

# Taxonomy of Evolution and Dependability

Massimo Felici\*

LFCS, School of Informatics, The University of Edinburgh  
JCMB, The Kings Buildings, Mayfield Road  
Edinburgh EH9 3JZ, United Kingdom  
massimo.felici@ed.ac.uk

## Abstract

Evolution is one of the major issues affecting system dependability as well as engineering activities and environments. The most common understanding considers evolution as a phenomenon that needs to be avoided. By contrast this paper takes into account evolution as a necessary feature of computer-based systems. This paper reviews a taxonomy of evolution identifying a conceptual framework to analyse evolutionary phenomena of computer-based systems as well as models of evolutions and their limits. The taxonomy of evolution points out different dependability aspects of computer-based systems. In conclusion, this paper provides a conceptual framework to analyse evolution and its influence on the dependability of computer-based systems.

## 1 Introduction

Computer-based systems are evolving citizens of the modern electronic mediated society. They are continuously (re)designed and (re)deployed in order to capture environmental evolution. Development life cycles in software system engineering recognise the need to capture evolution by iterative processes. Evolution of computer-based systems is a twofold concept. On the one hand evolution is an inevitable and a needed aspect of computer-based systems. On the other hand the degradation of the dependability of computer-based systems may be due to evolution and it may cause, in the worst case, catastrophic failures [39, 46, 59].

The relationship between Evolution and Dependability is still vaguely understood. This is because evolution and dependability are both complex concepts. There could be many different definitions of evolution that apply to computer-based systems. Evolution can occur from the early stages of a system (e.g., requirements evolution) to its deployment, use and decommission (e.g., corrective or perfective maintenance). The existence of different (definitions of) evolutions is often misunderstanding and it gives rise to practical issues as well. Dependability is defined as that property of a computer system such that reliance can justifiably be placed on the service it delivers [34, 35]. With respect to the dependability of computer-based systems evolution may be a means to increase it. On the other hand evolution is also a major source of undependability for computer-based systems.

This paper takes account of the evolution of computer-based systems. Section 2 reviews a taxonomy of evolution identifying a conceptual framework to analyse different evolutionary phenomena of computer-based systems. Section 3 points out how the evolutionary phenomena differently relate to the dependability of computer-based systems. In conclusion, this paper provides a conceptual framework to analyse evolution and its influence on the dependability of computer-based systems.

---

\*This work has been partially funded by a grant of the Italian National Research Council (CNR) for research in Mathematical Science, Bando n. 203.01.72, Codice n. 03.01.04, Research Programme: "A Formal Framework for Requirements Evolution".

## 2 Taxonomy of Evolution

This section reviews a taxonomy of evolution of computer-based systems. Figure 1 shows the evolutionary space drew by the taxonomy.

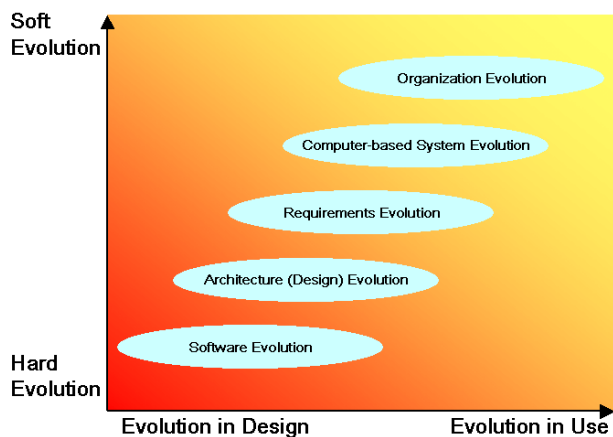


Figure 1: Evolutionary space.

The two dimensions of the evolutionary space are:

from *Evolution in Design* to *Evolution in Use* - stressing the life cycle perspective (or, temporal dimension), i.e., evolution can occur during different phases of a system life cycle, since design until deployment, use and further on.

from *Hard Evolution* to *Soft Evolution* - stressing where in the system evolution takes place (or, physical dimension).

A point within the evolutionary space represents a tradeoff among different evolutionary phenomena. The main evolutionary phenomena forming the evolutionary space are:

*Software Evolution* takes account of evolution from a product viewpoint.

*Architecture (Design) Evolution* describes how evolution is perceived at the design level.

*Requirements Evolution* represents an intermediate viewpoint. Requirements are used as a means of interaction between stakeholders, thus requirements represent a natural place where to capture information about the evolution of computer-based systems.

*Computer-based System Evolution* takes into account a systemic viewpoint that emphasises human factors with respect to evolution.

*Organization Evolution* emphasises the interaction between the computer-based system and the surrounding environment.

Notice that the evolutionary phenomena may overlap one another. Thus they need to be instantiated case by case. These five different evolutionary phenomena have some similarities with other reference models (e.g., [23, 47]) that categorise and structure engineering aspects of computer-based systems. The remains of this section describe the different phenomena forming the taxonomy of evolution.

## 2.1 Software Evolution

The problem of software evolution has been extensively investigated by Lehman et al [37, 38]. The investigation of software evolution as a natural phenomenon has led to the identification of the *E-type programs* (i.e., software systems). According to the Lehman's laws of software evolution, E-type programs continuously evolve in order to be satisfactory (for users) and (need) to accommodate environmental changes. E-type programs have an increasing complexity if maintenance is not performed. They represent multi-level, multi-loop, multi-agent feedback systems. In spite of the progress in software engineering managing evolution is still challenging [36]. Software evolution (and in general evolution) is often considered just a management issue [61] having little emphasis on software evolutionary features. Recent research in Object-Oriented software [20] stresses that a product viewpoint may enhance our understanding of software evolution. Three main patterns of evolutions have been identified:

*Software Tectonics* emphasises that software systems need to accommodate arising changes. It will involve not just fixing bugs, but also fixing flaws introduced early in the design. Hence software has to be implemented in order to support evolution, otherwise software changes will just increase software complexity and destroy fundamental software structures. Moreover, software systems should be adaptable to requirements changes by a series of small and controlled steps in order to avoid software degradation.

*Flexible Foundations* catalogues the need to construct systems out of stuff that can evolve along with them. That is, the basics (e.g., tools, languages, frameworks, etc.) of each system should be able to evolve themselves in order to support the software system evolution as a whole. Notice that the Flexible Foundations pattern focuses on the system's infrastructure (or architecture) and not on the specific implementation code.

*Metamorphosis* shows how equipping systems with mechanisms that allow them to dynamically manipulate their environments can help them better to integrate in these environments in order to fulfil evolving requirements. There exist different mechanisms supporting the Metamorphosis pattern. For instance, extendible systems allow users to add new features. Differently mutable systems allow users to change their existing features.

Software Metrics may capture evolutionary properties of software and relate them to software quality [18, 27, 28, 30, 54]. Metrics quantify how software has been modified, e.g., in terms of Lines Of Code (LOC). Specific metrics assess software reliability and probabilistic models, namely Reliability Growth Models, predict the evolution of software reliability [43]. With respect to evolution Software Reliability Growth Models present some applicability limits [17, 42]. The basic assumptions (e.g., features of the models inadequately fit real conditions in software development processes.

In spite of the limits of quantitative approaches recent research [5, 13, 22, 32] in empirical Software Engineering has emphasised how the use of specific quantitative models may estimate evolutionary software features. Antoniol et al. in [5] propose a quantitative model in order to estimate the size of changes for evolving Object-Oriented systems. The model has been assessed and evaluated on a case study. Coleman et al. in [13] use software metrics to evaluate the maintainability of 11 industrial software systems. Their experience points out that even simple models may support the decision making process at different stages of the software life cycle. Graves et al. in [22] devise different statistical models to evaluate the likelihood of faults introduced into modules based on the history of changes. The results emphasise the relationship between software changes and faults. Moreover, they evaluate the proposed models in order to assess their performance and accuracy. Kemerer and Slaughter in [32] propose an empirical approach to studying software evolution. They furthermore emphasise how it is difficult to conduct empirical analyses on software evolution due to the lack of evolutionary data.

## 2.2 Architecture (Design) Evolution

At the design level, the role of the system architecture may be unclear with respect to evolution. Architecture evolution is risky if it is not controlled around specific variation points that may characterise product lines [10], even if the evolutionary process and the architecture are well defined and understood [33, 56]. The architectures of product lines [10, 62] represent the extent to which system families can

capture future needs. Our ability to predict evolution is therefore crucial in the definition of product lines. The architecture should implement the most suitable trade off between generality and specificity.

Although the architecture is so important, its evolution may be misleading. There exist in general two (three) types of evolutions: the architecture evolves or its components evolve (everything evolves). Despite what people may think about the architecture's evolution, empirical results [1] show that the architecture is a stable part of those systems that have stringent safety and security requirements (constraints). Related research in requirements engineering [48] supports these empirical results by emphasising that stable requirements have origin in the business core. The more constrained is the business, the more evident is the relation between the architecture's stability and the business core. Further progress [44] on the evolution of Object-Oriented architectures supports these results by stating that: "...A software architecture is a collection of elements that share the same likelihood of change. Each category contains software elements that exhibit shared stability characteristics. ...A software architecture always contains a core layer that represents the hardest layer of change. It identifies those features that cannot be changed without rebuilding the entire software system."

## 2.3 Requirements Evolution

Requirements evolution has been mainly considered a management problem, rather than a feature of computer-based systems. Requirements Engineering literature has little emphasis on software evolutionary features [26, 53, 57, 63]. Despite this empirical analyses of industrial case studies point out that it is impossible to frozen requirements, but it is possible to analyse the extent to which requirements evolve in order to identify the stable ones and the most likely to change [1, 2, 3, 25, 26, 48, 58]. Since its recognition Requirements Engineering has received an increasing interest within Software Engineering and has acquired a multiperspective role towards model-based software engineering [60]. There exist structures that support the construction of models towards Requirements Evolution.

*Types of Requirements* allow to classify the origins of stable and changing requirements and to identify a conceptual framework for changing requirements [48]. The Proteus Project [48] suggests three main strategies dealing with requirements changes. The proposed strategies are *reducing changes*, *facilitating incorporation* of changes and *identifying needs* for changes as early as possible. These strategies make use of enabling technologies like predictive analysis of changes, traceability of requirements, formal framework for representing and reasoning about changes, and prototyping. Guidelines and checklists support the strategies dealing with requirements changes.

*Dependencies between Requirements* [3] may be redefined across subsequent releases in order to minimise them and effectively to assess the impact of changes.

*Types of Changes*, e.g., adding, deleting and modifying requirements, define how changes alter requirements specifications. It is possible to quantify requirements evolution by counting occurring (types of) changes. Requirements specification templates (e.g., [29, 52, 53]) may identify other (types of) attributes that are worthwhile monitoring in order to understand requirements evolution.

further refinements of documents as well as analyses.

*Requirements Traceability* [14, 15, 31, 49] provides further information to analyse requirements evolution, but it is insufficient. Requirements traceability is an important aspect to deal with evolution [6, 31]. Traceability represents not only an additional information to be collected (and maintained), but also a strategic policy involving an entire software organisation [49]. Traceability maintenance consists of strict processes and tools, which allow to identify relationships between the software organisation and the software product [31]. We can then enhance our ability to specify and to represent requirements evolution by combining traceability with other evolutionary information.

We can then represent requirements evolution in terms of the above structures. The combination of structured requirements evolution together with formal requirements specifications [48, 21, 65] may be useful in order to support formal as well as quantitative reasoning on requirements evolution. The integration of formal requirements specifications with empirical information is still challenging in requirements engineering. There are still few quantitative approaches that successfully capture requirements evolution

[1, 4, 27, 28, 55]. The lack of practical experience in monitoring long term information and the lack of evolutionary data often limit our ability in monitoring and understanding evolution as well as to estimate the cost of changes [7, 8, 9].

## 2.4 Computer-based System Evolution

The term “computer-based system” emphasises the human aspects within socio-technical systems. Computer systems (consisting of hardware and software) do not exist in isolation. They are part of the information society. On the other hand, an holistic view of computer-based systems may identify and better understand the drivers for evolution. For instance, figure 2 shows the SHEL model [16], which defines any productive process as performed by a combination of *Hardware* (e.g., any material tool used in the process execution), *Software* (e.g., procedures, rules, practices, etc.) and *Liveware* (e.g., end-users, managers, etc.) resources embedded in a given *Environment* (e.g., socio-cultural, political, etc.). Thus, any process requires some knowledge that is distributed among the system resources. Hence a productive process may be regarded as an instantiation of the SHEL model for a specific process execution. An holistic view, like that of the SHEL model, emphasises how drivers for software evolution may reside outside software artefacts. On the other hand evolution across resources may allow new artefacts to emerge as resulting behaviour of the evolved systems.

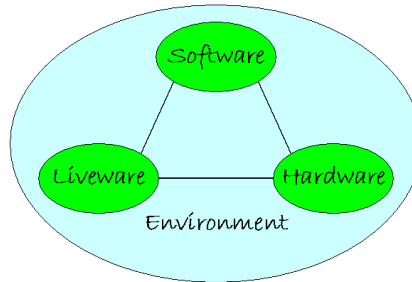


Figure 2: The SHEL model.

The link between technical evolution and socio evolution of computer-based systems has not yet been fully registered. It still remains vaguely understood and challenging for research and practice [12]. There is not any model that fully specifies the evolution of socio-technical systems. To some extent we can represent the evolution of socio-technical systems by collections of models, which do not complement each other in most of the cases.

Few models arose in order better to understand human factors with respect to computer-based system evolution. For instance, *Social Learning* [64] explains how human beings perceive machines in order to acquire computational artefacts and accomplish specific tasks. Social Learning consists of two main processes:

*Innofusion* is a practical activity of learning by trying [19] that allows to customise the computational artefacts (behind a computer-based systems) to meet social needs. The underlying hypothesis is that workers, individually as well as collectively, develop more efficient ways of employing machinery through their experience from usage. This kind of learning curve effect is well-known. Though often taken to be limited to the initial introduction of new equipment, it is now recognised that learning by doing may improve the efficiency of production over a very long period of time.

*Domestication* addresses the creative role of the user in integrating new artefacts within their everyday activities and meanings. Domestication, where people learn by situated activity, is a practical activity of learning by interacting. This activity stresses the complex interaction between technology

supply and use by applying evolutionary metaphors of the generation of variation and selection of artefacts.

*Distributed Cognition* [45] recognises the complex settings of socio-technical systems and analyses how humans work, operate and create external and internal artefacts. This approach re-elaborates the long lasting thesis that human cognition is mediated by artefacts (e.g., rules, tools, representations, etc.) that are both internal and external to the mind. The central tenet of the Distributed Cognition approach is that knowledge is distributed across people and artefacts; cognition is not a property of individuals but rather a property of a system of individuals and artefacts carrying out some activity. According to these theoretical assumptions, human activities and artefacts are the two inseparable sides of the same phenomenon: human cognition.

The above models are quite useful in order holistically to analyse the evolution of computer-based systems. But they still need to be fully integrated with classical engineering methodologies (e.g., those reviewed in the previous subsections).

## 2.5 Organization Evolution

The previous sections point out the strong connection between technical and social aspects of computer-based systems (or socio-technical systems). The evolution of socio-technical systems influences the organizational context as well. Thus system evolution implies an organization co-evolution, and vice versa. The mechanisms driving organization evolution characterise the new electronic mediated society [11, 12, 40]. The link between socio-technical evolution and organization evolution has not been yet fully registered. It is still challenging for future research and practice.

## 3 On the Dependability of Evolution

Dependability models capture evolution in different ways. For instance, fault tolerance models [35, 50] rely on failure distributions (e.g., Mean Time Between Failures) for computer-based systems. Monitoring this type of measure allows to characterise the evolution of system properties (e.g., Reliability, Availability, etc.). Probabilistic models [43] may predict how dependability measures evolve according to the estimations of attributes and the assumptions about the operational profile of the system.

By contrast other models link dependability features with system structures and development processes (e.g., [41, 42]). This allows us to link failure profiles with design attributes (e.g., diversity) and system structures (e.g., redundancy). Structured models (e.g., FMEA, HAZOP, FTA) assess the hazard related to system failures and their risk [59]. These models extend the *Domino* model, which assumes that an accident is the final result of a chain of events in a particular context [24]. Similarly the *Cheese* model consists of different safety layers having evolving undependability holes. Hence system failures completely arise and become catastrophically unrecoverable when they propagate throughout all the safety layers in place [51]. Despite of these models capture a dynamic view of system failures, they do not capture evolution.

The evolutionary phenomena presented in the previous section differently contribute to dependability. Table 1 summarises the different dependability evolutionary perspectives. The taxonomy of evolution points out that each methodology has different assumptions about the evolution of computer-based systems. This may trigger practical issues related to evolution:

**Inconsistency:** The basic assumptions of all adopted models (in order to characterise system dependability) may be inconsistent all together.

**Coverage:** The entire spectrum of models may be insufficient to cover all evolutionary aspects of system dependability.

**Relational:** The different evolutionary phenomena relate to each other. It is difficult to understand the different relationships between the evolutionary phenomena.

**Emergent:** New (or unexpected) system features may emerge from the different evolutionary phenomena.

Table 1: Dependability perspectives of Evolution.

Evolution	Dependability Perspective	Engineering Hint
Software Evolution	Software evolution can affect dependability attributes (e.g., Reliability). Nevertheless software evolution can improve dependability attributes by faults removal and maintenance to satisfy new arising requirements.	Monitor software complexity; Identify the change-prone parts of the software; Carefully manage basic software structures; Monitor dependability metrics.
Architecture (Design) Evolution	Architecture evolution is usually an expensive phenomenon. It does not affect directly dependability, but there is high risk if the evolution process is unclear and little understood. Architecture evolution may be needed to support specific system properties (e.g., redundancy, performance, etc.).	Assess the stability of the software architecture; Understand the relationship between the architecture and the business core; Analyse any (proposed or implemented) architecture change.
Requirements Evolution	Requirements evolution does not directly affect dependability, but non-effective management of the requirement process may allow undesired changes to fall down into the product affecting its dependability. On the other hand requirements evolution may enhance system dependability across subsequent releases.	Classify requirements according to their stability/volatility; Classify requirements changes; Monitor requirements evolution.
Computer-based System Evolution	System evolution may give rise to undependability. This is due to incomplete evolution of system resources. Evolution of some resources (e.g., software) should be taken into account by the other resources (e.g., liveware and hardware) in order to register a new configuration for the system. Hence the interactions among resources serve to effectively deploy a new system configuration. On the other hand human can react and learn how to deal with undependable situations, but continuous changes in the system configuration may give rise to little understanding about the system. Hence the human-computer interaction may become quite undependable as well.	Acquire a systemic view (i.e., Hardware, Software, Liveware and Environment); Monitor the interactions between resources; Understand evolutionary dependencies; Monitor and analyse the (human) activities supported by the system.
Organization Evolution	Organization evolution should reflect system evolution. Little coordination between system evolution and organization evolution may give rise to undependability.	Understand environmental constraints; Understand the business culture; Identify obstacles to changes.

The relationships between the evolutionary phenomena may identify a framework for analysing the evolution of computer-based systems. If the coordination between evolutionary phenomena is too weak, dependability may deteriorate. On the other hand evolutionary phenomena introduce diversity and may prevent system failures. Table 1 proposes some engineering hints in order to construct an evolutionary framework. The resulting framework may classify evolution for computer-based systems. The framework may furthermore provide evidence of the relationships between the different evolutionary phenomena and relate them to dependability. Notice that it is difficult to capture and analyse evolutionary data, because they are usually incomplete, distributed, unrelated and vaguely understood. The resulting evolutionary framework may support the empirical process as well. A taxonomy of evolution is then a starting point to understand how computer-based systems evolve.

## 4 Conclusions

Software evolution represents just one aspect of the evolution of computer-based systems. This paper describes a taxonomy of evolution: *Software Evolution*, *Architecture (Design) Evolution*, *Requirements Evolution*, *Computer-based System Evolution*, *Organization Evolution*. It provides a holistic viewpoint in order to analyse and to understand the evolution of computer-based systems. The taxonomy points out the different aspects and drivers for the evolution of computer-based systems. The review of different models and methodologies that take into account the evolution of computer-based systems is useful to further understand and stress the relationship between system evolution and dependability. The different assumptions about the evolution of computer-based systems may trigger practical issues affecting system

dependability:

**Inconsistency:** The basic assumptions of all adopted models (in order to characterise system dependability) may be inconsistent all together.

**Coverage:** The entire spectrum of models may be insufficient to cover all evolutionary aspects of system dependability.

**Relational:** The different evolutionary phenomena relate to each other. It is difficult to understand the different relationships between the evolutionary phenomena.

**Emergent:** New (or unexpected) system features may emerge from the different evolutionary phenomena.

The dependability analysis with respect to evolution identifies a framework. The engineering hints related to each evolutionary phenomenon may serve as basics in order empirically to acquire a taxonomy of evolution. The idea of the taxonomy of evolution has originated from previous experiences and collaborations. Future work aims to acquire practical experiences by using the the framework in industrial case studies. In conclusion, the taxonomy of evolution represents a starting point to analyse evolving computer-based systems. It draws a framework in order to further understand how computer-based systems evolve. It provides a holistic viewpoint that suggests future directions for research and practice on computer-based system evolution.

**Acknowledgements.** I would like to thank Stuart Anderson and John Dobson for the useful comments on early drafts of this paper and the discussions on evolution. This work has been conducted within the UK EPSRC DIRC<sup>1</sup> project, Interdisciplinary Research Collaboration in Dependability of Computer-Based Systems, grant GR/N13999.

## 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 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, September 2001. Springer-Verlag.
- [4] Stuart Anderson and Massimo Felici. Quantitative aspects of requirements evolution. In *Proceedings of the 26th Annual International Computer Software and Application Conference, COMPSAC 2002*, pages 27–32, Oxford, England, 2002. IEEE Computer Society.
- [5] G. Antoniol, G. Canfora, and A. De Lucia. Estimating the size of changes for evolving object oriented systems: A case study. In *Proceedings of the Sixth International Symposium on Software Metrics, METRICS '99*, pages 250–259, Boca Raton, Florida, November 1999. IEEE Computer Society.
- [6] K. Suzanne Barber et al. Requirements evolution and reuse using the systems engineering process activities (sepa). *Australian Journal of Information Systems (AJIS)*, 7:75–97, 1999. Special Issue on Requirements Engineering.

---

<sup>1</sup><http://www.dirc.org.uk/>

- [7] Barry W. Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.
- [8] Barry W. Boehm. Software engineering economics. *IEEE Transaction on Software Engineering*, 10(1):4–21, January 1984.
- [9] Barry W. Boehm et al. *Software Cost Estimation with COCOMO II*. Prentice-Hall, 2000.
- [10] Jan Bosch. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley, 2000.
- [11] David Bustard, Peter Kawalek, and Mark Norris, editors. *Systems Modeling for Business Process Improvement*. Artech House, 2000.
- [12] Elayne Coakes, Dianne Willis, and Raymond Lloyd-Jones, editors. *The New SocioTech: Graffiti on the Long Wall*. CSCW. Springer, 2000.
- [13] Don Coleman, Dan Ash, Bruce Lowther, and Paul Oman. Using metrics to evaluate software system maintainability. *IEEE Computer*, 27(8):44–49, August 1994.
- [14] Giorgio De Michelis et al. A three-faceted view of information systems. *Communications of the ACM*, 41(12):64–70, December 1998.
- [15] Ralf Dömges and Klaus Pohl. Adapting traceability environments to project-specific needs. *Communications of the ACM*, 41(12):54–62, December 1998.
- [16] E. Edwards. Man and machine: Systems for safety. In *Proceedings of British Airline Pilots Associations Technical Symposium*, pages 21–36, London, 1972. British Airline Pilots Associations.
- [17] Massimo Felici, Alberto Pasquini, and Mark-Alexander Sujan. Applicability limits of software reliability growth models. In *MMR'2000, Deuxième Conférence Internationale sur les Méthodes Mathématiques en Fiabilité: Méthodologie, Pratique et Inférence*, volume 1, pages 397–400, Bordeaux, France, July 2000.
- [18] Norman E. Fenton and Shari Lawrence Pfleeger. *Software Metrics: A Rigorous and Practical Approach*. International Thomson Computer Press, second edition, 1996.
- [19] James Fleck. Learning by trying: the implementation of configurational technology. *Research Policy*, 23:637–652, 1994.
- [20] Brian Foote and Joseph Yoder. Evolution, architecture, and metamorphosis. In John M. Vlissides, James O. Coplien, and Norman L. Kerth, editors, *Pattern Languages of Program Design 2*, chapter 13. Addison-Wesley, 1996.
- [21] Aditya K. Ghose. Managing requirements evolution: Formal support for functional and non-functional requirements. In *Proceedings of the International Workshop on Principles of Software Evolution*, pages 77–84, Fukuoka, Japan, 1999.
- [22] Todd L. Graves, Alan F. Karr, J. S. Marron, and Harvey Siy. Predicting fault incident using software change history. *IEEE Transactions on Software Engineering*, 26(7):653–661, July 2000.
- [23] Carl A. Gunter, Elsa L. Gunter, Michael Jackson, and Pamela Zave. A reference model for requirements and specifications. *IEEE Software*, pages 37–43, May/June 2000.
- [24] Herbert William Heinrich. *Industrial accident prevention: a scientific approach*. McGraw-Hill, 3rd edition, 1950.
- [25] Hubert F. Hofmann and Franz Lehner. Requirements engineering as a success factor in software projects. *IEEE Software*, pages 58–66, July/August 2001.
- [26] Ivy F. Hooks and Kristin A. Farry. *Customer-Centered Products: Creating Successful Products Through Smart Requirements Management*. Amacom, 2001.

- [27] IEEE. *IEEE Std 982.1 - IEEE Standard Dictionary of Measures to Produce Reliable Software*, 1988.
- [28] IEEE. *IEEE Std 982.2 - IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software*, 1988.
- [29] IEEE. *IEEE Std 830 - IEEE Recommended Practice for Software Requirements Specifications*, 1993.
- [30] ISO/IEC. *ISO/IEC 9126 - Information Technology - Software quality characteristics and metrics*.
- [31] Matthias Jarke. Requirements tracing. *Communications of the ACM*, 41(12):32–36, December 1998.
- [32] 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.
- [33] Juha Kuusela. Architectural evolution. In Patrick Donohoe, editor, *Software Architecture, TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, pages 471–478, San Antonio, Texas, USA, 1999. IFIP, Kluwer Academic Publishers.
- [34] Jean-Claude Laprie. Dependable computing: Concepts, limits, challenges. In *FTCS-25, the 25th IEEE International Symposium on Fault-Tolerant Computing - Special Issue*, pages 42–54, Pasadena, California, USA, June 1995.
- [35] Jean-Claude Laprie et al. Dependability handbook. Technical Report LAAS Report no 98-346, LIS LAAS-CNRS, August 1998.
- [36] M.M. Lehman. Software’s future: Managing evolution. *IEEE Software*, pages 40–44, Jan-Feb 1998.
- [37] M.M. Lehman and L.A. Belady. *Program Evolution: Processes of Software Change*, volume 27 of *A.P.I.C. Studies in Data Processing*. Academic Press, 1985.
- [38] 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.
- [39] Nancy G. Leveson. *SAFWARE: System Safety and Computers*. Addison-Wesley, 1995.
- [40] Frederick Levine, Christopher Locke, David Searls, and David Weinberger. *The Cluetrain Manifesto: The end of business as usual*. ft.com, 2000.
- [41] Bev Littlewood, Peter Popov, and Lorenzo Strigini. Modelling software design diversity: a review. *ACM Computing Surveys*, 33(2):177–208, 2001.
- [42] Bev Littlewood and Lorenzo Strigini. Software reliability and dependability: a roadmap. In A. Finkelstein, editor, *The Future of Software Engineering*, pages 177–188. ACM Press, Limerick, June 2000.
- [43] Michael R. Lyu, editor. *Handbook of Software Reliability Engineering*. IEEE Computer Society Press, 1996.
- [44] Tom Mens and Galan Hassan Galan. 4th workshop on object-oriented architectural evolution. In A. Frohner, editor, *Proceedings of the ECOOP 2001 Workshops*, LNCS 2323, pages 150–164. Springer-Verlag, 2002.
- [45] Donal A. Norman. *The Invisible Computer*. The MIT Press Cambridge, Massachusetts, 1998.
- [46] Charles Perrow. *Normal Accidents: Living with High-Risk Technologies*. Princeton University Press, 1999.
- [47] Dewayne E. Perry. Dimensions of software evolution. In *Proceedings of the IEEE International Conference on Software Maintenance*. IEEE Computer Society Press, 1994.
- [48] PROTEUS Project. Meeting the challenge of changing requirements. Deliverable 1.3, Centre for Software Reliability, University of Newcastle upon Tyne, June 1996.

- [49] Balasubramaniam Ramesh. Factors influencing requirements traceability practice. *Communications of the ACM*, 41(12):37–44, December 1998.
- [50] Brian Randel. Facing up to faults. *Computer Journal*, 43(2):95–106, 2000.
- [51] James Reason. *Managing the Risks of Organizational Accidents*. Ashgate Publishing Limited, 1997.
- [52] James Robertson and Suzanne Robertson. Volere: Requirements specification template. Technical Report Edition 6.1, Atlantic Systems Guild, 2000.
- [53] Suzanne Robertson and James Robertson. *Mastering the Requirements Process*. Addison-Wesley, 1999.
- [54] W. J. Salamon and D. R. Wallace. Quality characteristics and metrics for reusable software. Technical Report NISTIR 5459, NIST, 1994.
- [55] Norman 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.
- [56] Lui Sha, Rangunathan Rajkumar, and Michael Gagliardi. A software architecture for dependable and evolvable industrial computing systems. Technical Report CMU/SEI-95-TR-005, CMU/SEI, July 1995.
- [57] Ian Sommerville and Pete Sawyer. *Requirements Engineering: A Good Practice Guide*. John Wiley & Sons, 1997.
- [58] George Stark, Al Skillicorn, and Ryan Ameen. An examination of the effects of requirements changes on software releases. *CROSSTALK The Journal of Defence Software Engineering*, pages 11–16, December 1998.
- [59] Neil Storey. *Safety-Critical Computer Systems*. Addison-Wesley, 1996.
- [60] 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.
- [61] Gerald M. Weinberg. *Quality Software Management. Volume 4: Anticipating Change*. Dorset House, 1997.
- [62] David M. Weiss and Chi Tau Robert Lai. *Software Product-Line Engineering: A Family-Based Software Development Process*. Addison-Wesley, 1999.
- [63] Karl Eugene Wiegers. *Software Requirements*. Microsoft Press, 1999.
- [64] Robin Williams, Roger Slack, and James Stewart. Social learning in multimedia. Final report, EC targeted socio-economic research, project: 4141 PL 951003, Research Centre for Social Sciences, The University of Edinburgh, January 2000.
- [65] Didar Zowghi and Ray Offen. A logical framework for modeling and reasoning about the evolution of requirements. In *Proceedings of the Third IEEE International Symposium on Requirements Engineering*, pages 247–257, Annapolis, Maryland, USA, January 1997. IEEE Computer Society Press.