Scaling Agile Estimation Methods with a Parametric Cost Model

Carl Friedrich Kreß
Scientific Services
Cost Xpert AG
Augsburg, Germany
carl.friedrich.kress@costxpert.de

Oliver Hummel
Software Design and Quality
KIT
Karlsruhe, Germany
hummel@kit.edu

Mahmudul Huq
Scientific Services
Cost Xpert AG
Augsburg, Germany
mahmudul.huq@costxpert.de

Abstract— **Estimating the likely cost of a software development project is important with every process model. In agile settings, story points have proven being a useful tool to predict effort for small and medium sized projects or a few iterations. However when projects grow larger, their effort usually grows faster than a linear projection with story points would suggest. This can be attributed to so-called diseconomies of scale, e.g., caused by the growing communication overhead and need for refactoring in large projects. Although these effects are supported by all long-established parametric cost models, such as COCOMO, they are not yet taken into account with agile story point estimation. In this paper, we show how to calculate the magnitude of these non-linear effects to create awareness for this problem in the agile community. As a remedy, we propose three solutions to combine story points with COCOMO II in order to create advanced estimation methods that can be applied to large agile projects.**

Keywords-software cost estimation; COCOMO; agile; Scrum.

I. INTRODUCTION

Agile development approaches were initially aiming on software projects with manageable size and complexity. Due to their iterative and priority-driven implementation approach, they have become popular in many organizations. Even though – to our knowledge – there has been no scientific evidence that agile development projects are more successful than traditional approaches [1] so far, agile methods – like every other method or technique in software engineering – only seem to be helpful when conditions are right [2]. Nevertheless, it is not yet clear whether their perceived success is caused by increased development efficiency or just by the ability to steadily deliver working increments of a system under development. Nevertheless, this perceived success after decades of failed waterfall projects has raised the demand for scaling agile development approaches for larger undertakings. The scaling of agile projects is usually done organically, i.e., in a stepwise manner by splitting one team after a sprint to form the nuclei for two new teams that can then be filled with new additional people. As often reported in literature (see, e.g., [3]), this approach seems to work reasonably well in practice.

At the time being, agile approaches seem quite successful when it comes to estimating and planning two or three sprints ahead by analyzing the remaining user stories with the next highest priorities. Estimating the effort for complete agile projects, however, is a non-trivial challenge for various reasons. Many agile practitioners hence argue that it does not make sense to estimate a moving target (i.e., steadily changing requirements), but advocate to utilize a best effort design to cost approach that delivers as much functionality as the budget allows, billed on a time and material basis. Clearly, however, this is not satisfying from a management and controlling point of view: thus, it has led to the recommendation to elicitate and analyze more requirements in early iterations than can be implemented in order to quickly gain a coarse overview after a project has started [4]. Assuming that the size of each user story has finally been estimated in so-called story points [14], this would allow the prediction of a project's overall effort with the help of a burndown chart as soon as an initial velocity of the development team has been established after some initial sprints. The underlying estimation approach is similar to so-called expert judgments [9] that are a popular estimation method in non-agile environments.

From the perspective of large projects, however, both approaches suffers from a severe limitation that has only rarely been considered so far, especially in agile contexts: story points (as well as expert judgments) are a form of bottom-up estimation that predicts the overall project effort based on a linear projection ignoring so-called *diseconomies of scale*. The latter term describes the fact that larger development projects usually require disproportionally more effort than smaller projects, which can mainly be attributed to the growing communication and coordination overhead in larger undertakings [5]. Thus, although story-point-based estimation has proven to work reasonably well for projects of manageable size, it comes with significant drawbacks when it should be put to use in larger development efforts. Another issue that has recently been reported by practitioners is the increasing amount of refactoring required in growing agile projects. Although common sense clearly suggests that an incrementally extended system will require regular refactorings in order to remain maintainable and extensible, this continuously increasing technical debt [6] is ignored by the current, linear effort prediction via story points.

In order to highlight and overcome these limitations, the remainder of this paper is organized as follows: after going into more detail on the problem of diseconomies of scale and technical debt in Section II, we propose a set of three enhancements for agile estimation in Section III that will support agile developers in overcoming this challenge. The basic idea is to use some mathematics of the parametric estimation method COCOMO II in combination with the story point method to achieve more reliable effort estimates. Since our proposals can be used in different project contexts, we briefly highlight their intended area of application in Section IV before we conclude our paper in Section V.

II. Background

Various surveys have shown that even companies that adopted agile development methods still rely on traditional upfront project estimation and planning in many cases for business reasons. This is mainly because of the need to provide a project budget and status reports to customers or middle and top management, as, e.g. discussed by Sillitti and Succi [7]. We suppose that this necessity is not going to change anytime soon since project management standards like, PRINCE2 [17], that demand for a business justification, i.e., a cost-benefit analysis become more and more mandatory. Hence, even if the "domestic policy" of a development team is a settled agile methodology, in large enterprises and most customer relationships there will always be the need for a plan-based "foreign policy" justifying the expected effort to stakeholders outside the development team. The same need for planning in advance holds true from a strategic point of view. Assume, for example, that a company wants to evaluate a time-to-market strategy for a certain product. This again underlines the necessity for an estimation that quickly enables strategic planning before or at least soon after project start.

The practical need for dependable estimation in large agile projects is also enforced by emerging agile development models like the so-called *agile fixed price*. The clue is already in the name: in this model, a fixed price is agreed upon by suppliers and customers before or soon after the project is started [8]. Obviously, in order to be able to fix a price, the entire project scope must be determined in advance. Within the fixed price project the customer can then still decide what parts of the whole Information Technology (IT) product are to be developed with higher priority in an agile manner. This combination allows minimizing risks by setting a clear scope while at the same time providing a flexible –that is agile– project environment.

A. Diseconomies of Scale

As mentioned previously, agile estimates for the whole product backlog are relying on a linear effort projection: Agile teams measure how many story points they can deliver within a sprint and how much effort is required to do so. If, for example, a team can deliver 50 story points with 10 Person Month (PM) of effort, it can be concluded that, e.g., 300 remaining story points will roughly require 60 PM. Of course, one needs to steadily live with the risk that changing or misunderstood requirements will permanently disrupt this prediction and hence, most agile practitioners limit their estimations on the next two or three sprints. However, especially the management in larger organizations, usually, requires an upfront or early estimation of the whole project effort. The important aspect from an estimation point of view is that the pragmatic approach described above fully ignores the non-linearity of the size-effort relation in large software development projects, as already pointed out by Brooks [5] and Boehm [13]. This so-called diseconomy of scale has been confirmed subsequently by the regression analyses of every major parametric cost estimation model in use today, such as COCOMO II, REVIC, PRICE, and SEER [10]. However, it has to be acknowledged that there has been some controversy around this issue (see, e.g., [11]). Results that indicate slight economies of scale in smaller projects [12] are reflected in the COCOMO II model, which allows exponents smaller than 1 (see next section). But, even if such economies of scale can be reached in smaller projects, this amplifies schedule risks when projects need to scale as it could mean switching from economies of scale to diseconomies of scale.

The following Figure 1 demonstrates how an "over-linear" effort increase in large projects can indeed become a severe risk for the accuracy of agile effort estimations. It illustrates this graphically by contrasting a linearly growing effort curve (red lines), where effort is growing proportional to the expected size, with a nominal effort curve (shorter purple lines) calculated with Boehm's COCOMO II model [9]. Moreover, we have added two curves showing COCOMO II estimates under more and less complex project settings.

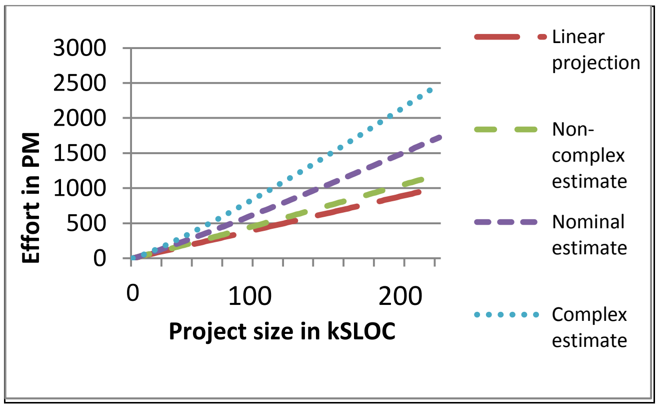

Figure 1. Linearly projected effort estimate against non-linear estimates using COCOMO II.

Clearly, in large projects the inherent empirical process control of agile methods would detect a decrease of development velocity over the course of the project (or at least the increasing refactoring efforts that have been reported by many practitioners once the code base has reached a significant size) and hence will better approximate the real effort over time. However, as described above the driver for estimation is the need to look ahead into the future for a significant amount of time and to present a realistic estimate for the overall effort expected for a project. This is where agile estimation as just presented has its weaknesses, especially when projects become larger. Ignoring this non-linear effort growth may lead to a dangerous underestimation of effort and in turn project duration that can endanger at least the business case of a project, if not the whole project itself.

B. Parametric Cost Models

Parametric cost models, like COCOMO II that we exemplarily present in this section, are based on a regression analysis of numerous projects. They usually require the estimated size of a project under consideration in kilo Source Lines of Code (kSLOC) and various complexity factors as input parameters. As soon as these are determined, the following formula (1) can be applied to calculate the expected development effort in person months:

$$\mathrm{PM} = \mathrm{A}^*\mathrm{kSLOC}^{\mathrm{E}}{}^* \prod_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{EM}_{\mathrm{i}} \tag{1}$$

with *A* being a calibration constant describing the productivity, the expected size of a system is usually derived with the help of a functional size measure for the requirements, such as Function Points [9]. The other factors need to be determined by analysts from a project's context. Values and explanations for the Scale Factors (SF) required for calculating *E* and the Effort Multipliers (EM) can be looked up in the COCOMO II model definition [9].

However, function points, the conversion to Lines of Code, and the determining of the project parameters all bear an inherent inaccuracy so that estimation is also not a trivial task for traditional (i.e., non-agile) development approaches. As is visible in formula (1), COCOMO directly reflects non-linear growth through the exponent E (which is usually larger than one, but can also be slightly smaller, cf. Banker et al. [12]). Moreover, it also distinguishes between effort caused by the functional size (in kSLOC) with its exponent E on the one hand and the effort caused by the product of various so-called EMs, on the other hand. The effort multipliers represent the difficulty caused by non-functional requirements such as reusability needs or constraints in execution time as well as cost drivers such as overall product complexity or a desired internationalization. This distinction is nevertheless important since COCOMO II assumes that effort caused by the functional size grows in a non-linear fashion while the effort multipliers (although they themselves are discrete values) have a linear effect on the effort projection, as they just multiply the effort without an exponent.

As mentioned before, COCOMO II requires rating several scale factors that are deemed responsible fordiseconomies of scale. The following list gives a brief and simplified summary of these ratings:

- Precedentedness: rates if the product or project type is similar to previous ones.
- Development Flexibility: rates the software conformance to requirements and external interfaces.
- Team Cohesion: accounts for communication overhead because of difficulties in synchronizing stakeholders.
- Process Maturity: rates the maturity of the development process according to CMMI levels.
- Architecture / Risk Resolution: rates the maturity of the risk management concerning development risks as well as the percentage of development schedule devoted to establishing the software architecture.

Even though COCOMO II [9] was not developed with agile projects in mind, especially the last parameter reflects a circumstance that all software development projects do have in common, and that agile project are especially prone to: not putting (enough) upfront effort into the development of a decent software architecture can drastically increase the technical debt of a project and will increase refactoring overhead over the course of the project.

C. Error Calculation

The COCOMO II model can also be used to calculate the magnitude of error of a story point estimate like it was depicted in Figure 1 before. For that purpose, we assume that the functional size of the project simply increases by an arbitrary factor g, for the moment. Thus, the linear model used by agile teams would estimate the expected effort as:

$$\mathrm{g}^*\mathrm{Story\ Points}/\ \mathrm{Velocity} = \mathrm{PM}_{\mathrm{new}} \tag{2}$$

Here, velocity is given as story points per person month. There is no normalization factor that describes how to measure a single story point. This means, the number of story points can be of arbitrary size, depending on the habits of the agile team and it will only become meaningful when set in relation to person month needed per story point, thus describing the velocity of a specific team [15]. On the other hand, integrating the growth factor *g* into the COCOMO II formula causes a non-linear effort increase. This is shown in the following formula: since *g* is multiplied to the size it is under the influence of the exponent *E*:

$$\mathrm{A}^*(\mathrm{g}^*\mathrm{kSLOC}_{\mathrm{ref}})^{\mathrm{E}}{}^*\mathrm{EM} = \mathrm{PM}_{\mathrm{new}} \tag{3}$$

To better illustrate the difference, we calculated the error as the fraction between both calculations. A 10 PM reference project would match a functional size of about 3.25 $\mathrm{kSLOC}_{\mathrm{ref}}$. This value is determined by "backward calculation" of COCOMO II for an effort of 10 PM, with scale factors set to high and all effort multipliers set to nominal, see (5). If, for example, the functional size growth factor between the reference project and another one is $g = 10$ the difference between the linearly interpolated estimate of 100 PM and the non-linear nominal scaling estimate equates to:

$$\frac{2.94^*\left(\frac{100}{10}{}^*3.25785\right)^{0.91+0.01^*18.97}}{100} \approx \ 36\%. \tag{4}$$

In other words, the calculation in (4) demonstrated that even for a relatively small project a story point based effort prediction is prone to underestimate effort about one third.

III. SCALING AGILE ESTIMATION METHODS

In this section, we propose three solutions with increasing accuracy to better represent the growing communication overhead in large agile projects. They all work by combining the parametric cost model COCOMO II with common agile estimation practices. As such they are simply based on the common agile artefacts like user stories and story points. However, since there is no absolute size reference for one story point, it is not possible to simply use absolute story point counts in the formulas that we are suggesting in the following three solutions. In order to circumvent this problem, we have to work with the *relation* between story points and not the story points themselves. The same issue and solution must be considered for even simpler user story based estimates.

All presented solutions make use of the following data points. This reference data can be gathered during regularly sized projects (or initial sprints *before* scaling up the team) and allows determining the regular productivity of the team. In the following Sub-sections III.A, III.B and III.C, we will show three ways how to use this information to calculate the productivity of the upcoming larger project, that is, when the overall team size is scaled up. The following reference data is needed to determine the initial productivity:

- Story points delivered,
- Number of user stories and
- Effort in person month needed or kSLOC written.

A. *Analogy-based Estimation using the Number of User Stories*

If only a ballpark figure is needed (e.g., early in a project), we suggest the following simple approach to derive a coarse estimate that is merely based on the number of user stories and an analogy to a previous project or an initial increment. Using the COCOMO II formula presented above and the effort actuals of the previous project/increment, a functional size analogue for the functionality that the team delivered before can be derived by rearranging the COCOMO II formula using kSLOC:

$$kSLOC_{ref} = \left(\frac{PM_{ref}}{A*EM_{ref}}\right)^{\frac{1}{E_{ref}}} \qquad (5)$$

When we consider the separation between growth caused by functional size and effort multipliers as explained before, this approach can only be applied under the following circumstances: 1) The user stories of the reference and the current project are of comparable size, that is their size differences are small for both projects. 2) The stories for the upcoming project are written in the same manner as in the reference project, especially in terms of the average size of a user story. 3) The effort multipliers do not change between the reference project and the current project, which is implicitly given when the reference data is coming from the same project.

Usually, these conditions will hold true when the upcoming project refers to the same class of products as the last project, e.g., when building a company's standard product like an interactive web application merely for a different customer. In these cases it is usually not necessary to rate the effort multipliers again. Combined with the assumption that the user stories are similar in size, the ratio between the number of user stories of the reference project ($\#\mathrm{US}_{\mathrm{ref}}$) and the number of user stories in the upcoming project ($\#\mathrm{US}_{\mathrm{new}}$) would then represent the change in functional size:

$$\mathrm{kSLOC}_{\mathrm{new}} = \mathrm{kSLOC}_{\mathrm{ref}} * \frac{\#\mathrm{US}_{\mathrm{new}}}{\#\mathrm{US}_{\mathrm{ref}}} \qquad (6)$$

This calculated value can then be used for effort estimation with the help of the COCOMO II equation:

$$\mathrm{PM}_{\mathrm{new}} = \mathrm{A} * \mathrm{kSLOC}_{\mathrm{new}}{}^{\mathrm{E}_{\mathrm{new}}} \qquad (7)$$

Although we are omitting a potential change of effort multipliers between the reference project and the upcoming project for sake of a quick estimate (i.e., $EM_{\mathrm{new}} = EM_{\mathrm{ref}}$), we do take the non-linear scaling factors (SF_j included in E) into account that are responsible for non-linear effort growth.

B. *Analogy-based Estimation using Story Points*

In order to improve the accuracy of the previous approach, it should be obvious that user stories weighted with story points produce better results as they also take size and complexity of the requirements into account. Concerning the distinction between functional size and effort multipliers discussed before, we can assume that a story point estimate is actually an amalgam of the functional size of the IT product and effort multipliers corresponding to the IT product.

Since the agile team judges effort multipliers implicitly when assigning story points, COCOMO II's effort multiplier ratings can be used to make this explicit as explained in the following. First, in order to eliminate this effect from the story point estimate and to gain a value for the pure functional size of the product we need to determine the effort multipliers [9] and "remove" them from the story point value through the following division:

$$\text{Functional size} = \frac{\text{Story Points}}{EM} \qquad (8)$$

Second, based on these considerations we suggest the following steps to combine the "purified" story point estimate with the COCOMO II model in order to gain a reliable estimate for larger projects that also incorporates non-linear scaling effects:

1. Determine the functional size a team is able to deliver using kSLOC by backward calculation of COCOMO II (5).
2. Again it is necessary to determine how the functional size of the upcoming project changes in relation to the reference project. Thus, in order to merely relate the functional size of both projects with each other, we need to eliminate the effort multipliers from both story point estimates. The new functional size can then be derived as:

$$\mathrm{kSLOC_{new}} = \left(\frac{\mathrm{SP_{new}}}{\mathrm{EM_{new}}} \Big/ \frac{\mathrm{SP_{ref}}}{\mathrm{EM_{ref}}}\right) * \mathrm{kSLOC_{ref}} \qquad (9)$$

3. Now, these values can be used to calculate the expected effort, this time including the effort multiplier rating for the new product $\mathrm{EM_{new}}$ as well as a new evaluation of the scale factors $\mathrm{E_{new}}$ depending on the new team constellation and product environment:

$$\mathrm{PM_{new}} = \mathrm{A} * \mathrm{kSLOC_{new}}^{\mathrm{E_{new}}} * \mathrm{EM_{new}} \qquad (10)$$

C. Parametric Estimation Measuring SLOC

The previous two approaches are simple in the regard that they only use parameters well known to every agile developer and some algebra. However, the "backward" calculation (5) of the functional size of the reference system may lead to an additional estimation uncertainty that can actually be avoided. When reference code (or an initial increment of a new project) is available, it becomes possible to directly measure the size of the existing code base with some metric tool, ideally the COCOMO II code counting tool of Boehm's group at the University of Southern California [16]. This will increase the accuracy of the estimates with concrete numbers.

Based on such a concrete SLOC measure, it becomes possible to project the expected size of the new project again using the rule of three and the ratio of story points and effort multipliers as before (9). As shown above, it is then merely required to determine the scale factors and effort multipliers for the reference project and the new project. The result can then be easily used to estimate the overall effort required for the project by using the COCOMO II formula already used in (10). In other words, this third approach predicts the non-linear effort portion to be expected in a large development project with the COCOMO II model, based on a typical agile size measurement with story points.

IV. DISCUSSION

As the number of user stories is usually available before a concrete story point estimate, the method from III.A can be used in quite early stages, perhaps even before a project is actually started. When presenting the solution above, we suggested that the effort multipliers would not change between the reference project and the upcoming project. This makes sense as it might be difficult to rate the effort multipliers at such an early point in time. However, if time, resources, and the necessary information are available, this solution can even be refined by rating the effort multipliers and integrating them into the estimate as in the story-point-based solution (analogue to (9)).

The latter was designed with the goal in mind to be easily applicable by any agile team while providing good estimation results. It can be used after all user stories have been written and assigned a story point value. This, obviously, requires analyzing all user stories close to the beginning of a project even when they merely have a low priority. While analyzing more requirements than can be implemented in order to gain an overview of a project quickly is sometimes recommended in literature [4], many agile practitioners merely look ahead for two or three sprints and leave further stories untouched until they become relevant for short term planning. This approach obviously clashes with the business need of effort prediction. We do not see a simple solution for this dilemma, but regard the upfront analysis of user stories as a viable compromise that allows effort predictions without generating too much overhead.

Besides the value that the estimate provides from a business point of view for reliable product planning, we see an even higher value for agile teams: When asked to come up with an estimate, traditional agile estimation methods do not provide the means for anything else than a linear scaling. Thus, early in a project when the team size is still small, agile teams may be trapped by the self-created benchmark without a chance to predict reduced productivity when the project is scaled up later. Using our solution they can make the diseconomies of scale transparent and understandable to management by rating the COCOMO II scale factors and using the non-linearity of this model.

As mentioned above, we currently assume that a direct SLOC measurement would yield the most promising estimation results (although this measure is admittedly not undisputed itself it is probably the most accurate approach that is available today). This is because the SLOC that have been delivered during a reference project or a number of reference sprints best describe the actual productivity of a team. In addition the SLOC value could for example allow gathering historical data to determine a mean productivity factor. It could thus also be used to do a full COCOMO II calibration as described by Boehm et al. [9] in order to match a team's productivity even better. Thus, the SLOC-based solution is probably best suited for advanced agile teams that want to further improve their estimation accuracy.

Moreover, since COCOMO II defines SLOC very carefully, it should be made sure that the tool used to measure the reference SLOC complies with this definition in order to reduce sources of potential deviations. Whether the measurement of SLOC conducted with an organization's code metrics tool largely differs from the original COCOMO II SLOC counting definition can easily be verified by calibrating it with values delivered by the COCOMO II counting tool mentioned in Section III.C at least once or by directly using the latter to measure the SLOC.

Another interesting question that should certainly become subject of future research is the question how "pure" story point estimates reflect the functional size of a user story or how far they are "polluted" with the extra functional effort multipliers identified in COCOMO II. In Section III.B we have made the latter assumption, however, a closer look in the COCOMO II manual [9] suggest that some may be implicitly considered during story point estimation and others may be ignored. Hence, we feel that even common story point estimation could benefit from explicit consideration or exclusion of these factors. As our mathematical solution only evaluates the change between the effort multipliers of the reference project and the upcoming project, our model should fortunately not be directly affected by the outcome of this future work.

V. Conclusion

The Agile Manifesto's intention [18] was not to create a reliable estimation method. It was about values and work culture, thus hit the nerve of the time and has inspired several successful agile development approaches. However, basically all agile methodologies were initially aiming on smaller projects with small teams and only recently ideas for scaling them in a stepwise manner have been added. As we have described in this paper, even agile projects are often under a significant outside pressure to deliver reliable effort estimates. The larger projects, the larger this pressure will usually become. Exactly such larger software development projects have to deal with so-called diseconomies of scale caused by the growing need for communication and coordination amongst their personnel due to the growing size and complexity of the software system. This non-linear increase of development effort with project size, is not reflected in current agile estimation techniques based on story points and hence poses a serious risk of under-estimation for larger projects.

In this paper, we described three advanced ideas to better deal with this challenge by combining agile estimation techniques with elements from the proven parametric cost model COCOMO II, as initially developed by Barry Boehm. Although in this early stage, the ideas look promising; it is obvious that the next step must be an investigation of their practical relevance. To our knowledge, there is no study that would have looked into the non-linear effort increase in large agile projects and hence no empirical data is readily available that could be used to validate our model. However, well managed agile projects that have tracked their development efforts should allow applying all three proposed approaches in retrospect so that predicting their overall effort based on the velocity of, e.g., the first three sprints and COCOMO II should become possible.

Even though a lot of work still needs to be done, we conclude that the combination of agile estimation methods and parametric cost models can be seen as a promising way for agile estimation in the 21st century software engineering that might help better predicting the growing communication and refactoring overhead in large agile projects.

VI. References

[1] T. Dyba and T. Dingsøyr, "Empirical studies of agile software development: A systematic review.", Information and Software Technology, vol. 50, iss. 9–10, August 2008, pp. 833-859.

[2] T. Chow and D.B. Cao, "A survey study of critical success factors in agile software projects", The Journal of Systems and Software, vol. 81, iss. 6, June 2008, pp. 961-971.

[3] Paasivaara, S. Durasiewicz, and C. Lassenius, "Distributed agile development: Using Scrum in a large project", Proc. IEEE International Conference Global Software Engineering, 2008, pp. 87-95.

[4] C. Larman, Applying UML and Patterns, Prentice Hall, Upper Saddle River, 2005.

[5] F. Brooks, The Mythical Man Month, Essays on Software Engineering, Addison-Wesley, 1975.

[6] P. Kruchten, R.L. Nord, I. Ozkaya, "Technical Debt – From Metaphor to Practice", IEEE Software, vol. 29, iss. 6, November/December 2012, pp. 18-21.

[7] A. Sillitti and G. Succi, "The Role of Plan-Based Approaches in Organizing Agile Companies", Cutter IT journal, vol. 9, iss. 2, 2006, pp. 14-19, URL: http://goo.gl/CYRvJK (retrieved: January 2014).

[8] A. Opelt, B. Gloger, W. Pfarl, and R. Mittermayr., Agile Contracts: Creating and Managing Successful Projects with Scrum, Hoboken, 2013.

[9] B.W. Boehm et al., Software Cost Estimation with COCOMO II, Prentice Hall, Upper Saddle River, 2000.

[10] R. Jensen, A.W. Armentrout, and R.M. Trujillo, "Software Estimating Models: Three Viewpoints.", CrossTalk, February 2006, pp. 23-29, URL: http://goo.gl/ACfzDk (retrieved: January 2014).

[11] B.A. Kitchenham, "The Question of Scale in Software – why cannot researchers agree?", Information and Software Technology, vol. 44, iss. 1, January 2002, pp. 13-24.

[12] R.D. Banker, H. Chang, and C.F. Kemerer, "Evidence on economies of scale in software development", Information and Software Technology, vol. 36, iss. 5, May 1994, pp. 275-282.

[13] B.W. Boehm, Software Engineering Economics, Prentice Hall, 1981.

[14] R.L. Nord and J.E. Tomayko, "Software Architecture-Centric Methods and Agile Development", IEEE Software, vol. 23, iss. 02, March-April 2006, pp. 47-53.

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

[16] URL: http://sunset.usc.edu/research/CODECOUNT (retrieved: January 2014).

[17] Great Britain. Office of Government Commerce: "Managing successful projects with PRINCE2", TSO 2009

[18] URL: http://agilemanifesto.org/ (retrieved: January 2014).