Stanisław Wrycza (Ed.)

Information Systems: Development, Research, Applications, Education

LNBIP 264

9th SIGSAND/PLAIS EuroSymposium 2016
Gdansk, Poland, September 29, 2016
Proceedings

Springer

Architectural Element Points: Estimating Software Development Effort by Analysis of Logical Architectures*

Luís M. Alves[1], Pedro Ribeiro[2], Ricardo J. Machado[2]

[1] Polytechnic Institute of Bragança, School of Technology
and Management Dept. of Informatics and Communications, Bragança, Portugal
lalves@ipb.pt
[2] Centro ALGORITMI, Engineering School of University of Minho, Guimarães, Portugal
{pmgar, rmac}@dsi.uminho.pt

Abstract. Empirical studies are important in software engineering to evaluate new tools, techniques, methods and technologies. In object-oriented analysis, use case models describe the functional requirements of a software system, so they can be the basis for software measurement and sizing. The purpose of this study is to develop a new metric called *Architectural Element Points* (AEPoint) that enables to calculate the effort required to develop a software solution, using the *4-Step Rule Set* (4SRS) method. This paper describes a case study with 60 undergraduate students grouped in four teams that developed a software system (Web application) for a real customer. In this study, we used the AEPoint metric to estimate the resources needed to develop a software system. The results of the AEPoint and *Use Case Points* (UCP) metrics and the real software development effort are compared, conclusions drawn and recommendations are proposed.

Keywords: empirical studies; software engineering management; software quality; software requirements; software metrics.

1 Introduction

In last decades, the software engineering community researchers have developed several empirical studies to evaluate tools, techniques, methods and technologies. The main goal is to provide to the practitioners the research results found in laboratories in order to improve their software products and processes. Thus, the practitioners can have evidence about innovative products and processes in order to assess their value and the risks that must be managed during their institutionalization.

In 1992, Basili *et al*. introduced the concept of *experience factory* [1]. Basically, the *experience factory* is an organizational schema that shows how to institutionalize

* This work has been supported by COMPETE: POCI-01-0145-FEDER-007043 and FCT – Fundação para a Ciência e Tecnologia within the Project Scope: UID/CEC/00319/2013.

the collective learning of an organization [2]. This schema presents two entities, the *project organization* that plan and execute the project and the *experience factory* that collect and package experiences and reuse the empirical results in order to diffuse, generalize, analyze the knowledge contained.

That Basili *et al*. schema was designed based on many years of the *Software Engineering Laboratory* (SEL) work. Empirical studies developed within the SEL involved students from different USA (United States of America) universities and industry partners.

In our case, to perform empirical studies we created an environment similar to an industrial set involved graduated and undergraduate students. All this students attend computer science and information systems program degrees of University of Minho (Portugal). We collected knowledge from the literature to build an environment where each student knows very well her/his role in the project. In a previous work we presented in detail our research environment [3].

As a factor of success it is important to estimate the total amount of resources early in developing process in order to ensure compliance, in terms of cost, schedule and quality of an IT (*Information and Technology*) project. In the development of a software system, the most complex activity is probably the transformation of a requirements specification into an architectural design. The process of designing software architectures is less formalized and often is greatly an intuitive ad-hoc activity, poorly based on engineering principles [4]. Since the architecture of a software system constrains the solution space, the design decisions made during the architectural design should be done very carefully, whereas they typically have a large impact on the quality of the final system. The *4-Step Rule Set* (4SRS) method employs successive model transformations in order to obtain a logical architecture that satisfies the previously elicited user requirements. It is based on the mapping of UML (*Unified Modeling Language*) use case diagrams into object diagrams. The iterative nature of the method and the usage of diagrammatic models help to ensure that the obtained logical architecture reflects the user requirements [4].

The *Use Case Points* (UCP) can predict the total amount of resources at the beginning of a software development process. It is easier to plan and predict the remaining project if there is a metric that allows to know the expected effort during the development phase of the project. With these metrics we can make a better analysis of the costs and the time it will take to complete a project [5].

The existence of a metric on the 4SRS process gives an estimate of the software development effort more precise than the UCP. With 4SRS implementation there is already an architecture aligned with the solution and not just an alignment with the problem. For this purpose we developed a metric to apply on 4SRS that allows to estimate the effort required in the software process development. This metric will be called *Architectural Element Points* (AEPoint).

After the development of the metric, the next step was to test it. In this sense, we considered to perform an empirical study in educational context. We intend to answer the following research question: the AEPoint metric gives a rough estimate of the resources that we should allocate to projects with this complexity level? Mainly, we hope that this metric gives an approximate number of hours to perform all steps of the software development.

In our empirical study we used graduate and undergraduate students of our university. The students were distributed in random groups in four teams. Each team developed a software system of medium/high complexity. We applied the original UCP method for estimate the effort needed to develop each one of that software systems.

In this paper, a description of 4SRS is presented in Section 2. Section 3 describes related work with *Use Case Points* methods. In Section 4, we present in detail the AEPoint metric that we use in the next section. In section 5 we briefly describe the empirical study we have developed to initially assess the effectiveness of using AEPoint metric in educational context. Finally, in Section 6 we present the conclusions and future work.

2 Purpose of the 4-Step Rule Set Method

The 4SRS purpose is to make the bridge between user requirements and the design elements of a complex system. The most complex activity during development of software systems is probably the transformation of a requirement specification into an architectural design [4]. These diagrams represent the logical architecture of the system, integrating the system-level entities, their responsibilities and the relationships between them. The logical architecture captures the functional and nonfunctional requirements of the system.

The 4SRS method for identifying the system components for such architecture (the conceptual model) requires the software engineer to start the development by defining the functional model (use case diagram) that reflects the system functionalities offered to its users from their perspectives. The method is based on a sequence of steps that are inscribed in a tabular representation that is used to derive the software architecture for a focused part of the global system. The method's iterative nature and the use of graphical models ensure that architectures reflect user requirements [4].

The 4SRS method is based on use cases and transformation rules that create other elements (objects). The elements are created and their names are prefixed with a code in brackets, which is used to ensure the uniqueness and easy visual identification. It uses an approach whereby a use case is realized by a collaboration of three types of components: *control*, *data* and *interface*. After this initial transformation, a series of steps with rules is proposed to transform the initial component model in a consistent component model, which is compatible with the requirements. Basically, at each step a set of refactoring rules are applied by modifying the initial model components across groups, divisions or removal of components. Some of these rules can be automated, but others depend on human intervention [4].

As advantages for 4SRS approach, we can point out that the identified classes properly represent the system requirements as they are identified through a recursive process embedded in the 4SRS method that ensures the elimination of redundancy and the identification of missing requirements. Additionally, the recursive nature of the 4SRS method permits that several components of a system can be treated one at a time (each one with its own 4SRS execution) [4].

This approach reduces the complexity of the overall system design, avoiding the construction of a global and massively complex class diagram for the whole system.

Instead, we obtain a single class diagram for each system component, when the 4SRS executions adopt the recursive approach. When compared to the existing approaches, the current version of the 4SRS method adopts a complementary approach by using both object-driven artefacts and use cases to support the complex process of identifying class diagrams from user requirements. A complete description of the 4SRS method can be found in [4].

3 Effort Estimation Methods

We can find in the literature great efforts and contributions to measure the size of a software system and estimate the effort needed to develop it. Measuring the size of a software system is different to estimate the effort needed to develop it, although the two concepts are connected. The first is an activity that consists to assign a measurement unit to represents the size of a software system while the second estimates the effort required to developing it. The relationship between the size of a software system and effort required to develop it is given by the productivity of the software development team.

Metrics are measurement methodologies whose main objective is to estimate the size of software system and assist, as an indicator, the project management of software system development. The estimated size is one of the most commonly used metrics for software size, since has direct impact on development effort and project management. It is an indicator of the amount of work to be performed and this kind of knowledge can be used to help us to estimate the cost and the lead time for the project [6]. According to Pressman, measurement enables managers to plan, monitor, improve and enhance the software process development [7].

The size of the software system means the amount of work to be performed in a project development. Each project can be estimated according to the physical size (which is measured through the requirements specification, analysis, construction and testing), based on the functions that the user gets, in the complexity of the problem that the software system will solve and in the reusability of the project, which measures how much the product will be copied or modified from another existing product [8, 9].

3.1 *Use Case Points*

The UCP metric was defined to estimate *Object Oriented* (OO) projects based on the same philosophy of *Function Points* and in the process "*Objectory*", where the use case concept was developed. Later, Ivar Jacobson developed "*Object-Oriented Software Engineering (OOSE)*", methodology based in use cases, a technique widely used in industry to describe and collect the functional requirements of the software. Considering that the use cases model was developed to collect the requirements based on use and users vision, it makes sense to base the estimation of size and resources of software projects in use cases [10].

The UCP metric was developed to predict the resources needed for a specific project in early developing process e.g. after the requirements analysis. With such an early estimation, one could more easily plan and predict for the rest of the project [6]. The first description of the method was published by Gustav Karner [6] with the aim of creating a model that would allow estimating the resources required to develop a software system under *Objectory AB* (later acquired by Rational Software).

The UCP method consists in calculating a metric called *Use Case Points* that give us an estimation of the size and complexity of a software project. If we know the development team productivity (to be obtained based on previous projects), we can derive an estimate of the effort required to develop the software project. The UCPs are related to functional, technical and environmental complexity of the software project.

When applying the method, we must first calculate the complexity of actors and use cases in the system to quantify the variables *Unadjusted Actor Weight* (UAW) and *Unadjusted Use Case Weight* (UUCW), respectively. When combined with their weight, we obtain an inadequate measure of the size and complexity of the system called *Unadjusted Use Case Points* (UUCP). The next step is to adjust this measure with a number of technical factors and environmental factors given by *Technical Complexity factor* (TCF) and *Environmental Factor* (EF) variables, respectively. These factors combined with the UUCP variable will produce the effective number of UCPs that reflect the size and complexity of the software project. In the following subsections we detail the steps needed to calculate the UCPs. For space reasons we do not present in detail the UCP method, however, it can be found in [6, 11, 12].

4 The Architectural Element Point (AEPoint)

In software projects there is a significant conceptual difference between the problem domain and solution domain. When there are such differences between the specified requirements and design decisions, the architecture can become unsynchronized with the specific requirements of the system. Thus, the 4SRS method support the transition of the system requirements to software architectures and elements of design. In this sense, it is useful that there is a method that allows to predict the total amount of resources at the beginning of the development process. Although there is already the UCP method, in this paper we propose a new metric, called *Architectural Element Points* (AEPoint). This metric will provide some advantages: it provides an estimate of the total amount of resources closer to reality because the metric runs on 4SRS method and there was already an alignment between the problem domain and solution; it provides data that allow allocate and reallocate resources, since we already take into account the software architecture and its elements of design.

The AEPoint metric will run on 4SRS method and is based on the UCP method. This metric is based on three key factors, namely: *Unadjusted Architectural Element Point* (UAEP), *Technical Complexity Factor* (TCF) and *Environmental Factor* (EF). The AEPoint are then calculated by the product of these three factors.

$$\text{AEPoint} = \text{UAEP} \times \text{TCF} \times \text{EF} \quad (1)$$

After calculating the AEPoint, we obtain a result about the architectural elements, lacking a factor to convert estimation points to effort in person hours. After application of the UCP method in three real projects, Karner found that it takes 20 man hours to complete one UCP [6]. This factor has been accepted as an historically collected figure representing productivity [10, 13]. In a previous work, also in an educational context, we find that in average it takes 9 man hours to complete one UCP [11]. This discrepancy between Karner value and our value to the productivity can be explained by several factors. In our view, the main factors are the environment context, using students instead of professionals and the partial implementation of the uses cases. Since we are evaluating metrics applied early in software development process and that there is no more literature about this subject we consider that an AEPoint also corresponds to 9 man hours of work.

4.1 *Unadjusted Architectural Element Point* (UAEP)

This factor must be calculated based on two components, namely, the *Architectural Element Impact* (AEI) and the *Architectural Element Complexity* (AEC). As equation (2) shows the calculation of UAEP is obtained by adding the two components that comprise it.

$$UAEP = AEI + AEC \tag{2}$$

The AEI indicates the impact that a given element has to architecture. The impact is related to the links of this elements. We created a scale to quantify the impact. This scale is then applied to the effect that Software Architects, Analysts and Developers attribute to a particular architectural element. Note that the impacts attributed by the work team's members consider a scale from 0 to 1 at intervals of 0.1. Table 1 shows the levels of impact considered, as well as a correlation with impact attributed by the work team members and their respective weights. The assignment of the impact by the team members will be obtained by averaging the impact that members assign to a particular architectural element.

Table 1. Architectural Element Impact (AEI)

Impact	Impact assigned by team members	Weight
Low	0 to 0.3	1
Average	0.4 to 0.7	2
High	0.8 to 1	3

The AEI variable is calculated by the sum of the products of the weight impacts by the number of architectural elements in each category impact.

Another component that allows us to infer the size and effort of a software architecture is the complexity of an architectural element. The complexity is related to the internal structure of this elements. As greater the complexity of an architectural element, greater the effort required to develop it and consequently its weight in the architecture. We created a scale to quantify the impact. This scale is then applied to the effect that Software Architects, Analysts and Developers attribute to a particular

architectural element. Note that the impacts attributed by the work team's members consider a scale from 0 to 1 at intervals of 0.1. Table 2 shows the levels of impact considered, as well as a correlation with impact attributed by the work team members and their respective weights. The assignment of the impact by the team members will be obtained by averaging the impact that members assign to a particular architectural element.

Table 2. Architectural Element Complexity (AEC)

Impact	Impact assigned by team members	Weight
Very Low	0 to 0.2	1
Low	0.3 to 0.4	2
Average	0.5 to 0.6	3
High	0.7 to 0.8	4
Very High	0.9 to 1	5

The AEC variable is calculated by the sum of the products of the weight impacts by the number of architectural elements in each category impact.

4.2 *Technical Complexity Factor* (TCF)

The size of the software system depends also, on the quality characteristics of the system. Therefore, there are a number of technical or non-functional factors that must be considered to measure the size of the system based on what was agreed between the customer and the supplier. A list describing all technical factors proposed by Karner is shown in Table 3 [6].

Table 3. *Technical Factors* contributing to complexity

Factor	Description	Weight
T1	Distributed systems	2
T2	Performance	2
T3	Efficiency	1
T4	Complex internal processing	1
T5	Code reusability	1
T6	Installation ease	0.5
T7	Usability	0.5
T8	Portability	2
T9	Changeability	1
T10	Concurrency	1
T11	Security	1
T12	Accessibility of others	1
T13	Training	1

The impact of each factor within the project is rated on a scale 0, 1, 2, 3, 4 and 5, where 0 means that it is irrelevant and 5 means it is essential. If the factor is neither important nor irrelevant it must be rated with the value of 3 [5, 12, 14, 15]. The rate

of the factor must be multiplied by the associated weight as shown in Table 3. The sum of all the products calculates the *TFactor* value, which is used to calculate the *Technical Complexity Factor* (TCF):

$$TCF = 0.6 + (0.01 \times TFactor) \quad (3)$$

Karner based the constants 0.6 and 0.01 on the adjustment factors of *Function Points* created by Albrecht [16].

4.3 *Environmental Factor* (EF)

The characterization of the software development teams is also important to obtain a measure of the size and complexity of the software project; thus there is a set of aspects related to the development environment that must be weighed. A list describing all environmental factors proposed by Karner is shown in Table 4 [6]. We believe that as on the UCP method the environment factors also influence the software architecture development, then we decide to consider these factors in our AEPoint metric.

Table 4. *Environmental Factors* contributing to efficiency

Factor	Description	Weight
E1	Experience with a software development process	1.5
E2	Development experience with similar projects	0.5
E3	Experience with OOP	1
E4	Maturity in OO analysis	0.5
E5	Motivation	1
E6	Stability of requirements	2
E7	Part time workers	-1
E8	Experience with technologies adopted	-1

The impact of each factor within the project is rated on a scale 0, 1, 2, 3, 4 and 5, where 0 means that it is irrelevant and 5 means it is essential. If the factor is neither important nor irrelevant it must be rated with the value of 3. The rate of the factor must be multiplied by the associated weight as shown in Table 4. The sum of all the products calculates the *EFactor* value, which is used to calculate the *Environmental Factor* (EF):

$$EF = 1.4 + (-0.03 \times EFactor) \quad (4)$$

The constants 1.4 and -0.03 were obtained by Karner from interviews performed to the resources projects analyzed. The evaluation of UAEP and technological factors occurred by contact with development teams. The environmental assessment factors occurred by online questionnaires realized in milestones defined by teacher's staff.

5 Case Study

Based on the previously described approach for calculating the AEPoint, a case study was developed to determine the productivity of some student software development teams.

The teams were constituted by second year students of the course 8604N5 Software System Development (SSD) from the Integrated Master's in Engineering and Management of Information System in University of Minho (the first University to offer in Portugal, DEng, MSc and PhD degrees in Computing). The teams had 60 people (1 team with 16, 2 teams with 15 and 1 team with 14). Each team receives a sequential identification number (Team 1, Team 2, Team 3 and Team 4) and the description of the customer problem.

The teams developed a software project of medium complexity, using the *Unified Modeling Language* (UML) notation encompassed in an iterative and incremental software development process, in this case, the *Rational Unified Process* (RUP). The teams followed the guidelines established by the RUP reduced model, executing the phases of inception, elaboration and construction according to the best practices suggested by CMMI-DEV v1.3 ML2. The project lasted 4 months. This software project was to develop a Web solution using object-oriented technologies (Java or C#) and relational databases (SQL Server or MySQL), to support the information system of one local professional handball team that provided all the information about the organization and interacted directly with the teams. The logical architecture of the software project was built through the 4SRS method presented in section two.

The main goal of case study was to apply the AEPoint metric in a real software development project although in an educational context. The impact and complexity analysis of 4SRS elements in order to calculate the AEI and AEC components was made based on documentation provided by the teams. Table 5 shows the results of AEPoint and UCP metrics obtained from the software development projects.

Table 5. Results of the software development teams

Description	**Team1**	**Team2**	**Team3**	**Team4**
Number of elements	15	16	15	14
Total effort [hours]	3321	3542	3321	3099
Architectural Element Impact (AEI)	144	58	209	51
Architectural Element Complexity (AEC)	132	79	325	57
Unadjusted Architectural Element Point (UAEP)	276	137	534	108
TFactor	47.5	52.5	48.0	45.0
Technical Complexity Factor (TCF)	1.075	1.125	1.08	1.05
EFactor	10	10	10	10
Environmental Factor (EF)	1.1	1.1	1.1	1.1
***Architectural Element Point (*AEPoint)**	326	170	634	125
Use Case Points (UCP)	477	225	513	281

In order to validate the results obtained with the AEPoint metric we decided also to calculate the UCP of the four project teams. Table 6 shows the comparison of the AEPoint and UCP metrics results. In this table, we also present the average of the real effort recorded by the teams.

Table 6. Comparison of the AEPoint and UCP metrics results

	AEPoint	Effort (Hours)	UCP	Effort (Hours)	Real Effort
Team1	326	2934	477	4293	3321
Team2	170	1530	225	2027	3542
Team3	634	5706	513	4619	3321
Team4	125	1125	281	2526	3099

As referred previously, based on our previous work, we consider that it takes 9 man hours to complete one UCP [11]. We used that reference value to estimate the software development effort by the teams, thus, we assume that the time to apply one UCP is the same as the time to apply one AEPoint.

From the analysis of Table 6 we can draw some interesting conclusions. It is in Team2 that is a greater closeness between the values obtained from the two metrics, approximately 500 hours of difference. In the case of Team1 and Team4 the difference is greater, approximately 1400 hours of difference. Regarding Team3 we can see that there is a difference of about 1000 hours, but the effort is almost 6000 hours so the difference is not as significant. We can also verify from Table 6 analysis that the efforts obtained by the metrics presents some difference from the real effort declared by the teams. AEPoint metric gives a close value to the real effort for the Team1 and UCP metric gives a close value to the real effort for the Team4 and also for the Team1. For both metrics and for the other teams the difference is above of 1000 hours. In Fig. 1 we can check this analysis throughput with a graph.

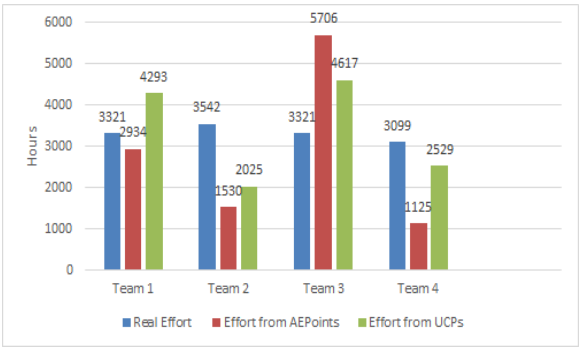

Fig. 1. Comparison of the real effort and effort got from AEPoint and UCP metrics

AEPoint metric gives a considerable different value to the real effort for the Team3. In our opinion, this happen because it is the team that has a more robust software architecture, but, in its final application, its implement only part of that architecture. Analyzing the traceability matrix we can check that the final solution only implements part of the architecture designed. In this matrix we can analyze what were modeled, what were designed and what were implemented in the final solution.

Table 7 shows the number of architectural elements of the teams software applications and their final grade in the SSD course. This table shows that team3 has the highest number of architectural elements, but they has not the highest grade. This

reinforce the previous analysis, in the sense that this team presented a very optimistic architecture, but in their final application we can find implemented just some parts of the global architecture. In other and, Team1 has the most realistic architecture, in terms of what they intended to implement and what they implemented. This team got the best grade because, among other factors, they built the most complete application. Team2 and Team4 have the lowest number of architectural elements. As the Team3, Team4 just implemented some part of the architecture, thus, your grade was the lowest.

Table 7. Number of architectural elements versus student assessment

	Architectural Elements	**Grade (0-20)**
Team1	59	17
Team2	21	15
Team3	84	14
Team4	22	13

Fig. 2 shows the software architecture of the Team2 and Team3 applications. This figure is intentionally not zoomed (and thus not readable), just to show the complexity of each one of the architectures.

The 4SRS method execution results in a logical architecture diagram, presented in Fig. 2. This logical architecture diagram represents the architectural elements, from which the constructors can be retrieved, their associations and packaging. The architectural elements derive from the use case model by the execution of the 4SRS method.

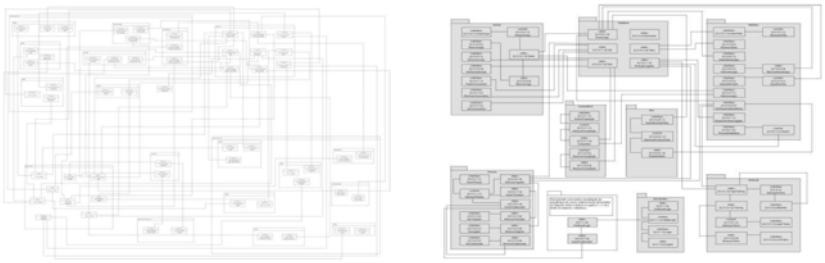

Fig. 2. Software Architecture of Team2 and Tema3 applications

We find possible causes of the discrepancy between the values of the metrics (see Table 6): (1) incorrect development of the AEPoint metric; (2) non-implementation of the overall architecture previously defined (3) poor control of the real effort record by the project manager and team members; (4) poor quality of inputs, including use cases and 4SRS models. The quality of artifacts, mainly the use case models, are limited by the reduced experience of the teams in requirements specification. We could observe in all teams, use cases with poor quality, leading to a subjective interpretation to identify transactions in the main and alternative courses. Additionally, the majority of the use cases were classified as *simple* according the UCP method because it had few transactions. It was not easy to analyze the use cases diagrams to

identify transactions. Sometimes we had to analyze activity and sequence diagrams as a way to validate the existence of transactions.

Throughout the semester, we observed a difference between use cases and architectural elements between projects of different teams, although the purpose of the project was the same. Furthermore, we observed some discomfort in the technological factors evaluation by team members due to lack of knowledge. These facts might have led to a poor quality of inputs.

Since the quality of inputs was one of the factors that most influenced the results, the use of methods that achieve better quality in defining architectures would be an asset. In our research, we propose the *Active Review for Intermediate Designs* (ARID), *Architecture Tradeoff Analysis Method* (ATAM) and *Software Architecture Analysis Method* (SAAM). As a future work, we can use one or a combination of that methods to improve the inputs collected from uses case and 4SRS models.

6 Conclusions

The software development effort in man hours obtained from UCP and AEPoint metrics are relatively close. For some teams, the estimate effort obtained by the metrics is so close to the real effort declared by the teams, but, for other teams there is a considerable difference. It should be needed further research to find the reason of this discrepancy. With some tune in the metric parameters we believe that we can use AEPoint metric to estimate the resources that we should allocate to projects with the medium/high complexity. Furthermore, we also believe, that empirical studies involving students on these subjects are important for the scientific community and the industry.

As future work, we suggest the following actions: (1) partial implementation of ARID, SAAM and ATAM methods early in development process to improve the quality of inputs and solution architecture; (2) application of the AEPoint metric on projects that have teams with more experienced and mature elements; (3) Reviewing the AEPoint metric trying to tune the weights and variables needed to calculate it; test AEPoints metric in different approaches, away from the UCP method. We suggest for future editions of the SSD courses that all teams use a development tool, for example *Teamwork Project Manager* [17] in order to accurately determine the effective involved effort. We intent to assess the influence of this tool in the teams performance.

As a strong validation of the AEPoint metric, we will test it in real design of information systems and we will carry out action research in order to allow practitioners to develop metric as well.

Acknowledgments

The authors would like to thank the referees for their valuable comments.

References

1. Basili, V., Caldiera, G., McGarry, F., Pajerski, R., Page, G., Waligora, S.: The Software Engineering Laboratory-an Operational Software Experience Factory. In: 14th ICSE, pp. 370-381. ACM, (1992)
2. Visaggio, G.: Empirical Experimentation in Software Engineering. In: Lucia, A.D., Ferrucci, F., Tortora, G., Tucci, M. (eds.) Emerging Methods, Technologies and Process Management in Software Engineering, pp. 227. John Wiley & Sons, Inc., Hoboken, New Jersey (2008)
3. Alves, L.M., Ribeiro, P., Machado, R.J.: Project-Based Learning: An Environment to Prepare IT Students for an Industry Career. Overcoming Challenges in Software Engineering Education: Delivering Non-Technical Knowledge and Skills, pp. 230-249. IGI Global (2014)
4. Machado, R.J., Fernandes, J.M., Monteiro, P., Rodrigues, H.: Transformation of UML models for service-oriented software architectures. In: 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS '05), pp. 173-182. (2005)
5. Karner, G.: Resource Estimation for Objectory Projects. Objectory Systems SF AB (1993)
6. Karner, G.: Use Case Points: Resource Estimation for Objectory Projects. Objective Systems SF AB. (1993)
7. Pressman, R.S.: Software Engineering: A practitioner's approach. McGraw-Hill, New York, USA (2005)
8. Fenton, N.E., Neil, M.: Software metrics: roadmap. In: CFSE 2000, pp. 357-370. ACM, 336588 (2000)
9. Fenton, N.E., Pfleeger, S.L.: Software Metrics: A Rigorous and Practical Approach. PWS Publishing Company, Boston, USA (1997)
10. University of Houston-Victoria, http://bfpug.com.br/Artigos/UCP/Damodaran-Estimation_Using_Use_Case_Points.pdf
11. Alves, L.M., Sousa, A., Ribeiro, P., Machado, R.J.: An empirical study on the estimation of software development effort with use case points. In: 2013 Frontiers in Education Conference, pp. 101-107. IEEE, (2013)
12. Ochodek, M., Nawrocki, J., Kwarciak, K.: Simplifying effort estimation based on Use Case Points. Information and Software Technology 53, 200-213 (2011)
13. Anda, B., Dreiem, H., Sjoberg, D.I.K., Jorgensen, M.: Estimating Software Development Effort Based on Use Cases-Experiences from Industry. In: 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools (UML' 2001), pp. 487-502. Springer-Verlag, (2001)
14. Anda, B., Benestad, H.C., Hove, S.E.: A multiple-case study of software effort estimation based on use case points. In: International Symposium on Empirical Software Engineering, pp. 407-416. (2005)
15. Yavari, Y., Afsharchi, M., Karami, M.: Software complexity level determination using software effort estimation use case points metrics. In: 5th Malaysian Conference Software Engineering (MySEC 2011), pp. 257-262. (2011)
16. Albrecht, A.J.: Measuring application development productivity. In: IBM Application Development Symposium, pp. 83-92. IBM Press, (1979)
17. http://www.teamworkpm.net/