Quantitative Aspects of Requirements Evolution

Stuart Anderson   Massimo Felici*

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

Abstract

Requirements Evolution is considered one of the most critical issues in developing computer-based systems. Despite the recognised role of requirements in Software Engineering, Requirements Evolution still remains a phenomenon little understood from both quantitative and qualitative perspectives. This paper investigates our ability to understand the Requirements Evolution Process by means of metrics. The empirical investigation of an avionics industrial safety-critical case study assesses our ability to monitor Requirements Evolution by evolutionary trends and the Requirements Maturity Index (RMI). We propose an enrichment of the RMI in order to take into account process aspects and history of changes. The empirical results support the models underlying the proposed metrics. The empirical nature of this work allows to replicate the experiment in other industrial contexts and to benefit of our results.

1. Introduction

Requirements Evolution [5, 6, 7, 13, 18, 19] is considered one of the most critical issues in developing computer-based systems. Requirements Evolution may cause increased cost as well as system degradation. This phenomenon arises in any industrial context developing computer-based systems. Requirements Evolution is due to both social and technical aspects. The social viewpoint is related to the stakeholders involved in the system, they range from end-users to software engineers, project managers and other business actors (e.g., standards regulators, market competitors, etc.). All stakeholders change their understanding of the ongoing system during its life cycle, hence requirements evolve. On the technical viewpoint, requirements may evolve due to production constraints, usage experience and feedback from other phases of the system life cycle (e.g., testing). Despite the recognised phenomenon there are few methodologies, available in practice, to understand Requirements Evolution. Requirements Engineering [14, 17, 20] relies on process oriented activities to manage evolution. Thus Requirements Evolution has been mainly tackled as a management problem and not as an engineering feature. Related research in engineering requirements [6, 13] and software [10, 11, 12, 16] shows promising results in tackling evolution as a natural feature instead of an issue to be solved and eliminated.

Accepting evolution as an inherent feature of requirements changes our view of Requirements Evolution. This paper provides a quantitative analysis of a safety-critical case study drawn from the avionics industry. The analysis assesses our ability to monitor Requirements Evolution by evolutionary trends and the Requirements Maturity Index (RMI) [8, 9]. We propose an enrichment of the RMI in order to take into account process aspects and history of changes. The proposed metrics are evaluated on the avionics case study dataset. The empirical results support the models underlying the proposed metrics. This paper is structured as follows. Section 2 introduces the case study. Section 3 provides an empirical investigation of the case study. These results provide input to the modelling in Section 4. Section 5 evaluates the modelling against the dataset. Section 6 summarises our conclusions and identifies further work.

2 An Avionics Safety-Critical Case Study

The description of the avionics case study identifies critical features with respect to evolution. The data we have covers the development of the software that meets the requirements. Each revision of the requirements is matched by a corresponding software release and the development process is a major driver for requirements evolution.

Safety Requirements. A system safety assessment anal-

yses 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: software design and code; and verification. Two main elements complicate the situation. Firstly there are feedback loops created by 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 in later activities (e.g., testing) and deliverables (e.g., code). There is a strict policy for configuration management, which also maintains traceability to ensure that the final code is complete and consistent with its higher level requirements, and that verification has been performed to the correct standard in the final code.

Product Line Aspects and Standards. Hardware dependent software requirements arise not from the system requirements directly but from the implementation of those system requirements. The hardware dependent software requirements characterised the specific product-line in terms of hardware constraints and safety requirements. A certification plan for the software ot the case study has been produced according to the specific guidelines in the standard RTCA/DO-178B [15].

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

3 Measuring Requirements Evolution I

The initial empirical investigation [1, 2, 3] of the avionics case study is based on analyses of data repositories constructed to demonstrate traceability. Figure 1 shows the total number of requirements changes, i.e., added, deleted and modified requirements, over 22 software releases.
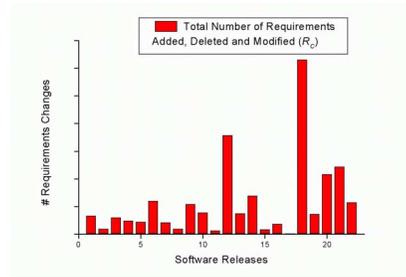

Figure 1. Number of requirements changes.

The analysis of requirements evolution points out that requirements changes are not uniformly distributed over the three types of change (i.e., added, deleted and modified requirements). Figure 2 shows the total number of requirements constantly increases over the software releases.
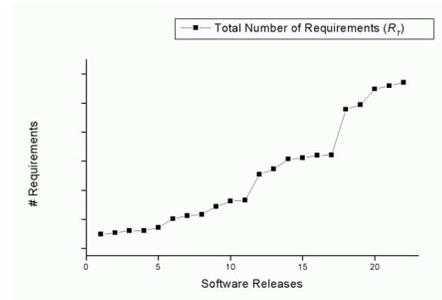

Figure 2. Total number of requirements.

Metrics [4] may be used to quantify properties of requirements evolution. The standard IEEE 982 [8, 9] 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[1].

$$RMI = \frac{R_T - R_C}{R_T} \; .　\qquad (1)$$

Figure 3 shows the RMI calculated for each software release with respect to the total number of requirements (R_T)

---

[1] R_T is the total number of software requirements in the current release; R_C is the number of software requirements in the current release that are added, deleted or modified from the previous one.

and the total number of changes (R_C). The RMI is sensitive to requirements change in successive releases, but it does not take into account historical information about change. Because most requirements change is due to added requirements we believe that the RMI is too pessimistic.
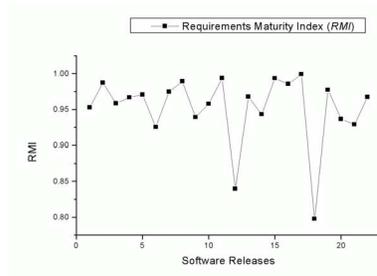

Figure 3. Requirements Maturity Index.

The software functional requirements fall into 8 functions for which separate documents are maintained. Figure 4 shows the scatter plot of number of cumulative requirements changes against the size of each function (at the 22nd release) in terms of number of requirements.
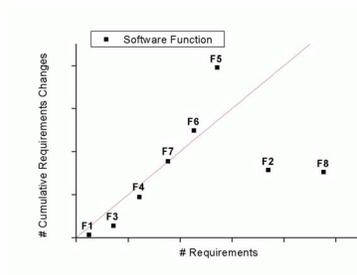

Figure 4. Cumulative number of requirements changes against the number of requirements.

Figure 4 shows that there is a linear relationship between the number of changes occurring in a requirements specification and its size. But there are outliers such as, e.g., F2, F5 and F8. Moreover, Figure 4 illustrates the intuitive interpretation that the functions above the diagonal dividing the graph are more unstable than that ones under the diagonal. In particular F1 turns to be a stable part of the system. This is interesting, because the specific function describes the hardware architecture of the system onto which the software architecture is mapped.

Figure 5 shows the different distribution of requirements changes (in percentage of the total number of changes occurred in the corresponding release) for three functions.
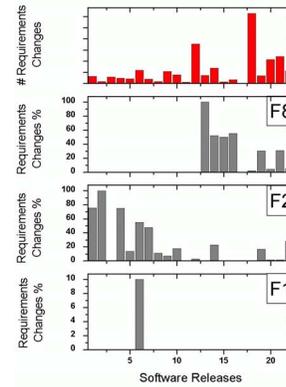

Figure 5. Distributions of requirements changes for three functions.

The different distributions point out some interesting intuitive properties: The likelihood that changes occur into specific functions is not constant over the software releases; Functions that are likely to change in early software releases change less during later releases, and vice versa. These aspects help to relate requirements change to the software life cycle. The different distribution of requirements change throughout the life cycle points to dependencies between functional requirements. Understanding these dependencies may improve the requirements process by improving the impact analysis of changes.

4 Requirements Evolution Modelling

This section introduces our empirical model of Requirements Evolution and its process. Requirements Evolution is modelled according to three different aspects aiming to classify, structure and quantify evolutionary features.

Taxonomy of Requirements Evolution. The inspection of the history of changes points out the specific changes in requirements. Table 1 shows a taxonomy of Requirements Evolution in the case study. Changes affect requirements attributes like variables, functionality, explanation, traceability and dependency. A structured way of representing, collecting and organising requirements attributes may be useful to identify information for controlling and monitoring requirements evolution. The taxonomy of Requirements Evolution 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 clarifying (i.e., modify) their specifications. Hence a 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.

Table 1. Type of change.

Type of Change	Description
Add, Delete and Modify requirements	Requirements are changed due to the specification process maturity and knowledge.
Explanation	The paragraphs that refer to a specific requirement are changed for clarity.
Rewording	The requirements itself does not change, but it is rephrased for clarity.
Traceability	The traceability links to other deliverables are changed.
Non-compliance	A requirement that is not applicable for a new software package. This is the case when the requirements specification is based on that one of a previous project.
Partial compliance	A requirement that is applicable partially for a new software package. This is the case when the requirements specification is based on that one of a previous project.
Hardware modification	Several changes are due to hardware modifications. This type of change applies usually to hardware dependent software requirements.
Range modification	The range of the variables within the scope of a specific requirements is modified.
Add, Delete, Rename parameters/variables	The variables/parameters to which a specific requirement refers can change.

Structuring Requirements Evolution. The backward reconstruction of the history of change identifies a structured work flow of Requirements Evolution. We introduce a graphical model that has been identified by reasoning on data repository of the case study and by inspecting the requirements documents and their respective history of changes. The graphical model shows a representation of Requirements Evolution in terms of requirements changes (i.e., added, deleted and modified requirements) together with a software life cycle perspective. Requirements are organised in a document or a collection of documents (one per each function) in which all requirements are uniquely identified by a numeric id. Figure 6 shows a graphical work flow representation of the three operations to change requirements.

Add: $Add[n]$ introduces a subset of n new contiguous requirements ordered according to their index into the current requirements document.

Delete: $Del[n]$ deletes a subset of n contiguous requirements from the current requirements document.

Modify: $Mod[n]$ modifies a subset of n contiguous requirements into the current requirements document.
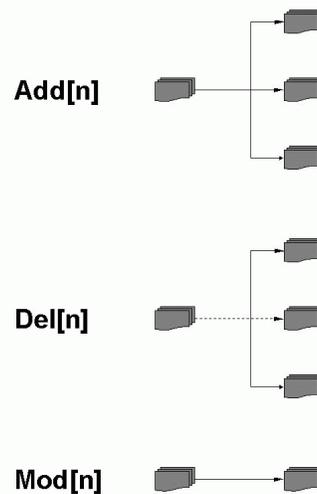

Figure 6. Operations to change requirements.

Here n represents the size of the subset changed, whereas the specific changed requirements are identified by the pointed subset in the middle of the three. The subset of n modified requirements is not identified by the graphical representation because the Mod operation does not actually change the structure of the requirements document.

At this stage the graphical model aims to capture the Requirements Evolution Process and its complexity intuitively. Figure 7 shows the representation of the work flow evolution for the function F1.
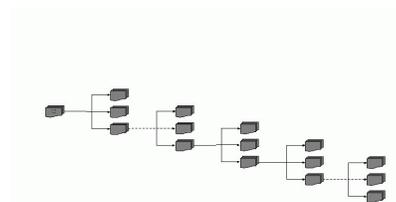

Figure 7. Requirements evolution for F1.

The shape (even without detailed information) of the requirement evolution captures the complexity of the Requirements Evolution process.

Measuring Requirements Evolution. The behavioural analysis of the RMI on the dataset points out that the RMI is not sensitive to the history of change. We propose a refinement of the RMI by taking into account historical evolutionary information and not just the requirements change

since the previous release. The simplest historical information consists of the *Cumulative Number of Requirements Changes* (CR_C) and the *Average Number of Requirements Changes* (AR_C), Eq. 2, over the software releases.

$$AR_C = \frac{CR_C}{n} \ . \qquad (2)$$

Based on the above information we propose two refinements of the RMI. The first refinement is named *Requirements Stability Index* (RSI), Eq. 3. The RSI is sensitive not just to the total number of requirements, R_T, but also to the cumulative number of changes, CR_C. The RSI can be also negative in cases where there have been more changes than the total number of requirements, R_T.

$$RSI = \frac{R_T - CR_C}{R_T} \ . \qquad (3)$$

The second refinement is named *Historical Requirements Maturity Index* ($HRMI$), Eq. 4. It is defined on the total number of requirements and the average distribution of requirements change over the software releases (AR_C).

$$HRMI = \frac{R_T - AR_C}{R_T} \ . \qquad (4)$$

Figure 8 shows the simulation of the above metrics on a sample case.
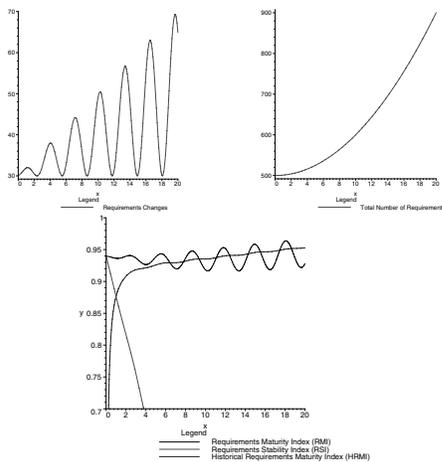

Figure 8. Evolutionary Scenario.

The $HRMI$ is less sensitive and more stable than the RMI, which is more sensitive to the number of changes occurring release after release.

5 Measuring Requirements Evolution II

This section assesses our proposed quantitative models of Requirements Evolution using the case study. Figure 9 shows the average number of changes over the software releases. The graph shows clearly an increasing trend.
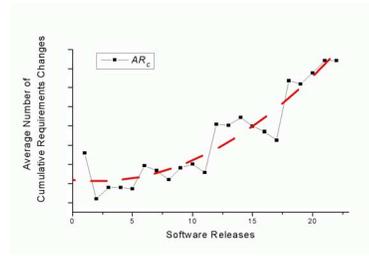

Figure 9. Average number of requirements changes.

As we would expect the RSI has a decreasing trend. It is more interesting to analyse the RSI at the last release (i.e., the 22nd release) of the case study, Figure 10.
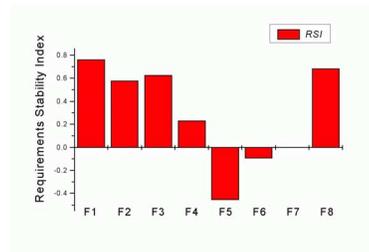

Figure 10. Requirements Stability Index.

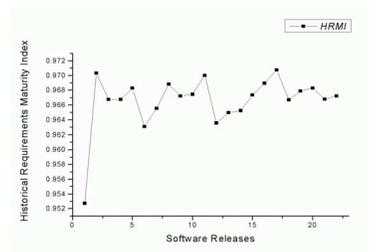Figure 11 shows the $HRMI$ for the entire set of requirements.

Figure 11. Historical Requirements Maturity Index.

In conclusion this section assesses the proposed metrics, RSI and $HRMI$. The analysis supports the intuition behind the two metrics.

6   Conclusions and Further Work

This paper shows an empirical investigation of an avionics industrial case study. The empirical results provide input to our modelling of Requirements Evolution. In summary, our main conclusions are:

Empirical Analysis. The empirical analysis of Requirements Evolution provides practical experience of measuring evolutionary information. Industry collects data, which are not always analysed with a life cycle and evolutionary perspective. The analysis of evolutionary datasets should be carefully studied and understood within the specific industrial context.

Requirements Maturity Index. The empirical analysis provides experience of how difficult is effectively to apply even simple metrics like the RMI.

Requirements Evolution Modelling. The proposed model captures three different aspects, which classify, structure and quantify Requirements Evolution. The quantitative modelling refines the RMI by taking into account historical information.

Evolution Drivers. The empirical evaluation of the two proposed metrics, that is, *Requirements Stability Index* and *Historical Requirements Maturity Index*, identifies the distribution of Requirements Changes as an important driver for the requirements process and for the Maturity of requirements. This relates Requirements Evolution to the iterations of the development life cycle.

Further work aims to: replicate the experiment in other industrial contexts; identify other drivers for the requirements process; model an Evolution Oracle based on the identified drivers and process information.

In conclusion this work provides new insights into Requirements Evolution. The proposed evolutionary model may be used to monitor Requirements Evolution. The empirical nature of this work allows the experiment to be replicated in other industrial contexts and to benefit from our results.

References

[1] S. Anderson and M. 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, Oct. 2000. Springer-Verlag.

[2] S. Anderson and M. Felici. Requirements changes risk/cost analyses: An avionics case study. In M. Cottam, D. Harvey, R. 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] S. Anderson and M. Felici. Requirements evolution: From process to product oriented management. In *Proceedings of Profes 2001, 3rd International Conference on Product Focused Software Process Improvement*, LNCS 2188, pages 27–41, Kaiserslautern, Germany, Sept. 2001. Springer-Verlag.

[4] N. E. Fenton and S. L. Pfleeger. *Software Metrics: A Rigorous and Practical Approach.* International Thomson Computer Press, second edition, 1996.

[5] T. F. Hammer, L. L. Huffman, and L. H. Rosenberg. Doing requirements right the first time. *CROSSTALK The Journal of Defense Software Engineering*, pages 20–25, Dec. 1998.

[6] S. Harker, K. Eason, and J. Dobson. The change and evolution of requirements as a challenge to the practice of software engineering. In *Proceedings of the IEEE International Symposium on Requirements Engineering*, pages 266–272, San Diego, California, USA, Jan. 1993. IEEE Computer Society Press.

[7] I. F. Hooks and K. A. Farry. *Customer-Centered Products: Creating Successful Products Through Smart Requirements Management.* Amacom, 2001.

[8] IEEE. *IEEE Std 982.1 - IEEE Standard Dictionary of Measures to Produce Reliable Software*, 1988.

[9] IEEE. *IEEE Std 982.2 - IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software*, 1988.

[10] M. Lehman. Software's future: Managing evolution. *IEEE Software*, pages 40–44, Jan-Feb 1998.

[11] M. Lehman and L. Belady. *Program Evolution: Processes of Software Change*, volume 27 of *A.P.I.C. Studies in Data Processing*. Academic Press, 1985.

[12] M. Lehman, D. Perry, and J. Ramil. On evidence supporting the feast hypothesis and the laws of software evolution. In *Proceedings of Metrics '98*, Bethesda, Maryland, Nov. 1998.

[13] PROTEUS Project. Meeting the challenge of changing requirements. Deliverable 1.3, Centre for Software Reliability, University of Newcastle upon Tyne, June 1996.

[14] S. Robertson and J. Robertson. *Mastering the Requirements Process.* Addison-Wesley, 1999.

[15] RTCA. *DO-178B Software Considerations in Airborne Systems and Equipment Certification*, 1992.

[16] N. F. Schneidewind. Measuring and evaluating maintenance process using reliability, risk, and test metrics. *IEEE Transactions on Software Engineering*, 25(6):769–781, November/December 1999.

[17] I. Sommerville and P. Sawyer. *Requirements Engineering: A Good Practice Guide.* John Wiley & Sons, 1997.

[18] G. Stark, A. Skillicorn, and R. Ameele. An examination of the effects of requirements changes on software releases. *CROSSTALK The Journal of Defence Software Engineering*, pages 11–16, Dec. 1998.

[19] A. 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.

[20] K. E. Wiegers. *Software Requirements.* Microsoft Press, 1999.