Financial Management and Control of Iterative Software Processes

Advances in Similarity-Based Calculation Methods

Alexander Baumeister
Chair of Managerial Accounting
Saarland University
Saarbruecken, Germany
a.baumeister@con.uni-saarland.de

Markus Ilg
Department of Management and Business Administration
Vorarlberg University of Applied Sciences
Dornbirn, Austria
markus.ilg@fhv.at

***Abstract* — Combining mainstream software process models and activity based costing enables new and improved approaches to the financial management and control of software processes. A similarity-based approach is presented and a solution to one of the key problems, the sufficient size of the similarity database is given – by introducing the concept of normalized iteration types. It is shown how to generate normalized iterations as well as how these have to be applied to calculate new software processes. Iteration signatures are defined and their application to form clusters of iterations as well as to easily search the database for similar iteration types is discussed. To further support management and control of projects up and running, ideas taken from the earned value method are adopted.**

Keywords — management accounting; activity based costing; earned value method; iterative software processes; unified process; agile software development

I. Introduction: Shortcomings in Financial Management of Iterative Software Processes

Software development processes have a long tradition and at the same time show a remarkable development and adaption to the advances in the development of software. Starting with the well-known and rather sequential waterfall process [36] software processes became more detailed and have been enriched with document frameworks, process models, design templates and much more. Important examples are *Boehm's* spiral model [10], which can be described as a risk-oriented approach, the German V-model XT [19] and the Unified Process [23,29], both models have significant iterative elements, and in recent years one can find a variety of so called agile methods [31]. *Sommerville* distinguishes specification-based models in contrast to evolutionary development models [40], the latter covering the agile approaches as well as for instance the Unified Process. Evolutionary models take into consideration, that it is difficult, if not impossible, to fully describe the requirements of a software system at the beginning of a software process.

Currently, agile methods are discussed intensively and there is some hope, that they will help to improve the software development process in terms of more satisfied customers, less cost overruns and better organized development processes [31]. Agile methods are incremental development methods with small increments and new system releases provided to the customer every two or three weeks. The early and intensive customer involvement allows rapid feedback und results in better software [31,41]. Well known agile approaches are Extreme Programming [8], Scrum [38,39], Chrystal [14] and Feature Driven Development [34].

However, the popular paradigm of agility, very well known as the agile manifesto [9], also arises some problems, especially when focusing on the cost-and time-dimensions of a project. At the heart of agile processes is the immediate customer feedback to even the earliest results of the software development. Providing working software is one of the most important aspects of agile software processes. In consequence it is rather difficult to estimate cost and time the software development will require, because "embracing change" can and also will result in changing cost and time requirement to finalize a project [33,41].

Modern software development processes therefore require new or improved management tools supporting the financial planning and control of iterative software developments. It is of great relevance, that these tools will take into account the structure of the development process, because this will allow tracking the progress made over time in a natural way. However, typical approaches for software cost estimation are ignoring the structure of the software process underlying and are concentrating on other more or less influential variables, ranging from counting source lines of code to the estimation of the experience of the development team. For an overview see [11,26,27].

An alternative, activity based approach not only to estimate software costs, but also to manage and control software processes is summarized in section II – it has been developed and explained elsewhere [5,22]. The focus then shifts to the foundations needed to calculate software projects based on similarity calculations, because these are an important element of the described activity based approach. Section III shows practical applications using this approach, implementing some concepts related to earned value analysis. Section IV summarizes the paper and outlines some further research topics.

II. Taking Financial Software Process Management and Control One Step Further

A. Brief Introduction to the Unified Process

An important and widely used software process is the rational **Unified Process** [23,29], which is a use-case-driven, user-centric and incremental development approach. It consists of iterations which can be thought as small, time-boxed mini-projects, each of them adding at least a small piece of functionality to the final product [22]. The Unified Process has therefore implemented important agile features. Despite that, it is sometimes called a heavyweight approach [18], because it is highly structured and offers numerous (but optional) templates as well as software support.

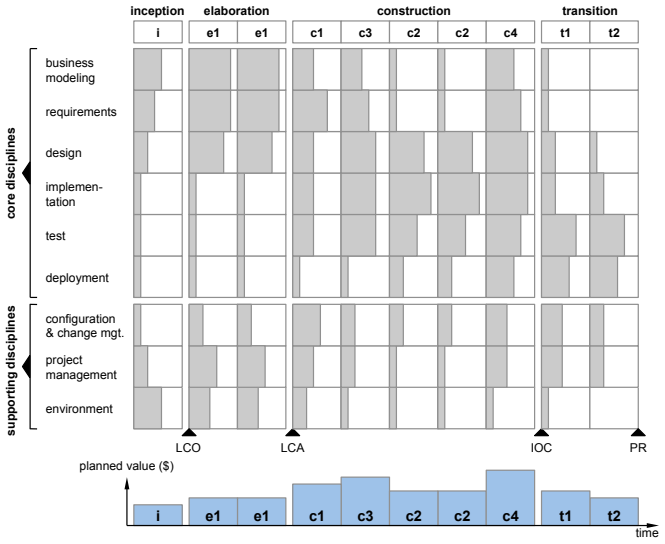

Figure 1. Dynamic (horizontal) and static (vertical) perspective of the Unified Process and corresponding planned cost values

In its **dynamic perspective** along the time axis it consists of four phases: inception, elaboration, construction and transition (see Figure 1); all phases except inception are made up of one ore more iterations of equal length, each iteration delivering a tested and integrated subsystem of the final product [30]. **Inception** defines the vision and the business case and provides an approximate time and cost estimate. In **elaboration** the core architecture and most requirements are defined, though on a rather general level, high risk and high value aspects are tackled with priority. Use cases are refined and time and cost estimates are recalculated based on the improved understanding of the product under development. In **construction** most implementation takes place, but business modeling, requirements and design are still of importance, because new or changed customer needs have to be considered. Beta tests and deployment are the main tasks in **transition**. Each phase terminates with a milestone; these are the lifecycle objective milestone (LCO), the lifecycle architecture milestone (LCA), the initial operational capability milestone (IOC) and the product release milestone (PR). [28,30,42] The **static structure** of the Unified Process is described by six core and three supporting disciplines (see Figure 1). All disciplines can be found in each iteration, however, their intensity differs. Within early iterations, business modeling and requirements are more important than implementation or test, to give an example.

Unlike the usual visualizations of the Unified Process our representation can be described as iteration focused, because it carves out the typical intensities of disciplines within different iterations, showing that sometimes iterations are repeated more or less identically (there are two iterations of type e1 and two iterations of type c2 in Figure 1).

B. Activity Based Costing of Iterative Software Processes

As shown earlier [5,6] **structural affinities** between the Unified Process and activity based costing (ABC) can be found (for an ABC-overview see [20,21,24,25]). The Unified Process' disciplines may be interpreted as activity groups known in the context of activity based cost management and - even more important - if it is accepted, that a software process consists of typical iteration types, these correspond to the cost drivers required to calculate planned values in activity based management.

Planning and estimating costs of software processes changes fundamentally using the **activity based approach.** Traditional measures like source lines of code, function points [1,2,35] or story points [33] are dismissed in favor of counting iteration types. Obviously, experience is required to transform the idea of a new or adapted software product into a plan of iterations – but in contrast to other proposals this approach is more closely linked to the core competences of every software engineer: the development process itself. Therefore better estimates can be expected and – as will be shown in section III – it serves as an ideal basis for the financial management and control of software processes.

But before new software processes may be calculated using the activity based approach on the basis of typical iteration types, a concept for the calculation, storage and retrieval of suitable iteration types has to be defined. The software engineer is confronted with several problems:

- How can iterations of different software projects be described in a generic way?
- Is it possible to refer to iterations of projects of different iteration length and of different staffing intensity?
- Is it possible to apply the calculation method despite of only few historic projects?
- How to changing wages have to be taken into account?

C. Normalization of Iteration Data as the Key to Establish Similarity Databases

The problem of a representative set of software projects to start the similarity calculation is not new [13] and its importance seems to increase, the greater the variety of software developed within a software company or development department is.

However, we suggest creating normalized descriptions of iterations, which focus on their disciplines' intensities. **Normalized iterations** are advantageous, because they accommodate for different iteration length in different projects as well as for different average intensities of personnel placement, expressed in full time equivalents (FTE).

Figure 2 shows the database input for an exemplary iteration (name CP2012-34-C02). It is recorded, that this iteration is part of the construction phase and that it started at the end of October. The iteration belongs to a development project for an IFRS Valuation Tool; the iteration length is two weeks. On an additional screen, which can be reached by clicking on the FTE-cells, all project members and the time, they worked within each discipline, is recorded resulting in an overall FTE average per week for each discipline and a corresponding weekly average for the respective personnel costs.

Iteration Input — Iteration Calculator v1.03

project	CP2012-34	IFRS 9 Valuation Tool	
project head	Anette Scharf		
duration (start, end)	4-Jun-12	14-Dec-12	*28 weeks*
iteration length	2 weeks		*14 iterations*
iteration	CP2012-34-C02		
phase, signature	Construction		***116952211***
duration (start, end)	29-Oct-12	9-Nov-12	*2 weeks*

core disciplines	FTE per week	personnel costs (avg per week)	personnel costs (total)
business modeling	0,4	1.120 $	896 $
requirements	0,4	1.311 $	1.049 $
design	2,1	1.253 $	5.263 $
implementation	3,5	768 $	5.376 $
test	2,0	768 $	3.072 $
deployment	1,0	768 $	1.536 $
supporting disciplines			
conf. & change mgt.	0,7	1.202 $	1.683 $
project management	0,5	1.520 $	1.520 $
environment	0,5	768 $	768 $
iteration total	11,1	953 $	21.162 $

additional information	
modelling enironment	UML, ARIS 9.0
development platform	eclipse 3.0
languages	C#, Java
application type	app

Figure 2. Acquisition of iteration data in the similarity database

The normalization procedure further transforms the weekly FTE averages of each discipline into the iterations' signature. Signatures have been described earlier in the context of activity based software cost estimation [22] and can also be found in some older software cost estimation approaches, though with different implementation details; for details see the text on EDB-methods in [13].

We define signatures as 9-digit-numbers characterizing an iteration, because each digit represents the relative intensity of its corresponding discipline. To calculate the signature, the weekly FTE for each discipline is divided by the maximum weekly FTE of all disciplines and multiplied by 10, resulting in a number in the range from 0 to 10. The integer part of the result is defined as the digit representing the disciplines' relative intensity; 10 is assigned the digit 9. This procedures results in 10 intervals [0;1[, [1;2[, [2;3[… [9;10] with 0, 1, 2, … 9 as interval names. In Figure 3, the signature of iteration CP2012-34-C02 is 116952211. It describes this iteration as one with low intensity in business modeling (1), requirements (1), moderate intensity in design (6), high intensity in implementation (9) and so on. Obviously, the signature does not provide any information concerning the absolute FTEs: it only tells us something about the relative intensity of its disciplines.

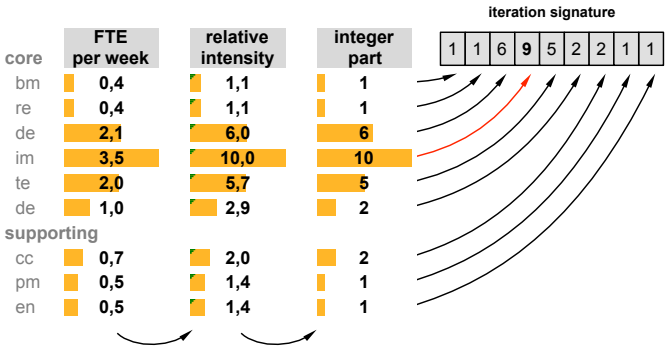

Figure 3. Calculation of the iteration's signature

D. *Applications for Iteration Signatures*

Having captured all historic projects, their iteration signatures offer a smart access to clustering iterations, creating meaningful iteration types and searching for similar iterations within the similarity database. First of all, the signatures can be used to calculate a **similarity index** when comparing different iterations. To do so, we recommend calculating the sum of the squared differences of each of the signatures' digits as shown in Figure 4. Iterations c2 and c3, both of them are typical iterations of the construction phase, show a far smaller similarity index than the comparison of iterations c2 und e1: the smaller the index, the greater the similarity is. The similarity index can be used to calculate a predefined number of clusters of similar iterations [7], each cluster serving as the basis to calculate **typical** (average) **iteration types**.

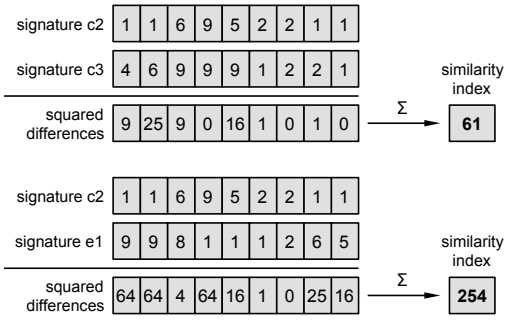

Figure 4. Calculation of similarity indices

Because of the normalization procedure described above, it is possible to consider iterations of projects with different iteration length and FTE-intensities simultaneously. This is of great importance with respect to the need of a similarity database of significant size. Because the similarity approach implemented here focuses not on projects but on iterations and because all iterations are normalized, a meaningful initial database can be gained by starting three to five projects (assuming there are at least ten iterations per project).

The similarity index is also used to **identify similar iterations** stored in the similarity database. The software engineer may describe an iteration by providing a rough FTE-estimate

for each discipline; then the similarity database can easily search for iterations with small similarity indices and give information about their respective costs, the projects they belong to, the staffing and much more.

E. Introduction of Scaling Factors Based on Real Prices

To be a useful tool to calculate, manage and control software processes the methods adopted must allow the software engineer to apply scaling factors accounting for different iteration lengths and FTE-intensities. Finally, wages may have changed and have to be considered with their current values. The **scaling factors** necessary (see also section III) can be calculated by applying multiple regression with iteration length and FTE-intensity as independent variables (on regression see [17]). The regression results will only provide useful results if it is possible to separate the **influences of price changes**, for instance due to the raise of wages, from the independent variables. When adding new iteration data to the similarity database it is therefore mandatory not only to enter time and cost, but also information about the current price level, for instance by memorizing the current wage plan or a price index. This will allow filtering out changes in iteration costs due to price movements.

III. Concepts and Tools for Output-Oriented Management and Control of Software Processes

A. Cost Estimation for Software Processes

Based on clustered iterations, identified iteration types and determined respective iteration costs, a variety of financial management tasks may be accomplished. Most important is the pricing of new software processes, either to back up the submitting of a quote in a bidding process or to be able to decide about a make-or-buy-situation. Other applications are the continuous monitoring of the advances made in the software process as well as the analysis of occurring deviations [22].

The **calculation scheme** applicable in the context of normalized iteration types like the ones developed in section II is shown in Figure 5. In this example a web-based enterprise valuation tool has to be developed. Carab Amabo, project head and experienced software engineer, made good experiences with iterative software processes with short 2-week-iterations. In his calculation, which is part of the feasibility study in the inception phase of the project, Amabo expects one iteration in elaboration because of experiences made in earlier projects. The construction phase consists of five iterations (but only four iteration types). The transition phase, which will complete the project, consists of two iterations. Each of the two-week-iterations completes with a piece of operational software, (starting with a reusable prototype of the user interface in elaboration) and immediate customer feedback to the results produced. To calculate the project, suitable iteration types have to be entered, followed by the count of repetitions and the planned average FTE per week. Obviously, this step of the calculation requires an experienced software engineer, considering findings from earlier projects and taking into account the complexity of the problem as well as the experience of his or her team members.

The same time, there is no need to think about the difficult calculation of iteration costs – these are derived by applying

Project Calculation Iteration Calculator v1.03

project	CP2013-001	Webbased Enterprise Valuation		
project head		Carab Amabo		
duration (start, end)		7-Jan-13	*24-May-13*	*20 weeks*
iteration length		2 weeks		*10 iterations*
personell cost scheme		2013-01		

	type	#	FTE	costs	pv	main tasks
inception		per week				
	i1	1	1,5	4.593 $	8%	feasability study, use cases
	new	new	new			new
elaboration						
	e1	2	2,0	5.380 $	18%	core architecture; high risk issues
	new	new	new			new
construction						
	c1	1	4,0	8.272 $	33%	user interface
	c3	1	5,0	11.000 $	52%	dcf-based valuation engine
	c2	2	3,5	6.650 $	64%	valuation database
	c4	1	5,0	12.500 $	86%	benchmark engine
	new	new	new			new
transition						
	t1	1	2,0	4.040 $	94%	integration of data from old system
	t2	1	2,0	3.640 $	100%	rollout to all valuation desks
	new	new	new			new
subtotal		10	31	56.075 $		
direct costs				3.900 $		object 2013x3 valuation library
				1.200 $		consultation fee p2 communications
				new		new
direct costs total				5.100 $		
project costs				**61.175 $**		

Figure 5. Financial calculation of an iterative software process based on normalized iteration types

the scaling factors iteration length and FTE to the used iteration types, combined with the current wage plan or price index to transform real prices into nominal ones (price index and the real prices are not shown in Figure 5, they are part of global parameters valid for all software projects).

Adding **direct project costs** will frequently complete the estimation process based on iteration types; the example in Figure 5 shows two items of direct costs, one of them being an object library with predefined enterprise valuation methods.

B. Project Management and Control

Cost estimation of software processes is an important task, but the financial tools adopted have to go further: ongoing support in the management and control of the projects up and running has to be given. Iterative software processes offer a natural way to track the progress made: the end of every iteration may serve as a checkpoint. If it holds, the project is in time and budget. The time required to complete an iteration is fixed: two weeks in our example. However, iteration costs may differ significantly from iteration to iteration. But how could one measure the degree of project completion?

Currently the idea of story points is discussed [15,33]. They serve as a relative complexity measure for all the items to be realized within a project. We are aware of the advantages this approach has, especially when discussing the project plan in its entireness. For the ongoing monitoring and control of the project we prefer a rather direct approach: the percentage of budget cumulated can also be interpreted as proportion of output that will be created. Figure 6 shows the value creation over time the project planned in Figure 5 will take. According to the terms used by the **earned value analysis** (for an overview of the earned value method see [4,16,32]), we call the cumulated values created the **planned value** (pv).

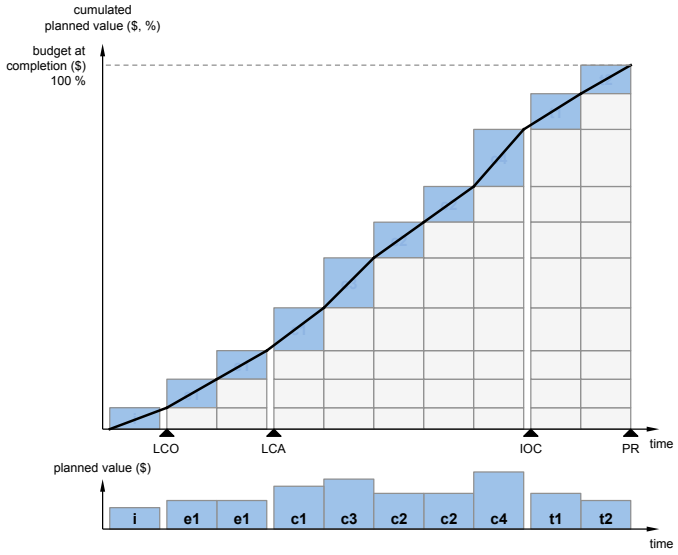

Figure 6. Cumulated planned values as a measure for project completion

However, in the majority of software processes changes will be inevitable, which is a fact acknowledged by agile project management methods. The financial management tools adopted have to meet these needs. Again, we are borrowing concepts related to the earned value method for the financial management of iterative software processes.

As iterative software processes are built from time-boxed, fixed-length iterations [30], it is unlikely that one will find cost overruns in early project stages signaling that something goes wrong. The only reasonable way problems can be identified is by comparing the functionality delivered to date (**earned value**) with the functionality promised in the initial cost estimate.

After each iteration customer feedback is encouraged and may either confirm 1) that everything is fine, 2) that the functionality provided does not meet expectations or 3) that some new or additional functionality is required. In the latter case this means an extension of the project's scope. If both, developer and customer agree on this extension, project completion degrees have to be recalculated. A project status of 50 % completion will diminish to 40 % by extending the project's scope by 25% to 125 %.

If the required functionality has not been provided, this will regularly imply additional and sometimes unpaid extra work for the development team. In Figure 7 it is assumed that at the end of the fifth iteration it becomes obvious that customer requirements are not met and an additional iteration of type c3 has to be considered. Despite this modification, the original planned values are left unchanged (dashed grey line) – and are therefore still a correct measure for the planned project completion over time. The additional iteration (c3, shown in red) forcefully implies a project delay of two weeks (one iteration), other things left unchanged. The blue line indicates a new estimate for the project completion grade (**completion estimate**). Because of the time-boxed approach, the **estimated actual costs** differ only slightly from the planned values until the originally planned ending of the project, however, they continue to rise resulting in the variance at completion after the last (and delayed) iteration.

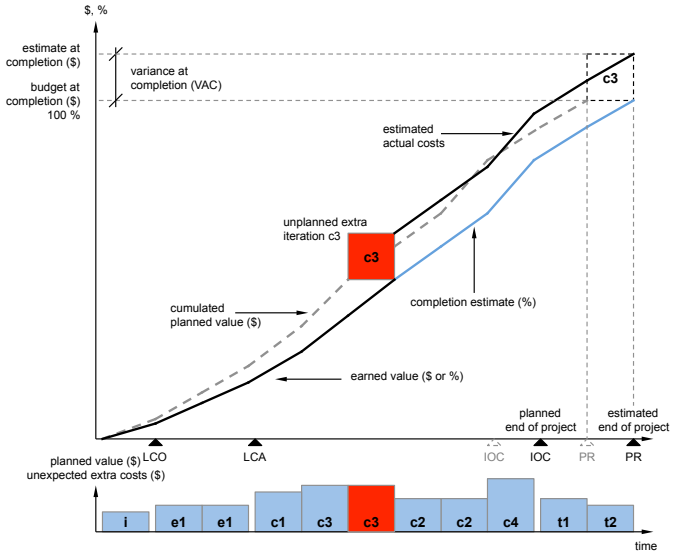

Figure 7. Adjustments necessary due to an additional iteration of type c3

C. *Selected Further Problems*

Financial management and control of software processes may in some cases require additional techniques and concepts to cope with selected problems. If software processes require two or more years to complete, it becomes more and more important to correctly **account for time preferences**, i.e. planned costs should be discounted with a risk adjusted discount rate, considering appropriate opportunity costs (for a discussion of opportunity costs see [12]). This will highlight the relevance of the order in which the tasks are completed if iteration costs differ significantly. Another important area is **variance analysis**, which requires special attention if financial incentives based on target costs are granted to improve efficiency of project management. Here it is relevant to eliminate the unavoidable appearance of higher order variances, because these cannot be correctly accredited to the project manager. Finally the application of methods stronger focusing on assessing the project's risk should be considered, especially **simulation approaches** may provide additional insights at moderate time and effort [3,37]. For a deeper discussion of these topics see [22].

IV. Conclusions

The financial management approach for software processes developed, shows advantages that can be gained by exploiting structural affinities of software process models on one hand and established costing techniques on the other. Due to the nature of similarity calculations, which are one important ele-

ment of the approach presented, the size of the underlying database is critical for creating meaningful results. The concept of normalized iterations dramatically reduces this problem. Describing iterations by signatures opens up further possibilities to cluster iterations, calculate meaningful iteration types and to search the database for similar iterations easily. To further improve financial management and control of software processes, concepts related to earned value analysis are adopted.

In the future, the concept developed may be further broadened. Taking the viewpoint of project managers responsible in a multi-project-context, the development of management cockpits quickly and informatively giving access to the current status of all projects under responsibility is required. No need to emphasize that new technologies like Web 2.0 will find their application in this context and will offer access to the most important facts at the fingertip of mobile devices.

Especially if problems are requiring sourcing decisions (because of a shortage of experienced engineers, for instance), management tools could support the complex situation of the multi-project environment, where benefitting one project may be disadvantageous for the other. This will be an excellent area to apply sophisticated simulation models.

Finally, we want to emphasize the increasing relevance of project management in an international context. Powerful technologies and highly productive management tools will not help solving intercultural problems, but they can at least provide the information necessary to conduct a meaningful discussion.

REFERENCES

[1] A.J. Albrecht, Measuring Application Development Productivity, in: Proceedings of the Joint SHARE, GUIDE, and IBM Application Development Symposium, October 14–17, IBM Corporation, Monterey, CA, 1979: pp. 83–92.

[2] A.J. Albrecht, J. Gaffney, J.E., Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation, IEEE Transactions on Software Engineering. SE-9 (1983) 639 – 648.

[3] L. Angelis, I. Stamelos, A Simulation Tool for Efficient Analogy Based Cost Estimation, Empirical Software Engineering. 5 (2000) 35–68.

[4] A. Baumeister, A. Floren, Optimizing the Configuration of Development Teams Using EVA, International Journal of Information Technology Project Management. 2 (2011) 62–77.

[5] A. Baumeister, M. Ilg, Entwicklungsbegleitende Kalkulation von Softwareprojekten nach dem Unified Process, Wirtschaftsinformatik. 46 (2004) 188–195.

[6] A. Baumeister, M. Ilg, A Technique to Forecast and Control Software Development Costs, in: Proceedings of the IASK International Conferences. E-Activity and Leading Technologies 2009. InterTIC 2009, Seville, 2009.

[7] A. Baumeister, M. Ilg, Activity Driven Budgeting of Software Projects, International Journal of Human Capital and Information Technology Professionals. 1 (2010) 14–30.

[8] K. Beck, Embracing Change with Extreme Programming, Computer. 32 (1999) 70–77.

[9] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, et al., Manifesto for Agile Software Development, (2001).

[10] B.W. Boehm, A spiral model of software development and enhancement, ACM SIGSOFT Software Engineering Notes. 11 (1986) 14–24.

[11] B.W. Boehm, C. Abts, A.W. Brown, Software Cost Estimation with Cocomo II, Prentice Hall, 2000.

[12] R. Brealey, S. Myers, F. Allen, Principles of Corporate Finance, 10th ed., McGraw-Hill/Irwin, 2011.

[13] M. Burghardt, Projektmanagement: Leitfaden für die Planung, Überwachung und Steuerung von Projekten, 9th ed., Publicis Publishing, 2012.

[14] A. Cockburn, Crystal Clear: A Human-Powered Methodology for Small Teams: A Human-Powered Methodology for Small Teams, Addison-Wesley Professional, 2004.

[15] M. Cohn, Agile Estimating and Planning, Prentice Hall, 2005.

[16] Q.W. Fleming, J.M. Koppelman, Using Earned Value Management, Cost Engineering. 44 (2002) 32.

[17] D.A. Freedman, Statistical Models: Theory and Practice, 2nd ed., Cambridge University Press, 2009.

[18] E. Hanser, Agile Prozesse: Von XP Über Scrum Bis MAP, Springer, Berlin, Heidelberg, New York, 2010.

[19] R. Höhn, S. Höppner, Das V-Modell XT Grundlagen, Methodik und Anwendungen, Springer, Berlin, Heidelberg, New York, 2008.

[20] C.T. Horngren, G. Foster, S.M. Datar, M.V. Rajan, C. Ittner, Cost Accounting: A Managerial Emphasis, 13th ed., Prentice Hall, 2008.

[21] P. Horváth, R. Mayer, Was ist aus der Prozesskostenrechnung geworden?, Controlling & Management. 55 (2011) 5–10.

[22] M. Ilg, A. Baumeister, Performance Management in Software Engineering, International Journal of Information Technology Project Management (IJITPM). 1 (2011) 1–18.

[23] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process, Addison-Wesley Longman, Amsterdam, 1999.

[24] R.S. Kaplan, S.R. Anderson, Time-Driven Activity-Based Costing, Harvard Business Review. 82 (2004) 131–138.

[25] R.S. Kaplan, A.A. Atkinson, E.M. Matsumura, S.M. Young, Management Accounting, Times Prentice Hall, 2011.

[26] B. Kitchenham, Measuring Software Development., in: Software Reliability Handbook, Amsterdam, 1990: pp. 303–332.

[27] B. Kitchenham, What's up with software metrics? - A preliminary mapping study, Journal of Systems and Software. 83 (2010) 37–51.

[28] P. Kroll, P. Kruchten, The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP, Addison-Wesley Longman, Amsterdam, 2003.

[29] P. Kruchten, Rational Unified Process: An Introduction, 3rd ed., Addison-Wesley Longman, Amsterdam, 2003.

[30] C. Larman, Applying UML and Patterns an Introduction to Object-Oriented Analysis and Design and the Unified Process, 2nd ed., Prentice-Hall, Upper Saddle River, NJ, 2002.

[31] C. Larman, Agile and Iterative Development: A Manager's Guide, Addison-Wesley Longman, Amsterdam, 2003.

[32] J.A. Lukas, Earned Value Analysis - Why it Doesn't Work, AACE International Transactions. (2008) 1–10.

[33] A. Opelt, B. Gloger, W. Pfarl, R. Mittermayr, Der agile Festpreis: Leitfaden für wirklich erfolgreiche IT-Projekt-Verträge, Hanser, 2012.

[34] S.R. Palmer, M. Felsing, S. Palmer, A Practical Guide to Feature-Driven Development, Prentice Hall, Upper Saddle River, NJ, 2002.

[35] R. Park, Software Size Measurement: A Framework for Counting Source Statements, Software Engineering Institute, Carnegie Mellon University, 1992.

[36] W. Royce, Managing the development of large software systems: concepts and techniques, in: Proceedings of the IEEE Wescon, 1970: pp. 328–338.

[37] R.Y. Rubinstein, D.P. Kroese, Simulation and the Monte Carlo Method, 2nd ed., John Wiley & Sons, 2008.

[38] K. Schwaber, SCRUM Development Process, in: Proceedings of the 10th Annual ACM Conference on Object Oriented Programming Systems, Languages, and Applications (OOPSLA), 1995: pp. 117–134.

[39] K. Schwaber, M. Beedle, Agile Software Development with Scrum, Pearson Studium, 2008.

[40] I. Sommerville, Software process models, ACM Computing Surveys. 28 (1996) 269–271.

[41] I. Sommerville, Software Engineering, 9th ed., Addison-Wesley Longman, Amsterdam, 2010.

[42] G. Versteegen, Projektmanagement mit dem Rational Unified Process, Springer, Berlin, Heidelberg, New York, 2000.