Evidence-Based Software Processes

Barry Boehm and Jo Ann Lane

University of Southern California
941 W. 37th Street
Los Angeles, California, 90089 USA
1-213-740-8163
{boehm,jolane}@usc.edu

Abstract. Many software projects fail because they commit to a set of plans and specifications with little evidence that if these are used on the project, they will lead to a feasible system being developed within the project's budget and schedule. An effective way to avoid this is to make the evidence of feasibility a first-class developer deliverable that is reviewed by independent experts and key decision milestones: shortfalls in evidence are risks to be considered in going forward. This further implies that the developer will create and follow processes for evidence development. This paper provides processes for developing and reviewing feasibility evidence, and for using risk to determine how to proceed at major milestones. It also provides quantitative result on "how much investment in evidence is enough," as a function of the project's size, criticality, and volatility.

Keywords: Evidence-based development, feasibility evidence, Incremental Commitment Model, evidence-based reviews, risk assessment, risk management.

1 Introduction

We have found through experience, case study analysis, and cost-benefit analysis that up-front investments in providing evidence that the specifications and plans for a software-intensive system satisfies a set of desired properties produce positive payoffs later. A good way to capture this evidence is in a Feasibility Evidence Description (FED). The FED's evidence, when validated by independent experts, can also be used as a basis for assuring system stakeholders that it is reasonable for the project to proceed into development.

Providing such validated evidence is generally considered to be a good practice, but it generally fails to be done well. This is because of a lack of proof criteria, proof-generation procedures; measures for monitoring proof generation; methods, standards, and contractual provisions that make proof generation optional; and an appreciation of the consequences of proceeding into development with unsubstantiated specifications and plans. The main contributions of this paper are to provide experience-based approaches for dealing with each of these concerns, and to show the consequences of their use via case studies and parametric analysis.

Y. Yang, J. Münch, and W. Schäfer (Eds.): ICSP 2010, LNCS 6195, pp. 62–73, 2010.

2 Evidence Shortfalls in Current Software Practices

Evidence shortfalls can be found in both technical and management practices. The key to success is identifying critical risk elements and focusing engineering efforts to reduce risks through evidence-based methods. The following sections elaborate on the types of technical and management issues that should drive a project to collect further evidence of feasibility.

2.1 Technical Shortfalls

Current software design and development methods focus strongly on the inputs and outputs, preconditions and post-conditions that a software function, object, or service operates by as a product. They generally lack adequate capabilities to support analyses about how well the elements perform, how expensive they will be to develop, or how compatible are their underlying assumptions. In principle, they support reasoning about off-nominal performance, but in practice their descriptions generally focus on sunny-day scenarios. As a result, many software project reviews tend to focus on exhaustive presentations of PowerPoint charts and UML diagrams. They provide little evidence that the system they describe could handle rainy-day scenarios; perform adequately on throughput, response time, safety, security, usability, or other desired quality attributes across a range of representative mission scenarios; be buildable within the available budgets and schedules in the plan; or to generate positive returns on investment for the stakeholders.

Most current versions of model-driven development, for example, strongly focus on expressing product capabilities and relationships, and on reasoning about their combined incompatibilities and incompleteness. However, analyses of failed projects such as the one shown in figure 1 of the Bank of America (BofA) Master Net project [7, 8] find that incompatibilities among product models and other stakeholder value models (process models, property models, success models) are at least as frequent and important as product-product (PD-PD) model clashes.

For example, the MasterNet users specified 3.5 million source lines of code (3.5 MSLOC) worth of wish-list features that were put into the project's requirements specification. Even at an extremely optimistic development cost of $30/SLOC for a project of this size, this would cost $105 million. Thus, the users' product model was in serious conflict with the acquirers' property model of a $22 million budget. Also, many of the wish-list items had no mission effectiveness rationale, conflicting with the acquirers' success model.

Faced with this dilemma, the acquirers searched for a supplier who could reuse a related solution to provide useful features at a low cost. They found one in Premier Systems, who had built similar applications for small banks. However, their product model only worked on Prime computers, which conflicted with the BofA users' and maintainers' product-model value propositions of applications compatibility and ease of maintenance, given that BofA was an IBM mainframe operation. Also, the Prime host could not scale up to BofA's throughput, response time, and reliability needs, causing a property-product evidence shortfall that should have been addressed earlier.

Overall, the implications here, both with the BofA model clashes in red /gray in figure 1 and with further model clashes in black from analyses of other failed projects, is that there are many potential value-proposition conflicts just among the four main stakeholders in a project, and that product-product evidence shortfalls are only a moderate fraction of the total set of issues of concern.

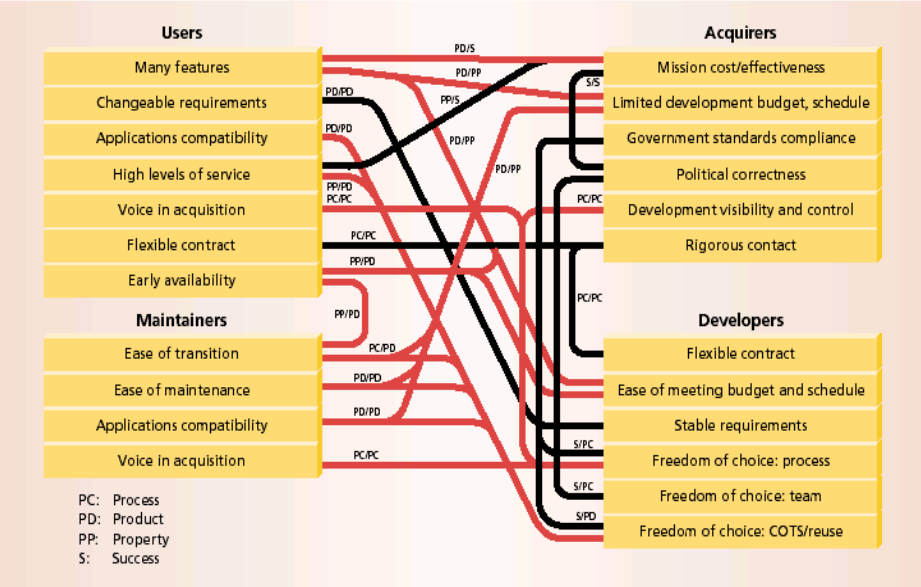


Fig. 1. Stakeholder Value Model Clashes. The red or gray lines show model clashes from the failed MasterNet system.

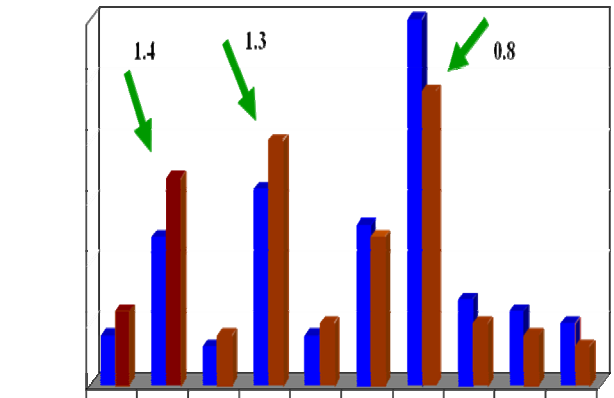
Al Said's analysis shown in Figure 2 of the risk patterns across 35 electronic-services projects found that product-product model incompatibilities accounted for only 30% of the model clashes and only 24% (ratio of 0.8) of the risk-priority values. [1]

2.2 Management Shortfalls

A major recent step forward in the management of outsourced government projects has been to move from schedule-based reviews to event-based reviews. A schedule-based review says basically that, "The contract specifies that the Preliminary Design Review (PDR) will be held on April 1, 2011, whether we have a design or not."

In general, neither the customer nor the developer wants to fail the PDR, so the project goes into development with the blessing of having passed a PDR, but with numerous undefined interfaces and unresolved risks. As we will show below, these will generally result in major project overruns and incomplete deliveries.

An event-based review says, "Once we have a preliminary design, we will hold the PDR." Such a review will generally consist of exhaustive presentations of sunny-day PowerPoint charts and UML diagrams, and focus on catching the 30% of the project



	Success-Property	Success-Product	Success-Success	Product-Property	Process-Property	Property-Property	Product-Product	Process-Process	Product-Process	Success-Process
%of Clashes	4	12	3	16	4	13	30	7	6	5
%of Risk	6	17	4	20	5	12	24	5	4	3

Fig. 2. Frequency of Project Model Clash Types

model clashes of the product-product form. But in general, it will still have numerous other model clashes that are unidentified and will cause extensive project rework, overruns, and incomplete deliveries.

Similarly, most outsourcing contracts focus on product-oriented deliverables and reviews. These reinforce paths toward project disaster, as in the quote from a recent large-project manager, *"I'd like to have some of my systems engineers address those software quality-factor risks, but my contract deliverables and award fees are based on having all of the system's functions defined by the next review."* Similar overfocus on product definition is found in project earned-value management systems for tracking project progress and data item descriptions (DIDs) for deliverables. Most contract DIDs cover function, interface, and infrastructure considerations, but place demonstration of their feasibility in optional appendices where, as with the project manager above, they are the first to go when time and effort are running out.

3 Consequences of Evidence Shortfalls

The biannual Standish Reports consistently identify shortfalls in evidence of feasibility with respect to stakeholder objectives as the major root causes of project failure. For example, the 2009 Standish Report [14] found that only 32% of the 9000 projects reported delivered their full capability within their budget and schedule; 24% were cancelled; and 44% were significantly over budget, over schedule, and/or incompletely delivered. More detail on the top critical success factors distinguishing successful from failed software projects was in the 2005 Standish Report. There, 71% of the sources of failure were primarily due to evidence shortfalls with respect to stakeholder objectives (lack of user involvement, executive support, clear requirements, proper planning, and realistic expectations).

Recent further analyses of the COCOMO II database on the effects of incomplete architecture definition, risk resolution, and resulting feasibility evidence are shown in Figure 3. They show the results of a risk-driven "how much architecting is enough" analysis, based on the COCOMO II Architecture and Risk Resolution (RESL) factor [4]. This factor was calibrated along with 22 other factors to 161 project data points. It relates the amount of extra rework effort on a project to the degree of evidence that the project had demonstrated its architecture feasibility and resolved its technical and management risks. This also correlates with the percent of project effort devoted to software-intensive system architecting and risk resolution.

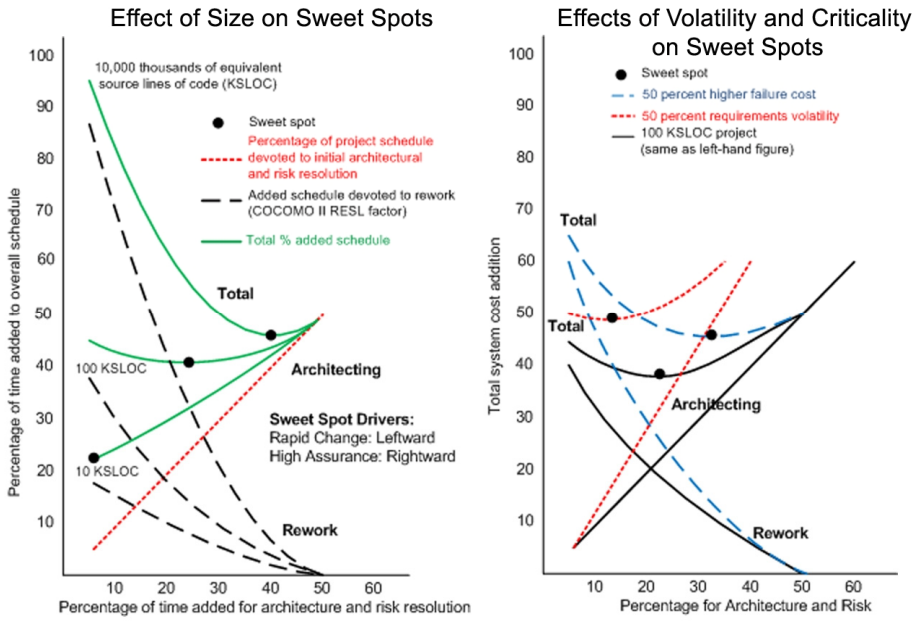


Fig. 3. Architecture Evidence Shortfall Penalties and Resulting Architecting Investment Sweet Spots

The analysis indicated that the amount of rework was an exponential function of project size. A small (10 thousand equivalent source lines of code, or KSLOC) project could fairly easily adapt its architecture to rapid change via refactoring or its equivalent, with a rework penalty of 18% between minimal and extremely thorough architecture and risk resolution.

However, a very large (10,000 KSLOC) project would incur a corresponding rework penalty of 91%, covering such effort sources as integration rework due to undiscovered large-component interface incompatibilities and critical performance shortfalls.

The effects of rapid change (volatility) and high dependability (criticality) on the architecture evidence shortfall penalties and resulting architecting investment sweet spots are shown in the right hand graph. Here, the solid black lines represent the average-case cost of rework, architecting, and total cost for a 100-KSLOC project as

shown at the left. The dotted red lines show the effect on the cost of architecting and total cost if rapid change adds 50% to the cost of architecture and risk resolution. Quantitatively, this moves the sweet spot from roughly 20% to 10% of effective architecture investment (but actually 15% due to the 50% cost penalty). Thus, high investments in architecture, feasibility analysis, and other documentation do not have a positive return on investment for very high-volatility projects due to the high costs of documentation rework for rapid-change adaptation.

The dashed blue lines at the right represent a conservative analysis of the external cost effects of system failure due to unidentified architecting shortfalls. It assumes that the costs of architecting shortfalls are not only added development project rework, but also losses to the organization's operational effectiveness and productivity. These are conservatively assumed to add 50% to the project-rework cost of architecture shortfalls to the organization. In most cases for high-assurance systems, the added cost would be considerably higher.

Quantitatively, this moves the sweet spot from roughly 20% to over 30% as the most cost-effective investment in architecting and development of feasibility evidence for a 100-KSLOC project. It is good to note that the "sweet spots" are actually relatively flat "sweet regions" extending 5-10% to the left and right of the "sweet spots." However, moving to the edges of a sweet region increases the risk of significant losses if some project assumptions turn out to be optimistic.

The bottom line for Figure 3 is that the greater the project's size, criticality, and stability are, the greater is the need for validated architecture feasibility evidence (i.e., proof-carrying specifications and plans). However, for very small, low-criticality projects with high volatility, the proof efforts would make little difference and would need to be continuously redone, producing a negative return on investment (the same could be said for proof-carrying code). In such cases, agile methods such as Scrum [15] and eXtreme Programming [2] will be more effective. Overall, evidence-based specifications and plans will not guarantee a successful project, but in general will eliminate many of the software delivery overruns and shortfalls experienced on current software projects.

4 Evidence Criteria and Review Milestone Usage

Having shown that the regions of high payoff for evidence-based specifications and plans are extensive and enterprise-critical, it is now important to define the criteria for the evidence that will be associated with the system development specifications and plans. The criteria are extensions to those for the Anchor Point milestones defined in [3] and adopted by the Rational Unified Process (RUP) [9, 12]. The extensions have been incorporated into the recently-developed Incremental Commitment Model; details of this model can be found in [6, 11].

The evidence criteria are embodied in a Feasibility Evidence Description (FED). It includes *evidence* provided by the developer and validated by independent experts that, *if the system is built to the specified architecture it will:*

1. Satisfy the specified operational concept and requirements, including capability, interfaces, level of service, and evolution
2. Be buildable within the budgets and schedules in the plan

3. Generate a viable return on investment
4. Generate satisfactory outcomes for all of the success-critical stakeholders
5. Identify shortfalls in evidence as risks, and cover them with risk mitigation plans.

A FED does not assess a single sequentially developed system definition element, but the consistency, compatibility, and feasibility of several concurrently-engineered elements. To make this concurrency work, a set of Anchor Point milestone reviews are performed to ensure that the many concurrent activities are synchronized, stabilized, and risk-assessed at the end of each phase. Each of these reviews is focused on developer-produced and expert-validated *evidence*, documented in the FED (or by reference to the results of feasibility analyses), to help the system's success-critical stakeholders determine whether to proceed into the next level of commitment. Hence, they are called Commitment Reviews.

The FED is based on evidence from simulations, models, or experiments with planned technologies and increasingly detailed analysis of development approaches and project productivity rates. The parameters used in the analyses should be based on measured component performance or on historical data showing relevant past performance, cost estimation accuracy, and actual developer productivity rates.

A shortfall in feasibility evidence indicates a level of program execution uncertainty and a source of program risk. It is often not possible to fully resolve all risks at a given point in the development cycle, but known, unresolved risks need to be identified and covered by risk management plans, including the necessary staffing and funding to address them.

The nature of the evidence shortfalls, the strength and affordability of the risk management plans, and the stakeholders' degrees of risk acceptance or avoidance will determine their willingness to commit the necessary resources to proceed. A program with risks is not necessarily bad, particularly if it has strong risk management plans. A program with no risks may be high on achievability, but low on ability to produce a timely payoff or competitive advantage.

A FED needs to be more than just traceability matrices and PowerPoint charts. Evidence can include results of:

- Prototypes: of networks, robots, user interfaces, COTS interoperability
- Benchmarks: for performance, scalability, accuracy
- Exercises: for mission performance, interoperability, security
- Models: for cost, schedule, performance, reliability; tradeoffs
- Simulations: for mission scalability, performance, reliability
- Early working versions: of infrastructure, data fusion, legacy compatibility
- Previous experience
- Combinations of the above.

The type and level of evidence generated depends on perceived risks and their associated levels with no evidence. To determine what feasibility evidence to generate, one must evaluate the expected value of the evidence with respect to lowering risks going forward and the cost and schedule to generate the evidence versus the costs if the perceived risk is realized. If there is significant risk exposure in making the wrong decision and feasibility evidence can be cost-effectively generated, then steps should be initiated

to reduce risks early. Feasibility evidence costs can be minimized by focusing on prototypes that can become part of the system or early working versions of key parts of the system. Spending time and money on early feasibility evidence should be viewed as buying information to reduce both risk and the cost of potential rework (assuming that the problem can be resolved later in the project) or to avoid canceling the project altogether after expending considerable resources. Likewise, if risk exposure is low, then comprehensive, detailed feasibility evidence is of less value.

Not only does the evidence need to be produced, but it needs to be validated by independent experts. These experts need to determine the realism of assumptions, the representativeness of scenarios, the thoroughness of analysis, and the coverage of key off-nominal conditions. The risk of validating just nominal-case scenarios is shown in the next section.

The following is a set of criteria that cover the various dimensions of validity for the evidence:

- Data well defined
  a. What is counted
  b. How it is derived (e.g., how measured, calculated, or inferred)
- Representative mission scenarios
  a. Operational environment
  b. Adversaries
  c. Component reliability, scalability, etc.
  d. Nominal and off-nominal scenarios
  e. Treatment of uncertainty
  f. Composability of results; interference effects
  g. Scale
- Parameter values realistic
  a. Based upon measured results
  b. Inferred from representative projects/activities
  c. Derived from relevant scenarios
- Outputs traceable to mission effectiveness
  a. Directly/indirectly
  b. Based on appropriate models, scenarios
- Models verified and validated
  a. Via representative scenarios
  b. Limiting cases, off-nominals realistic

This should be used as a checklist and not as a one-size-fits-all set of criteria, which would generally be overkill.

5 FED Development Process Framework

The most important characteristic of evidence-based software specifications and plans is that *if the evidence does not accompany the specifications and plans, the specifications and plans are incomplete.*

Thus, event-based reviews, where the event is defined as production of the specifications and plans, need to be replaced by *evidence-based reviews*.

This does not mean that the project needs to spend large amounts of effort in documenting evidence of the feasibility of a simple system. As described above, the appropriate level of detail for the contents of the FED is based on the perceived risks and criticality of the system to be developed. It is NOT a "one size fits all" process, but rather a framework to help developers and stakeholders determine the appropriate level of analysis and evaluation. As with reused software, evidence can be appropriately reused. If a more complex system than the one being reviewed has been successfully developed by the same team, a pointer to the previous project's evidence and results will be sufficient. Table 1 outlines a process that can be used for developing feasibility evidence.

The steps are designated by letters rather than numbers, to indicate that they are carried out with considerable concurrency rather than sequentially. The process clearly depends on having the appropriate work products for the phase (Step A). As part of the engineering work, the high-priority feasibility assurance issues are identified that are critical to the success of the system development program (Step B). These are the issues for which options are explored, and potentially viable options further investigated (Step C). Clearly, these and the later steps are not performed sequentially, but concurrently.

Since the preliminary design and plans are incomplete without the FED, it becomes a first-class project deliverable. This implies that it needs a plan for its development, and that each task in the plan needs to be assigned an appropriate earned value. If possible, the earned value should be based on the potential risk exposure costs, not the perceived available budget.

Besides monitoring progress on developing the system, the project needs to monitor progress on developing the feasibility evidence. This implies applying corrective action if progress falls behind the plans, and adapting the feasibility evidence development plans to changes in the project objectives and plans. If evidence generation is going to be complex, it is generally a good idea to perform pilot assessments.

6 Experiences with Evidence-Based Reviews

AT&T and its spinoffs (Telcordia, Lucent, Avaya, and regional Bell companies) have been successfully using versions of evidence-based reviews since 1988. On average, there has been a 10% savings per reviewed project, with substantially larger savings on a few reviewed projects. More detail on their Architecture Review experience is in [10].

The million-line TRW CCPDS-R project summarized in [12] by Walker Royce, and several similar TRW follow-on projects were further successful examples. Evidence-based anchor point milestone reviews are also an integral part of the Rational Software Process, with many successful implementations [13], although a good many RUP applications have been unable to succeed because of the type of contractual constraints discussed in Section 2.

The highly successful Abbott Laboratories' next generation intravenous infusion pump documented in chapter 5 of [11] is a good commercial example of evidence-based specifications and plans.

Table 1. Steps for Developing a FED

Step	Description	Examples/Detail
A	Develop phase work-products/artifacts	For a Development Commitment Review, this would include the system's operational concept, prototypes, requirements, architecture, life cycle plans, and associated assumptions
B	Determine most critical feasibility assurance issues	Issues for which lack of feasibility evidence is program-critical
C	Evaluate feasibility assessment options	Cost-effectiveness; necessary tool, data, scenario availability
D	Select options, develop feasibility assessment plans	The preparation example in Figure 5 below is a good example
E	Prepare FED assessment plans and earned value milestones	The preparation example in Figure 5 below is a good example
F	Begin monitoring progress with respect to plans	Also monitor changes to the project, technology, and objectives, and adapt plans
G	Prepare evidence-generation enablers	Assessment criteria Parametric models, parameter values, bases of estimate COTS assessment criteria and plans Benchmarking candidates, test cases Prototypes/simulations, evaluation plans, subjects, and scenarios Instrumentation, data analysis capabilities
H	Perform pilot assessments; evaluate and iterate plans and enablers	Short bottom-line summaries and pointers to evidence files are generally sufficient
I	Assess readiness for Commitment Review	Shortfalls identified as risks and covered by risk mitigation plans Proceed to Commitment Review if ready
J	Hold Commitment Review when ready; adjust plans based on review outcomes	See Commitment Review process overview below.

NOTE: "Steps" are denoted by letters rather than numbers to indicate that many are done concurrently.

Lastly, the University of Southern California (USC) has used evidence-based specifications and plans to achieve a 92% on-time, in-budget, satisfied-client success rate on over 50 fixed-schedule local-community e-service projects [6].

7 Conclusions

The most important characteristic of evidence-based software specifications and plans is that *if the evidence does not accompany the specifications and plans, the specifications and plans are incomplete.*

Thus, event-based reviews, where the event is defined as production of the specifications and plans, need to be replaced by *evidence-based reviews*.

The Anchor Point milestones and Feasibility Evidence Descriptions presented in this paper provide an approach for successfully realizing the benefits of proof-carrying software specifications and plans. Further, the Anchor Point milestones enable synchronization and stabilization of concurrent engineering.

The parametric analysis in Section 3 concludes that the greater the project's size, criticality, and stability are, the greater is the need for validated architecture feasibility evidence. However, for very small, low-criticality projects with high volatility, the evidence generation efforts would make little difference and would need to be continuously redone, producing a negative return on investment. In such cases, agile methods such as Scrum and eXtreme Programming will be more effective. Overall, evidence-based specifications and plans will not guarantee a successful project, but in general will eliminate many of the software delivery overruns and shortfalls experienced on current software projects.

Some implications of defining the FED as a "first class" project deliverable are that it needs to be planned (with resources), and made part of the project's earned value management system. Any shortfalls in evidence are sources of uncertainty and risk, and should be covered by risk management plans. The main contributions of this paper are to provide experienced-based approaches for evidence criteria, evidence-generation procedures, and measures for monitoring evidence generation, which support the ability to perform evidence-based software engineering. And finally, evidence-based specifications and plans such as those provided by the FED can and should be added to traditional milestone reviews.

References

1. Al Said, M.: Detecting Model Clashes During Software Systems Development. Doctoral Thesis. Department of Computer Science, University of Southern California (2003)
2. Beck, K.: Extreme Programming Explained. Addison-Wesley, Reading (1999)
3. Boehm, B.: Anchoring the Software Process. IEEE Software, 73–82 (July 1996)
4. Boehm, B., et al.: Software Cost Estimation with COCOMO II. Prentice Hall, Englewood Cliffs (2000)
5. Boehm, B., Lane, J.: Incremental Commitment Model Guide, version 0.5 (2009)
6. Boehm, B., Lane, J.: Using the ICM to Integrate System Acquisition, Systems Engineering, and Software Engineering. CrossTalk, 4–9 (October 2007)
7. Boehm, B., Port, D., Al Said, M.: Avoiding the Software Model-Clash Spiderweb. IEEE Computer, 120–122 (November 2000)
8. Glass, R.: Software Runaways. Prentice Hall, Englewood Cliffs (1998)
9. Kruchten, P.: The Rational Unified Process. Addison-Wesley, Reading (1999)
10. Maranzano, J., et al.: Architecture Reviews: Practice and Experience. IEEE Software (March/April 2005)
11. Pew, R., Mavor, A.: Human-System Integration in the System Development Process: A New Look. National Academy Press, Washington (2007)
12. Royce, W.: Software Project Management. Addison-Wesley, Reading (1998)

13. Royce, W., Bittner, K., Perrow, M.: The Economics of Iterative Software Development. Addison-Wesley, Reading (2009)
14. Standish Group 2009. CHAOS Summary (2009), http://standishgroup.com
15. Schwaber, K., Beedle, M.: Agile Software Development with Scrum. Prentice Hall, Englewood Cliffs (2002)