Requirements Evolution
From Process to Product Oriented Management

Stuart Anderson and Massimo Felici[*]

LFCS, Division of Informatics, The University of Edinburgh
Mayfiled Road, Edinburgh EH9 3JZ, United Kingdom
{soa, mas}@dcs.ed.ac.uk

Abstract. Requirements Evolution represents one of the major problems in developing computer-based systems. Current practice in Requirement Engineering relies on process-oriented methodologies, which lack of product features. The resulting scenario then is a collection of general methodologies, which do not take into account product features that may enhance our ability in monitoring and controlling Requirements Evolution. This paper shows empirical investigations of two industrial case studies. The results point out evolutionary product features and identify an Empirical Framework to analysing Requirements Evolution. This work represents a shift from process to product-oriented management of Requirements Evolution.

1   Introduction

The requirements process is one of the most important of the entire system life cycle. Most of the faults into computer-based systems (the term computer-based systems draws attention to the involvement of human participants in most complex systems) can be traced backup to the early requirements stage when a recovery action can still be cost-effective [4, 5]. Hence the management of requirements is an important activity for Software Process Improvement (SPI). It is furthermore a key action of many maturity models applied in industry (e.g., CMM [16]). Despite this *Requirements Evolution* [1, 2, 10, 11, 18, 23] still remains one of the most critical issues for software organizations and is challenging for research and practice in Requirements Engineering [14, 25].

Current practice in Requirements Engineering [20, 22, 27] relies on general processes (e.g., elicitation, verification, etc.) and methodologies (e.g., standards and templates for specifying requirements) that are adapted according to specific needs by plugging in other general processes (e.g., task analysis) and methodologies (e.g., formal methods). This view can be really articulated and complex depending on the industrial context. Thus within industrial contexts Requirements Engineering practice

is a collection of processes and methodologies driven by stakeholders. The resulting view lacks in taking into account product features that may enhance our ability in engineering requirements for evolving systems. Current practice in engineering requirements demands a shift from process to product-oriented engineering [26]. Unfortunately this shift in Requirements Engineering has not yet been fully registered. The lack of product perspectives in managing Requirements Evolution gives rise to issues degrading both software process and product in terms of dependability. This view becomes even more complex within safety-critical contexts in which evolution can lead to undependable situations [15, 24] causing loss.

Our work is based on investigations of live industrial contexts to devise product-oriented approaches supporting Requirements Evolution. This paper shows a product-oriented analysis of Requirements Evolution in two industrial case studies. The paper is structured as follows. Section 2 introduces the two case studies. The research methodology in Section 3 has driven the empirical analyses presented in Section 4. Section 5 summarises the conclusions and identifies further work.

2   Industrial Case Studies

This section introduces two industrial case studies, an avionics and a smart card case study, which have contributed to our work. The investigation of different industrial case studies is motivated by our research hypotheses. Our work aims to characterise industrial settings in order to identify product-line methodologies [26] for Requirements Evolution. Origins of Requirements Evolution can be identified in stakeholder [18] interaction, which characterises the industrial context, and technical issues [10, 14, 23]. The link among stakeholder interaction, technical issues and Requirements Evolution is still vaguely understood. The analysis of different case studies aims to identify product features that may enhance our ability in monitoring and controlling Requirements Evolution to devise computer-based systems. Our interest is in differences and commonalities across industrial contexts. This aims to define taxonomy of evolution in industrial contexts, which may be characterized by evolving patters for product-lines. This work contributes to identify dependable Requirements Evolution in terms of process and product, that is, dependable process for deploying evolving systems and Requirements Evolution enhancing system dependability. The following subsections introduce the two industrial case studies, which provide interesting aspects due to their stringent requirements (e.g., safety and security) and to their product-line nature.

2.1   An Avionics Safety-Critical Case Study

The description of the avionics case study identifies critical features with respect to evolution. We describe the case study in general terms, because our focus is on methodologies. On the other hand the case study provides practical issues that may interest practitioners and researchers. The following features are identified in the case study.

Safety Requirements. A system safety assessment analyses the system architecture to determine and categorise the failure conditions. Safety related requirements are determined and flowed down to the software and hardware requirements.

Functional and Operational Requirements. The customer provides the system requirements, which contain information needed to describe the functional and operational requirements for the software. This includes timing and memory constraints and accuracy requirements where applicable. Requirements also contain details of inputs and outputs the software has to handle with special reference to those where non-standard data formats are used.

Software Development Process. The bulk of the software development task can be split into two broad areas, that of software design and code and the other of verification. Two main elements serve to complicate the situation. Firstly there are feedback loops occurring due to problems or modifications. Secondly there is an expansion of information down the design chain to the code. As design progresses, the software requirements are partitioned into smaller more manageable items. This fragmentation is also reflected into further down activities (e.g., testing) and deliverables (e.g., code). There is a strict policy for configuration management, which also requires maintaining traceability on the project to ensure that the final code is complete and consistent with its higher-level requirements, and the verification has been performed on the correct standard of the final code. Software verification consists of two main elements testing and review/analysis. At the start of the project, the verification activities have been planed in specific documents detailing the level of verification performed at each phase of the software development.

Product-Line Aspects and Standards. Hardware dependent software requirements arise not from the system requirements directly but from the implementation of those system requirements. This is normally due to the way that the hardware has been designed to meet its requirements and as an indirect result of safety related requirements. The hardware dependent software requirements characterise the specific product-line in terms of hardware constraints and safety requirements. A certification plan for software is the primary means used by the designed authorities to assess whether the software development process proposed is commensurate with the software level proposed. The plan of the case study has been produced according to the specific guidelines in the standard RTCA/DO-178B [21].

Evolution and Maintenance. During the development life cycle, changes and modifications arise causing feedback loops. As the size and the scope of these changes will be different for each modification and for each project, only general guidelines have been defined. For modifications to a certified standard of software the full life cycle process of testing and reviews should be complied with. The extent to which all activities have to be repeated will depend upon the time elapsed since certification. Modifications that are introduced prior certification will be incorporated during the development life cycle.

2.2 A Smart Card Case Study

People use smart card systems in their daily life. Credit cards, Pay-TV systems and GSM cards are some examples of smart card systems. Smart card systems provide interesting case studies of distributed interacting computer-based systems. Behind a simple smart card there is a complex distributed socio-technical infrastructure. Smart card systems are

Real-Time Systems. The transactions of smart card systems occur in real-time with most of the services operating on a 24-hour basis. The availability of smart card systems is fundamental to support business (e.g., e-money) and obtain customer satisfaction.

Interactive Systems. Most of smart card systems operate on demand. The request of any service provided depends on almost random human factors. Operational profiles show that human-computer interaction turns to be one of the critical factors for smart card systems.

Security Systems. Smart card systems often manage confidential information (e.g., bank account, personal information, phone credit, etc.) that need to be protected from malicious attacks.

The considered smart card context is certified according to many quality and security standards. Among these its management process conforms to PRINCE2. PRINCE[1] (PRojects IN Controlled Environments) is a structured method for project management [6]. It is used extensively by the UK Government and is widely recognized and used in the private sector, both in the UK and internationally. PRINCE2 is a process-based project management approach integrating a product-based project planning. The investigation of the smart card context aims to identify general aspects in requirements management. We do intend neither to validate any particular methodology nor to assess the specific industrial context. Our aim is to identify general practical aspects that can improve our ability in dealing with Requirements Evolution.

3 Empirical Research Methodology

Our research methodology consists of two main steps named *Empirical Analysis* and *Product-oriented Refinement*. The Empirical Analysis aims to gather information from the industrial case studies by analyses of industrial data (e.g., requirements document, data repository). The Product-oriented Refinement is to focus our empirical methodology in order to gather further information by product-oriented analyses.

The analyses consist of both qualitative and quantitative approaches. The qualitative approaches consist of stakeholder interviews focusing on requirements

---

[1] PRINCE is a registered trademark of CCTA (Central Computer and Telecommunications Agency).

engineering practice (e.g., requirements management policy) in the industrial context and product features (e.g., requirements evolution). Stakeholder interviews are also integrated with a Requirements Engineering Questionnaire [3] consisting of 152 questions organized in terms of *Business*, *Process* and *Product* viewpoints. Qualitative analyses are furthermore performed on documents (e.g., requirements documents, history of changes, repository, etc.) by inspections [9]. Finally, quantitative approaches (e.g., Software Metrics [8]) aim to identify additional information and to support empirical results obtained by qualitative analyses.

We perform incremental empirical analyses and product-oriented refinements, that is, the empirical analyses are incrementally conducted by product-oriented refinements. The better our understanding of the case study, the better our ability in defining product-oriented analyses. The empirical results point out which product-oriented refinements to be implemented in subsequent analyses.

4 Empirical Analyses

This section shows the results of the empirical analyses of the case studies. The analyses investigate different aspects of the case studies depending on the industrial context and the available data. The analysis of the avionics case study aims to identify product features characterising Requirements Evolution, whereas the analysis of the smart card case study aims to assess viewpoints for Requirements Evolution management.

4.1 Avionics Case Study

The empirical investigation of the avionics case study is based on analyses of data repositories of requirements evolution. The aim is to identify requirements properties [1, 2] that may enhance our ability in controlling Requirements Evolution [1]. The investigation goes from a general viewpoint towards a product-oriented viewpoint. The incremental investigation is summarised in what follows.

General Requirements Evolution. Figure 1 shows the total number of requirements changes, i.e., added, deleted and modified requirements, over the 22 software releases[2]. The trend of requirements changes does not give enough information about evolutionary features, but it emphasises the problem of Requirements Evolution. The analysis of Requirements Evolution points out that requirements changes are not uniformly spread out over the three basic changes (i.e., added, deleted and modified requirements), in fact the total number of requirements constantly increases over the software releases. This is because the requirements become clearer to the stakeholders, who split complex requirements into smaller and more precisely stated requirements. Another reason is that new requirements arise during the progress of the project, because there are requirements that cannot be defined at the beginning due to

---

[2] There is a correspondence one-to-one between versions of the requirements specification and software releases.

lack of information. Finally, design, implementation and testing activities provide additional feedback to the requirements.
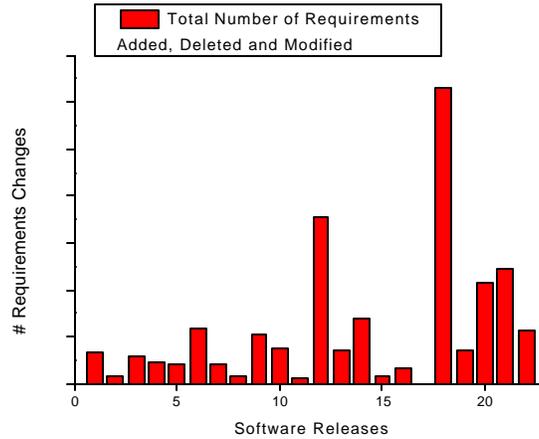

Fig. 1. Number of requirements changes per software release

Taxonomy of Requirements Evolution. The inspection of the history of changes points out the specific changes occurring into requirements. Table 1 shows the taxonomy of requirements changes in the case study. Changes affect requirements attributes like variables, functionality, explanation, traceability and dependency. These attributes are easily collected and represented by requirements templates (e.g., [19, 20]). These attributes are usually embedded within paragraphs specifying requirements. A structured way of representing, collecting and organising requirements attributes may be useful to identify information for controlling and monitoring Requirements Evolution.

Table 1. Taxonomy of requirements evolution

Type of Changes		
General	**Domain Specific**	**Product Line**
− Add, delete and modify requirements − Explanation − Rewording − Traceability	− Non-compliance − Partial compliance	− Hardware modification − Variables range modification − Add, delete, rename parameters/variables

Changes fall into three main classes named *General*, *Domain Specific* and *Product Line*. This rough classification may help to identify requirements issues. For instance, if changes overlap two categories, the affected requirements may need to be refined in order to fit in one category. This may identify different evolving paths, e.g., splitting

requirements in smaller and more detailed requirements or clarify (i.e. modify) their specifications. This is the case when there are requirements overlapping software and hardware. The decision whether to allocate requirements to software or hardware may be delayed till there is a clear understanding of the system. Hence taxonomy of Requirements Evolution may classify not only an industrial context, but it can be used as a tool during design to identify requirements issues.

Requirements Evolution Measurement. Metrics [8] may be used to quantify some properties monitoring Requirements Evolution. The standard IEEE 982 [12, 13] suggests a Software Maturity Index to quantify the readiness of a software product. The Software Maturity Index can be used for software requirements, hence a Requirements Maturity Index (RMI) to quantify the readiness of requirements. Equation (1) defines the RMI[3].

$$RMI = \frac{R_T - R_C}{R_T} \tag{1}$$

Figure 2 shows the RMI calculated for all the software functional requirements. In this case the RMI results to be misleading to assess the readiness of the software functional requirements. The RMI does not have an increasing regular trend. Hence any assessment based only on the RMI could be misleading and risky. This result points out that it is not obvious how to apply even widely used software metrics. Metrics that are suitable at the software level can became unusable at the requirements level.

Fig. 2. Requirements Maturity Index

---

[3] R_T quantifies the number of software requirements in the current delivery; R_C quantifies the number of software requirements in the current delivery that are added, deleted or modified from a previous delivery.

Functional Requirements Evolution. To provide a more detailed analysis our focus moves from the total number of requirements changes to the number of requirements changes in each function that forms the software functional requirements. The software functional requirements fall into 8 functions for which separate documents are maintained. Figure 3 shows the trend of the cumulative number of requirements changes for each function. The figure points out that the likelihood that changes can occur into specific functions is not constant over the software releases. An outcome of this functional analysis is that the function F1 is not likely to change, therefore, it could be considered a stable part of the system. This aspect becomes interesting, because the specific function describes the hardware architecture of the system onto which the software architecture is mapped. It seems furthermore that functions that are likely to change during early software releases change less during later releases, and vice versa. This aspect helps to relate requirements changes with the software life cycle. The different occurrences of requirements changes throughout the life cycle points out some dependencies among functional requirements. Understanding these dependencies may improve the requirements process.
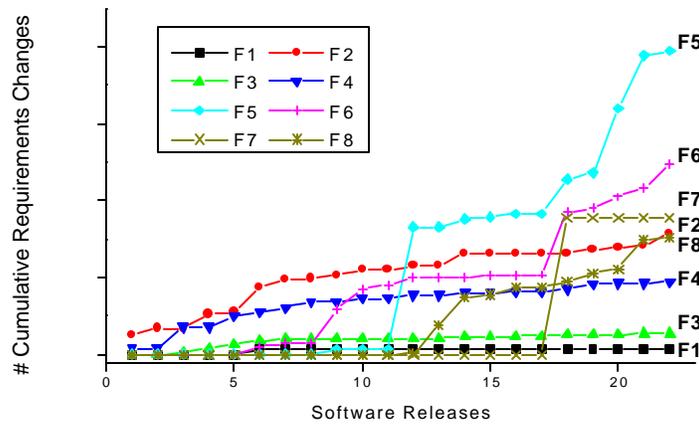

Fig. 3. Cumulative number of requirements changes for each function

Requirements Dependencies. The analysis of Requirements Evolution per software function points out some dependencies among functions. We have evaluated the dependencies among software functions by the number of common fault reports arose during the software life cycle. The number of fault reports overlapping pairwise functional requirements has quantified their dependencies. The underlying hypothesis implies that if two functions are modified due to the same fault report, then there are some dependencies between them. Table 2 shows the dependencies matrix that has been obtained according the above assumption.

We take the number of fault reports in common between two functions as *Dependency Index* between the corresponding requirements. For example, the Dependency Index between the functions F4 and F8 is 5, which means that there have

been 5 fault reports in common between F4 and F8. The blank entries in Tab. 2 mean that there are not common fault reports between the corresponding crossing functions. The requirements dependencies matrix, Table 2, gives a practical tool to assess to which extent software functional requirements depend each other. Moreover it identifies those particular fault reports that trigger changes into different functions. Thus an analysis of such identified fault reports may give important information about requirements. In order to provide feedback into the software organisation the matrix may be analysed to assess the ability of the organisation in identifying software functional requirements for a system or a series of systems for a particular product-line. A tool as the dependencies matrix supports software product-line engineering [26]. Future refinements over product-line projects may identify an effective set of modular system functions such that to reduce disturbing dependencies (e.g., Dependency Index equals to 1). Moreover the Dependency Index may be used to refine impact changes estimates based on traceability.

Table 2. Requirements dependencies matrix

F1	F1							
F2	2	F2						
F3		3	F3					
F4	3	1	1	F4				
F5	1	4	2	6	F5			
F6				1	1	F6		
F7				1	1	1	F7	
F8	1	4	3	5	9	2		F8

4.2 Smart Card Case Study

The analysis of the smart card case study focuses on requirements viewpoints [22]. Viewpoints provide different perspectives to manage Requirements Evolution. The case study provides the opportunity to identify hierarchical viewpoints within the organisation to which correspond different levels of management and different requirements. Viewpoints analysis furthermore points out both process and product divergences, which can be used for further investigations. The investigation analysis is based on interviews and questionnaires [3].

Requirements Evolution Viewpoints. The analysis of the smart card organisation points out three different viewpoints named *Business*, *Process* and *Product Viewpoints*. They correspond to different management levels and responsibilities within the organisation. Each level corresponds to different processes and different requirements. All the three viewpoints together contribute to the overall management of Requirements Evolution.

The business viewpoint is associated to a high management level within the organisation. This level is where projects are originated and the interaction between the organisation and customers takes part. Customer requirements are integrated together with general requirements to complete the smart card system requirements.

Requirements are then negotiated between the customer and the bureau (i.e., the department responsible for the spin-off of smart card projects). It provides production constraints (i.e., additional requirements). When customer and bureau agree on the system requirements the project is declared LIVE, that is, from this moment onward the production of the smart cards starts. The production is organised in terms of subsequent deliveries. During the production of each delivery new requirements may arise due to feedback from users of the new smart cards (e.g., misbehaves, request of new services, etc.) or to business constraints (e.g., production issues, request of additional cards, etc.). The business viewpoint therefore deals with the management of system and business requirements, which are not directly related to software requirements.

The process viewpoint consists of all the management processes adopted within the organisation [6]. Every time a request of change arises the process for changes management starts. The initial part of the changes management process is a macro-process of the negotiation activity. If changes require some software development a set of analyses is performed. Each of these analyses corresponds to a different subsystem consisting of a part of the whole smart card system. For each subsystem an impact report is produced estimating also the cost of changes in terms of man-day. The set of impact reports serves as basis for the negotiation of changes. The agreed software changes are given as input to the software development life cycle, which represents the gate to another viewpoint, namely Product, in the organisation.

The product viewpoint identifies the level producing software embedded in the smart card system. Here the software follows its own development process (i.e., a V model [17]). Software requirements are elaborated through the process and requirements changes are allocated to subsequent releases of the software behind the smart card system. At this level software changes are drawn down through the development process. Differently from the Process viewpoint that estimates changes in terms of man-day, the Product viewpoint takes into account software changes in terms of activities (e.g., coding and testing).

Hierarchical Requirements Evolution. The three viewpoints identified within the organisation represent a hierarchy through Requirements Evolution propagates. This hierarchy identifies different types of requirements and their granularity. Despite this hierarchical structure current methodologies in requirements engineering flatly capture requirements viewpoints [22].

The interviews point out to seek different abilities to support different viewpoints. Different responsibilities seek different types of support. The Business viewpoint needs to support project visibility. Most of the process-oriented methodologies in Software Engineering allow better to plan project activities, but they do not completely clarify the link between software features and project activities. This motivates a shift from process to product-oriented software engineering [26]. The Process viewpoint seeks support to enhance its management ability by measuring (requirements) evolution. The management process registers requirements changes, but a quantitative approach to measure Requirements Evolution needs to be identified within the specific environment.

The Product viewpoint would like to enhance its ability in identifying reusable (product-line) functions and repeatable processes to allocate functions to smart card system requirements. At this level there exist two different opponent processes. Good

practice in Software Engineering addresses that requirements are divided into smaller and more manageable items. This triggers an information flow expansion throughout the development process. On the other hand specific functions would be allocated to software requirements according also to past experience. The gap between these two processes represents the extent to which an organisation is able to identify an optimal and effective set of software functions. The smaller the gap, the better the ability in reusing software functions and identifying product-line features.

Viewpoints Analysis. The empirical analysis also investigates Requirements Engineering viewpoints by a questionnaire [3] to assess the general understanding of the requirements process within the organisation. The questionnaire consists of 152 questions grouped according the categories in Fig. 4.

1. Requirements Management Compliance	10. Requirements Description
2. Business Tolerance Requirements	11. System Modelling
3. Business Performance Requirements	12. Functional Requirements
4. Requirements Elicitation	13. Non Functional Requirements
5. Requirements Analysis Negotiation	14. Portability Requirements
6. Requirements Validation	15. System Interface
7. Requirements Management	16. Requirements Viewpoints
8. Requirements Evolution & Maintenance	17. Product-Line Requirements
9. Requirements Process Deliverables	18. Failure Impact Requirements

Fig. 4. The groups of requirements engineering questions

People with different responsibilities within the organisation filled in the questionnaire. Figure 5 shows the profiles of the questionnaire for two persons with similar experience and with different responsibilities within the organisation. They correspond to two different management levels within the organization and are respectively associated to the Product and Process viewpoints. The questionnaire captures how the requirements process is interpreted from different viewpoints within the organisation.
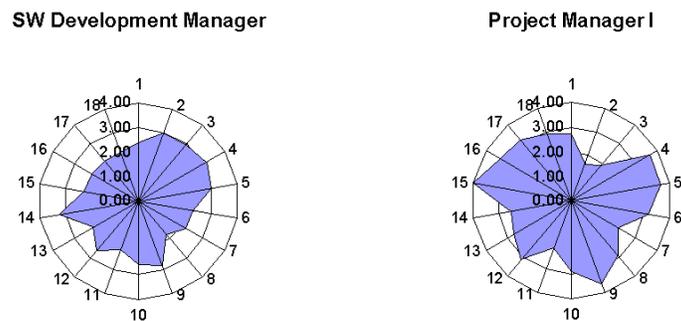
SW Development Manager   **Project Manager I**

Fig. 5. Comparison of two different requirements engineering viewpoints

Figure 6 compares two similar viewpoints (i.e., the viewpoint associated to the project manager) with a third viewpoint (i.e., the viewpoint associated to the software development manager) to take into account some bias in the analysis. The similar viewpoints have similar trends, whereas there are some divergences between the different viewpoints. The largest ones are those associated with the groups of questions 2, 3, 8, 15, 16 and 17 (see Fig. 4). The group 2 and 3 identify business aspects of the requirements process. The distance between the answers to the questions in the group 8 points out that there are different levels of confidence in the management of requirement evolution. This is probably because the management of changes takes into account process aspects that better fit the process viewpoint. This is also due to some issues in transmitting requirements changes through the management hierarchy within the organisation. The groups 15, 16 and 17 identify product-oriented questions. The divergences might be due to the fact that the two viewpoints deal with different requirements. The process viewpoint deals with system requirements, which are different from the software requirements taken into account at the product level. The software embedded in a smart card system represent one aspect, which is not completely visible at the process level. These divergences identify product-oriented refinements for further investigations. They furthermore represent awareness for the organisation. Issues arising from these divergences may cause undependable software and process degradation.
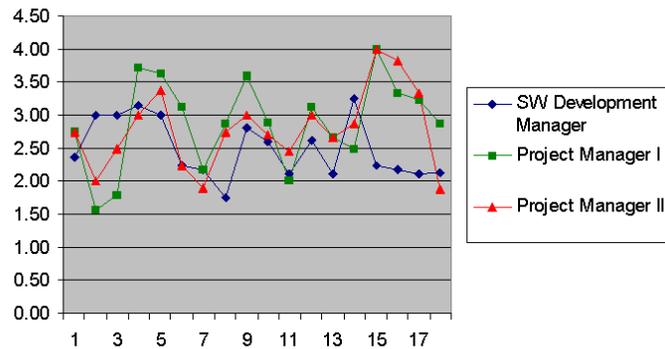

Fig. 6. Comparison of viewpoints

5 Conclusions and Further Work

This paper shows two empirical analyses of two industrial case studies respectively. Our incremental research methodology aims to identify product-oriented empirical analyses. The complexity of Requirements Evolution does not allow adopting general techniques or methodologies, moreover current practice in Requirements Engineering

has not yet fully registered a shift from process to product-oriented approaches. This work tackles Requirements Evolution from a product point of view.

The analysis of the avionics case study points out product evolutionary features that improve our understanding of Requirements Evolution. The analysis gathers information starting from a general analysis of requirements evolution towards a product-oriented analysis via taxonomy of Requirements Evolution and measurement. The analysis furthermore identifies general practical issues and remarks that may be investigated in other industrial contexts. The analysis of the smart card case study identifies viewpoints within the organisation. These viewpoints represent not only sources of requirements and evolutions, but also a management hierarchy for Requirements Evolution. Viewpoints analysis may also identify inconsistencies in the requirements process and areas for further investigations. Figure 7 shows the steps of the two analyses. The joining of the two analyses identifies an Empirical Framework to analyse Requirements Evolution within industrial contexts. The framework consists of analyses that can be repeated within industrial contexts with affordable costs.
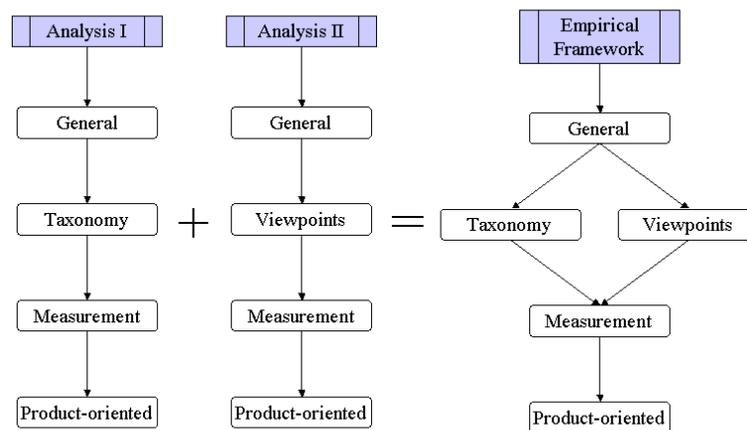

Fig. 7. Empirical framework for Requirements Evolution

Further work aims to validate the entire framework in other empirical investigations of industrial case studies. The empirical framework can then be used to analyse industrial contexts to enhance our ability in managing Requirements Evolution. On the other hand the framework can be reversed, in the sense that it can identify an effective way to structuring Requirements Evolution. This aspect will be assessed by further empirical analysis also supported by formal modelling.

In conclusion, this work identifies an empirical framework for the analysis of Requirements Evolution. The framework represents a valuable tool to implement a feedback into software process and product. Such feedback is the basis for a continuous software process improvement. The empirical nature of the work allows industry to benefit of our experience and studies.

References

1. Stuart Anderson and Massimo Felici. Controlling requirements evolution: An avionics case study. In Proceedings of SAFECOMP 2000, 19th International Conference on Computer Safety, Reliability and Security, LNCS 1943, pages 361-370, Rotterdam, The Netherlands, October 2000. Springer-Verlag.
2. Stuart Anderson and Massimo Felici. Requirements changes risk/cost analyses: An avionics case study. In M.P. Cottam, D.W. Harvey, R.P. Pape, and J. Tait, editors, Foresight and Precaution, Proceedings of ESREL 2000, SARS and SRA-EUROPE Annual Conference, volume 2, pages 921-925, Edinburgh, Scotland, United Kingdom, May 2000.
3. Stuart Anderson and Massimo Felici. Requirements engineering questionnaire, version 1.0, January 2001.
4. Barry W. Boehm. Software Engineering Economics. Prentice-Hall, 1981.
5. Barry W. Boehm. Software engineering economics. IEEE Transaction on Software Engineering, 10(1):4-21, January 1984.
6. CCTA, editor. Prince2 Manual - Managing successful projects with PRINCE 2. CCTA - Central Computer and Telecommunications Agency, 1998.
7. DIRC Project. Interdisciplinary research collaboration in dependability of computer-based systems. http://www.dirc.org.uk/.
8. Norman E. Fenton and Shari Lawrence Pfleeger. Software Metrics: A Rigorous and Practical Approach. International Thomson Computer Press, second edition, 1996.
9. Tom Gilb and Dorothy Graham. Software Inspection. Addison-Wesley, 1993.
10. Theodore F. Hammer, Leonore L. Huffman, and Linda H. Rosenberg. Doing requirements right the first time. CROSSTALK The Journal of Defense Software Engineering, pages 20-25, December 1998.
11. Ivy F. Hooks and Kristin A. Farry. Customer-Centered Products: Creating Successful Products Through Smart Requirements Management. AMACOM, 2001.
12. IEEE. IEEE Std 982.1 - IEEE Standard Dictionary of Measures to Produce Reliable Software, 1988.
13. IEEE. IEEE Std 982.2 - IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software, 1988.
14. M.M. Lehman. Software's future: Managing evolution. IEEE Software, pages 40-44, Jan-Feb 1998.
15. Nancy G. Leveson. SAFEWARE: System Safety and Computers. Addison-Wesley, 1995.
16. Mark C. Paulk et al. Key practices of the capability maturity model, version 1.1. Technical Report CMU/SEI-93-025, Software Engineering Institute, Carnegie Mellon University, February 1993.
17. Shari Lawrence Pfleeger. Software Engineering: Theory and Practice. Prentice-Hall, 1998.
18. PROTEUS Project. Meeting the challenge of changing requirements. Deliverable 1.3, Centre for Software Reliability, University of Newcastle upon Tyne, June 1996.
19. James Robertson and Suzanne Robertson. Volere: Requirements specification template. Technical Report Edition 6.1, Atlantic Systems Guild, 2000.

20. Suzanne Robertson and James Robertson. Mastering the Requirements Process. Addison-Wesley, 1999.
21. RTCA. DO-178B Software Considerations in Airborne Systems and Equipment Certification, 1992.
22. Ian Sommerville and Pete Sawyer. Requirements Engineering: A Good Practice Guide. John Wiley & Sons, 1997.
23. George Stark, Al Skillicorn, and Ryan Ameele. An examination of the effects of requirements changes on software releases. CROSSTALK The Journal of Defense Software Engineering, pages 11-16, December 1998.
24. Neil Storey. Safety-Critical Computer Systems. Addison-Wesley, 1996.
25. Axel van Lamsweerde. Requirements engineering in the year 00: A research perspective. In Proceedings of the 2000 International Conference on Software Engineering (ICSE'2000), pages 5-19, Limerick, Ireland, June 2000.
26. David M. Weiss and Chi Tau Robert Lai. Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley, 1999.
27. Karl Eugene Wiegers. Software Requirements. Microsoft Press, 1999.