

Controlling Requirements Evolution An Avionics Case Study

Stuart Anderson and Massimo Felici

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

Abstract. This paper presents an empirical investigation of the control of requirements evolution in an avionics safety-critical system. Metrics can be used to manage (e.g., control and predict) requirements evolution. The results point out issues in the use of metrics for controlling requirements evolution in the case study. Moreover, they provide new evidence, which suggests a product line oriented management method for requirements. The empirical analysis supports a number of remarks that are described through the paper.

1 Introduction

Current practice in Software Engineering [1, 13, 17] emphasises the need for iterative development processes refining the software product in all its related representations (e.g., software requirements specification, design specification, testing activities, source code, etc.). Even small software projects require subsequent releases in order to fulfil changing user expectation and environmental constraints. Researchers and practitioners in Software Engineering have recognised the important role of requirements [2, 3, 16] leading to the development of the field of Requirements Engineering [18, 20].

Recent research [14] in Requirements Engineering identifies the origins of requirements change in the development environment. Changing requirements are classified according to five types, which are related to the development environments, stakeholders, development processes, requirements understanding and requirements relations. Requirements change raise problems that affect all aspects of software production [14], i.e., processes as well as deliverables (e.g., software). Thus, software development environments from an external viewpoint look like a collection of processes, stakeholders and deliverables giving rise to instability in the product. The extremes of this turbulence are requirements evolution and software evolution. Previous studies [11] have identified a set of laws of software evolution, but understanding the relationship between requirements evolution and software evolution still remains a challenge for researchers and practitioners [5, 19]. This relationship is very important. Confirmation of its importance can be found in reported accidents in safety-critical systems, which have emphasis the relationship between requirements evolution, software evolution and system safety [12].

Metrics [4] can be used as a means to manage (e.g., control and predict) requirements evolution. They have been used to investigate software evolution [11, 8], but there is still little evidence how similar metrics can be effectively applied at the requirements level as well. The effective use of metrics requires a careful plan and a deep analysis of the specific problems for which a measurement program is developed [4]. Previous analyses [9, 10] emphasise the need for domain specific approaches in requirements engineering. The use of general measurement approaches could be misleading and could lead to misunderstanding. It is strongly recommended to develop specific measurement programs for each context. Only experience within a specific context can improve our level of confidence in a measurement approach.

The work reported in this paper aims to assess how metrics can be used to give quantitative evidence of the effectiveness of the management of requirements evolution. The empirical investigation aims to relate requirements features to the specific product line. The paper is organised as follows. Section 2 describes the avionics safety-critical case study. Section 3 summarises the empirical investigation. The empirical analysis supports a set of remarks that are reported in the Section 3. Section 4 summarises the work and identifies possible areas of future work.

2 An Avionics Case Study

The case study consists of software for an avionics system. There are stringent safety requirements for the software, which has been certified according to the standard used in the avionics context [15]. The software project has been developed as part of a sub-contract. There are 22 successive releases of the software requirements specification each corresponding to a software release. The evolution of these requirements specifications is analysed in the empirical investigation summarised in the next section.

Figure 1 shows a representation of the phases of the development process and its deliverables (i.e., system requirements, software functional requirements, etc.). The *System Requirements* cover the whole system, that is, they are specified in terms of system and not software functions. Then the *System Process* translates the requirements and allocates them into the *Software Functional Requirements*. The software functional requirements are organised in terms of the software functions identified in the system requirements. These software functions will be integrated further in the development process. After the definition of the software functional requirements, the development process travels through design and coding. All the anomalies (e.g., faults, failures, misbehaviours, etc.) encountered during the development are reported by a *Fault Report*. Fault reports consist of a document reporting all the information useful to the development team in order to assess the possible faults. When a fault has been recognised actions are taken to fix it and the needed changes are allocated to a specific software release. The scope of these actions ranges from requirements to software code. Thus continuous feedback is provided by the fault reports. Notice that

certification requires that all the changes are traced. This data is the basis for the empirical investigation described in the next section.

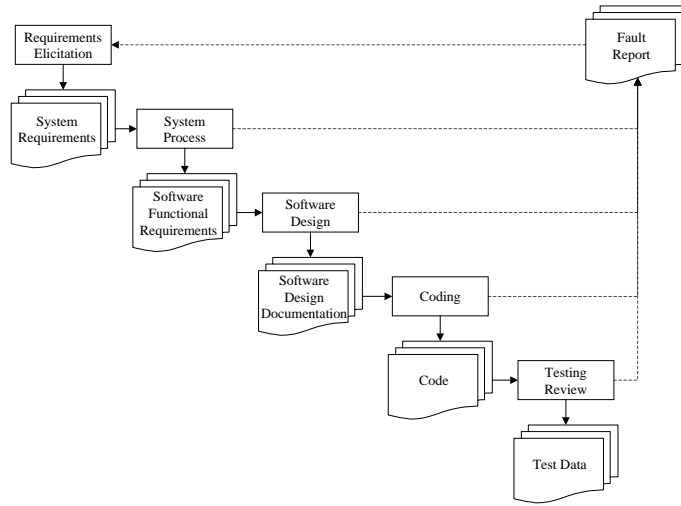


Fig. 1. Development activities and deliverables

3 Measures and Trends

This section summarises an empirical investigation aiming to identify quantitative features of requirements evolution. The empirical results are summarised in a number of remarks.

Figure 2 shows the total number of requirements changes (i.e., added, deleted and modified requirements) over the 22 software releases¹. The trend of requirements changes does not give enough information about evolution features, but it emphasises the problem of changing requirements. Figure 3 shows the trend of the total number of requirements over the software releases. The view given by the total number of requirements is more interesting than that one by the total number of changes. The total number of requirements increases over the software releases.

Remark 1 (Size of Requirements). The number of requirements tends to grow over the software releases.

¹ There is a correspondence one-to-one between versions of the requirements specification and software releases. We will refer only to software releases along the rest of the paper to avoid any confusion.

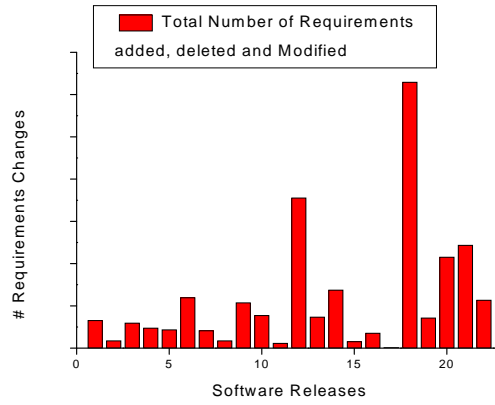


Fig. 2. Number of requirements changes per software releases

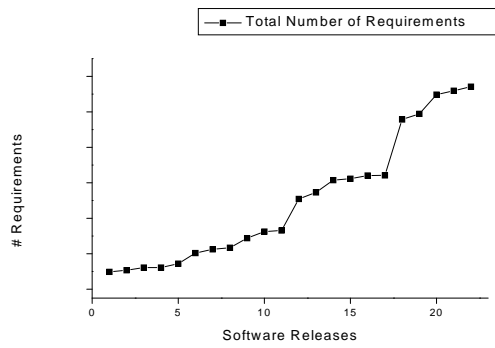


Fig. 3. Number of requirements per software releases

This is probably then because the requirements become clearer to the stakeholders, who split complex requirements into smaller, more precisely stated, requirements. Another reason could be that new requirements arise during the progress of the project, in other words, there could be requirements which cannot be defined at the beginning for the lack of information. Finally, design, implementation and testing provide additional feedback to the requirements.

A question arises from the trend of the number of requirements. What is a suitable metric to assess the readiness of requirements? The stability of requirements [4] can be assessed by (1).

$$Requirements\ Stability = \frac{Number\ of\ initial\ requirements}{Total\ Number\ of\ requirements} . \quad (1)$$

Equation (1) is not sufficiently expressive, because it does not take into account modified requirements. Hence, it is not adequate in cases where there are

many modified requirements or there are roughly the same number of added and deleted requirements. The standard IEEE 982 [6, 7] suggests a Software Maturity Index to quantify the readiness of a software product. Hence, the idea of using a similar Requirements Maturity Index (RMI) to quantify the readiness of requirements. Equation (2) defines the RMI².

$$RMI = \frac{R_T - R_C}{R_T} . \quad (2)$$

Figure 4 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 shape. Hence any assessment based only on the RMI could be misleading and risky, see Remark 2.

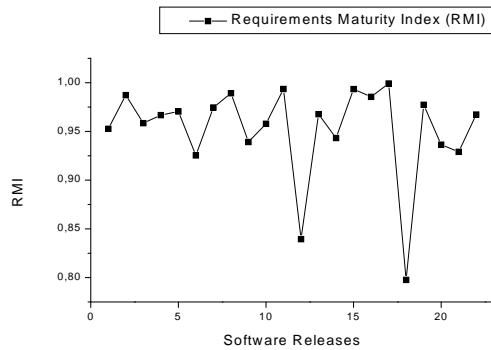


Fig. 4. Requirements Maturity index

Remark 2 (Requirements Maturity Index). The Requirements Maturity Index can be misleading in assessing the readiness of requirements. The metrics user should always assess its applicability in the specific context.

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 become unusable for the requirements. Metrics are context sensitive, therefore an aware metrics user should always assess their applicability for the specific case.

To provide a more detailed analysis our focus moves from the total number of requirements changes to the total 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.

² R_T = number of software requirements in the current delivery ; R_C = number of software requirements in the current delivery that are added, deleted or modified from a previous delivery.

Figure 5 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.

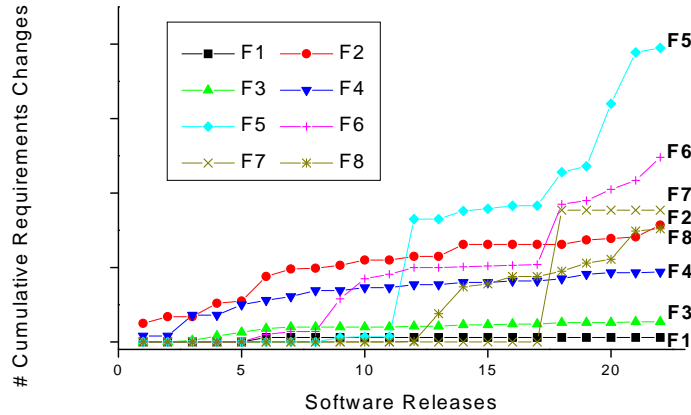


Fig. 5. Cumulative number of requirements changes for each function

The first outcome of this new viewpoint 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. Hence, Remark 3, the software architecture is a stable part of the requirements.

Remark 3 (Architecture). The software requirements of the system architecture represent a stable part of the requirements. The likelihood of architecture changes is low and changes usually occur in early releases of the software, when they can still be accommodated with affordable costs.

Another outcome, Remark 4, of this functions oriented view is 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 identified in the functions with the software life cycle. Moreover, the trends show an interesting switching feature that needs further investigation to clarify its consistency. The assumption is that there could be possible dependencies between subsets of the requirements, that is, the requirements identified by those functions which change later in the development process depend on those functions containing early changing requirements. Thus, there are requirements that are dependent upon others. It is not possible to define all the requirements at once due to these dependencies over requirements. At this level of granularity the empirical investigation does not allow easy analysis of these dependencies between

requirements. We intend to validate Remark 4 by replicating the experiment in other case studies and in different industrial contexts.

Remark 4 (Requirements Dependencies). Functions that are likely to change during early software releases change less during later releases, and vice versa. The evolution trends show some dependencies between requirements.

Figure 6 shows a scatter plot relating the cumulative number of requirements changes for each function and the respective size in terms of number of requirements in the last release³. For most of the functions there exists a linear relation between these two measures. Figure 6 identifies the functions that have a non-linear relation upon these measures (i.e., F2, F5 and F8). Thus, the graph helps to identify those functions which are most likely to change (i.e., F5, F6 and F7) and those which are not. Questions arise from this result. In particular, can a product oriented approach to allocate changes help in managing requirements changes? In other words, can a product oriented requirements management be useful in obtaining a cost-effective allocation of requirements changes? Remark 5 summarises the last result of the empirical investigation.

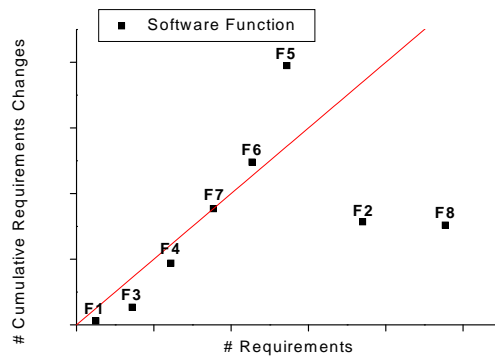


Fig. 6. Scatter plot of number of changes against number of requirements

Remark 5 (Product Oriented Perspective). Some functions are more likely to experience requirements change than others. For most of the functions there exists a linear relation between cumulative number of requirements changes and final number of requirements. The management of requirements changes could be improved by a product oriented perspective.

4 Conclusions and Further Work

The paper shows an empirical investigation of an avionics safety-critical case study. The empirical investigation points out interesting aspects of changing re-

³ The number of requirements in the 22nd release.

quirements. The results identify requirements features, which suggest product oriented management of requirements changes. Effective integration of current management processes together with a product oriented view offers the potential to improve our ability to bound the scope of change in order to guarantee continuous process improvement. Moreover, a product oriented view can provide a useful baseline to devise new tools for assessing the risk of requirements changes. The main remarks are summarised in what follows.

Size of Requirements. The number of requirements tends to grow over the software releases.

Requirements Maturity Index. The Requirements Maturity Index can be misleading in assessing the readiness of requirements. The metrics user should always assess its applicability in the specific context.

Architecture. The software requirements of the system architecture represent a stable part of the requirements. The likelihood of architecture changes is low and changes usually occur in early releases of the software, when they can still be accommodated with affordable costs.

Requirements Dependencies. Functions that are likely to change during early software releases change less during later releases, and vice versa. The evolution trends show some dependencies between requirements.

Product Oriented Perspective. Some functions are more likely to experience requirements change than others. For most of the function there exists a linear relation between cumulative number of requirements changes and final number of requirements. The management of requirements changes could be improved by a product oriented perspective.

The above results suggest further work aiming to

- validate the results in different case studies
- validate the results in different product line and industrial contexts
- extend the results by specific analyses of the features identified through the paper
- investigate relationships with other product aspects (e.g., reliability, safety, dependability, etc.)
- devise product oriented requirements engineering tools.

In conclusion, the paper shows a detailed quantitative analysis, which points out product oriented features that may improve the specification of requirements as well as their management. The results identify some research directions defining a work plan to understand requirements evolution.

5 Acknowledgements

The authors wish to thank the industrial partner, who has provided the case study. Due to the high safety level of the case study, the agreement with the industrial partner does not allow to show any numerical information. Despite this the analysis remains still valid and the results are clearly expressed.

References

- [1] Lowell Jay Arthur. *Rapid Evolutionary Development: Requirements, Prototyping & Software Creation*. John Wiley & Sons, 1992.
- [2] Daniel M. Berry and Brian Lawrence. Requirements engineering. *IEEE Software*, pages 26–29, March 1998.
- [3] Alan M. Davis and Pei Hsia. Giving voice to requirements engineering. *IEEE Software*, pages 12–16, March 1994.
- [4] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. International Thomson Computer Press, second edition, 1996.
- [5] S.D.P. Harker and K.D. Eason. 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, January 1993. IEEE Computer Society Press.
- [6] IEEE. *IEEE Std 982.1 - IEEE Standard Dictionary of Measures to Produce Reliable Software*, 1988.
- [7] IEEE. *IEEE Std 982.2 - IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software*, 1988.
- [8] Chris F. Kemerer and Sandra Slaughter. An empirical approach to studying software evolution. *IEEE Transactions on Software Engineering*, 25(4):493–509, July/August 1999.
- [9] W. Lam. Achieving requirements reuse: A domain-specific approach from avionics. *The Journal of Systems and Software*, 38(3):197–209, September 1997.
- [10] W. Lam, J.A. McDermid, and A.J. Vickers. Ten steps towards systematic requirements reuse. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering*, pages 6–15, Annapolis, Maryland, USA, January 1997. IEEE Computer Society Press.
- [11] M.M. Lehman, D.E. Perry, and J.F. Ramil. On evidence supporting the feast hypothesis and the laws of software evolution. In *Proceedings of Metrics '98*, Bethesda, Maryland, November 1998.
- [12] Nancy G. Leveson. *SAFWARE: System Safety and Computers*. Addison-Wesley, 1995.
- [13] Shari Lawrence Pfleeger. *Software Engineering: Theory and Practice*. Prentice-Hall, 1998.
- [14] PROTEUS Project. Meeting the challenge of changing requirements. Deliverable 1.3, Centre for Software Reliability, University of Newcastle upon Tyne, June 1996.
- [15] RTCA. *DO-178B Software Considerations in Airborne Systems and Equipment Certification*, 1992.
- [16] J. Siddiqi and M.C. Shekaran. Requirements engineering: The emerging wisdom. *IEEE Software*, pages 15–19, March 1996.
- [17] Ian Sommerville. *Software Engineering*. Addison-Wesley, fifth edition, 1995.
- [18] Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practice Guide*. John Wiley & Sons, 1997.
- [19] George Stark, Al Skillicorn, and Ryan Ameele. An examination of the effects of requirements changes on software releases. *CROSSTALK The Journal of Defence Software Engineering*, pages 11–16, December 1998.
- [20] Karl Eugene Wiegers. *Software Requirements*. Microsoft Press, 1999.