as the plan to be used for directing the testing work
- Provide all the necessary information for stakeholders participating in the testing procedure so as to ensure that the testing activity is subject to proper governance and planning, as well as to deliver the necessary results
- Serve as a plan for testing, subject to approval and validation from the stakeholders

The objectives of the Test Management Plan will be the following:

- Identify the items that should be tested for the concerned project
- Identify and describe the test strategy that will be used to cover the test requirements
- List the major test activities

Two different types of tests are envisaged:

- Factory Acceptance Testing (FAT) hosted at the Contractors environment
- Site Acceptance Testing (SAT) hosted at the client's environment

A short description of the envisaged work and procedures is presented in the following paragraphs.

- Factory Acceptance Testing (FAT): The FAT sessions are executed in the Contractor's Test environment, by the implementation team. They aim to verify that the application meets its specifications, all associated development activities are completed, and all test scripts to be executed during the next testing phase (SAT) have been carried out successfully.
- Site Acceptance Testing (SAT): SAT will be performed in client's test environment. Site Acceptance Testing aims to determine whether the system satisfies its acceptance criteria and is required in order to validate that the system, satisfy the specified requirements and objectives of the project. In particular, the following series of tests shall be carried out:
    - Functional tests
    - Stress Tests
    - Performance tests

### *6.5.1 Functional Testing*

Functional testing concerns the testing of the functionalities of the system. This is a critical aspect of the testing effort, as it ensures that the system meets the functional specifications, and thus, ensures acceptance by the users. For completeness, each requirement shall be associated with a set of test cases and scenarios. Any issues, bugs or failures that are identified during functional testing period, are recorded so as to be addressed and resolved on later stages.

### *6.5.2 Functional Security Testing*

This section aims to verify that actors are restricted to specific functions or use cases and are limited to the data that is available to them. These tests will verify the access control

mechanisms and the defined permission levels throughout the "Legislation Compliance" System.

| Description | Condition | User Action | Expected result |
|---|---|---|---|
| Users who maintain an account to the "Legislation Compliance" system is denied access when trying to access the system | User does not maintain account | User provides wrong combination of credentials and wrong 2FA pass code | Fail |
| Logged in User to the "Legislation Compliance" system is denied access when trying to access a Workgroup | User does not have permissions to access the selected Workgroup | User provides the URL of a selected Workgroup on a browser | Fail |
| User with permissions only on College X, will not retrieve search results for content of College Y | User has permissions to Workgroup X AND User does not have permissions to Workgroup Y | User searches for content of Workgroup Y from Workgroup X | Fail |
| User with permissions on Workgroup X and on Workgroup Y, will retrieve search results for content of both Workgroups | User has permissions to Workgroup X AND User has permissions to Workgroup Y | User searches for content of Workgroup Y from Workgroup X | Pass |

**Table 1: Functional Security tests for "Legislation Compliance" system**

# 7. CONCLUSIONS

## 7.1. Introduction

This essay's objective is to study how existing methodologies, as well as requirements' capturing methods contribute to the development of trustworthy software. To this aim, this essay firstly focused on the clarification of the concept of trustworthiness, and subsequently elaborated on the current practices of software development and requirements engineering in the context of supporting the building of trustworthy systems.

## 7.2. Conclusions

Trustworthy software is stable software, sufficiently fault-tolerant that it does not crash at minor flaws. Trustworthiness is a set of properties, encompassing security, safety and reliability. Several different approaches to software development exist, some of them follow a more structured approach, while others follow a more incremental approach. However, there is no suitable way for software systems design that could to be used in any project. Nevertheless, in order to build a trustworthy software system, one of the most important parts is the thorough analysis of trustworthiness requirements.

Much software engineering focuses on schedule, performance as well as functional requirements and scarcely is trustworthiness considered. Trustworthiness is a set of properties, encompassing security, safety and reliability, etc.; however addressing only some of those dimensions is not sufficient to assure trustworthiness, nor is it adequate to assemble components that are trustworthy on their own.

Existing software development methods are able to ensure security; however, only if they were followed wisely, they could result in systems with assurance that security properties are indeed satisfied. Security is a very important component for the successful development of trustworthy software but not the only one and thus it should not be taken into consideration as an isolated quality that a system must possess in order for it to be considered trustworthy.

Understanding the requirements is often the hardest part of the whole process, since their incorrect identification will induce many issues and may result in improper design leading

to untrustworthiness. Therefore, the thorough identification, analysis and documentation of a system's requirements is of crucial importance, in order to ensure that the delivered system will satisfy its users' needs, as well as meet the required security level.

Future work on this subject could include the study of measurements and metrics in order to assess the trustworthiness of systems during their implementation as well as measures that indicate how a system's design is able to support the perception of a good security level.

# 8. BIBLIOGRAPHY

[1] Allen J. H. et al (2008), 'Software Security Engineering: A Guide for Project Managers (The SEI Series in Software Engineering', Addison-Wesley Professional

[2] Anderson R. (2003), 'Cryptography and Competition Policy - Issues with `Trusted Computing'

[3] Beck K. (1999), 'Extreme Programming Explained: Embrace Change', Addison-Wesley (http://www.extremeprogramming.org)

[4] Bishr M. et al. (2013), 'OPerational Trustworthiness Enabling Technologies - Catalogue of trust and trustworthiness events and mitigation actions'

[5] Boehm B. W. (1988), 'A Spiral Model of Software Development and Enhancement' IEEE Computer

[6] Devanbu P.T., Stubblebine S. (2000), 'Software Engineering for Security: a Roadmap', ICSE

[7] Hasselbring W., Reussner R. (2006), 'Toward Trustworthy Software Systems', IEEE

[8] Jayaswal B.K., Patton P.C. (2011), 'Design for Trustworthy Software: Tools, Techniques and Methodology for Developing Robust Software', Prentice Hall

[9] Kruchten P. (2003), 'The Rational Unified Process: An Introduction', Published by Addison Wesley, 3rd edition

[10] Kotonya G, Sommerville I. (1998), 'Requirements Engineering: Processes and Techniques', Published by The Wiley

[11] Larman C., Basili V. R. (2003), 'Iterative and Incremental Development: A brief History', IEEE Computer Society

[12] Mili A. (1999), 'Combining fault avoidance, fault removal and fault tolerance: an integrated model', 14th IEEE International Conference on Automated Software Engineering

[13] Motet G., Powell D. (1999), 'Fault Avoidance and Fault Removal in Real-Time Systems & Fault-Tolerant Computing', Proceedings of 5th International Euro-Par Conference, France

[14] Mohhammadi N.G. et al. (2012), 'An analysis of Software Development Methodologies on how they support Trustworthiness'

[15] Nuseibeh B., Easterbrook S. (2000), 'Requirements Engineering: A Roadmap' In: Proceedings of the International Conference on Software Engineering, Limerick, Ireland

[16] Neil C. J., Laplante P.A. (2003), Requirements Engineering: The state of the practice, IEEE Computer Society

[17] Paulus S., Mohammadi N. G., Weyer T. (2013), 'Trustworthy Software Development', 14th IFIP – International Federation for Information Processing, Germany

[18] Royce W. W. (1970), 'Managing the Development of Large Software Systems', Proceedings, IEEE WESCON

[19] Schach S. R. (2002), 'Object-Oriented and Classical Software Engineering', McGraw-Hill

[20] Sommerville I. (2011), 'Software engineering', Published by Addison Wesley, 9th edition

[21] Young R.R. (2004), 'The Requirements Engineering Handbook', Artechouse

[22] Zhang H., Kitchenham B., Jeffrey R. (2010), 'Toward trustworthy software process models: an exploratory study on transformable process modeling', Journal of software maintenance and evolution: research and practice

[23] White Paper, 'Rational Unified Process - Best Practices for Software Development Teams', Rev 11/01

[24] Wiegers K.E. (2010), 'More about software requirements'

[25] sOWASP CLASP, v1.2 - https://www.owasp.org/index.php/Category:OWASP_CLASP_Project

[26] ISO/IEC 15408 (2009) – 'Information technology - Security techniques - Evaluation criteria for IT security - Part 1: Introduction and general model', Switzerland

[27] Manifesto of Agile Software Development, (2001) - http://agilemanifesto.org/

[28] Microsoft (2002), “Trustworthy Computing", http://research.microsoft.com/ur/us/twc/default.aspx

[29] NIST (2010), ‘Toward a Preliminary Framework for Assessing the Trustworthiness of Software’

[30] TCSEC (1985) - Department of defence trusted computer system evaluation criteria. Dept. of defence standard, Department of Defence

**END OF DOCUMENT**