Patterns in the Requirements Engineering: A Survey and Analysis Study

PRANAY MAHENDRA
Arizona State University
School of Computing, Informatics, and Decision Systems Engineering
7171 E. Sonoran Arroyo Mall, Mesa, AZ 85212
USA
Pranay.Mahendra@asu.edu

ARBI GHAZARIAN
Arizona State University
School of Computing, Informatics, and Decision Systems Engineering
7171 E. Sonoran Arroyo Mall, Mesa, AZ 85212
USA
Arbi.Ghazarian@asu.edu

Abstract: - Requirements patterns have recently been gaining popularity among research communities to help users in identifying, analysing and structuring requirements of a software system. While still in their early stage of development, organizing these patterns would not only assist practitioners in accessing patterns but also provide developers with a direction to design their patterns accordingly. In this paper, an extensive study of various pattern based methods for the requirements phase is reported. Their features, content, structure, purpose, role in assisting requirements engineering activities, similarities and differences are analyzed as we subsequently classify the constituting requirements patterns based on the kind of artefact they present to the user.

Key-Words: - Requirements Patterns, Requirements Engineering, Classification

1. Introduction

Software engineering has been significantly impacted since the concept of patterns was adopted by researchers and practitioners. When a problem occurs over and over in an environment, its generalized solution along with certain forces that govern the application of this solution is coupled to form a pattern [2]. One can reuse pattern knowledge to solve recurring problems, without ever doing it in the same way twice. In their own way, patterns as tools represent knowledge and experience that underlies many redesign and reengineering efforts of developers that have struggled to achieve greater reuse and flexibility. Since patterns encourage knowledge reuse and avoid reinventing the wheel, they show great potential to build software better, faster and at low cost [6].

Although the concept of patterns existed in the 1980s, it was after mid 1990s that patterns became widely known and adopted. Over the last two decades, a large body of pattern knowledge has been amassed, spanning every phase of the software development life cycle [60] since the Gang of Four's (GOF) design patterns [20] were first proposed for the design and architecture phases. While an overwhelming majority of them focus on the solution space [60], the growth trend for patterns in the problem space specifically the ones that aid requirements engineering has slowly but steadily been on the rise. Organizing them at this stage would benefit both pattern users as well as developers. Business analysts and requirements engineers can find the pattern that solves their problem more easily with the help of guidelines that relate other patterns and apply them sequentially. For pattern writers, a good categorization is the basis of cross reference so that newer patterns can be placed into correlated sets. Also, by organizing the existing body of pattern knowledge, the need to understand how requirements patterns tie into modern development methodologies can be satisfied.

For the above purposes, the contribution of this research is threefold – Firstly, we present an extensive survey of the research in the field of patterns in the problem space with a special focus on requirements patterns, by investigate how and what knowledge do patterns capture to help users overcome inherent requirements engineering problems. We then classify various pattern methodologies found broadly into 3 main categories based on the varying context of their applicability and the kind of artefact they offer. Finally, we draw an analysis of various approaches by comparing and contrasting them based on general software pattern criteria as well as general requirements engineering parameters. The paper is structured as follows. Section 2 provides a brief background and motivation for requirements patterns. In section 3, we present an extensive study of various requirements patterns, classifying them into various classes and subclasses. We discuss our findings in section 4 following which, we conclude the paper in section 5.

2. Requirements Engineering and Patterns

Requirements Engineering is the branch of software engineering where business requirements that a system must satisfy first need to be gathered, refined using various analysis techniques before being verified against stakeholder needs and eventually be transformed into a functional software product [63]. Though described as individual and ordered sub-processes namely elicitation, analysis, negotiation and validation, these activities in practice are interleaved, iterative and span the entire development process depending on the organization, application domain, and people involved [44]. Natural language is the primary medium of requirements communication among project stakeholders. However, for later stages of development, such descriptions are rendered imprecise and not complete enough so as to be transformed into software. Hence requirements are also represented in a variety of other formats with the help of various modelling tools and languages, which provide a more concrete basis for designing and consequently building the required system. In the real world however, engineers face various difficulties in this process.

1) The challenge of building a detailed specification from abstract and often conflicting ideas from various stakeholders [44].
2) Since a large group of stakeholders lack the knowledge about computing but have to understand various artefacts of the system, defining requirements at different levels of understandability is a tedious task [6].
3) Unlike the solution space, the problem space is unconstrained which makes defining a system requirement completely and unambiguously, a rigorous task [6].
4) Often the end users themselves do not know what they need from the system and the scenarios they describe are vague and incomplete, failing to identify all possible events that occur during the same interaction [63].

Also, unlike other phases, effectiveness in handling major social factors like communication, strategies and guidelines plays a bigger role in the initial phase than in later stages [52, 69]. As time and money are the ultimate limiting factors for almost every software project, one solution to these onerous tasks would be to use reference frameworks and models that have been applied successfully to similar situations in the past. Artefact reuse in fact continues to be a prominent subject of research at especially at the requirements elicitation and modelling level [6]. Among all such methods, patterns have been deemed the most prominent as they record and reuse ‘best practices’ from recurring problems [6, 38, 19]. Their potential is further justified by recent workshops [30, 31] and conferences [48] dedicated to the solely for the development of patterns that aid requirements processes.

Since requirements engineering can, from one perspective, be seen as defining the problem which software engineers must solve, it is not clear precisely what a requirements pattern should capture. Therefore, there is no strict definition for a requirements pattern. Through the survey we conducted, it has become apparent that the word ‘pattern’ in software is contextually overloaded. As a consensual and very generic definition,

A requirements pattern is a reusable, experience based framework that aids a requirements engineer write or model better quality requirements in as less time possible.

Using a requirements pattern, a set of quality attributes of a requirements specification as defined and standardized by the IEEE [34] can be addressed. The major goals of any requirements pattern are

a) To guide analysts and product developers to most appropriately apply a set of techniques and methods so as to produce a more thorough analysis and understanding of the problem area.

b) To provide a framework upon which to define and capture requirements before and during development and upon which a proposed software product can be evaluated, designed, built and tested.
c) To be able to trace the design of the system back to the original business and system objectives.

A majority of the patterns we found are documented in a template that has been largely adopted from the one prescribed in either the GOF [20] or the POSA (Pattern Oriented Software Architectures) book [9]. Nonetheless, due to the wide variety of knowledge which engineers have felt the need to capture, distinct templates have emerged to support expression of such requirements knowledge. In an effort to bring uniformity, requirement pattern conferences encourage pattern writers to adopt a canonical template to express their patterns though it is subject to revision. In this paper, we compare all our findings with the latest proposed specification by the RE conference [40] which we refer to it as the *General Pattern Format*. This prescribed template is mentioned below. By and large, most requirement patterns follow this template and contain a subset of the below attributes

Name: A self-explanatory name
Also known as: Alias, which is optional
Author: To manifest plausibility of the pattern
Problem: Objective or intent of the pattern
Context: Condition(s) justifying valid use of the pattern.
Forces: Side-effects, constraints or conflicts that arise when the pattern is applied
Solution: Text describing the situation where the problem is solved within the described context and forces, sometimes accompanied by diagrams
Applicability: Describes how and when the pattern can be applied
Classification: In case the pattern conforms to a category of existing software artefacts.
Known Uses: Contains the sources of knowledge captured by the pattern
Examples: Pattern instantiations to aid better use of the pattern
Related patterns: A list of patterns that relate to the pattern semantically or conceptually.

3. Survey of Requirements Patterns

Solution space patterns are mainly for developers to decide on the system design and code structure [20, 9]. Requirements patterns on the other hand, involve not only the developer but end users, project managers, business stakeholders, architects and requirements analysts. In a recent study of requirements pattern [24], 4 approaches have been mentioned. Our search has led to the discovery of many other pattern catalogues as we expanded our search to a broader level. A summary of surveyed pattern books can be seen in Table 1. We believe that a comprehensive survey of requirements patterns would require considerable effort and time but at the same time not be worth the effort considering the pace at which they continue to grow. Nonetheless, the patterns mustered for this research are a concrete representative set and provides enough insight to develop stable classification.

A general agreement among researchers is that a good classification is based on multiple metrics and has henceforth been the approach for the methods described in [25], [8], [27] and the only other known attempt to consolidate and classify requirements patterns [42]. Understandably such a categorization helps break the target space comprehensively into precise sections and provides flexibility with respect to extending the classification. At the same time, a multi-dimensional categorization may be difficult to comprehend and also complex to navigate if each classification dimension is a unique property of the sample space. While there undoubtedly is merit in a classification scheme that distinguishes between individual elements of representative set, we have decided to classify pattern catalogues instead of dissecting them individually. This is mainly because each pattern book has been developed in such a way that it intrinsically imposes some level of uniformity and predictability in the resulting artefacts. There are many general criteria to classify patterns like application domain or requirements phase at which they can be used. Ours is based on the kind of artefact these patterns present to their user. In the following subsections, we describe each category of patterns, providing relevant examples to justify our grouping. We also present their intended purpose and various similarities among individual pattern collections.

3.1. Processes and Guidelines

With the emergence of several patterns in different development phases and at various levels, Ambler [3] differentiates and formalizes *Process patterns* as "a collection of general techniques, actions and/or tasks for developing software". He describes processes patterns at 3 levels of granularity – Phases, Stages and Tasks. He relates them in a way that one type of pattern may consist of one or more of the other. He also identifies recurring processes in an organization that have negative impact on the

project as *Process Anti-patterns*. A process anti-pattern describes an approach and/or series of actions for developing software that is proven to be ineffective and often detrimental to an organization. Patterns listed under this section are recurring instances of workflows, guidelines and best practices aimed specifically for various RE activities.

One of the earliest references to the word 'pattern' in relation to requirements engineering was made by Whitenack in his RAPPeL framework [69]. RAPPeL is a pattern language which initially consisted of 20 interconnected patterns, but has grown over time to now contain over 100 patterns [18], that provide direction and rationale for guiding analysts, developers and project managers in determining as well as defining requirements of business applications in the object oriented (OO) paradigm. The framework is a mixture of general guidelines as well as low level principles which are fairly technical. For instance, in the *Defining Requirements* pattern, a guide to produce a detailed specification is provided. It advocates the capture structural and behavioural requirements along with constraints from customers, making explicit various relationships between them. To facilitate this process, this pattern suggests an in-depth analysis of the problem domain and also modelling prototypes to expand and clarify requirements. Holding true to its pattern language criterion, this pattern tightly couples other constituting patterns like *Prototyping*, *Sponsor Objectives*, *Requirements Specification* and *Problem Domain Analysis*. On the other hand, an activity with a comparatively narrow scope like what needs to be done to handle a state change of a business object while modelling real world entities is discussed in the *Object Aging* pattern. Inspired by RAPPeL, Rawsthrone [50] documented 12 patterns that outline requirements analysis process adopted for various defence projects, which can be used in conjunction with Whitenack's patterns. It is noteworthy that his *Requirements Analysis*, *Prototyping*, *Requirements Specification* and *Domain Analysis* patterns are significantly similar to Whitenack's *Defining Requirements*, *Prototypes, Requirements Specification* and *Problem Domain Analysis* patterns respectively. They aim to solve the same problem and also internally refer to other lower level, standalone patterns. Since both these pattern sets were some of the early ones, the pattern format is very simplistic with just the name, a problem description, a section defining the context and the solution.

Hagge and Lappe's RE patterns [39] are more contemporary as their format is inspired from the general pattern template with sections describing the pattern's objective, context of applicability, the solution supplemented by figures, a listing of applicable areas of the pattern, the resulting consequences of using the pattern, example implementations and a section that describes the impact of the pattern. They focus on capturing experience in the form of certain procedures applicable throughout the requirements phase that "offer guidance for organizing the specification procedure and for eliciting, specifying and verifying requirements". Each of their 4 patterns in [39] presents requirements engineers and project managers with a solution to a requirement management hurdle in terms of a set of tasks. Consider the *Generate Approval Checklists* [39] pattern which is prescribed to facilitate user driven requirements validation. While customers would appreciate checklists as approval certificates, it may not serve the purpose if a requirement is to be satisfied by multiple components of the system. As a solution to better manage overlapping requirements for various systems, they suggest building a checklist for each product which can be used during requirements verification and validation. A repository of such patterns [26] has been developed to facilitate learning and optimizing RE processes within an organization.

Some methodologies subscribe to a more general definition of a pattern, not conforming to any of the standard pattern templates like [9] or [20]. A study of recorded online communication of geographically distributed teams has led to identification *Clarification patterns* [21] which can be used to counter the problem of misinterpreting requirements which is commonly encountered in project environments where there is minimal or no face to face communication between stakeholders. Eric Knauss defines a clarification pattern as 'a recurring trajectory of communication throughout the lifetime of a requirement' [21]. Each pattern is a line graph between the amount of communication and the degree of clarification about certain aspect of a requirement. A similar assessment of various RE artefacts of a certain organization – requirements documents, procedures to collect and validate these requirements - was conducted to identify patterns and their causes that lead to construction of 'complete' requirements artefacts [15]. Such patterns do not prescribe a process but can be used as indicators to correct existing processes by exposing potential threats of requirements definition early in the development phase.

3.2. Models and Templates

As discussed in section 2 of this paper, for detailed requirements definition, different levels of formality need to be used to represent a single requirement. Patterns belonging to this category are reusable snippets of functionality which aid business analysts in discovering, documenting and elaborating requirements. This class of patterns was found to be highly concentrated in terms of sheer amount as it amasses over two-thirds of the total pattern space surveyed. Each of the independent pattern books however addresses a smaller problem. To cope with this skewness and make the classification more robust, we studied individual pattern books looking for factors that define this pattern class. The various roles of these patterns ranged from

a) Documenting high level functional and/or nonfunctional requirements (NFRs)
b) Identify requirements of a specific application domain
c) Identifying requirements pertaining to specific feature(s) of the system like security or reliability.
d) Formalizing high level requirements into analysis models

To reflect as many of the aforementioned properties and by maintaining the rationale of our overall classification scheme, we divide these pattern sets into elicitation patterns and analysis models keeping the pattern users in mind. With this distinction, a pattern user can make decisions based on the RE activity being carried out.

3.2.1. Elicitation Patterns

The main motive of these patterns is to facilitate easy and faster requirements communication among various stakeholders of a project, especially the nontechnical audience. These patterns typically contain a checklist of items to be documented for a requirement. The general template of these patterns is slightly more enriched that the one described in section 2 with additional sections to capture necessary requirement metadata. In this section, we describe the intended use of these patterns, their structure in relation with the General Pattern Template.

Wahono and Cheng [49] prescribe templates to capture requirements for web based applications. The proposed pattern template has a constraint section which needs to be satisfied by an extensible part. While constraints contain the forces and context of the problem, the extensible parts of the pattern are defined by semiformal models describing behaviour, user interaction, architecture and security. It is noteworthy that these patterns don't capture previous experience. Neither do they account for any guidelines about how each section is to be documented. They just provide a template for documenting a requirement with a small degree of homogeneity. Moreover, the pattern names suggest that each is used to define an entire system or subsystem, rather than a single requirement and is subsequently categorized based on the type of system being built. For instance, the *Online Game pattern* and *Registration Form pattern* are classified as interactive patterns while instances of transactional patterns include *Electronic Shopping* and *Online Banking*. Duran *et al.'s* [1] propose a requirements documentation technique with their fill-in-the-blank templates that capture high level system features as a sequence of interactions between the user and the system. This template tracks requirement metadata like id, name, version, author, source, purpose of the requirement and a priority indicator. They introduce linguistic patterns or L patterns to supplement their template with controlled language sentences, constraining the way a requirement can be worded. As requirements were documented for several information systems overtime, a recurring thread of functionality was identified. These recurring functionalities have been termed R Patterns or Requirements Patterns, making them directly reusable when a demand for similar functionality in a new arises. R patterns for **C**reate, **R**ead, **U**pdate and **D**elete (CRUD) operations and information storage have been identified from several instantiations of their pattern based methodology.

Creel's [14] work can be seen as more aligned towards the contemporary definition requirements patterns as his patterns adopt a format from that of [20] but exclude design specific sections like structure, collaborations, participants and sample code. The catalogue he proposed consists of *Specify, Presentation* and *Prioritize* patterns, named after the intent of constituting elements in the software environment. The *Presentation* pattern [51] for instance can be used to extract what the output layer of an application should display to a user but not how it should be presented. The applicability and motivation sections of these patterns have a limiting effect on their usage as they are described in terms of vague examples. Being relatively primitive, they do not ensure high quality or homogeneous requirements but fulfil the basic requirement of a pattern of capturing experience aiding elicitation.

The most popular requirements patterns among researchers today are the ones developed by Withall [70] as several other patterns draw inspiration from

his work. His collection of 37 patterns has been crafted to identify and document functional and non-functional requirements of a wide range of software systems. Each pattern is available as an electronic document, containing a rich set of guidelines that help analysts extract information from the end user. His pattern setup is one of the major contributors to the standard requirement pattern guidelines [40]. Additionally, Withall supplements his patterns with information that developers can use while modelling and a guide for testers on how to test the given requirement. All this information is the basis to fill out one or more templates available within the pattern, similar to the one proposed in [1]. Some significant adaptations of his work can be seen in Roher and Richardson's *Sustainability Requirements Patterns* [55] to specify aspects of a software system that can be used to impact the surrounding physical environment in a positive way and in patterns described for eliciting requirements of online based examination systems [64], which also draw major structural adaptations from [1],[57] and [20].

The patterns by Renault *et al.* [57] capture knowledge of recurring functionalities affecting qualitative aspects of the system. They are primarily designed to allow trade-off decisions during selection of commercial-off-the-shelf (COTS) products for custom development. The *Failure Alerts* pattern [57] can be used to check the kind of reliability feature a system possesses in notifying failures. As they are designed for interview based elicitation, patterns contain one or more 'forms' to allow a requirement to be phrased as required. Each form is further split into a fixed part and one or more extended parts, the latter of which is optional. The core of each pattern is its fixed part as it contains information that is to be defined by the requirement. An elaborate process [57] and set of tools [12] has been developed using which these patterns can be efficiently utilized to maintain completeness and consistency of the resulting SRS. While their 27 patterns cover quality soft goals like reliability, usability, efficiency and portability, they apply to software systems across application domains. Juristo *et al.* [43] conducted a more controlled study and identified patterns that help elicit functional goals that a Human-Computer Interface (HCI) based system can incorporate to make it more usable. They focus on 8 usability features defined in terms of 15 patterns, though using a minimalistic template. Each of their patterns presents a checklist of questions for project stakeholders as a solution, which in turn helps developers model these usability features. On similar lines, requirements patterns to understand rules that a system must follow, from privacy and trust compliance standpoint have been proposed [29]. Laws and regulations for socio technical information management systems form the rationale for the pattern catalogue [28]. In a recent study by Slavin *et al.* [62], security requirements patterns were surveyed to identify heterogeneity in pattern structure. They propose a standard security requirement pattern format which is flexible enough to capture major components of a security requirement along different levels of abstraction.

Compliance business requirements for the domain of e-commerce and banking systems can be defined using Turetken *et al.*'s control patterns. Control patterns are high-level, domain-specific templates that represent desired properties that apply to process specifications [45]. These patterns however do not help identify scenarios but help structuring high level compliance goals using formalization rules to relate scenarios. For instance, the statement {**Receive_Invoice** *LeadsTo* **Make_Payment**} is an instantiation of the *LeadsTo* pattern which assures that Make_Payment logically follows Receive_Invoice. To address the specification of more complex controls, simple expressions are combined and nested via Boolean operators (such as and, or, and xor). Web based tools that verify and enforce these compliance rules have been developed to automate the process [45].

By narrowing the problem space, patterns can provide a more precise and straightforward solution, also taking into consideration various aspect that relate to the solution. This feature is seen in Withall's patterns and is listed as the 'Consideration for Development' section. Mahfouz *et al.'s* patterns [6] to elicit requirements concerned with how communication should take place between various modules in a service based computing environment use this strategy. Their *Deadline* pattern encapsulates the solution to deal with asynchronous service communication by suggesting that a calling service should wait for an expected resource for a specified time, exceeding which an alternate course of action is to be taken. They list a number of considerations that engineers may also want to ponder upon when this pattern is applied – (a) the number of times the service may want to retry before taking the alternative, (b) whether or not to wait out the deadline if the required resource is obtained early, (c) the relative or absolute value of the delay to be set, (d) cases where deadline may want to be postponed and finally (e) expiration time of the requested resource. Due to their fairly technical nature, the text in the solution is supplemented by UML (Unified Modelling

Language) object diagrams [19] which provide a better understanding of the solution.

3.2.2. Analysis Models

There are some patterns that use models to capture domain and operation specific knowledge. While patterns reviewed so far deal with identifying and documenting high level user goals in an ad hoc form, the dynamic aspects of the system still need to be defined so that the development team understand the process flow of a given system feature. Visual representations of a requirement are more intuitive and also best suited to capture such process flows. UML [19], Problem frames [35], i* (pronounced 'i star') framework [22] are some that provide a strong platform to define both static and dynamic aspects of a system. In this subsection, we discuss patterns that capture domain structures and modelling principles, which are more translatable to software design than ad hoc requirements.

Although templates are useful tools to document requirements, ad-hoc descriptions of NFRs are rendered vague and often incomplete [1, 13]. Unlike functional requirements which can be traced to a certain part of the software system, NFRs are qualities of the system as a whole. To standardize the way NFRs are defined, the NFR framework [13] defines quality requirements as soft goals that need to be 'satisficed'. Suppakul *et al.* use this framework as the basis for their approach to capture, organize and reuse knowledge in model based requirements engineering [65]. To capture security requirements patterns, an NFR model of the soft goal is to be created based on the real world information available. Based on the NFR model, 4 kinds of patterns with specific roles are mined which can be applied sequentially to model the quality goal in future systems. Every pattern contains only 3 sections, unlike the general pattern template, to capture the initial and resultant state in the form of soft goal interdependency graphs [13]. The transformation is rationalized by the supplemented refinement rules which are mined from the NFR model. Major objectives of the system are identified using the *Objective pattern* which also captures applicability information of the pattern in the form of answers to who, what why, when, where, how and how much. Various threats to the system are identified using the *Problem Pattern.* The *Alternatives pattern* is then applied to define multiple solutions for the problem for flexibility. Alternatives generally impact other soft goals positively or negatively, hence possible side effects are captured as well. After trade-off analysis, the most suitable solution is chosen using the *Selection pattern*. Quality attributes for a given software system may be defined differently as there are no standard definitions for them. Hence these patterns can be used to address any such soft goals provided their context is understood. The effectiveness of i* models to elaborate quality requirements has also been exploited in context of submarine navigation systems [46]. These 4 patterns discuss tradeoffs between non-functional requirements like accuracy and noise, accuracy and maintainability using specific scenarios based on i* models specified by the pattern.

Functional reuse by capturing recurring behaviours was proposed by Robertson in the form of UML use case diagrams. According to her method, patterns can be abstracted as required - complex ones like making an online credit card transaction or a low level instruction like a trigger to store data into the system's database. When systems for the same domain or functionality are engineered, these use cases can be reused instead of reinventing the wheel. The examples provided in [54] capture this knowledge using a very simplistic template containing a pattern name, context, solution and related patterns. These patterns define relationships between various actors, activities, their data flow and states. More contemporary patterns take this modelling approach forward for specific application domains like embedded systems [58] and mobile and radio communication systems [4]. Patterns in [58] provide a model to a specific embedded system problem. For example, embedded systems generally contain various actuators and sensors to receive inputs that trigger a processing unit responsible for certain computation. When there are large number of acutators or sensors making requests, the load on the computing component could cause the system to crash. The *Mask pattern* helps model a solution for such situations. The patterns help identify components of the system and also suggest how to use these components in the larger picture of the system. In addition to UML class diagrams to structure various components and sequence/state diagrams to define interactions between these components, each pattern also uses problem frames [35] to explicitly state the context of the problem. The collaborations and participants section supplement the UML diagrams by describing the role of each component in the pattern and how they interact. The constraints section is very similar to the one described in [5] with a comprehensive checklist of considerations and restrictions of the pattern application. For embedded systems, these cover hardware, timing, environmental and safety

conditions. The various considerations listed for implementing an embedded system *fault handler* [58] are that it be hardware implemented in case of performance constraint and unlikelihood of change, it be protected in intensive geo environmental conditions, user interface to indicate errors clearly, safety actions mapped to error conditions and that the hardware and related software components should have a high degree of reliance on each other. It is observed that due to domain specificity, there are a finite number of patterns which share common components. Hence, unlike Robertson's patterns, pattern interrelations are rationalized enforcing users to consider technical dependencies. Additionally, one or more design patterns are also listed to help developers, resulting in an easily transformable and traceable system.

Andrade's patterns [4] aim at elaborating problems in systems of mobile and radio communication domain using Use Case Maps [63], which illustrate a scenario based model relative to optional components involved in the scenario. Use case maps can be thought of as a combination of use cases but are represented nothing like concentional use cases. With them, it is easy to represent how a specific scenario affects the entire system. She brings to light that a recurring problem in mobility management is to ensure privacy and secure communication over insecure wireless channels (see *Chiphering* pattern in [4]). As a common solution and best practice, encryption of data being transferred between subscribers is suggested. However, the application of the encyption algorithm is a joint responsibility of other components in the field, which inturn are enlisted as patterns in her collection. Hence, to describe such a scenario, use case maps provide major benefits. Although visible differences in template sctructure exist between these patterns and requirements pattern for embedded systems [58], a high degree of similar traits overshadow them. Both patterns try to complement design patterns and affect system architecture directly. Both use sequence diagrams to describe behaviour. Pattern interrelations are driving forces for each pattern. Both catalogues are classified into structural and behavioural patterns based on the solution they provide. While it can be argued that both these pattern catalogues can be deemed analysis patterns true Fowler's definition [23] as they help model conceptual structures of their respective domains, they extend the definion by stressing extensively on the software design and architectural implementations.

Though semiformal modelling languages are helpful in representing system behaviour and provide a platform for operationalization, complex and safety critical systems need very precise definitions which are at a much lower level of abstraction. The patterns presented by Zhou *et al.* [11] can be distinguished from other patterns in this section as they focus on describing discrete and continuous dynamics of the system rather than high level functional behaviour. The pattern catalogue is extracted from Simulink stateflow [61] modules created to model systems designed by Honeywell [32] which include components for aircrafts, medical devices, automobiles and nuclear power systems. These patterns are structured very differently than conventional software patterns and capture model specific information. A requirements description section lists a set of inputs, outputs, constraints, parameters and the defined functionality of a component of the system while logical assertions are used to define the relationship between these components. Other patterns that facilitate analysis of requirements with the help of strict logical notations are [53] and [37].

3.3. Language Refinements

As mentioned earlier, natural language descriptions of technical software requirements are imprecise. This imprecision of natural language requirements specifications can be attributed to 3 main causes – the inherent ambiguity, incompleteness and inaccuracy of natural language [17]. The way requirements are specified in modern development methodologies, especially the ones in line with the agile movement which stresses on focussing on writing software instead of supporting documentation, are ambiguous and do not capture all the nuances of the systems functionality [47]. Methods to address such ambiguities in requirements specifications have been surveyed and classified into post documented and prewritten refinements [17]. These methods are applied by requirements engineers to an SRS based on its state of completion with the aim of improving requirements inspection and preventing language imprecision respectively. In [66], the universe of all such methods has been classified into 3 groups

- Approaches that define linguistic rules and keywords
- Methods that define guidelines for vocabulary and sentence formation
- Language patterns used in requirements specifications.

In this section, we discuss language refinements or *Language patterns* which the most granular classes of patterns are in RE Because of the scope of the

patterns, the templates mentioned earlier in this paper are not applicable here.

Denger's patterns [16] enforce the usage of certain sentence parts that help writers frame a requirement based on the action being performed by the system. This ensures that requirements in a specification are streamlined. These sentence patterns have been extracted specifically for the domain of embedded systems. All 38 patterns are consolidated into 9 classes based on the type of requirement being documented. The categories are Functional Requirements, Events, Reactions, Computations, Conditions, Relationships, Exceptions, Non-functional Requirements and Special aspects patterns. Below are few variations of TCP i.e. time condition pattern as described in [17].

TCP1: for *time* (variable of type Time)
TCP2: for at least *time* (variable of type Time)
TCP3: {for <not> more than | for at most} *time* (variable of type Time)
TCP4: TCP2 {but | and} TCP3

Similarly, Tjong *et al.* [66] identified more language patterns and suggests combining them with a controlled language for defining requirements as described in [1]. Their patterns are very similar Denger's but can be applied to systems across domains. They are discussed in detail in the main author's doctorate dissertation [67]. Language refinement patterns for specifying requirements similar to Denger's method have been proposed by others. ProjectIT-RSL [10] is one such pattern language that provides a definitive structure to requirements statements based on the kind of requirement being documented. The patterns identified in the ProjectIT-RSL have been abstracted and generalized to form a metamodel which helps practitioners apply these patterns when applicable.

4. Discussion

The presented requirement pattern catalogues provide ways to identify and define different kinds of software requirements at varying levels of formality. In this section, we discuss the benefits and tradeoffs of our classification scheme. According to Buschmann *et al.* [9], a good categorization of patterns has the below properties.

I. Fairly simple and easy to learn
II. Few classification criteria
III. Reflects pattern properties
IV. Provides a roadmap to pattern selection
V. Flexibility to accommodate new patterns
VI. Shows relationship between patterns

The only attempt so far to organize the entire corpus of requirements patterns is presented in [42]. The classification is based on 4 general facets i.e. purpose, domain specificity, content and RE phase at which they are used, resulting in a high degree of flexibility to classify individual patterns. Various pattern properties can be recognized very easily as individual classifications reflect precise properties of the catalogue. On the contrary, such a scheme doesn't allow a comprehensive study of pattern properties as these four facets are exclusive of each other. It is considered sufficient if a given pattern is classified under any of the 4 facets and not necessarily under all. Not only does this isolation make navigating through various patterns a challenge but also poses a threat to understanding relationships between patterns across facets.

In contrast, our categorization divides patterns into 3 major groups as seen in Figure 1. An advantage of such a general scheme is that instead of classifying requirements patterns individually, this scheme allows an entire pattern catalogue to be classified as a single unit and make use of the internal classification scheme provided as part of the pattern book.

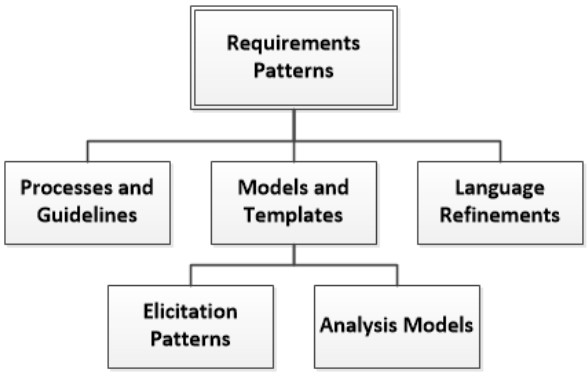

Figure 1: Classification of Requirements Patterns based on artefact.

Data organized in a single tree structure is easier to read and found to be more intuitive than in dispersed sets. Due to this aspect and the number of classes in our scheme, it can be said that our classification meets criteria I and II as defined earlier. Although very general in their outlook, each class of the proposed categorization reflects certain properties that concern pattern users in the requirements creation process. Though multidimensional classifications are better in this aspect, it can be argued that our classification satisfies criteria III and IV. All the patterns of a given catalogue have several commonalities - structural and semantic. After studying the large body of requirements

patterns collected, we are in agreement with Hafiz *et al.* [25] that regardless of the classification scheme, some patterns will always belong to more than one category. However, these 2 criterions are closely related with pattern applicability and the user would have to go through patterns individually as applicability varies from patterns to pattern. Since we are classifying pattern collections, defining precise relationships among constituting patterns may not be possible. However, each class of patterns proposed can be related to each other in more than one way based on the development methodology used, more precisely the order of RE activity. Figure 2 shows a general order of pattern application to satisfy criteria VI. As for V, since we employ very general criteria to divide the problem space, there is sufficient reason to believe that our classification can accommodate new patterns but it can only further research can test the validity of our proposed scheme.

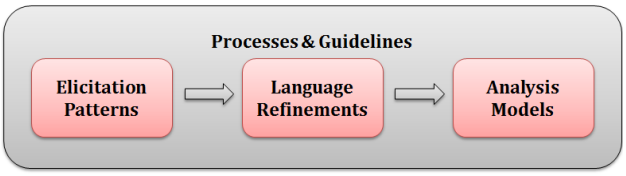

Figure 2: Relationship between various classes of Requirements Patterns

Though patterns can be found in all walks of life across disciplines, domains and paradigms, it is argued that good patterns are hard to write [2, 4, 6]. Below we discuss various properties that govern the selection of a requirements pattern or pattern catalogue.

Scope – While we outline the major purpose of each kind of pattern in the beginning of each category, the usage of many of these patterns is not restricted to a certain RE phase. Devedzic classifies process patterns that shape new organizations or evolve existing ones as *Organizational Patterns* [19] which apply to pragmatics concerning problems that lean towards the functioning of the team and the relationships between them. Other catalogues that capture Customer Interaction patterns [52] and Business Patterns [36, 56] prescribe processes which are not specifically related to the technical aspect of requirements gathering. A distinction can hence be made between requirements and managerial processes which underscores the impact of the pattern on the quality of requirements creation. In general, process patterns seem to have a broad context exceeding that of just the RE specific aspect but as a general guideline of distinguishing, RE specific process patterns suggest methods that directly facilitate requirements capture, analysis and documentation. Patterns from [69] namely *Customer Rapport, Sponsor Objective* and *Envisioning*, to name a few, could be thought of as managerial processes.

Similarly, several patterns described under analysis models section of our classification [4, 46, 37, 58] contain rich sets of information that impacts the architecture and design of a system. Some of the patterns also provide metrics that can be collected as the pattern is applied to test the validity of the resulting solution [39, 5, 70] and also discuss potential risks that may need to be mitigated during and after their application.

Applicability – A common misconception among pattern developers and users is that patterns are exact solutions to a problem in a defined context when in fact each instance of a pattern can produce a correct solution to the problem [4]. With that being said, the only feasible way to increase accuracy and precision of a pattern, especially in a vast discipline like requirements engineering, is to narrow the problem space [16, 70]. The key to optimal pattern reuse is to strike a balance by not making a pattern too generic or open to interpretation like the ones in [49] and [14], at the same time not making it very rigid like the ones in [54] where a pattern can be abstracted to a specific use case of a system.

Apart from factors like application domain, RE task being carried out or organizational policies and rules, several of the patterns we surveyed explicitly describe specific situations and factors in which they can be applied. Different patterns specify these conditions under different sections. Some list them as part of the *Context* while some have an applicability section. There are others that implicitly define applicability, mostly because of the nature of the patterns themselves. Henceforth the pattern template is an important factor that impacts applicability.

Access to pattern catalogue - Patterns have been documented and published as part of books and articles. Even dedicated pattern repositories have been created to store some of them like [26] and [65]. Patterns and ready-to-use requirement templates by Withall [70], for instance, are available online and can be downloaded as required however not all the patterns are freely accessible to users. Getting access to all patterns in the proposed pattern catalogues is still a major challenge as not all of them have been published publicly. Another reason

for this is that many of the collections that we studied are still a work in progress or incomplete.

Dependence on Frameworks and Tools – Adopting a requirements pattern catalogue into a generic development process isn't always possible. While all elicitation patterns can be used as a means to identify and document high level requirements of any general system, several goal and scenario based requirements modelling requires users to have considerable knowledge of the modelling languages like i* [13] and KAOS [53]. According to Alexander [2], a pattern does not need tool or methodical support to be effective. However, the power of certain patterns can only be harnessed when a user is well acquainted with the underlying framework that the patterns are based on. For instance, Supakkul *et al.*'s [65] NFR patterns are heavily based on the NFR framework proposed by Chung [13]. Similarly, Compliance control patterns tie in with the Business Process Compliance Management (BPCM) Framework [45]. Also, several of the patterns are supported by methodical processes and a set of tools that help automate tasks.

Functional vs. Non-functional requirements - It is noted that non-functional requirements tend to remain unchanged across domains [57]. While this is a true statement, we have come to the conclusion that it is only the definition of high level goals in natural language that remain unchanged. Their implementations are seen to differ based on the type of system and domain. Hence, elicitation patterns are subjected to wider reuse than other classes of patterns.

Another issue is that, it is difficult to classify any requirements patterns into functional or non-functional because non-functional requirements do not pertain to a specific part but to the entire system as a whole and are eventually refinement into functional goals during elaboration. Consider the *Data Validation* pattern in [70], for instance, is a functional requirement. It can be described in natural language and also in terms of a UML sequence diagram based on the need of the user. At the same time, the *Watchdog Pattern* [58] from the embedded systems requirement pattern catalogue can be seen as a functional as well as non-functional because it helps users structure components in a system that monitor and take correct action when required which is a functional requirement but also adds to the reliability, fault tolerance and safety of the system.

Notation used for requirements definitions - All requirements patterns use natural language as the primary medium of communication. The emphasis on it varies based primarily on the requirement characteristics that the pattern addresses. Analysis models that help elaborate requirements, i.e. define their static or dynamic behaviour as a component in the system and how they interact with corresponding components are represented as diagrams or as formal specifications. Patterns using formal or semiformal constructs are seen to address more requirement quality attributes than others. However, the notation used is not a function of the type of pattern. Patterns in [45] use formal assertions like the patterns proposed by Zhou *et al.* [11] but we classified the later as an analysis model pattern, the former helps identify and structure requirements and is hence an overlapping of an elicitation pattern and a language rule. Note that natural language patterns can be seen as formal logic rules applied on natural language and have a very specific role of reducing ambiguity in a requirements specification.

Domain specific patterns - Patterns which contain semi-formal or formal representation of system goals generally tend to be domain specific. Of the 12 pattern books that captures a recurring functional aspect, 9 of were found to have some kind of formal representation were identified as application domain specific with the exception of [53], [7] and [37]. In [53], patterns are essentially formal refinement rules defined in the KAOS language that are used in the elaboration of system goals. They suggest that domain specific frameworks should be considered when adapting them to systems of specific domains. As for [37], Klop defines *Organizational Patterns* for requirements analysis, more precisely modelling the domain of the system to be built. Though the patterns are domain independent, instances of each pattern contain components to extract domain specific information. It is important to note that patterns in [53] and [37] do not follow the Alexandrian definition of a pattern. Patterns in [7] may not be domain specific but are targeted to identify only security requirements. The application domain specified for some domain specific patterns is not clearly defined. Domains like business systems or information systems have a very broad definition.

Internal classification – Requirements patterns which tend to follow a classification scheme are found to be more helpful to the end user as it provides a user to navigate and understand the pattern catalogue better. Of the 28 pattern books

surveyed, 12 of them contain negligible (less than 7) or no defined number of patterns. Of the remaining 16, 13 of them provide a way to classify the constituent requirements patterns based on a metric or property. However, the resulting classification is precise or broad based on the metric chosen for classification. Withall's own pattern categorization is based on high level features of software systems which shows inter-pattern relationships specified structurally ("has", "uses", "is-a") and semantically ("is across", "displays").

Pattern interdependence – Patterns have been documented as individual solutions to a commonly recurring problem under a very specific context. This is true with most elicitation patterns and we have referred to them as a pattern catalogue or pattern book throughout our research. In many others, patterns work in tandem with others to form a vocabulary to solve a bigger problem than the one they are designed for individually. Such systems of highly coupled patterns constitute a *Pattern Language* [19]. In the approaches surveyed, these set of relations were either semantic or strict in terms of a meta-model.

In a pattern language, a resulting context of one pattern becomes the starting context of other patterns. A fundamental view of a pattern language is the description of the pattern relationships that is also stressed in [2], as follows: "when you build a thing you cannot merely build that thing in isolation." These relationships constitute the whole system to be designed. In other words, the notion of sequence of patterns in a pattern language is crucial to explain how the language works.

5. Conclusion

Requirements patterns can be powerful tools to streamline the requirements engineering processes as they capture proven knowledge. In this paper, we have surveyed and categorized various requirements patterns according to the kind of artefact they present to the user to aid the process of requirements engineering. We also listed out some of our findings which we believe will have significant impact in a user's pattern selection process.

However, further research effort to integrate them into existing development frameworks is necessary. Despite the fact that the pattern concept has been applied to distinct fields, there are not enough experience reports that patterns can help everybody. Two major reasons for this is the lack of access to requirements pattern catalogues and the late growth trend of requirements patterns. Extensive experimentation by external sources such as the one done in [41] is needed prove their validity. Also, adapting a pattern or in certain cases, an instantiated pattern so that it fits the desired context is still considered an art. We believe that better examples of individual pattern implementations could help widespread adaptation of these patterns among practitioners.

References:

[1] A. Durán Toro, Bernárdez B Jiménez, A. Ruiz Cortés & M. Toro Bonilla, *A Requirements Elicitation Approach Based in Templates and Patterns*, Workshop em Engenharia de Requisitos, 1999.

[2] Alexander Christopher, *The Timeless Way of Building*, Oxford University Press, 1979.

[3] Scott W Ambler, *Process Patterns: Building Large-Scale systems using Object Technology*, Cambridge University Press, 1998.

[4] Andrade Rossana Maria De Castro Capture, *Reuse, and Validation of Requirements and Analysis Patterns* – PhD. Dissertation, University of Ottawa , 2001 .

[5] Ayman Mahfouz, Leonor Barroca, Robin Laney & Bashar Nuseibeh, *Patterns for Service-Oriented Information Exchange Requirements*, Proceedings of the 2006 Conference on Pattern Languages of Programs (PLOP '06), 2006.

[6] Betty Cheng & Joanne Atlee, *Research Directions in Requirements Engineering,* Future of Software Engineering (FOSE '07), 2007. - pp. 285 - 303.

[7] Betty Cheng, Sascha Konrad, Laura Campbell & Ronald Wassermann, *Using Security Patterns to Model and Analyze Security,* IEEE Workshop on Requirements for High Assurance Systems, 2003.

[8] Bunke Michaela, Koschke Rainer & Sohr Karsten, *Organizing Security Patterns Related to Security and Pattern Recognition Requirements,* International Journal on Advances in Security, 2012. - Vol. 5.

[9] Buschmann F, Meunier R, Rohnert H, Sommerlad P & Stal M, *Pattern Oriented Software Architecture: A System of Patterns*, John Wiley & Sons, 1996.

[10] Carlos Videira & Alberto Rodrigues Da Silva, *Patterns and metamodel for a natural-language-based requirements specification language*, Proc. of CaiSE'05 Forum, 2005.

[11] Changyan Zhou, Ratnesh Kumar, Devesh Bhatt, Kirk Schloegel & Darren D. Cofer, *A Framework of Hierarchical Requirements Patterns for Specifying Systems of*

Interconnected Simulink/Stateflow Modules, Proceedings of the Nineteenth International Conference on Software Engineering and Knowledge Engineering (SEKE'2007), 2007.

[12] Christina Palomares, Carme Quer & Xavier Franch, *PABRE-Proj: Applying patterns in requirements elicitation,* 21st IEEE International Requirements Engineering Conference (RE '13), 2013.

[13] Lawrence Chung, *Representing and Using Non-Functional Requirements: A Process-Oriented Approach,* IEEE Transactions on Software Engineering, 1992. - 6 : Vol. 18.

[14] Creel Christopher, *Requirement Patterns*, 30th International Conference on Technology of Object-Oriented Languages and Systems (TOOL '99), 1999.

[15] Daniel Méndez, Fernández Stefan Wagner, Klaus Lochmann & Andrea Baumann, *Field Study on Requirments Engineering Artefacts and Patterns*, 14th International Conference on Evaluation and Assessment in Software Engineering (EASE '10), 2010.

[16] Christian Denger, *High quality requirements specifications for Embedded Systems through authoring rules and language patterns* - Masters Thesis, Kaiserslautern University, 2002.

[17] Denger C., Berry D.M. & Kamsties E., *Higher quality requirements specifications through natural language patterns,* IEEE International Conference on Software: Science, Technology and Engineering., 2003.

[18] Devedzic Vladan, *Software Patterns* in Handbook of Software Engineering and Knowledge by Chang S.K., World Scientific Publishing Co, 2002.

[19] Documents Associated with Unified Modeling Language (UML), V2.4.1, Object Management Group 2014. Online at - http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF/.

[20] E. Gamma, R. Helm, R. Johnson & J Vlissides, *Design Patterns : Elements of Reusable Object Oriented Software,* Addison-Wesley Professional, 1994.

[21] Eric Knauss, Daniela Damian, Germán Poo-Caamaño & Jane Cleland-Huang, *Detecting and classifying patterns of requirements clarifications*, 20th IEEE International Requirements Engineering Conference (RE '12), 2012.

[22] Eric Yu & John Mylopoulos, *Understanding "Why" in Software Process Modelling, Analysis, and Design,* Proceedings of the 16th International Conference on Software Engineering, 1994.

[23] Martin Fowler, *Analysis Patterns: Reusable Object Models,* Addison-Wesley Professional, 1996.

[24] Xavier Franch, Cristina Palomares, Carme Quer & Samuel Renault, *A Metamodel for Software Requirement Patterns,* Requirements Engineering: Foundation for Software Quality (REFSQ '10), 2010. - Vol. 6182.

[25] Hafiz Munawar, Adamczyk Paul & Johnson Ralph E, *Organizing Security Patterns,* IEEE Software, 2007. - 4 : Vol. 24.

[26] Hagge Lars & Lappe Kathrin, *Using Requirements Engineering (RE) Patterns for Organizational Learning,* Journal of Universal Knowledge Management, 2006. - 2 : Vol. 1.

[27] Elke Hochmüller, *Requirements classification as a first step to grasp quality requirements,* Proceedings of the Third International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ' 97), 1997.

[28] Hoffmann, A., Schulz T., Hoffmann H., Jandt, S.; Roßnagel, A. & Leimeister J., *Towards the Use of Software Requirement Patterns for Legal Requirements,* 2nd International Requirements Engineering Efficiency Workshop (REEW '12), 2012.

[29] Hoffmann A., *A Pattern-based approach for analysing requirements in socio-technical systems engineering,* 20th IEEE International Requirements Engineering Conference (RE '12), 2012.

[30] Home: First International Workshop on Requirements Patterns (RePa '11) 2011 online http://www.utdallas.edu/~supakkul/rp11/index.html.

[31] Home: Second International Workshop on Requirements Patterns (RePa' 12) 2012 online - http://www.utdallas.edu/~supakkul/repa12/.

[32] Honeywell International Inc, 2013 online at - http://honeywell.com/Pages/Home.aspx.

[33] Ian Sommerville, David Martin & Mark Rouncefield, *Informing the Requirements Process with Patterns of Cooperative Interaction,* International Arab Journal of Information Technology, 2003.

[34] IEEE Recommended Practice for Software Requirements Specifications, IEEE Computer Society, 1998.

[35] Michael Jackson, *Problem Frames: Analysing & Structuring Software Development Problems*, Addison-Wesley, 2000.

[36] Allan Kelly, *Business Patterns Software Developers,* Wiley, 2012.

[37] Manuel Kolp, *Organizational Patterns for Early Requirements Analysis,* 15th International Conference on Advanced Information Systems Engineering (CaiSE'03), Springer, 2003.

[38] Axel van Lamsweerde, *Requirements Engineering in the year 00: A Research Perspective*, Proceedings of the 2000 International Conference on Software Engineering (ICSE '00), 2000.

[39] Lars Hagge & Kathrin Lappe, *Sharing requirements engineering experience using patterns*, IEEE Software, 2005. – 1 : Vol. 22.

[40] Lawrence Chung, Barbara Paech, Liping Zhao, Lin Liu & Sam Supakkul, *RePa Requirements Pattern Template,* International Workshop on Requirements Patterns (RePa '12), 2012.

[41] Markus Strohmaier, Jennifer Horkoff, Eric Yu, Jorge Aranda and Steve Easterbrook, *Can Patterns improve i* Modeling? Two Exploratory Studies* in Requirements Engineering: Foundation for Software Quality – Lecture Notes in Computer Science by Barbara E Paech & Colette E Rolland, Springer Berlin Heidelberg, 2008. – Vol. 5025.

[42] James Naish & Zhao Liping, *Towards a generalised framework for classifying and retrieving requirements patterns*, First International Workshop on Requirements Patterns (RePa '11), 2011.

[43] Natalia Juristo, Ana Moreno & Maria-Isabel Sanchez-Segura, *Moving Usability Forward to the Beginning of the Software Development Process* in Human Computer Interaction by Pavlidis Ioannis, InTech, 2008.

[44] Nuseibeh Bashar & Easterbrook Steve, *Requirements Engineering: a Roadmap,* Proceedings of the Conference on The Future of Software Engineering (ICSE '00), 2000.

[45] Oktay Turetken, Amal Elgammal, Willem-Jan van den Heuvel & Michael P. Papazoglou *Capturing Compliance Requirements: A Pattern-Based Approach*, IEEE Software, 2012. – 3 : Vol. 29.

[46] P. Pavan, N.A.M. Maiden & X. Zhu, *Towards a Systems Engineering Pattern Language: Applying i* to Model Requirements – Architecture Patterns,* Proceedings of the 2nd International Workshop Software Requirements to Architectures (STRAW' 03), IEEE Press, 2003.

[47] Paetsch F, Eberlein A & Maurer F, *Requirements Engineering and Agile Software Development,* Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 2003.

[48] PloP Conferences – The Hillside Group, 2013 online at http://hillside.net/conferences.

[49] R. S. Wahono & J. Cheng, *Extensible Requirements Patterns of Web Application for Efficient Web Application Development,* Proceedings of the First International Symposium on Cyber Worlds, 2002.

[50] Daniel A Rawsthorne, *A Pattern Language for Requirements Analysis,* Procs. Of Workshop in the Conference of Pattern Language of Programs, 1996.

[51] The Presentation Pattern Dr. Dobb's – The World of Software Development 2006 online at – http://www.drdobbs.com/architecture-and-design/requirements-by-pattern/196600223?pgno=3.

[52] Rising Linda, *Customer Interaction Patterns* in Pattern Languages of Program Design 4 by Neil Harrison, Brian Foote & Hans Rohnert, Addison Wesley, 2000.

[53] Robert Darimont & Axel v. van Lamsweerde, *Formal refinement patterns for goal-driven requirements elaboration*, Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering (SIGSOFT '96), ACM Press, 1996. – Vol. 21.

[54] Robertson Suzanne, *Requirements Patterns Via Events/Use Cases,* The Atlantic Systems Guild Ltd, 1996.

[55] Kristin Roher & Debra Richardson, *Sustainability Requirements Patterns,* Third International Workshop on Requirements Patterns (RePa '13), 2013.

[56] S. J. Bleistein, A. Aurum, K. Cox, & P. K. Ray, *Linking requirements goal modeling techniques to strategic e-business patterns and best practice,* Australian Workshop on Requirements Engineering (AWRE'03), 2003.

[57] Samuel Renault, Oscar Mendez-Bonilla, Xavier Franch & Carme Quer, *A pattern based method for building requirements documents in call for tender processes,* International Journal of Computer Science & Applications (IJCSA '09), 2009. – 5 : Vol. 6.

[58] Sascha Konrad & Betty Cheng, *Requirements Patterns for Embedded Systems,* Proceedings of the IEEE Joint International Conference on Requirements Engineering (RE' 02), 2002.

[59] Sascha Konrad, Laura Campbell, Betty Cheng & Min Deng, *A Requirements Patterns-Driven Approach to Specify Systems and Check Properties*, Model Checking Software, Springer Berlin Heidelberg, 2003. – Vol. 2648.

[60] Scott Henninger & Victor Corrêa, *Software Pattern Communities: Current Practices and Challenges,* Proceedings of the 14th Conference on Pattern Languages of Programs (PLOP '07), 2007.

[61] Simulink, MathWorks 2013 online at – http://www.mathworks.com/products/simulink/index.html.

[62] Rocky Slavin, Shen Hui & Niu Jianwei, *Characterizations and boundaries of security requirements patterns,* Second International Workshop on Requirements Patterns (RePa '12), 2012.

[63] Ian Sommerville & Pete Sawyer, *Requirements Engineering: A Good Practice Guide,* Wiley, 1997 .

[64] Sangeeta Srivastava, *A Repository of Software Requirement Patterns for Online Examination System,* International Journal of Computer Science Issues (IJCSI '13), 2013. – 3 : Vol. 10.

[65] Sam Supakkul, Tom Hill, Lawrence Chung,; Tun, Thein Than, Julio Cesar Sampaio do Prado Leite, *An NFR Pattern Approach to Dealing with NFRs*, 18th IEEE International Requirements Engineering Conference, 2010.

[66] Tjong S.F., Hallam N. & Hartley M., *Improving the Quality of Natural Language Requirements Specifications through Natural Language Requirements Patterns*, The Sixth IEEE International Conference on Computer and Information Technology (CIT '06), 2006.

[67] Sri Fatimah Tjong, *Natural Language Interfaces for Requirements Engineering* – PhD. Desertation, University of Nottingham, 2006.

[68] About Use Case Maps, User Requirements Notation (URN) Wiki 2013 – online at http://www.usecasemaps.org/aboutucms.shtml

[69] Bruce Whitenack, *RAPPeL: A Requirements-Analysis Process Pattern Language for Object-Oriented Development* in Pattern Languages of Program Design by James Coplien & Douglas C. Schmidt, ACM Press/Addison-Wesley, 1995.

[70] Stephen Withall, *Software Requirements Patterns,* Microsoft Press, 2008.

Table 1: Summary of different Requirements Patterns arranged chronologically.

Pattern Book	Primary Purpose	Domain	Notation	Pattern Count	Year	Requirement Type	Aspect captured
[69]	To profile the roles of project managers and developers and requirements analysts	General Purpose	Natural Language	NA	1995	NA	Software engineering and project management best practices
[50]	Suggest processes to gather, model and validate requirements	General Purpose	Natural Language	12	1996	NA	Requirements Engineering best practices
[54]	Requirements Analysis	Business Systems	Natural Language + UML	NA	1996	Functional	Structural and data models of requirements
[53]	Requirements Elaboration	General Purpose	Natural Language + logical assertions as AND Trees	NA	1996	Both	Goal refinement rules
[14]	Requirements Elicitation	General Purpose	Natural Language	3	1999	Functional	High level system goals
[1]	Document Requirements	Information Systems	Natural Language	4	1999	Both	Functionality of a system
[52]	Improving communication among project stakeholders	General Purpose	Natural Language	12	2000	NA	Social and psychological factors that affect stakeholder communication
[4]	Requirements analysis and modelling	Mobile systems	Natural Language + Use case maps	12	2001	Both	Recurring functional behaviour and architecture
[49]	Document Requirements	Web based systems	Natural Language + UML	30	2002	Both	Functionality of a system
[16]	Improving precision of natural language requirements specifications	Embedded systems	Natural language	38	2002	Functional	NA
[7]	Requirements Analysis	General purpose	Natural language + UML	8	2003	Nonfunctional	Structure and behaviour of Security Requirements
[58]	Requirements Analysis	Embedded Systems	Natural Language + UML	10	2003	Both	Structure and behaviour of the System
[33]	Facilitate communication and interaction among various resources in a team.	General Purpose	Natural Language + Diagrams	10	2003	NA	Cross ethnographic studies
[46]	Requirements modelling	Submarine Manouvering Systems	Natural Language + i* Models	4	2003	Both	Tradeoffs between requirments and architecture choices
[37]	Requirements modelling	General Purpose	Telos + i* Models	10	2003	Nonfunctional	Behaviour of a system based on architectural style

[5]	Identifying and Structuring Requirements	Information systems	Natural Language + UML	11	2006	Both	Structure of information exchange requirements
[66]	Formalize natural language in requirements specifications	General purpose	Natural Language	23	2006	Both	NA
[11]	Requirements Analysis	Systems modelled using Simulink	Natural Language + Temporal Logic	NA	2007	Functional	Behaviour of a system
[70]	Document Requirements	General purpose	Natural Language	37	2008	Both	System functionality
[43]	Requirements Elicitation	General Purpose	Natural Language	15	2008	Nonfunctional	Usability requirements for HCI based systems
[57]	Requirements Elicitation	General Purpose	Natural Language	29	2009	Nonfunctional	Functionality of COTS systems
[65]	Eliciting and modelling requirements	Web based systems	Natural language + Goal graphs	4	2010	Nonfunctional	Refinement rules and trade-off decisions for security requirements
[15]	To understand and customize RE processes for volatile project environments	Business Information systems	Natural Language	NA	2010	NA	NA
[21]	Detecting and rectifying miscommunication	General Purpose	Natural Language + Line/Bar graphs	6	2012	Both	NA
[29]	Requirements Elicitation	Socio-Technical information systems	Natural Language	6	2012	Both	Requirements of legal rules and regulations that a system is required to adhere to.
[45]	Verifying and managing requirements	E-business and banking systems	Natural language + logical assertions	27	2012	Both	Recurring compliance rules for business processes
[55]	Document requirements	General Purpose	Natural Language	3	2013	Both	Sustainability requirements
[64]	Document Requirements	Online Examination Systems	Natural language	30	2013	Both	Functionality of a system