Łabędzki, Maciej; Promiński, Patryk; Rybicki, Adam; Wolski, Marcin

Article

Agile effort estimation in software development projects - case study

The Central European Review of Economics and Management (CEREM)

Provided in Cooperation with:
WSB Merito University in Wrocław

Suggested Citation: Łabędzki, Maciej; Promiński, Patryk; Rybicki, Adam; Wolski, Marcin (2017) : Agile effort estimation in software development projects - case study, The Central European Review of Economics and Management (CEREM), ISSN 2544-0365, WSB University in Wrocław, Wrocław, Vol. 1, Iss. 3, pp. 135-152, https://doi.org/10.29015/cerem.359

This Version is available at:
https://hdl.handle.net/10419/229741

CENTRAL EUROPEAN REVIEW
OF ECONOMICS AND MANAGEMENT

ISSN 2543-9472; eISSN 2544-0365

Vol. 1, No. 3, 135-152, September 2017

www.cerem-review.eu
www.ojs.wsb.wroclaw.pl

Agile effort estimation in software development projects – case study

Maciej ŁABĘDZKI, Patryk PROMIŃSKI, Adam RYBICKI, Marcin WOLSKI

Poznań Supercomputing and Network Center, Poland

Abstract:

Aim: The purpose of this paper is to identify common mistakes and pitfalls as well as best practices in estimating labor intensity in software projects. The quality of estimations in less experienced teams is often unsatisfactory, as a result of which estimation as part of the software development process is abandoned. The decision is usually justified by misunderstanding "agility". This article is part of the discussion on current trends in estimation, especially in the context of the new "no estimates" approach.

Design / Research methods: The publication is a case study based on the experience of a mature development team. The author, on the basis of literature-based estimation techniques, shows good and bad practices, as well as common mistakes in thinking and behavior.

Conclusions / findings: The key to correct estimation is: understanding the difference between labor intensity and time, ability to monitor performance, as well as how to analyze staff requirements for the team.

Originality / value of the article: The publication helps to master confidence-boosting techniques for any estimation (duration, and indirectly, the cost of software development) where requirements are known, but mainly at the stage of project implementation (design and implementation).

Limitations of the research: The work does not address the problems of initial estimation of projects, i.e. the estimation made in the early stages of planning.

Key words: software development, estimation, effort, measurement, requirements engineering, story points, Scrum

JEL: L86

Correspondence address: Maciej Łabędzki, Poznańskie Centrum Superkomputerowo-Sieciowe, Kancelaria, ul. Jana Pawła II 10, 61-139 Poznań, Poland. E-mail: office@man.poznan.pl; labedzki@man.poznan.pl.
Received: 24-11-2017, Revised: 28-04-2017, Accepted: 05-06-2017
http://dx.doi.org/10.29015/cerem.359

1. Introduction

Estimating effort is an important issue in terms of ensuring a high quality of a software development project. The central motivation to create estimations is the need to answer the question, "How long does it take and with what resources to execute a particular functionality?". A developer team can occasionally see the need to create estimations, while at other times it will be a practice adopted at the organizational level. Depending on the circumstances and the team itself, not only what will be estimated but also the estimation method will vary. Still, the usefulness of the results obtained will be conditional on the correctness of the assumptions adopted and on choosing the right approach.

According to the scholarly literature, the problem involving the cost estimation of software dates back to 1960's (Nelson, Edward 1967; Farr, Zagorski 1964). The sheer number of available scientific publications attests to both the prevalence and far-reaching dimension of this issue (Jørgensen, Shepperd 2006: 33-53). For over 50 years, numerous estimation methods have been proposed. Amongst them, we can encounter complex as well as fairly simple calculation methods. Experimental implementations of tools automating such calculations have been created, e.g. using machine learning (Srinivasan, Fisher 1995: 126-137), or Bayesian network (Pendharkar et al. 2005: 615-624). The paper does not seek to analyze all of them, and even less so – to identify the best among them (Mahnič 2011: 123-128; Bjarnason et al. 2016: 61-79; Torrecilla-Salinas et al. 2015: 124-144). However, it presents examples of a successful and unsuccessful application of specific practices, their effectiveness and easiness of implementation.

The paper shows the experience of a group of programmers with a 10-year long experience of working with the Java programming language. In their vast majority, the projects the group implemented comprise "bespoke" systems. They tend to be built from the scratch, they solve unusual problems and fulfill individual requirements of the client. The team executes small and medium scale-projects, deploying an agile approach, Scrum (Schwaber, Beedle 2002; Schwaber, Sutherland 2013), and the experiences described in the paper refer to just such cases.

2. Analyzed projects

The case study presented in the paper draws on three reference projects (the names of the two have been changed). In each of the projects agile techniques were used, with the differences essentially referring to the way the requirements were defined and delivered.

OSW System – a project designed to rebuild and develop the already existing education system. It took a group working in two separate locations one year to implement the project. The average number of programmers engaged in it was 10. The main technology was the Java language. Detailed requirements were known in advance. The team estimated all user stories quite thoroughly, since they were similar to one another functionally. Still, for technical reasons, it was not possible to reduce them to one general user story by way of generalization. As a result of changing the requirements – which happened on numerous occasions in the course of the project – the user story received new estimation. The team was able to plan its work precisely for the next sprint. Moreover, it was capable of specifying the completion date, and, along with the changing requirements – to propose a new estimation.

TOPO System – building an information repository with computer network topology as a core service for external systems. A six-person team conducted work for six years. This was a project with user stories known in advance and defined at a very general level. The developer team basically broke down the general user stories into the smaller ones, within which the team identified subtasks. Next, it made estimations for the user stories thus defined. The more accurately the user story was estimated, the less time was left before beginning the implementation work.

FOODIE System – a platform for sharing knowledge and electronic resources for the agricultural industry[1]. Its development took a three-person team 16 months. There was a pretty general definition of the goal of the product's prototype. The functional requirements were identified on a day-to-day basis, often as a result of the verification of the realized increments. In the first phase of the project, there were no

[1] https://www.foodie-cloud.org/

clearly defined user stories– the user stories with estimations allowed the next two iterations to be planned.

3. Basic issues

Estimating the amount of work is a very common practice, and yet not that absolute as to be spread everywhere. Performing tasks without estimations can take place everywhere where the team does not have to report on the progress of its work and there is no firm deadline within which a particular undertaking has to be completed. From the team's perspective, the decision on conducting estimations belongs to the project manager; however, in general, it is necessitated by having to control costs and deadlines. In commercial projects, specific dates and sums are very likely to be an element of the contract, for enterprises allocate necessary funds in advance. For research projects, a scientific institution or a firm appoints a team, frequently the recipients group of the system is limited, with the effects being expected in a longer term perspective.

Whatever the actual goal behind making estimations, these values are always marred by an error. They are, after all, driven by the workings of the team's more or less precise intuition, and not by the calculation of hard input data. That is why plans and forecasts drawing on such data, by definition, are imprecise.

3.1 Work organization in Scrum

A very popular methodology applied in the work of software development teams is Scrum. As an agile methodology, Scrum forces one to work in iterations and regular functional increments. Although Scrum itself stems from the IT industry, where it is also most frequently deployed, it represents a method of organizing work on any complex product, as its creators tend to emphasize (Schwaber, Sutherland 2013).

Scrum introduces its own terminology, clearly defining its new concepts. One of such is Product Backlog. It is a list of tasks to be performed by the team working on a product. The lists contains all the tasks, whatever their nature, whose execution

will lead to meeting the set objectives. Product Backlog is created and maintained by a person responsible for setting the directions of work within the project. Such a person usually understands the best the needs of the project, and he/she is referred to as Product Owner in the Scrum terminology. This person's responsibilities comprise, among others: defining tasks at a substantial level (not the technical one), prioritizing and ensuring that the tasks are clearly understood by all persons involved in the project. In its structure, Product Backlog can be likened to a stack on whose top there are tasks to be completed first. Their priority arises from having a considerable business value, they are understandable, defined at a sufficiently detailed level, they are well identified by the team and, and further to that, there are no doubts as to why they have to be executed. Product Owner can redefine task priorities (moving them up or down within the stack) according to the changing business determinants. He/she also can delete tasks, creating new ones at any stage of the project work. This tool allows the Product Owner to optimize on a daily basis the value the project has for an organization.

Scrum does not impose the form within which tasks are to be defined, and it only requires that Product Backlog be measurable in terms of work effort. However, not infrequently scrum teams use a user story format to describe the Product Backlog items. User story is a well known approach to define functional and nonfunctional requirements. Since user stories are very informal in nature, they are understood by non-technical persons and can therefore be formulated by them.

A user story is a one-sentence description of a feature one expects from the product that is being built. An example of a user story looks as follows: The user must authenticate before starting to use the application. A particularized specification can be additionally associated with the user story (e.g. acceptance criteria, exceptions). Moreover, there is also a very formal form of the user story, providing a standard set of information in accordance with the template, "*As a (role) I want (something) so that (benefit)*". For example, "As an administrator, I want to delete the users' accounts in order to block access to people who fail to comply with the rules".

User story is provided by Product Owner, yet he/she need not be its author. User stories can be collected through discussion with stakeholders – persons who take

interest in the project's success (e.g. potential users, sponsors, analysts). In Scrum, however, it is Product Owner who is responsible for identifying and incorporating user stories in the Product Backlog.

The first requirements tend to be formulated at a very general level of abstraction. Among other things, this is due to the fact that at the beginning the vision of the product is rather vague. A more detailed image emerges over the course of time. This is the reason why the Product Backlog items are simultaneously very general and very large functionally. That is why the effort related to them is substantial and difficult to estimate, with the team being unable to execute a single task during an iteration. Such tasks are known as epics. Epics are not estimated. They need to be analyzed and broken down into smaller elements – the very user stories.

Besides, user stories can be grouped in so called themes. A single theme groups user stories which are linked with one another logically. For example, the theme "administration tools" aggregates functionalities intended for the administrator's daily work.

Scrum defines prescribed events in the scrum process. These are: planning sprint (iteration), daily scrum, sprint review and retrospective. While planning sprint, the team chooses the user story it is going to realize by way of coming to a consensus (negotiation) with the Product Owner. Next, the team plans technical tasks, at least for those user stories which it intends to perform in the first place. Thus, single user stories defined in a general way are attributed with specific subtasks which clearly identify how they will be implemented.

Definition of Done is a concept linked to Scrum. Definition of Done is a mutual agreement between the scrum team's members which specifies precisely when the work on a Product Backlog item will be completed. For example, the user story is realized if there are automated unit tests and integration tests, continuous integration server raises no objection, there are comments in the code and the code review has been conducted. The team usually starts the project with a minimum Definition of Done which it expands systematically during next iterations, thereby raising the quality standards the product is to meet. It is worth pointing out that estimating

effort required for a single user story should also take into account the tasks resulting from Definition of Done.

3.2 The distinction between effort, time-consumption and cost

The section below presents the definitions of the underlying concepts used further on in the paper, i.e. effort, time-consumption and cost. These concepts, although having different meanings, are quite frequently used interchangeably in the practice of software development teams.

Time consumption implies the amount of time required to execute a user story. Naturally it is expressed in time units. Time consumption, however, does not specify the amount of work itself, but the time needed to perform a task.

Effort means the volume of work required to execute a user story. In practice, it is expressed in various units, depending of the nature of the actions involved in the execution of a particular work. For example, these can be code lines, screens displayed to users or the forms they fill out. Still, it can also be a more abstract unit, such as, for example, the well-known story points (Coelho, Basu 2012). However, whatever the unit applied, the effort always answers the question as to how much work a person executing a user story must perform. The information on effort and the productivity of people engaged allow time-consumption to be determined.

Cost is to be understood as the resources an organization must allocate for the execution of user stories. Depending on the nature of the organization itself and practices adopted, cost can be expressed in, e.g. person-months, cash amount or any other convenient form.

3.3 Choosing the right metric

The choice of the measure used to express effort, and thus – of the unit, can be of relevance to the quality of the data collected and forecasts made. An ideal metric is such for which there is only one possible interpretation as to the values expressed with it. In order to have a better picture of it, it may suffice to compare the measurement unit, "meter" with the approach used in the past when the distance used to be expressed in days and weeks. Applying the time unit was reasonable at a time a journey was only possible on a horseback or foot, and its speed was always

predictable. Nowadays this kind of measure might be misleading, given the diverse means of transport of today.

The decision to express the effort required to execute user stories and technical tasks in days and weeks (Jørgensen 2016) might seem a natural choice, when the overriding goal is establishing work timetable and making plans with respect to the official editions of the product, already tested and free of errors (as was the case for the TOPO project). However, when the team is made up of persons whose experience varies substantially, we encounter differences in work productivity, and as a result – the time required for the execution of the same user stories and technical tasks tends to be different, depending on who in the team was appointed to perform them.

Further to that, the values thus estimated continuously become outdated, as the productivity of the entire team grows, which arises directly from autonomous expansion of the team's knowledge of the issue in question, and ever better familiarization with the technologies used. The effect is that these values become useless and need to be re-estimated.

The problems above can be avoided if, instead of time units, we use a more abstract measure – story points,

3.4 Estimating in story points

The name of this metric refers to the concept of the user story (Patton, Economy 2014). It specifies, albeit not directly, the amount of work needed to execute a single functionality of the product that is being built. This measure, on the other hand, is not directly linked to the execution time of the user story. The value expressed in story points establishes the complexity (difficulty) of the work necessary to complete the execution of a particular functionality. Story points metric can also be used in the estimation of the total cost of maintaining software (Choudhari, Suman 2012: 761-765).

Story points are an abstract measure and for this reason they may cause some difficulties when an inexperienced team decides to use them. This is confirmed by the experiences while working on the TOPO project, following the decision to

implement an agile approach. It is not quite clear what estimation value the first user story should get. In its case there is no point of reference.

The problem can be solved by launching an in-depth discussion on how some selected user stories are to be executed. The discussion should dispel any doubts as to the technical execution of the user stories. This knowledge provides the basis for assigning estimations to user stories while applying story points. At this stage, those values are used which the team considers to be appropriate. In this way a set of reference user stores is created.

It is crucial for the complexities of the user stories from the reference set to be defined as precisely as possible. The next user stories are estimated by comparing them with this set, with the quality of further estimations being largely dependent on this precision. A reference user story should, therefore, be chosen knowingly.

In addition, the practice consisting in estimating the reference set according to days and hours may prove to be helpful. The numbers thus determined are then assigned to user stories. From that moment on, these values are treated as story points. This makes things easier for people who are strongly attached to time units. Nevertheless, one should be aware that those values no longer refer to time-consumption but to effort. This arises from the already mentioned phenomenon of the increasing work productivity and team's experience.

The range of the numbers themselves also carries certain importance. The numbers which work well in practice are those ranging between 0 and 13. This is connected with the particularities of human perception, and to be more precise, with the easiness with which we make comparisons when numbers are not that much different from one another. Some also recommend using numbers from the Fibonacci sequence. The experience gained while working for the OSW, TOPO and FOODIE projects shows that bigger numbers are inconvenient. The analysis of large user stories is difficult and such user stories tend to be broken down into smaller ones or subtasks are identified

3.5 Planning poker, that is, team estimation

Planning Poker (Mahnič, Hovelja 2012: 2086-2095; Moløkken-Østvold et al. 2008: 2106-2117) is a technique allowing for estimation to be made within a group

of developers. Estimation is made using cards (hence the name planning poker). This technique brings about discussions which lead to the improvement in terms of the teams' awareness as to the amount of work involved in the execution of the user stories.

As the experience of working with this technique shows, individual estimations made by the team members for a single user story can vary substantially. This becomes very clear in the FOODIE project. In extreme cases, one user story could get estimations of 5 and 13 points simultaneously. Planning poker solves the problem of disparities while demonstrating how risky it is to appoint just one person to do the estimations.

This practice involves yet another trap. There are actual cases where the discussion did not even take place. The value that is then chosen is the one proposed by the majority or a person enjoying a considerable respect. Such a person can put unintentional pressure on the rest of the team – this was the case in the TOPO project. Yet another reason for falling into this trap is the team lacking sufficient engagement in the estimation process.

4. Effort estimation

4.1 Estimation by comparison

According to this technique, the effort value (e.g. a story points number) for US_X is defined by having it compared to a similar US_Y which was executed in the past. In optimistic cases, the similarity between the user stories becomes apparent at the level of the description included in the requirement specification provided by the client. For example, in the OSW project, the complexity of the user stories depended on the number of operations in the implemented web service interface, on the structure of the input data and the number of the conditions for those data validation. All this information was included in the functional requirements documentation. In practice, the specification having that degree of detail is very rare; however, the client ordering the product was very informed, also

technically informed, as to the final solution. The thorough analysis of the requirements documentation allowed for a very precise effort estimation.

Dealing with such a comfortable situation is, however, seldom the case. The functional requirements themselves are very often clear, with the team having the competences to execute them. However, it is still not easy to compare user stories when there are no similarities in the requirement definition. Then, a more in-depth analysis is needed focusing on the identification of specific subtasks within the user stories and afterwards, do the comparing at this level. What most frequently transpires is that an analogy can be spotted, for example, in the number of new modules, classes, methods or configuration elements.

4.2 Estimating tasks difficult to estimate

In practice we come across user stories to which it is difficult to assign a specific effort value using the comparison method. The problem may occur when the goal is not defined clearly. If this is the case, we should identify the reasons behind it and then select a suitable solution.

One of the reasons why a user story may be too difficult to estimate is its being too broadly defined. What appears to be a solution in such a situation is to re-examine the requirement and/or split the user story into the smaller ones, or to identify subtasks whose estimation is easier.

There are also user stories for which the developer team does not have complete knowledge with respect to the technical aspects of their execution. If there is at least one team member who is capable of saying how complex the user story is, he/she can take the other developers through its technical aspects. The developer team must then devote more time to discuss in more detail the complexity of that user story and estimate it.

However, if the difficulty arises from having no idea how to solve a user story, then it is worthwhile introducing an auxiliary task aimed at the examination/identification of the problem. The task is not given an estimation. From this perspective, it is treated similarly to an error (see “Error estimation”). And just like an error, it reduces the productivity, since it is necessary to work on it, with this work, however, being impossible to foresee.

Scrum methodology provides definition of the concept of Backlog Refinement. This is a practice according to which there should be regular meetings of the team with a view to identify and solve all doubts and problems linked to the definition of requirements. This is when, for example, the effort required for the Product Backlog items is being estimated. The auxiliary task already mentioned, being an element of the backlog improvement, allows for a better insight into the requirement and the method of its execution, consequently – the effort required.

This strategy worked well in the FOODIE project, which was based on a ready-made framework, which from the very start provided a ready-made implementation for a large portion of the project's general requirements. How effective the estimations of the next user stories were was largely dependent on the knowledge of the framework and on its flexibility. It very often happened that the user stories which seemed easy to execute ultimately required much more work than one would believe based on the estimations. Employing auxiliary tasks before assigning estimations contributed to having more accurate estimations.

4.3 Error estimation

The approach to error (bug) estimation may vary and it is closely dependent on the nature of the cooperation between the contractor and the client.

In the OSW, TOPO and FOODIE projects, it was taken for granted that neither bugs nor tasks which did not contribute to functional increment were to be estimated. The assumption was that the projects were accounted for only on the basis of the functionalities ordered, while all other tasks were necessary to ensure the product's high quality. Examples of those tasks include: bug fixing, preparing technical documentation, comparison or reconfiguration of the infrastructure (e.g. continuous integration server). Moreover, the tasks involving bug fixing are usually difficult to estimate, for most of the work is devoted to looking for the cause of an error with the fixing itself being relatively quick.

As a result, the effort required to realize the tasks which are not given estimations becomes visible in the velocity of work. Every new functionality carries the risk of bugs and requires to prepare, e.g. a part of technical documentation.

Every now and then, it is necessary to reconfigure the environment. The actual velocity is then the result of the work on functionalities and non-estimated tasks.

Sprints whose productivity substantially departs from the average may be the consequence of this kind of approach. These sprints are iterations in which one deals, for example, with bugs.

If, for some reason, one needs to know the actual effort required to fix bugs, it is useful to write down how much time was spent on all the tasks. Such tools as JIRA offer the relevant functionality. This kind of knowledge, for example, makes it possible to determine the ratio of the effort required to fix bugs to the rest of the work.

5. Planning and forecasting

5.1 Sprint size estimation

Once the initial estimation of user stories has been completed (e.g. using the story points metric), these user stories go to the first sprint chosen by the developers by way of discussion, drawing on their own subjective beliefs as to their capabilities. A completed sprint provides information about the team's capacities. The sum of estimations of the executed user stories is the first reasonable estimation of the size of the next sprint. The user stories which were not completed will be transferred to the next iteration, with the effort required for their realization being credited to the sprint in which they will be completed (e.g. they will be consistent with the Definition of Done that was adopted).

Further sprints can be planned drawing on the observation of the average work velocity expressed in points per sprint. The velocity refers to the team's ability to deliver complete functionalities. The experience obtained from the OSW, TOPO and FOODIE projects shows that this value may change over time. During the project's initial phase, work productivity increases with every iteration. This allows for increasing the size of the next sprints. To this end, one can use a velocity chart. A practice which might prove useful involves calculating the size on the basis of the weighted average of the last few iterations – earlier sprints receive smaller weights.

After a few, or a dozen or so iterations, the actual velocity can become stable. This happens when the team has already become acquainted with the technology and the user stories have been well defined, and they are similar to one another (as in the OSW and TOPO projects).

5.2 Changes in work velocity

Fluctuations or single rapid changes of velocity are possible due to random factors. The fluctuations can arise from imprecisely planned iterations, when the user stories which were not completed go from one sprint to the next, being classified as the results of the other sprint.

A rapid change in velocity reflects the changes in the method of work execution. Leaps upward can be observed when the framework-like solution is being implemented with the implementation of certain parts of the system becoming considerably facilitated. This could be interpreted in two ways. The team may recognize that its productivity has grown because it performs the same work faster (the similarity of tasks at the level of functional requirement). However, at the technical level, this is no longer the same work. User stories have become more simplified and less work is needed to deliver the same functionality, and by inference – the same business value for the product. It is then possible to consider estimating again the tasks that are still to be realized - the simplest way to do so is to multiply all the outstanding user stories of the similar type by a relevant coefficient. This approach ensures a better quality of the estimation and forecast. Yet, it requires more effort, for the team must keep comparing the user stories each time at the technical level.

Leaps downward may also signal a change in the way the work is being performed. This could be seen in the OSW project, which was executed using a modular approach. First, the user stories from module X were realized. As second, the similar work in the Y part was executed. However, in order to build the foundations of the Y module, the software developers applied different technical solutions which affected adversely the velocity in terms of the execution of very similar functionalities.

5.3 Changes in the team's composition

The risk that must be taken into consideration while starting software development projects is that the composition of a team may change. The situations where the team's composition changes or a completely different team is appointed to work on the project is a common practice. In those cases, the advantages of an abstract metric (such as, e.g. story points) are fully revealed. Once this kind of change has taken place, estimations are still up-to-date. Only velocity of work is adjusted, and ultimately all the forecasts.

5.4 Varying engagement of the team's members

There are instances when an organization allocates dynamically human resources in a project. There might be incidences when the aggregated engagement of staff, calculated, for example, in man days, changes from one sprint to another. This problem can be dealt with by controlling the total participation. At the planning meeting, the team declares its participation and on this basis the size of iterations is determined. Any indicators and forecasts should then be determined in reference to the average team engagement that was adopted.

However, there are some difficulties when it comes to determining the actual participation. One could assume that the team will be engaged at the level it has declared. Yet, in extreme cases one does not manage to follow the declaration. If this is the case, it is worth adjusting the values already recorded during the discussion. Another approach to measuring engagement is to keep the information on the time devoted to work on all the user stories (e.g. the log work function in JIRA system).

6. Conclusion

In the paper, we presented a set of techniques within the scope of effort estimation and the practical aspects of their application in the projects created in agile methodologies. The considerations were based on projects of diverse characteristics – taking into account, among other things, the industry, clients, the size of the developer team – and the approaches to estimations.

The conclusion to be made is that the approach adopted by a particular team to the issue of estimation is dependent on the very nature of the project. Planning poker is a technique which yields good results in any undertaking, increasing the estimation precision and raising the level of understanding user stories which await their realization. When functional requirements are clear and the method of their execution is well known, estimation may be based on brief descriptions of user stories. This is not the case for complex technologies or an inexperienced team, for then substantial attention must be paid to the planning of technical solutions. However, whatever the selected plane for considerations, estimation by making comparisons with the work already done delivers good results. In comparing, it is important to strive for building a good quality reference set of user stories. Moreover, using an abstract metric, such as, for example, story points, in contrast to time units, allows one to avoid the trap of the changing work productivity.

The considerations and recommendations presented in the publication could be a starting point for newly formed teams of developers and provide the basis for further modifications, tailored to individual needs and working conditions.

Acknowledgement

FOODIE is a project co-funded within the CIP (Competitiveness and Innovation Framework Programme) of the EU Seventh Framework Program (FP 7).

References

Bjarnason E. et al. (2016), A multi-case study of agile requirements engineering and the use of test cases as requirements, „Information and Software Technology", no. 77, pp. 61-79.

Choudhari J., Suman U. (2012), Story points based effort estimation model for software maintenance, „Procedia Technology", vol. 4, pp. 761-765.

Coelho E., Basu A. (2012), Effort estimation in agile software development using story points, „International Journal of Applied Information Systems (IJAIS)", vol. 3 no. 7, pp. 7-10.

Farr L., Zagorski H.J. (1964), Factors that affect the cost of computer programming. Vol. II. A quantitative analysis, System Development Corp, Santa Monica Ca, http://www.dtic.mil/dtic/tr/fulltext/u2/607546.pdf [23.09.2017].

Jørgensen M., Shepperd M. (2006), A systematic review of software development cost estimation studies, „IEEE Transactions on Software Engineering", vol. 33 no. 1, pp. 33-53.

Jørgensen M., Grimstad S. (2011), The impact of irrelevant and misleading information on software development effort estimates. A randomized controlled field experiment, „IEEE Transactions on Software Engineering", vol. 37 no. 5, pp. 695-707.

Jørgensen M. (2016), Unit effects in software project effort estimation. Work-hours gives lower effort estimates than workdays, „Journal of Systems and Software", no. 117, pp. 274-281.

Mahnič V. (2011), A case study on agile estimating and planning using scrum, „Elektronika ir Elektrotechnika", vol. 111 no. 5, pp. 123-128.

Mahnič V., Hovelja T. (2012), On using planning poker for estimating user stories, „Journal of Systems and Software", vol. 85 no. 9, pp. 2086-2095.

Moløkken-Østvold K., Haugen N.Ch., Benestad H.Ch. (2008), Using planning poker for combining expert estimates in software projects, „Journal of Systems and Software", vol. 81 no. 12, pp. 2106-2117.

Nelson E.A. (1967), Management handbook for the estimation of computer programming costs, System Development Corp, Santa Monica Ca.

Patton J., Economy P. (2014), User story mapping. Discover the whole story, build the right product, O'Reilly Media, Inc., Beijing, Sewastopol.

Pendharkar P.C., Subramanian G.H., Rodger J.A. (2005), A probabilistic model for predicting software development effort, „IEEE Transactions on Software Engineering", vol. 31 no. 7, pp. 615-624.

Schwaber K., Beedle M. (2002), Agile Software Development with Scrum, Prentice Hall, Upper Saddle River, NJ.

Schwaber K., Sutherland J. (2013), The Scrum Guide. The definitive guide to Scrum: the rules of the game, https://www.scrumguides.org/docs/scrumguide/v1/scrum-guide-us.pdf [23.09.2017].

Srinivasan K., Fisher D. (1995), Machine learning approaches to estimating software development effort, „IEEE Transactions on Software Engineering", vol. 29 no. 2, pp. 126-137.

Torrecilla-Salinas C.J. et al. (2015), Estimating, planning and managing Agile Web development projects under a value-based perspective, „Information and Software Technology", vol. 61, pp. 124-144.

Zwinne szacowanie pracochłonności w projektach programistycznych – studium przypadków

Streszczenie

Cel: Celem niniejszej pracy jest wskazanie powszechnych błędów i pułapek, a także sprawdzonych praktyk w zakresie estymacji pracochłonności w projektach programistycznych. Jakość oszacowań w mniej doświadczonych zespołach jest często niezadowalająca, wskutek czego estymacja jako element procesu wytwarzania oprogramowania jest porzucana. Decyzja zwykle uzasadniana jest błędnie rozumianą „zwinnością". Artykuł wpisuje się w dyskusję nad bieżącymi trendami w zakresie szacowania, w szczególności w kontekście nowego podejścia „no estimates".

Metodyka badań: Publikacja ma formę studium przypadków opartego o doświadczenia dojrzałego zespołu programistycznego. Autor, na podstawie znanych z literatury technik estymacji, wskazuje dobre i złe praktyki oraz często popełniane błędy w myśleniu i postępowaniu.

Wnioski: Kluczowe dla poprawnej estymacji okazują się: zrozumienie różnicy pomiędzy pracochłonnością i czasochłonnością, umiejętność monitorowania wydajności, a także sposób analizowania wymagań i sytuacja kadrowa zespołu.

Wartość artykułu: Publikacja pomaga opanować techniki podnoszące poziom zaufania do wszelkich oszacowań (czasu trwania, a pośrednio – kosztu wytwarzania oprogramowania) tam, gdzie znane są wymagania, jednak głównie na etapie realizacji projektu (projekt i implementacja).

Ograniczenia: Praca nie porusza problemów wstępnej estymacji przedsięwzięć, tj. estymacji dokonywanej na wczesnych etapach planowania.

Słowa kluczowe: wytwarzanie oprogramowania, szacowanie, pracochłonność, miary, inżynieria wymagań, story points, Scrum

JEL: L86