Industrial Engineering
& Management Systems
Vol 11, No 1, Mar 2012, pp.70-81
ISSN 1598-7248 | EISSN 2234-6473 |
http://dx.doi.org/10.7232/iems.2012.11.1.070

Design Requirements in Software and Engineering Systems

A. M. Eleiche*
Dept. of Mechanical Engineering, King Fahd University of Petroleum and Minerals
I. Ahmad
Dept. of Information and Computer Sciences, King Fahd University of Petroleum and Minerals
M. O. Elish
Dept. of Information and Computer Sciences, King Fahd University of Petroleum and Minerals

(Received: November 13, 2011 / Revised: January 10, 2012 / Accepted: January 13, 2012)

ABSTRACT

The subject of "Design Requirements" (DR) is central to the design of software and engineering systems. The main reason for this is that quality aspects are usually closely tied to requirements, among other things. In this review paper, we consider how the subject of requirements is being managed in these two seemingly different design disciplines. Two important aspects are covered, namely: (a) requirements development, describing various activities leading to requirements documentation, and (b) requirements change management, describing various activities needed for the proper treatment of the inevitable changes in requirements. Similarities and differences on how these two aspects are handled in software and engineering systems are highlighted. It is concluded from this literature survey that the management of software requirements is quite coherent and well established as a science. On the other hand, management of engineering systems requirements suffer from being unstructured, in particular when requirements changes are involved. Important gaps and future important research areas are identified.

Keywords: Software Requirements Engineering, Design Requirements in Engineering Systems, Requirements Development and Change Management

* Corresponding Author, E-mail: eleichea@kfupm.edu.sa

1. INTRODUCTION

In a previous paper (Eleiche, 2010), it was concluded that in order to support innovative energy research projects funded by the CCWCE (2008), innovators should be provided with a software program that can help manage the effects of possible changes in design requirements (DR) and hence facilitate the design of successful quality systems on schedule and within planned budgets. It was also stated that guidance would be sought from similar research being carried out in the software engineering arena.

In the present paper, we address in detail the subject of DR, which is central to the design of software and engineering systems. The main reason for this is that quality aspects are usually closely tied to requirements, among other things. In this review paper, we consider how the subject of requirements is being managed in these two seemingly different disciplines. Two important aspects are covered, namely: (a) requirements development, describing various activities leading to re-quirements documentation, and (b) requirements change management, describing various activities needed for the proper treatment of the inevitable changes in requi-rements. Similarities and differences on how these two aspects are handled in software and engineering systems are highlighted. Important gaps and future important research areas are also identified.

The next section introduces the topics related to DR. in a generic way. This is followed in Section 3 by

an overview of how DR development is treated in software and engineering systems, while Section 4 presents how DR management is being applied in these same two disciplines. Important conclusions are given finally in Section 5.

2. REQUIREMENTS

IEEE (1990) defines "Requirement" as a condition or capability that: (a) is needed by a user to solve a problem or achieve an objective, (b) must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document. The set of all requirements forms the basis for subsequent development of the system or system component.

Requirements Engineering (RE) is a crucial aspect for a successful development of software and engineering systems. The RE process is the primary work of the "Requirements" phase of the project. Figure 1 shows the sub disciplines of RE (Wiegers, 2000) consisting of two major types of activities. In *RE development*, "User Needs" are translated into "Requirements Specification" through five activities that form a sub-process, while the activities *in RE management*, namely traceability and change, do not flow into each other but are separate.

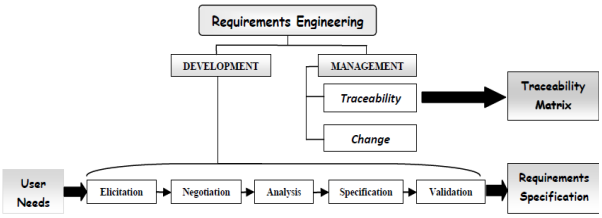


Figure 1. Sub Disciplines of RE, Adapted from Wiegers (2000).

3. REQUIREMENTS DEVELOPMENT (RD)

Requirements can be classified into: (a) Functional requirements that capture the intended behavior in terms of services, tasks or functions the system is required to perform; (b) Non-functional requirements (or system qualities) that capture required properties or qualities of the system; (c) Constraints (organizational, operational, economical, legislative, and ethical). All requirements must be carefully derived through analysis of user needs and documented. They should specify *what* is to be done, not *how* it is to be done**.**

A good requirement should be (Tavassoli, 2009): clear; complete; correct; consistent; verifiable; traceable; feasible; modular; adaptable; and design-independent.

Nowhere more than in the requirements process do the interests of all the stakeholders in a software or engineering system project intersect. These include: (a) external stakeholders (end users, customers acquiring the product, suppliers, distributors, subcontractors, legislators, policy makers); and (b) internal stakeholders (requirements analysts, project managers, developers, designers, manufacturing staff, testers, regulators and auditors, sales and marketing, purchasing and finance, and field support or help desk staff). Handled well, this intersection can lead to exciting products, delighted customers, and fulfilled developers. Handled poorly, it is the source of misunderstanding, frustration, and friction that undermine the product's quality and business value.

3.1 RD in Software

Many software problems arise from shortcomings in the ways that people acquire, document, agree on, and modify the product's requirements. Typical problem areas include informal information gathering, implied functionality, inadequately defined requirements, and a casual change process (SERENA, 2011). The widely quoted CHAOS report (Standish Group, 1995) relates the consequences of casual approaches to requirements engineering. Year after year, lack of user input, incomplete requirements, and changing requirements are the major reasons why so many software and information technology projects fail to deliver all of their planned functionality on schedule and within budget.

In software engineering, RE has already been accepted as an independent discipline and is done systematically. As will be seen in the following, many concepts and methods for handling of requirements have already been elaborated. Hence, a common terminology of process phases in RD has emerged, involving elicitation, negotiation, analysis, specification and validation. This is depicted in Fig. 2, as proposed by Kotonya and Sommerville (1998).

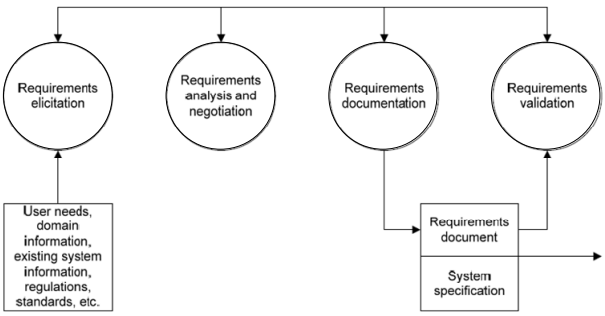


Figure 2. Software RD Phases (Kotonya and Sommerville, 1998).

During the ***requirements elicitation*** phase, different complementary techniques are applied to understand the application domain, and the problems and needs of all stakeholders. The most common software requirements elicitation techniques are interviewing, brainstorming, storyboarding, workshops, surveys/questionnaire, and ethnography (Braude, 2010; Lamsweerde, 2009; Press-

man, 2005; Sommerville, 2007; Wiegers, 2003). The full integration of users is a decisive factor for the success of this RD phase (Kujala *et al*., 2005).

During ***requirements negotiation***, the elicited requirements are discussed with the stakeholders. Requirements are then prioritized, unnecessary ones are removed, and conflicting and incomplete ones are resolved. Requirements scope/baseline is also defined and agreed upon (Kotonya and Sommerville, 1998). Prioritizing requirements "assists project managers with resolving conflicts, plan for staged deliveries, and make necessary trade-off decision" (Aurum and Wohlin, 2005).

During ***analysis***, models are created from different perspectives to develop an understanding of the system, as well as checking for necessity, completeness, consistency and feasibility. A system model abstracts the system leaving out the unnecessary details. Models enable to filter out the complexities of the real world not relevant for the time being, so that directed effort can be put towards the most important parts of the system under development (Giaglis, 2001; Kotonya and Sommerville, 1998; Sommerville, 2007). Moreover, visualizing requirements as models helps the customer better understand the requirements (Tavassoli, 2009).

Requirements ***specification*** refers to the production of a requirement document which can be systematically reviewed, evaluated, and approved (Bourque and Dupuis, 2004). Once the requirements are agreed upon between the stakeholders and the development team, they are specified in a document, popularly known as *Software Requirements Specification* (SRS) document. This is an official statement of what is required of the system developers. Its purpose is to be an authoritative statement of 'what' the software is to do. As far as possible, the document shall not address the design and implementations issues, and should be detailed enough to allow the design of the software without user involvement. In general, the size and content of the SRS document should reflect the size and complexity of the software product.

The many sections in the SRS document should detail various aspects of the software system to be developed, mainly: functional requirements, non-functional requirements, design constraints and interface specifications (Kotonya and Sommerville, 1998; Sommerville, 2007). IEEE (1998) has defined a standard known as "IEEE Std 830-1998" which provides guidelines for documenting software requirements specification.

The requirements document along with the models can be and should be managed using requirements management tools such as Telelogic DOORS or IBM Rational RequisitePro.

The requirements ***validation*** process ensures that the software engineer has understood the requirements. It is also important to verify that a requirements document conforms to company standards, and that it is understandable, consistent, and complete (Bourque and Dupuis, 2004). The process attempts to identify the errors in the SRS document before it is used as a basis for further system development. In other words, requirement validation is concerned with demonstrating that the requirements define the system that the customer really wants. Prototyping, reviews, inspections, and test case generation are the most commonly used requirements validation techniques.

3.2 RD in Engineering Systems

By engineering systems, we mean consumer or capital products and systems ranging from simple to quite complex ones. Especially in modern energy applications, these may also incorporate many kinds of software, thus forming "hybrid systems" that contain material and immaterial parts. The following discussion will be limited to systems consisting of hardware only.

In engineering systems design, RD is usually taken into account, as stated in most engineering design textbooks. For instance, Cross (1989) suggests the use of a goal tree to vaguely collect the initial requirements. The requirements are further refined as the problem understanding of the customer and engineers increases. Pahl and Bietz (1996) suggest a sequential process model in which the engineer has to extract the requirements from the customer's wishes. They also recognize that customers are often not able to express their requirements appropriately; however, methods for eliciting these requirements are not suggested. Ulrich and Eppinger (2008) collect the requirements in hierarchical weighted lists. They also state that it is important to reveal implicit customer needs, and that a common product understanding between customer and engineer is necessary.

In their comprehensive review, Jiao and Chen (2006) state that RD usually encompasses only the three activities of *elicitation, analysis and specification*. During ***Elicitation***, approaches used can be classified into:

- *Psychology-based approaches:* where techniques such as Kansei engineering; Kawakita Jiro method, affinity diagrams, and laddering can be employed.
- *Methods and tools from the field of knowledge acquisition*: where techniques employed can be:
  (a) "Contrived" (not-heavily dependent on natural language dialogue, but good at reducing systematic bias, eliciting implicit knowledge, representing declarative and procedural knowledge): e.g. sorting, laddering, repertory grids.
  (b) "Non-contrived" (traditional) techniques: e.g. surveys, observations, ethnography, self-reports, interviews.
- *AI-based approaches*: where fuzzy systems, regression analysis, and expert systems have been developed for eliciting customer requirements more accurately and objectively. Also, integrated approaches by combining picture sorts and laddering, fuzzy evaluation and neural network techniques.
- *Knowledge recovery*: from historical data.

Case studies of elicitation in practice are available in the literature. This is described by Mathelin *et al.* (2005) for the automotive domain, and by Ward *et al.* (2003) and Agouridas *et al.* (2006) for medical devices.

Activities in the ***Analysis*** phase consist of:

- *Understanding market and customer needs.*
- *Customer preference.*
- *Prioritization*: By assigning different importance weights for customer requirements. This affects the target values to be set for the engineering characteristics. Existing techniques are:
  (a) AHP (Analytic hierarchy process)
  (b) Fuzzy AHP
  (c) Using supervised learning with a radial basis function (RBF) neural network
  (d) Applying conjoint analysis to prioritize customer requirements through pairwise comparisons
- *Classification*: This helps guiding the designer in compiling, organizing, and analyzing product design issues. Existing techniques are:
  (a) Affinity diagrams
  (b) Ontology for representing requirements that supports a generic requirement process. Ontology defines parts, features, requirements and constraints
  (c) Taxonomic approach (developing a set of taxonomies to assist in gathering, storing, using and reusing requirements)

Specification is concerned with the creation of a structurally concrete and precise specification of product requirements based on functional knowledge that has been elicited from key stakeholders. Common techniques are:

- *Requirement transformation*: e.g. Customer optimization route and evaluation (CORE) model; methodology of organizing specifications in engineering (MOOSE)
- *QFD.* To translate customer requirements to technical design requirements
- *Fuzzy QFD*: To enhance handling of ambiguous requirement information and evaluating inputs. Subjective crisp variables are expressed as fuzzy numbers
- *Prioritization of design requirements.* Importance rating among engineering characteristics in the QFD; a linear programming model for the prioritization of design requirements in the QFD planning process; employing a fuzzy outranking approach to prioritize the design requirements in the QFD
- *Targets of design requirements*: (usually defined by design teams subjectively and empirically) Using a fuzzy set theoretic approach to determine optimum target values for the engineering characteristics in QFD with consideration of the relevant constraints; using fuzzy regression and fuzzy optimization.

Therefore, as the RD phases proceed, the informally expressed needs of stakeholders are explored, developed and expanded into a more complete and formal document, variously known as the product description, technical specification, or *Product Design Specification* (PDS), that is understood and agreed upon by all stakeholders, and from which a design solution can be proposed.

3.3 Discussion on RD

RD in software engineering is highly elaborated and many methods and process models are known. Software engineering sees RD as a continuous activity that is performed throughout the entire development process, while product/systems engineering considers RD as a phase at the beginning of the project.

The specification of requirements in software engineering results from not only the transformation of customer requirements from those end-users, but also considerations of many engineering concerns. This is consistent with the principle of viewpoint-oriented software requirement engineering, where multiple viewpoints encapsulate different types of requirement models natural to different stakeholders.

4. REQUIREMENTS MANAGEMENT (RM)

RM involves two main tasks, namely: (a) requirements traceability, performed once the specifications have been documented, in order to determine the links between various requirements; and (b) change impact analysis, conducted at any stage of the design process, once a change of any kind has been proposed, in order to determine the feasibility, implications, cost, etc, of such a change, and finally deciding on its approval, as illustrated in Figure 3.

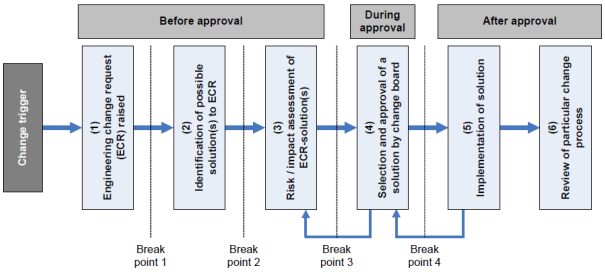


Figure 3. Generic Engineering Change Process (Jarratt *et al.*, 2004).

Changes in requirements are always expected. Indeed, they are the norm and not the exception, since most complex projects involve interdisciplinary system scenarios. In these cases, the stakeholders needs are usually not fully identified, and hence the requirements and the specifications are loosely-defined. Therefore, those initial requirements can be expected to change for a variety of reasons, e.g. altered market needs, safety concerns, problem corrections, new constraints, introduction of new technologies, uncertainty of resources, cost considerations, legislation changes, etc. On the other hand, in such systems, components are usually highly interconnected; a change to one requirement in one

component can propagate through the system and cause other changes in other components and parts. The change can also spread to other products (e.g. other family members) due to common platforms, other processes (e.g. manufacturing), and other businesses (e.g. suppliers, partners, etc.) (Jarratt *et al.*, 2011). Hence, the need of full-scale traceability between all system players.

In complex systems, it is important to identify and stabilize the requirements as early as possible in the process. Otherwise, inevitable changes will cause disruption of the product development schedule, increase of costs, and failure to meet the expected system quality. Table 1 shows the relative costs of fixing requirements defects in different phases of a project. Whatever it may cost to do things right in the requirements phase, it may be 3 to 1000 times more costly to fix later.

Table 1. Relative Costs of Fixing Requirement errors (Gause and Weinberg, 1989).

Phase in Which Fixed	Relative Cost
Requirements	1
Design	3~6
Coding	10
Development Testing	15~40
Acceptance Testing	30~70
Operations	40~1000

Similarly, Table 2 shows that return on investment (ROI) from practicing good requirements management is substantial, in terms of its results and benefits.

Table 2. ROI from practicing good RM (SERENA, 2011).

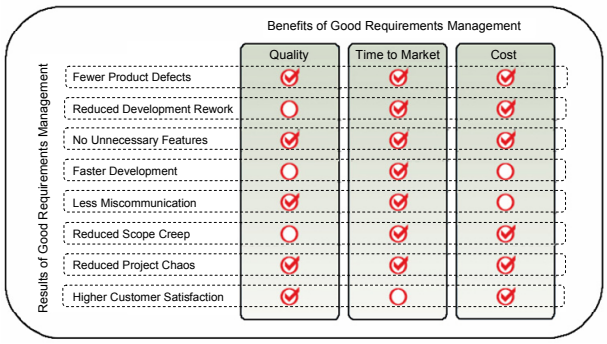


Table 3. Investing in Requirements Accelerate Development (Blackburn *et al.*, 1996).

Finally, a European study by Blackburn *et al.* (1996) showed that teams that developed products more quickly were found to have devoted more of their schedule and effort to requirements than did slower teams (Table 3).

4.1 RM in Software

With respect to managing requirements change in software development, the requirements can be broadly categorized into volatile (requirements that are likely to change), and non-volatile requirements (stable ones) (Sommerville, 2007; De Lucia *et al.*, 2008). As a proactive way to manage changing requirements, the volatile requirements are sometime further categorized. Categorizing requirements helps in change management as the reason and justification for change is better understood and proper attention can be given to different categories of non-volatile requirements (Sommerville, 2007; Harker *et al.*, 1993). Additionally, if a change can be categorized, we can understand its impact and how much control we have on this change (McGee and Greer, 2009). One of the popular categories of changing requirements is that proposed by Harker *et al.* (1993). Sometimes the changes are categorized from the developer's perspective too (Nurmuliani *et al.*, 2004). During categorization of requirements, it is also helpful to assess the probability of the change to that requirement (De Lucia *et al.*, 2008). A simplified view of the change management process is shown in Figure 4.

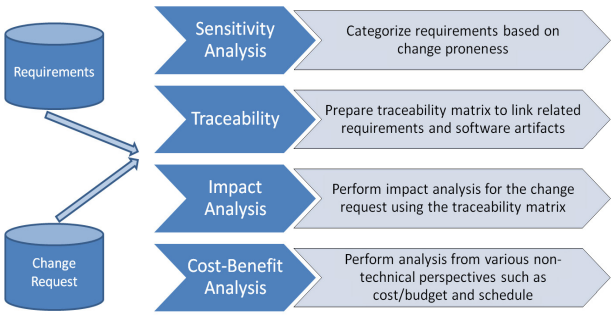


Figure 4. Requirements Change Management Process.

A requirements change normally begins as a request, either formal or informal. Informal change requests need to be checked and controlled as far as possible as it leads to many problems due to inadequate allocation of schedule and manpower to manage those changes. A formal change request needs to be documented from the beginning. A change request has many attributes apart from the requested change itself. Among the important attributes for a change request are type of change <add, delete and modify> (Strens and Sugden, 1996; McGee and Greer, 2009), the importance of change (O'Neal and Carver, 2001), the reasons and justification and the source of change (Nurmuliani *et al.*, 2004; McGee and Greer, 2009). It may not be possible to collect and document all the attributes for a given change request but the more

the information available on these attributes, the easier it is to handle them.

Once the change request is received, the next important activity is to assess the impact of implementing this change. This is popularly known as **impact analysis**. Conducting impact analysis helps answer many questions related to the impact of implementing this change. Among the important aspects are, when in development cycle the change needs to be implemented (Ramzan and Ikram, 2006; Imtiaz *et al*., 2008; Strens and Sugden, 1996), what artifacts are impacted by this change (Ramzan and Ikram, 2006; De Lucia *et al.*, 2008; O'Neal and Carver, 2001; Imtiaz *et al.*, 2008), the degree of change (De Lucia *et al.*, 2008; O'Neal and Carver, 2001; Strens and Sugden, 1996) and who among the stakeholders are impacted by the change (Ramzan and Ikram, 2005; O'Neal and Carver, 2001). Figure 5 shows the impact of change affecting different phases and artifacts.

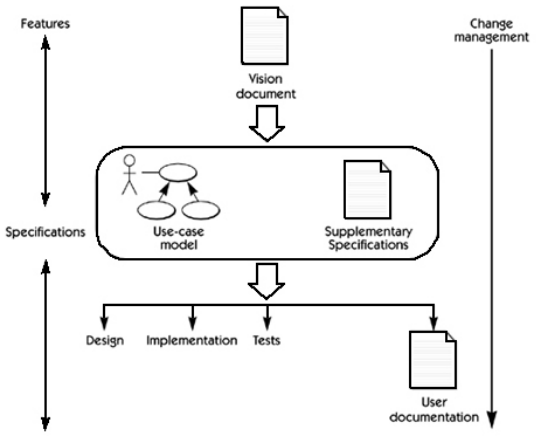


Figure 5. Ripple Effect on Artifacts and Phases Due to Change (Leffingwell and Widrig 2003).

One of the main techniques to do the impact analysis is the use of **traceability matrix**. A traceability matrix represents the dependency relationships between different requirements and also across different artifacts. This dependency relationship can be of different types like 'dependent,' 'related.' There are different types of traceability matrix depending on the level of details it works on, the type of information it stores, etc. (Ibrahim *et al.*, 2005; O'Neal and Carver, 2001). An example of a simple traceability matrix is shown in Figure 6. A common categorization of traceability matrix is vertical versus horizontal traceability matrix. A vertical traceability matrix includes dependency relationship between artifacts and entities from different phase of development whereas a horizontal traceability matrix includes relationship between artifacts and entities within a development phase.

To come up with the traceability matrix for a project is a major task. The core of this issue is to build the dependency relationship between different entities and artifacts. For relatively small projects it may be feasible to build these relationships manually, but for large complex systems, we need some level of automation (Ibrahim *et al*., 2005). There are various techniques employed for automation ranging from heuristics, information retrieval, data mining, domain knowledge and this is a hot area in research (Imtiaz *et al.*, 2008). Moreover, the relationship between artifacts and entities can be direct, where an artifact or an entity is directly affected by the change, or indirect, where an artifact or an entity may be affected indirectly due to its relationship to other artifacts and entities that are affected directly by the proposed change (Strens and Sugden, 1996). This phenomenon is sometimes termed as "ripple effect", and the chain of this ripple effect may not be limited to only one level but to a number of levels with different degree of effect (Bohner, 2002, De Lucia *et al.*, 2008, Ramzan and Ikram, 2005).

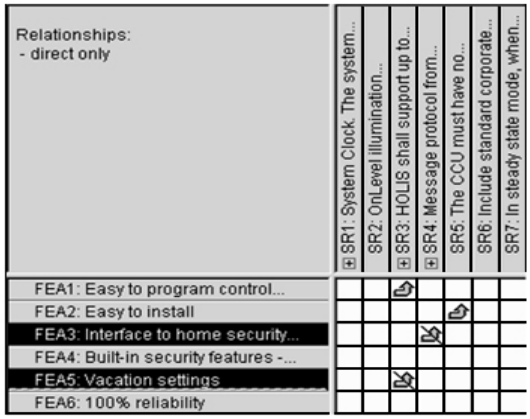


Figure 6. A Sample Traceability Matrix (Leffingwell and Widrig, 2003).

Once the impact analysis is conducted using the traceability matrix, a candidate impact set is obtained which includes all the artifacts and entities affected directly or indirectly by the proposed change. It shall be kept in mind that this impact set is not always accurate in the sense that it may not be able to include all the possible artifacts and entities, and accordingly some of them might be left out (*termed as false negatives*). Additionally, the impact set may include some artifacts and entities which may in reality not be affected by this change (*termed as false positives*) (Bohner, 2002). The effectiveness of a given technique for impact analysis and traceability matrix is mostly judged considering these false positives and false negatives. Figure 7 shows the process of impact analysis. The impact itself may be measured in terms of degree and probability of impact.

The results of impact analysis is further studied by employing cost benefit analysis which not only considers the technical implications of change but also other aspects like management, administrative, cost/budget and project schedule (Imtiaz *et al*., 2008). Once the complete analysis is done, a final decision is made on whether to implement this change request or to put the

change request on hold. It is important that all the major stakeholders participate in this process and be a part of the final decision.

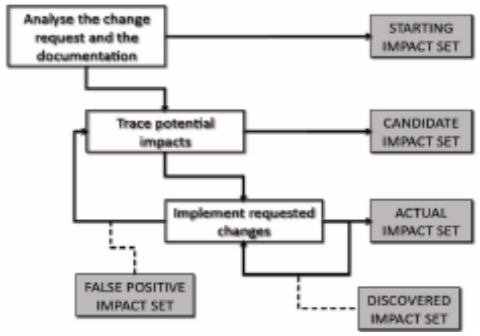


Figure 7. Impact analysis process (De Lucia *et al.* 2008).

It is also recommended that there be a clear process and change control body who takes care of the entire change management process instead of change creeping in from anywhere with no proper control on them. This may adversely affect the schedule, cost, and the final quality of the software product. The concept of having an independent *change control board* is very effective and popular in this respect and is practiced widespread (Leffingwell and Widrig, 2003). There are many requirements change management process proposed in literature and a good comparison on many of them is available in (Ramzan and Ikram, 2005). The process of change management involving change control board is shown in Figure 8. The change control board acts like a firewall by controlling the incoming change and preventing the change creeping in through informal change requests.

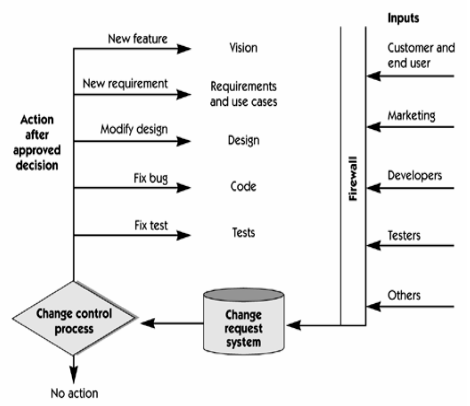


Figure 8. Change management through change control board (Leffingwell and Widrig, 2003).

4.2 RM in Engineering Systems

General engineering changes during a system's lifecycle have been studied intensively in the last decade. A comprehensive review has been reported recently by Jarratt *et al.* (2011). The subject is still under continuous development, hence no complete books have been devoted to the topic. Results have been published mainly in journal or conference papers and a few book chapters. The change process follows the plan given previously in Figure 3, with many iterations and break/stage-gate points.

As seen in Figure 9, the sources of changes throughout the design process are numerous, and can be emergent (i.e. arising from the properties of the system itself), or initiated (for improvements, enhancements, or adaptations of the system). The initiation source can be internal (operational experience, manufacture/assembly, production, build), or external (customer, supplier, contractual), with changes in customer requirements playing a substantial role (Table 4).

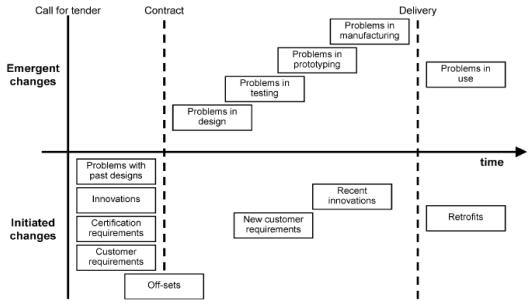
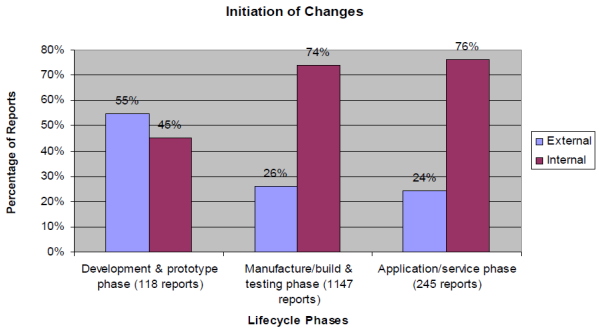


Figure 9. Sources of Change Throughout the Design Process (Eckert *et al.*, 2004).

Table 4. Initiation of changes. From Ahmed and Kanike (2007).



Eckert *et al.* (2004) have identified two types of change propagation: (a) Ending ones, consisting of small ripples of change, which are brought to a conclusion within an expected time frame, and (b) Unending ones, consisting of avalanches of change, typical of "a snowball effect," which occur when a major change initiates several other major changes, and all of these cannot be brought to a satisfactory conclusion.

With innovative systems, changes are much more likely to occur since everything is fuzzy, particularly requirements, at the front end. This is particularly so in hybrid systems governed by both technology push and market pull, as in many emergent energy systems. In theses cases, the best strategy is to isolate the innovation

process within R&D laboratories, where "organized chaos" could be controlled and maintained, until new systems (ideas, materials, processes, etc.) are fully developed and successfully tested.

As noted by Jarratt *et al.* (2011), changes during the design process result in "information deficiencies" for other development teams, whose decisions are then made about the system without up-to-date data (Fricke *et al*., 2000; Rouibah and Caskey, 2003). In order to remedy this situation, and provide help in decision-making, and the overall engineering change process, engineers need well-designed tools for their support. Such tools should help perform many important tasks, namely: track requirements status; communicate with stakeholders; store requirements attributes; manage versions and changes; facilitate impact analysis; control access; reuse requirements. Some of the academic support tools reported in the literature have been discussed in detail by Jarratt *et al.* (2011). A few of these will be highlighted here.

Keller *et al.* (2005, 2007, 2008) developed the Change Prediction Method (CPM) tool, which is a software program for predicting change propagation, by analyzing indirect changes and calculating the combined risk that a change to one component will affect others. The CPM tool makes use of Design Structure Matrices (DSMs) to provide a simple, compact, and visual representation of the probability that a change will propagate from one component to others. The tool can be used in all life-stages of a system, such as in the conceptual design, detail design, and operational life-stages.

In parallel, Koh and Clarkson (2009) presented a modeling method that aims to manage the effects of change propagation, and applied the method to the design of a jet engine fan. The method uses a matrix-based approach to model the dependencies between the solution alternatives, the potential change propagation brought about by the solutions, the affected product attributes, and the resources needed to carry out the change work. The method allows engineers to trace critical change propagation paths and manage them, and hence appears to be suitable for assessing solution alternatives during preliminary design and exploring the design space in the right direction.

Lemmens *et al.* (2007) analyzed the impact of change from different points of view e.g. requirements, physical or functional product architecture, design processes or activities, organization, with the aim of facilitating decision-making by improving the visibility and shared understanding of the interdependencies that exist within and between such viewpoints. Change Propagation Analysis (CPA) algorithms were proposed and implemented in a prototype software environment to allow the modeling and visualization of multiple dependencies across multiple information domains, including lifecycle considerations.

Tracking changes can always give an engineer valuable experience in designing future products. However, this approach is "*reactive*" and only addresses the problem of changing requirements after they have occurred and it is too late to adjust the current design. Designing systems should be "*proactive.*" Designing with "changing requirements in mind" can be especially effective when a customer has only loosely identified requirements or when requirements are not fully known. A few approaches have been proposed in that direction.

Qureshi *et al.* (2006) defined "product flexibility" as the adaptability of a system in response to changing factors. Since flexible products were realized with *ad hoc* methods that rely on the experience and intuition of the designer, they presented a set of formal principles for guiding the design of flexible products. These principles were derived from the results of an empirical study of the United States patent repository. They also validated the effectiveness of these principles using a Change Modes and Effects Analysis (CMEA) tool.

The Design For Variety (DFV) method uses product platform architecture to provide a structured approach to reduce the amount of redesign effort for future generations of a product. For large projects, system architecture can be used to break down the design into smaller subsystems at each level of the design hierarchy (Hintersteiner, 2000). The DFV has the advantage of being a simple and inexpensive technique to determine potential design changes. The methodology *makes use of standardization and modularization techniques* to reduce future design costs and efforts (Martin and Ishii, 2002). The design for variety method develops two indices to measure a product's architecture. The first, called the Generational Variety Index (GVI), is an indicator of the amount of redesign effort required for future iterations of a product. The other is called the Coupling Index (CI), and it is used to gauge the extent of coupling among the different components in a product. DFV can be used to help reduce the impact of variety on the lifecycle costs (Martin and Ishii, 2002).

Peterson *et al.* (2007) identified six product development strategies to cope with changing requirements and specifications. These strategies were tested while developing working product prototypes for their project. These six recommendations are:

- Establish and foster open communication between designers and customers.
- Develop and explicitly write down a complete list of design requirements..
- Analyze the list of requirements to identify which requirements are likely to change and which are stable.
- Predict future market/customer needs and requirement changes.
- Use an iterative approach. Quick turnover of designs and prototypes provides a method for testing requirements and discovering unanticipated ones.
- Build flexibility into the design by selecting product architectures that tolerate changing requirements. This can be achieved by over-designing components that are likely to change to meet future needs.

Freezing requirements is one way designers try to deal with changing requirements. One goal of a freeze is to reduce the likelihood of design changes. The major benefits from using design freezes are the ability to structure the design process and to control design changes (Eger *et al.*, 2005). A design freeze marks the end of a development stage where requirements become fixed before the design can continue (Eger *et al*., 2005). Early design freezes have the benefit of pushing any design changes to future product generations. This can be constructive in an iterative design process. Early design freezes can also force a design before it is beneficial to do so. When the exact requirements are uncertain, it may be advantageous to postpone a design freeze. Some changes due to safety concerns, problem corrections, or altered customer requests will still have to be carried out regardless of whether a component is frozen. Changes after a freeze are likely to be more costly, and the cost will continue to increase the later the change is implemented (Eger *et al*., 2005). Many designers feel it is best to keep parts flexible where changes are anticipated.

Information about design freezes is especially important when working in a team. Recognizing the dependencies between parts and acknowledging which parts may be frozen can avoid inadvertent changes to the overall design.

4.3 Discussion on RM

Some of the methods developed for software engineering to assess the impact of a design change can be applied to engineering systems with some adaptation. As noted by Peterson *et al.* (2007), the difficulty lies in that software design is only concerned with the transmission of information, while engineering systems design must also deal with material and energy transfer.

A couple of models used in software engineering consider changes in evolutionary software development (Schach and Tomer, 2000; Rajlich, 2000). However these models are not appropriate for engineering systems design where component interfaces are not as explicit and involve more than just information transmission. Generally, these programs only identify the immediate implications of change within the immediate sub system and are not capable of exploring the consequences of change propagation through complex systems with different mechanical interactions (Keller *et al.*, 2005).

An important concept that should be carried over from software to engineering systems is the adoption of a socio-technical approach which contends that communication problems can be reduced if all stakeholders are involved in all phases of the design process.

The idea of guessing future changes in software (*Future Analysis*) can also be applied to engineering systems design. A robust design is one that can cope with alternative futures (Land, 1982). In order to build robust systems the designer must attempt to consider all possible alternative futures. The outcome of the analysis of the system for future changes is a list of system features which are likely to be affected. Building flexibility into a system can be beneficial but is often expensive so it is important to determine where best to build flexibility into the system (Land, 1982). The target lifespan will determine how much flexibility the system should have to meet that target.

For both software and engineering systems, commercial software tools are available to handle RM. Some of these tools are compared in Table 5 and Table 6.

Table 5. RM Commercial Tools (Uspenskiy 2004)*.

CHARACTERISTIC	TOOL		
	DOORS	RDD-100	RTM
Requirements Capture	Full	Full	Poor
Multi-user Support	Full	Full	Full
Concurrency Support	Full	Full	Full
Batch Loading	Full	Full	Moderate
Change Management	Full	Full	Full
Version Control	Moderate	Moderate	Full
Interface w/Other Tools	Moderate	Moderate	Full
Multi-platform	Full	Full	Full
Access Control	Full	Full	Full

Note) * Dynamic Object Oriented Requirements System (DOORS) is the product of IBM Telelogic Inc.; Requirements Driven Development-100 (RDD-100) is a product suite of Holagent Corporation; Requirements Traceability Management (RTM) is a product of Integrated Chipware; System Modeling Language MagicDraw +SysML is a product of NoMagic.

Table 6. RM Commercial Tools (McLellan *et al*. 2010)*.

Requirement Capabilities	DOORS	MagicDraw+SysML
Refinement	The user can identify the decomposition schema of the requirements list using the hierarchal nature of DOORS.	A specific directional relationship can be created between requirements and their parent/child requirements.
History	The requirement change is recorded, including the date, by whom and the type of change made.	MagicDraw Team Server can record changes made to requirements. MagicDraw Standard Edition does not support requirement history documentation.
Satisfaction	Relationships between the system structure and the requirement are created using the "link" relationship.	Relationships between the system structure and the requirement are created using the "SatisfiedBy" relationship.
Verification	Relationships between the requirements and the test cases are created using the "link" relationship.	Relationships between the system structure and the requirement are created using the "VerifiedBy" relationship.
Coupling	Relationships between requirements are created using the "link" relationship.	Relationships between requirements are created using the "DeriveReq" relationship.
Prioritization	Requirements can be rated according to priority by creating a priority attribute. Requirement manipulation by sorting and filtering can then be done.	Includes a requirement "risk" attribute that can be treated as a priority attribute since higher risk requirements are of higher priority than lower risk requirements.
Input Validation	Input validation is not supported since there is no structure or standard to test requirements by.	Input validation is not supported since there is no structure or standard to test requirements by.
View Restriction	Viewing rights/limitations can be applied to users through their username privileges	Specialized requirement diagrams can be created with only certain requirements specified. It cannot create access rights for requirements.

Note) * Dynamic Object Oriented Requirements System (DOORS) is the product of IBM Telelogic Inc.; Requirements Driven Development-100 (RDD-100) is a product suite of Holagent Corporation; Requirements Traceability Management (RTM) is a product of Integrated Chipware; System Modeling Language MagicDraw +SysML is a product of NoMagic.

5. CONCLUDING REMARKS

Since requirements development and management are among the most important activities in any software or engineering system project, efforts towards improving these two tasks can always increase and accelerate the ROI. According to the "garbage in, garbage out" rule: If the requirements are not "good" and "properly managed," all subsequent efforts will only help design, make, and market the wrong unneeded product faster**.**

The analysis of the mentioned literature revealed that RE in product engineering is mostly restricted to the early phases of the development process. During the late phase, RE does not seem to play a substantial role. Most of these approaches state that the customer plays a central role during the entire development process. The type and degree of customer integration into the development process varies. The integration of the customer into the process of requirements elicitation is emphasized, but not in later phases.

RE is a well-established discipline in software development. This is clear from the many textbooks that have appeared on the subject and the many courses in curricula of various software engineering programs. In contrast, there are no books dedicated to this subject for designing engineering systems. A special journal issue of *Research in Engineering Design* on "Engineering Change" is forthcoming in 2011 (Eckert *et al.*, 2010).

Table 7. Comparing the Degree of Achievement of Various RD and RM Tasks in Software and Engineering Systems.

ITEM		STATUS*	
		Software	Engineering Systems
RD	Methods for formal specification of requirements	FC	PC
	Methods for requirements elicitation are described	FC	FC
	Tool-support is existing	FC	PC
	Follow all phases of RD	FC	PC
	Consider stakeholders throughout the system lifecycle	FC	PC
	All phases of the innovation process are covered	NC	NC
	Hybrid systems of software and hardware are covered	NC	NC
RM	Changes of the innovation process are covered	FC	FC
	Commercial software tools are available	FC	FC
	All phases of software and hardware are covered	NC	NC
	Hybrid systems of software and hardware are covered	NC	NC

Note) * Status: FC: Fully Covered, PC: Partlally Covered, NC: Not Covered.

Following is Table 7 compares the degree of achievement of various RD and RM tasks in software and in engineering systems.

Both software and engineering systems lack the coverage of RD and RN in complex innovative and/or hybrid applications.

As seen also in the Table, RD and RM are applied in a structured way in software systems, whereas a few areas are still under-developed for engineering systems; further research in that direction is needed.

ACKNOWLEDGMENT

Thanks are extended to the King Fahd University of Petroleum and Minerals for funding the research reported in this paper through the KFUPM Center of Excellence for Scientific Research Collaboration with Massachusetts Institute of Technology (MIT) (Project # MIT 09023).

REFERENCES

Ahmed, S. and Kanike, Y. (2007), Engineering change during a product's lifecycle, In: Bocquet, J-C (ed) *Proceedings of the 16th international conference on engineering design (ICED'07)*, Paris, France, 633-634.

Agouridas, V., Marshall, A., Mckay, A., and Pennington, A. D. (2006), Establishing stakeholder needs for medical devices, *ASME 2006 Int. Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Philadelphia, Pennsylvania, USA.

Aurum, A. and Wohlin, C. (Editors) (2005), *Engineering and Managing Software Requirements*, Springer, Heidelberg.

Blackburn, J. D., Scudder, G. D., and Van Wassenhove, L. N. (1996), Improving Speed and Productivity of Software Development: A Global Survey of Software Developers, *IEEE Trans. on Software Engineering*, **22**(12), 875-885.

Bohner, S. A. (2002), Software change impacts-an evolving perspective, *Proc. Int. Conf. on Software Maintenance*, *IEEE Comput. Soc*, 263-272.

Bourque, P. and Dupuis, R. (2004), *Guide to the Software Engineering Body of Knowledge,* IEEE Computer Society.

Braude, E. (2010), *Software Engineering: Modern Approaches*, 2nd ed., Wiley.

CCWCE (Center for Clean Water and Clean Energy Research) (2008), Online at: http://ccwce.mit.edu.

Cross, N. (1989), *Engineering design methods*, Wiley, Chichester.

Darlington, M. J. and Culley, S. J. (2002), Current research in the engineering design requirement, *Proc Instn Mech Engrs*, *Part B: J Engng Manufacture*, **216**, 375-388.

Darlington, M. J. and Culley, S. J. (2004), A model of factors influencing the design requirement, *Design Studies*, **25**(4), 329-350.

De Lucia, A., Fasano, F., and Oliveto, R. (2008), Traceability management for impact analysis, *Frontiers of Software Maintenance*, *IEEE*, 21-30.

Eckert, C. M., Clarkson, P. J., and Zanker, W. (2004), Change and customization in complex engineering domains, *Res Eng Des*, **15**(1), 1-21.

Eckert, C., Clarkson, J., and de Weck, O. (2010), Call for papers for a special issue on 'Engineering Change,' *Research in Engineering Design.*

Eger, T., Eckert, C. M., and Clarkson, P. J. (2005), The role of design freeze in product development: In Samuel, A., and Lewis, W. (eds) *Proc. 15th int. conf. on engineering design (ICED'05)*, Melbourne, 164-165.

Eleiche, A. M. (2010), Engineering change management in sustainable innovative projects, *APIEMS 2010: The 11th Asia Pacific Industrial Engineering and Management Systems Conference, and The 14th Asia Pacific Regional Meeting of Int. Foundation for Production Research*, 501.

Fricke, E., Gebhard, B., Negele, H., and Igenbergs, E. (2000), Coping with changes: causes, findings and strategies. *Syst Eng*, **3**(4), 169-179.

Gause, D. C. and Weinberg, G. M. (1989), *Exploring Requirements: Quality Before Design*, Dorset House, N. Y.

Giaglis, G. (2001), A Taxonomy of Business Process Modeling and Information Systems Modeling Techniques, *Information Systems*, **13**, 209-228.

Harker, S. D. P., Eason, K. D. and Dobson, J. E. (1993), The change and evolution of requirements as a challenge to the practice of software engineering, *Proc. IEEE International Symposium on Requirements Engineering*, San Diego, CA, USA.

Hintersteiner, J. D. (2000), Addressing Changing Customer Needs by Adapting Design Requirements. In *First International Conference on Axiomatic Design, ICAD2000*, Cambridge, MA, 290-299.

Ibrahim, S., Idris, N. B., Munro, M., and Deraman, A. (2005), Integrating software traceability for change impact analysis, *Int. Arab J. of Information Technology*, **2**(4).

IEEE-Standard-830 (1984), *IEEE Guide to Software Requirements Specifications.*

IEEE (1990), *IEEE Standard glossary of software engineering terminology,* IEEE Std. 610.

IEEE (1998), *IEEE Standard 830: Recommended Practice for Software Requirements Specifications.*

Imtiaz, S., Ikram, N., and Imtiaz, S. (2008), Impact analysis from multiple perspectives: Evaluation of traceability techniques, *The Third International Conference on Software Engineering Advances, IEEE*, 457-464.

Jarratt, T. A. W., Eckert, C. M., and Clarkson, P. J. (2004), Engineering change. In: Clarkson, P. J., and Eckert, C. M. (eds) *Design process improvement*, Springer, N. Y.

Jarratt, T. A. W., Eckert, C. M., Caldwell, N. H. M., and Clarkson, P. J. (2011), Engineering change: an overview and perspective on the literature, *Res Eng Des*, **22**, 103-124.

Jiao, J. and Chen, C.-H. (2006), Customer requirement management in product development: A review of research issues, *Concurrent Engineering: Research and Applications*, **14**(3), 1-25.

Keller, R., Eger, T., Eckert, C. M. and Clarkson, P. J. (2005), Visualising change propagation, *Int. Conf. on Engineering Design*, *ICED05*, Melbourne.

Keller, R., Alink, T., Pfeifer, C., Eckert, C. M., Clarkson, P. J., and Albers, A. (2007), Product models in design: A combined use of two models to assess change risks, *International Conference on Engineering Design*, *ICED*, *Cite des Sciences et de l'Industrie*, *Paris*, *France.*

Keller, R., Eckert, C. M., and Clarkson, P. J. (2008), Through-life change prediction and management. *Proceedings International Conference on Product Lifecycle Management*, *PLM-SP$*, Chapter **3**, 212-221.

Koh, E. C. Y. and Clarkson, P. J. (2009), A modelling method to manage change propagation, *Int. Conf. on Engineering Design*, *ICED Stanford University*, *Stanford*, *CA*, *USA.*

Kotonya, G. and Sommerville, I. (1998), *Requirements Engineering: Processes and Techniques*, Wiley.

Kujala, S. M., Kauppinen, L., and Lehtola, K. T. (2009), The role of user involvement in requirements quality and project success, *Proc. 13th IEEE Int. Conf. on Requirements Engineering*, *Paris*, 75-84.

Lamsweerde, A. (2009), *Requirements Engineering* John Wiley and Sons, Incorporated.

Land, F. (1982), Adapting to Changing User Requirements, *Information and Management*, **5**(2), 59-75.

Larson, A. L. (2000), Sustainable innovation through an entrepreneurship lens, *Business Strategy and the Environment*, **9**, 304-317.

Leffingwell, D. and Widrig, D. (2003), *Managing Software Requirements: A Use Case Approach*, 2nd ed.: Addison Wesley.

Lemmens, Y., Guenov, M., Rutka, A., Coleman, P., and Schmidt-Schaffer, T. (2007), Methods to analyse

the impact of changes in complex engineering systems, *7th AIAA Aviation Technology*, *Integration and Operations Conference (ATIO)*.

Martin, M. V. and Ishii, K. (2002), Design for variety: developing standardized and modularized product platform architectures, *Research in Engineering Design*, **13**, 213-235.

McLellan, J. M., Morkos, B., Mocko, G. G., and Summers, J. S. (2010), Requirement modeling systems for mechanical design: a systematic method for evaluating requirement management tools and languages, Proc. of IDETC/CIE 2010, ASME 2010 international design engineering technical conferences and computers and information in engineering conference, Montreal, Canada, Paper # DETC2010 -28989.

Mathelin, S., Boujut, J.-F., and Tollenaere, M. (2005), Improving collaborative design tools in automotive industry: A case study, *Int. Conf. on Engineering Design*, *ICED'05*, Melbourne.

McGee, S. and Greer, D. (2009), A software requirements change source taxonomy, *Fourth Int. Conf. on Software Engineering Advances, IEEE*, 51-58.

Nurmuliani, N., Zowghi, D., and Williams, S. P. (2004), Using card sorting technique to classify requirements change, *12th International Requirements Engineering Conference (RE)*.

O'Neal, J. S. and Carver, D. L. (2001), Analyzing the impact of changing requirements, *Proc. IEEE Int. Conf. on Software Maintenance*, *ICSM 2001, IEEE Comput. Soc*, 190-195.

Otto, K. and Wood, K. (2001), *Product design: techniques in reverse engineering and new product development,* Prentice Hall, Upper Saddle River.

Pahl, G. and Beitz, W. (1996), *Engineering design-a systematic approach*, 2nd edn. Springer, London.

Peterson, C., Paasch, R. K., Ge, P., and Dietterich, T. G. (2007), Product innovation for interdisciplinary design under changing requirements, *Int. conf. on engineering design*, *ICED*, *Cite Des Sciences et de l'Industrie*, *Paris*, *France.*

Pressman, R. (2005), *Software Engineering: A Practitioner's Approach*, 6th ed.: Mc Graw Hill.

Qureshi, A., Murphy, J. T., Kuchinsky, B., Seepersad, C. C., Wood, K. L. and Jensen, D. D. (2006), Principles of product flexibility. *Proc. IDETC/CIE 2006, ASME 2006 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, *Philadelphia, Pennsylvania*, *USA.*

Rajlich, V. (2000), Modelling software evolution by evolving interoperation graphs, *Ann Softw Eng*, **9**, 235-248.

Ramzan, S. and Ikram, N. (2005), Making decision in requirement change management, *Proc. Int. Conf. on Information and Communication Technologies, IEEE*, 309-312.

Ramzan, S. and Ikram, N. (2006), Requirement change management process models: activities, artifacts and roles, *Proc. IEEE Int. Multitopic Conf., IEEE*, 219-223.

Ross, A. M., Rhodes, D. H., and Hastings, D. E. (2008), Defining changeability: Reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value, *Systems Engineering*, **11**, 246-262.

Rouibah, K. and Caskey, K. R. (2003), Change management in concurrent engineering from a parameter perspective, *Comput Ind*, **50**(1), 15-34.

Schach, S. R. and Tomer, A. (2000), A maintenance-orientated approach to software construction, *J Softw Maint-Res Pract*, **12**(1), 25-45.

SERENA (2011), RTM product overview. Accessed at: www.sciti.com.ar/productos/pdf_serena/RTM%20overview.pdf, on 3 May 2011.

Sommerville, I. (2007), *Software Engineering*, 8th ed.: Addison-Wesley.

Standish Group Report (1995), 'CHAOS,' http://www.projectsmart.co.uk/docs/chaos_report.pdf, Accessed on 1 May 2011.

Strens, M. R. and Sugden, R. C. (1996), Change analysis: a step towards meeting the challenge of changing requirements, *Proc. IEEE Symposium and Workshop on Engineering of Computer-Based Systems*, *IEEE Comput. Soc. Press*, 278-283.

Tavassoli, D. (2009), Ten steps to better requirements management, *IBM white paper*. Accessed on 3 May 2011 at: ftp://ftp.boulder.ibm.com/software/uk/pdf/RAW14059-USEN-00-1.pdf.

Ullman, D. G. (2010), *The mechanical design process*, 4th edn, McGraw-Hill Education, Boston.

Ulrich, K. T. and Eppinger, S. D. (2008), *Product design and development*, 4th edn. McGraw-Hill HE, Boston.

Uspenskiy, D. (2004), Requirements management (RM) tools.

Verganti, R. (1997), Leveraging on systematic learning to manage the early phases of product innovation projects, *R&D Management*, **27**, 377-392

Ward, J., Shefelbine, S., and Clarkson, P. J. (2003), Requirements capture for medical device design. *Int. Conf. on Engineering Design, ICED'03*, Stockholm.

Wiegers, K. E. (2000), When telepathy won't do: Requirements engineering key practices, *Cutter IT Journal*, May 2000, www.processimpact.com/articles/telepathy.html.

Wiegers, K. E. (2003), *Software requirements*, 2nd ed. Microsoft Press.