# Evolution before Birth – A Closer Look on Software Deterioration

Andreas Bollin
Software Engineering Research Group
Alpen-Adria Universität Klagenfurt
Klagenfurt, Austria
Email: Andreas.Bollin@aau.at

***Abstract*—The better our requirements engineering process is, the higher is the quality of the systems we build. But, environments and requirements change, and, by time, we also do know more about the systems that we are going to create or that we have to maintain. In all, this means that our specifications and documents, including the software code, are usually not just written once.**

**This talk sheds light onto the topic whether it is not just a burden to invest a lot of effort into re-design activities and refactoring of code, but also a necessity to constantly modify and extend our documents. Additionally, by extending the versioned stage model of software evolution, it demonstrates that there is a lot of benefit when re-design and refactoring tasks are combined with suitable measurement systems.**

## I. Motivation

Today's systems become more and more software-intensive which typically means that software is the major component that provides the needed functionality. With that, quality considerations gain in importance, too, and the examination of other disciplines lets us now start with an analogy and a question. Would you cross a bridge when there is a sign saying "Enter at your own risk"? I assume not. The bad news is that the situation is quite comparable to a lot of software systems around. Our standard software comes with license agreements stating that the author(s) of the software is (are) not responsible for any damage it might cause, and the same holds for a lot of our hardware drivers and many other applications around. Basically, we use them at our own risk.

The issue of quality (in all it's facets) implies a rigorous requirements engineering process, with formal methods adding additional value to the development process. The use of formal specifications enables refinement steps, and the additional documents bring in the advantages of assurance and reliable documentation, too. Though there are a lot of myths around [1], formal methods can be used in practice as companies using a formal software development process demonstrate [2]. Formal modeling is also not as inflexible as one might believe. Changing requirements and a growing demand in software involve more flexible processes and it is good to see that a combination of formal methods and the world of agile software development is possible [3]. It enables the necessary shorter development cycles, but, and this is the key issue, it also means to start thinking about evolution right from the beginning.

## II. Back to the Roots

Let us again start with the analogy above: Why does one cross a bridge even without bothering about its safety? The answer is simple: normally, one trusts in the original design, the statics, the teams that built it and the officials that did the final checks. The trust stays the same when the bridge is going to be renovated, when the design changes and when some bridge railings are broken down (or new ones are erected). One naturally assumes that the old plans have been investigated, all dependencies and side-effects have been considered, and that structural engineers finally took a look at it.

With this we arrive at the crucial point: from an engineering perspective the same situation holds (or should hold) for our software systems, too. There is an overall design, there are teams that build and modify it and there are teams that test it before being sold. We trust in their professionalism.

Changing environments or new demands (requirements) lead to a change in the software – the software is undergoing a "renovation" process, called software evolution. In the year 2000, Bennet and Rajlich [4] introduced a staged model to describe this process in more details (see Fig. 1). Starting with the initial development and the first running version of the software, evolutionary changes happen, leading to servicing phases and then, finally, to the phase-out and close-down versions of the software. In their article, the authors also point out the importance of software change for both, the evolution and servicing phases. In fact, effort is necessary at every step of the phase to keep up with the quality standards and for keeping alive our trust in it.

Taking a third time a closer look at our analogy of building/reconstructing a bridge, it can be observed that there is a chain (or network) of trust and dependencies. The pedestrian on the bridge counts on the construction company, they by themselves *trust* in the quality of the building materials they use, and the team that builds the bridge *trusts* in the architects (just to mention some of the links). And, there are clear *relations* between construction elements (pillars, bolts) in real life and the design documents, from structural analysis documents to materials control. Moreover, only with a rigorous measurement methodology at hand, and by looking at the elements and their dependencies, trust can be achieved at all stages of development and construction.

When our (evolving) software is undergoing changes, then there is a similar chain of trust and dependencies. This is

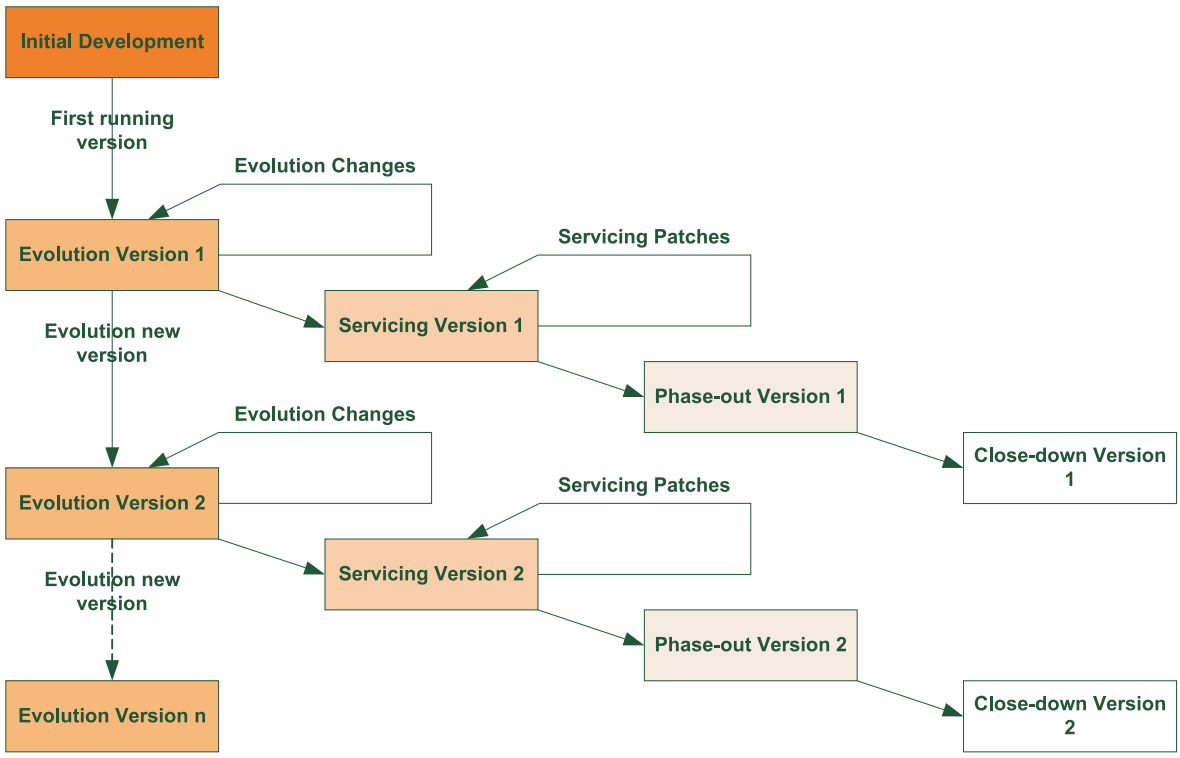


Fig. 1. The versioned staged model of Bennett and Rajlich [4]. Starting with the initial development and the first running version evolution is about to begin. The goal of the evolution phase is to adapt the software to ever changing user requirements or changes in the operating environment. When substantial changes are not possible anymore (at least not without damages to the system), then the servicing phase starts. Only small tactical changes are applied. When no more servicing is done, then the phase-out starts. Finally, with the close-down phase, the users are directed towards a new version.

the place where the following two issues mentions above come into play again: (a) a reliable and formal requirements engineering process, and (b) a constant assessment process, including measurement and prediction. To keep the trust, a change in the software has to be preceded by changes in the design documents and with it a change in the software specifications. And, all the measurement and calculations have to be improved and conducted again, allowing for new (and ideally, better) predictions and statements about the system.

One can also put it the other way round: when an architect does not update his or her plans, and the structural engineer is not involved anymore, then future renovations are (hopefully) impeded.

## III. An Extended Model of Evolution

Writing down requirements in a keen way is necessary, and the use of formality is not new to our community. In their article Black et al. point out that formal methods have already been used by Ada Lovelace's and Charles Babbage's work on the analytical engine when they verified the formulas [3]. Since then several success stories of the use of formal methods have been published [5], [6], [2], [7], [8]. However, traditional formal design is commonly seen as something that is just happening at the beginning, and most of us are tempted to think about the development of just *one* formal model. This viewpoint is exemplified in the versioned staged model of Bennett and Rajlich [4] in Fig. 1 as it only contains one stage of "Initial Development" (top left), with no change process (vertices) affecting the included artifacts.

As the analogy above demonstrates, this viewpoint is probably not correct. When drawing up a plan, an architect does not only draw a single version. He or she starts with a first version, modifies it, and plays around with it. Several versions are assessed and, finally, discussed with the customer. The plans are, in addition to that, revised when the bridge (...) is changed later on.

The same holds for our specification documents. They form the basis for incrementally pinning down the most important requirements (and eventually ongoing refinement steps). As already addressed in [9, p.243], only when kept up to date during evolutionary changes, specifications act as valid sources for comprehension activities. In the last years, our research group investigated a lot of software systems and specification documents, and we were able to demonstrate that the situation is not so trivial as assumed. In [10, p.77] it is then demonstrated that

*there is also evolution before the birth of the running version of the software system.*

The advances of continuous integration environments [12] and measurement approaches [13], [14] motivated us to take a closer look at properties of other types of documents, too. It turned out that the correlations between our specification documents and the related pieces of software code are high enough to allow for precise predictions. In recent works [15], [16] the feasibility of confidently predicting software measures based on specifications has been demonstrated. The correlations found between the different size-, structure-, and

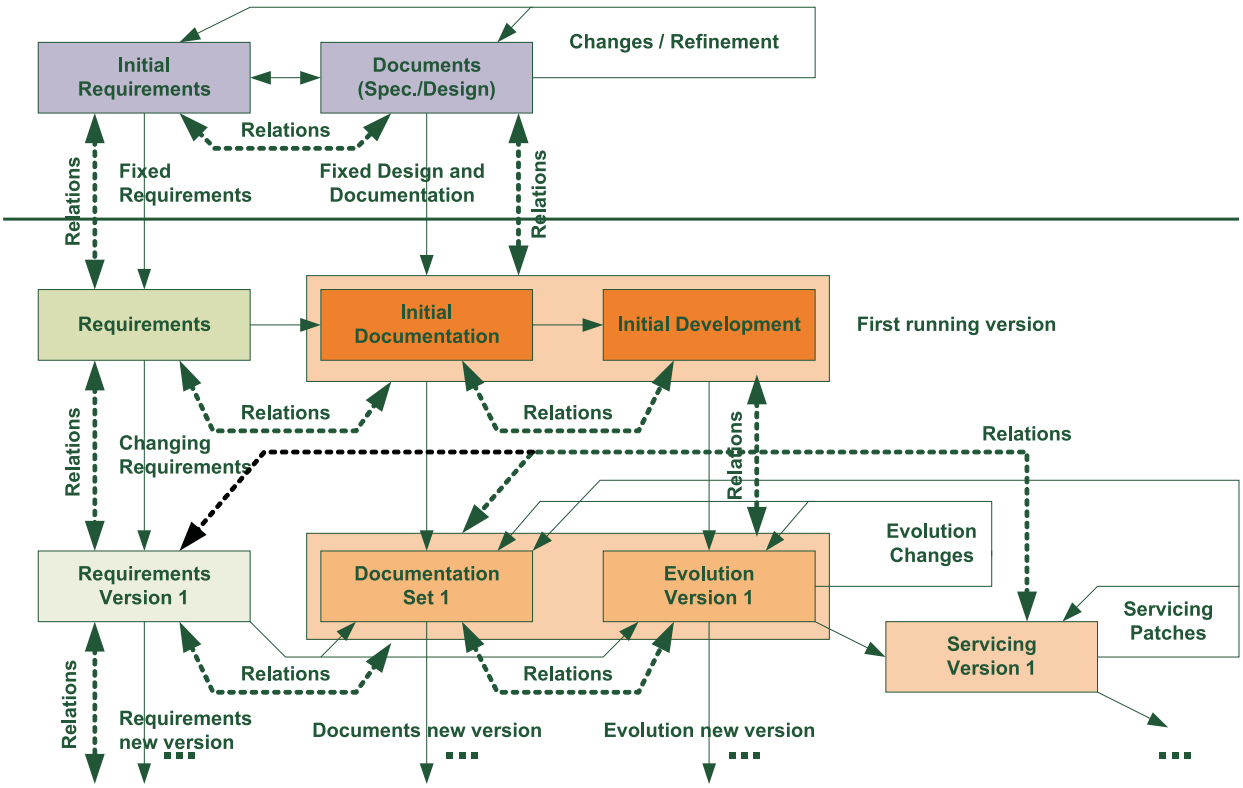


Fig. 2. A refined and extended version of the versioned stage model of Bennett and Rajlich [4] and Bollin [10, p.78]. Starting with a first set of initial requirements several versions of documents are created. Requirements are refined, and formal specifications are (among other design documents) also changed and modified. When the design is fixed, development is about to begin. Due to evolutionary changes after this phase, the existing documentation – including specifications and design documents – is (and has to be) changed, too. In addition to that, there are a lot of dependencies and correlations (referred to as "Relations") between parts of the documents. The form the basis for assessment activities and allow for estimates and predictions. The term "evolutionary change of a formal specification" is used in a rather general sense. Apart from the classification of system types of Lehman [11], the figure illustrates that essential changes might happen to documents before and after delivery.

semantics-based measures of formal Z specifications [17] and the implementation metrics promise of being able to predict size and complexity attributes as well as enables the estimation of likely costs and efforts. The work confirms the observations of Samson et al. [18] who conducted a similar study (but with only a fraction of experimental subjects) several years ago.

Fig. 2 now tries to exemplify these issues for the initial and evolutionary versions of the software. In this figure the original model of Bennett and Rajlich [4] has been extended by refining the boxes of the evolutionary versions. Documents and requirements have been added so that formal specifications are made explicit (as they do belong to the set of necessary documents). They are, depending on the changing requirements, also changed. The major difference is that these changes either happen before one has a first running version of the software or afterwards.

Another extension to the original model was to make the different relations between the documents even more visible. The relations are added to the figure on a very abstract level intentionally, pictured as dashed lines between the document boxes in Fig. 2. Relations, of course, can be available explicitly due to a descriptive text or the use of bug or issue tracking systems (where unique tokens or identifiers bind them together). But, they can also be hidden or unknown to the developer(s), like assumed correlations between the different quality or size-based measures which have to be calculated explicitly for the involved documents.

## IV. Implications

The implications of this (refined) picture are manifold and should be considered when using different types of documents in the software development life-cycle:

- Suitable size- and quality-based measures should be defined. This ensures that changes in the various documents – including formal specifications – can be detected and assessed. The measures introduced in the studies mentioned above are just an example of how it could be done for a specific specification language. For other languages the ideas can be reused. It might also be necessary to define new measures. However, the crucial point is that there is a measurement system around.
- Points of measurement should be introduced at every change/refinement loop. This ensures that the effects of changes can be assessed, and that the further direction of the development can be steered. By making use of a measurement system one is at least on the safe side.

- The terms "Fixed Design and Documentation" just designate the conceptual border between the initial development phase and the first running version. Nevertheless, changes to the documents happen before and after this milestone in the project (as evolution is about to begin). The previously introduced measure points should also be defined for the evolutionary phases, and measures should be collected during all the evolutionary changes and servicing activities (influencing the documents and specifications).

Basically, the extended model of software evolution makes one property of specifications (and other documents) explicit: they are no exception to aging. But, there are always a lot of relations we can count on. With this, it is obvious that measures, at the right point and extent, help in answering the question of what happened and, eventually, of what can be done for preventing unwanted deterioration effects.

The results of this contribution are interesting insofar as it turned out that, for the full benefits of a rigorous software development process, it makes sense to permanently take care of the quality of all the underlying document(s).

## REFERENCES


[1] J. P. Bowen and M. G. Hinchey, "Seven more myths of formal methods," *IEEE Software*, vol. 12, no. 4, pp. 34–41, July 1995.

[2] P. E. Ross, "The Exterminators," *IEEE Spectrum*, vol. 42, no. 9, pp. 36–41, September 2005.

[3] S. Black, P. P. Boca, J. P. Bowen, J. Gorman, and M. Hinchey, "Formal Versus Agile: Survival of the Fittest," *IEEE Computer*, vol. 42, no. 9, pp. 37–54, September 2009.

[4] K. Bennet and V. Rajlich, "Software Maintenance and Evolution: a Roadmap," in *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*. ACM New York, NY, USA, 2000, pp. 73–89.

[5] E. M. Clarke and J. M.Wing, "Formal Methods: State of the Art and Future Directions," Carnegie Mellon University, CMU-CS-96-178, Tech. Rep., 1996.

[6] J. Woodcock and J. Davis, *Using Z: Specification, Refinement, and Proof*, ser. Prentice-Hall International Series in Computer Science. Hemel Hempstead, Hertfordshire, UK: Prentice Hall, July 1996.

[7] M. Hinchey, M. Jackson, P. Cousot, B. Cook, J. P. Bowen, and T. Margaria, "Software engineering and formal methods," *Communications of the ACM*, vol. 51, no. 9, pp. 54–59, September 2008.

[8] R. M. Hierons, K. Bogdanov, J. P. Bowen, R. Cleaveland, J. Derrick, J. Dick, M. Gheorghe, M. Harman, K. Kapoor, P. Krause, G. Lüttgen, A. J. H. Simons, S. Vilkomir, M. R. Woodward, and H. Zedan, "Using formal specifications to support testing," *ACM Comput. Surv.*, vol. 41, no. 2, pp. 1–76, 2009.

[9] R. T. Mittermeir and A. Bollin, "Demand-Driven Specification Partitioning," *Lecture Notes in Computer Science*, vol. 2789, no. 2003, pp. 241–253, 2003.

[10] A. Bollin, "Is There Evolution Before Birth? Deterioration Effects of Formal Z Specifications," *Lecture Notes in Computer Science*, vol. 6991, no. Formal Methods and Software Engineering, pp. 66–81, 2011.

[11] M. M. Lehman, "On understanding laws, evolution, and conservation in the large-program life cycle," *Journal of Systems and Software*, vol. 1, no. 1, pp. 213–221, 1979.

[12] M. Fowler, *Continuous Integration: Improving Software Quality and Reducing Risk*. Pearson Education, 2007.

[13] T. M. Meyers and D. Binkley, "An Empirical Study of Slice-Based Cohesion and Coupling Metrics," *ACM Transactions on Software Engineering and Methodology*, vol. 17, no. 1, pp. 2:1–2:27, December 2007.

[14] A. Bollin, "Slice-based Formal Specifiation Measures – Mapping Coupling and Cohesion Measures to Formal Z," in *Proceedings of the Second NASA Formal Methods Symposium*, ser. NASA/CP-2010-216215, C. Muñoz, Ed. NASA, Langley Research Center, April 2010, pp. 24–34.

[15] A. Bollin and A. Tabareh, "Predictive Software Measures based on Z Specifications - A Case Study," in *2nd Workshop on Formal Methods in the Development of Software, co-located with the 18th International Symposium on Formal Methods, CNAM Paris, France*, 8 pages, August 2012.

[16] A. Bollin, "Metrics for quantifying evolutionary changes in Z specifications," *Journal of Software: Evolution and Process*, vol. 25, no. 9, pp. 1027–1059, September 2013.

[17] J. Spivey, *The Z Notation*, ser. C.A.R. Hoare Series. Prentice Hall, 1989.

[18] W. Samson, D. Nevill, and P. Dugard, "Predictive software metrics based on a formal specification," in *Information and Software Technology*, ser. 5, vol. 29, June 1987, pp. 242–248.