USER STORY SOFTWARE ESTIMATION: A SIMPLIFICATION OF SOFTWARE ESTIMATION MODEL WITH DISTRIBUTED EXTREME PROGRAMMING ESTIMATION TECHNIQUE

Ridi Ferdiana[1], Paulus Insap Santoso[1], Lukito Edi Nugroho[1], Ahmad Ashari[2]
[1]Electrical Engineering and Information Technology Department, Engineering Faculty, Gadjah Mada University, Grafika Street No. 2, Yogyakarta Indonesia
[2]Computer Science Department, FMIPA Faculty, Gadjah Mada University, Sekip Unit III, FMIPA Gedung Selatan, Yogyakarta Indonesia
Email: ridi@acm.org[1], insap@mti.ugm.ac.id[1], lukito@mti.ugm.ac.id[1], ashari@ugm.ac.id[1]

ABSTRACT

Software estimation is an area of software engineering concerned with the identification, classification and measurement of features of software that affect the cost of developing and sustaining computer programs [19]. Measuring the software through software estimation has purpose to know the complexity of the software, estimate the human resources, and get better visibility of execution and process model. There is a lot of software estimation that work sufficiently in certain conditions or step in software engineering for example measuring line of codes, function point, COCOMO, or use case points.

This paper proposes another estimation technique called Distributed eXtreme Programming Estimation (DXP Estimation). DXP estimation provides a basic technique for the team that using eXtreme Programming method in onsite or distributed development. According to writer knowledge this is a first estimation technique that applied into agile method in eXtreme Programming.

Keywords: *Software estimation, DXP estimation, eXtreme Programming.*

1. INTRODUCTION

The concerns of this paper are the overabundance of proposed software estimation technique when it’s applied into agile method like eXtreme Programming. Since the one of the principle of eXtreme Programming is simplicity [1], this paper will make an effort to observe previous research and create simplification software estimation technique for distributed extreme programming method.

Estimating software is somewhat challenging but essentially needed. For example, when the development team meets the client, they should be able to estimate how long the software will be developed, how much is cost, and how many resources needed. Another example that happen in software industry is estimating the retail process of the software, how can be a 100 KB software have a worth $1200, but a software with 46 MB have a same value. Those examples provides us that software estimation observe to calculate more than one dimension (i.e. line of code) but three dimensions which are process, product, and resources [5].

Good estimation provides team a wide-ranging forecast view in quantitative aspect such as time to finish the project, how many resources, and also project risks value. Jones [6] has stated that accuracy of good estimation can be achieved ±10% from the real one, but only on well-controlled project. In normal project the estimation accuracy can be within 25% for the actual result (product), and 75% for the actual time [4]. The variance happens since the project is in the phase of uncertainty when it’s executed, there some of unforeseen external events which make the projects late or in the risk. Requirement changes, staff changes, and priority changes are the most prominent changes that make the project far from the estimation. McConnell [7] states that a good estimation is an estimate which is provides a clear enough view of the “project reality” to allow the project leadership to make good decision about how to control the project to hit its target.

In software development project, the project reality can be achieved by seen its lifecycle. Software development lifecycle phases from gathering requirement, analysis, design, and development can be estimated through various research results. Table 1 shows some of estimation technique that designed for certain phases. Applying some of those estimation techniques in an agile method like extreme programming will give additional work for the team.

Table 1. Various Estimation Techniques and Its Lifecycle Fitness

Phases	Estimation Technique Samples
Requirement	Function Point [15], Mark II Function Point [18]
Analysis	Use Case Points [10]
Design	Uml Estimation [11]
Development	Line of Code [17]
Maintenance	COSMIC [8]
Overall Phases	COCOMO [2]

The additional work sometime will make the accumulation of the work rather than creating the quality code. eXtreme Programming has a simple approach of estimation technique during the planning game session. The entity which is being estimated is called as user story. User story is estimated through intuition of the developer. Although is provides a just enough estimation model, we see an opportunities that this paper will contribute.

- A novel approach to estimate the software by using user story and formal the approach as an artifact in extreme programming and distributed extreme programming,
- A modified user story estimation by including risk and others aspect that make the user story estimation more precisely and give a good estimation value to the team
- A proposed way to integrate the user story as basic information for project budgeting and costing.

We simply said our contribution as Distributed eXtreme Programming estimation technique (DXP Estimation).

2. PREVIOUS RESEARCH

2.1. Use Case Points

This research is started by seeing available simple approach to estimate the software. Carroll (2005) provides a simple way to estimate the software complexity by using use case points (UCP). UCP estimate technique provides a formal approach to estimate the software through use case diagram, a part of UML diagram that used in much software engineering method. Figure 1 provides a workflow to estimate the use case points.

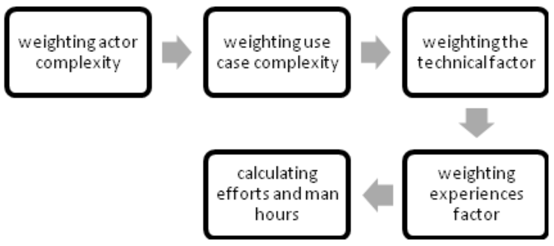
weighting actor complexity
weighting use case complexity
weighting the technical factor
weighting experiences factor
calculating efforts and man hours

Figure 1. Use Case Points Workflow

1. Weighting actor complexity

The process starts by considering the Actors. For each actor, determine whether the actor is a simple, average or complex actor. A simple actor represents another system with a defined Application Programming Interface (API). An average actor is either another system that interacts through a protocol such as TCP/IP, or it is a person interacting through a text-based interface. A complex actor is a person interacting through a graphical user interface (GUI). A simple actor is weighted by 1, API is weighted by 2, and GUI is weighted by 3. Those values will be multiplied by the sum of actor that identified.

2. Weighting use case complexity

For each use case, determine whether it is simple, average or complex based on the number of transactions in a use case, including secondary scenarios. For this purpose, a transaction is defined as an atomic set of activities which is either performed entirely or not at all. A simple use case has 3 or fewer transactions is weighted by 5, an average use case has 4 to 7 transactions is weighted by 10, and a complex use case has more than 7 transactions is weighted by 15. Those values will be multiplied by the sum of the use case that identified.

Both actor complexity and use case complexity is calculated as unadjusted use case points (UUCP).

$$uucp = use\ case\ complexity + actor\ comple \quad (1)$$

3. Weighting the technical factor

Technical factor is an exercise to calculate a Use Case Point modifier which will modify the UUCP by the weight of the technical complexity factors (TCF). TCF go through the following table and rate each factor from 0 to 5. A rating of 0 means the factor is irrelevant for this project, 5 means it is essential. For each factor multiply its rating by its weight from the table.

Table 2. Weighting Technical Factor

Technical Factor	Factor Descriptions	Weight Factor
T1	Distributed solution	2
T2	Specific performance objectives	1
T3	meet end-user efficiency desires	1
T4	complex internal processing	1
T5	code must be reusable	1
T6	must be easy to install	0.5
T7	must be easy to use	0.5
T8	must be portable	2
T9	must be easy to change	1
T10	must allow concurrent user	1
T11	special security features	1
T12	provides interoperability for 3rd parties	1
T13	special user training	1

Those technical factors is accumulated by using this formula

$$TFactor = weighting\ factor * \sum_{n=1}^{13} T \qquad (2)$$

TFactor then calculated to get technical complexity factor (TCF) by using this formula.

$$TCF = (0.01 * Tfactor) + \qquad (3)$$

The technical complexity factor provides basic way to calculate the size of software (szUC) which is multiplying a TCF and UUCP.

$$szUC = TCF * UU \qquad (4)$$

4. Weighting Experience Factor

The level of experience for each team member can have a great affect on the accuracy of an estimate. Consider the experience level for each team member, called the Experience factor (EF).

Table 3. Weighting Experience Factor

Experience Factor	Factor Descriptions	Weight Factor
E1	familiar with software process	1
E2	application experience	0.5
E3	paradigm experience (OO)	1
E4	lead analysis capability	0.5
E5	motivation	0
E6	stable requirements	2
E7	part time workers	-1
E8	difficulty of programming language	-1

To calculate EF, go through the table above and rate each factor from 0 to 5. For factors E1-E4, 0 means no experience in the subject, 3 mean average, and 5 means expert. For E5, 0 means no motivation on the project, 3 means average, and 5 means high motivation. For E6, 0 means unchanging requirements, 3 means average amount of change expected, and 5 means extremely unstable requirements. For E7, 0 means no part-time technical staff, 3 means on average half of the team is part-timer and 5 means all of the team is part-time. For E8, 0 means an easy to use programming language is planned, 3 means the language is of average difficulty, and 5 means a very difficult language is planned for the project.

For each factor, multiply its rating by its weight from the table above. Add together all of these factors to get the total E factor.

$$EFactor = weighting\ factor * \sum_{n=1}^{8} E \qquad (5)$$

Experience Complexity Factor (ECF) can be calculated using formula below

$$ECF = (-0.03 * Efactor) + \qquad (6)$$

Use Case Points is calculated from the multiplication between Experience Complexity Factor and Use Case Size.

$$UCP = ECF * sz \qquad (7)$$

5. Calculate efforts and man hours

Translating use case points into man-hours per UCP is a matter of calculating a standard usage or effort rate (ER) and multiplying that value by the number of UCPs. Carroll calculates the effort rate by 28 for small medium projects and 20 for complex or enterprise project. Those numbers is counting from the number of factor ratings of E1-E6 that are below 3 and the number of factor ratings of E7-E8 that are above 3. If the total is 2 or less, then use 20 man-hours per UCP. If the total is 3 or 4 use 28 man-hours per UCP. If the total is 5 or more then consider restructuring the project team so that the numbers fall at least below 5. A value of 5 indicates that this project is at significant risk of failure with this team.

$$total\ man - hours = ER * U \qquad (8)$$

Project budget then can be calculated by multiplying the man hours with hourly rate.

2.2. User Story

User story is defined as unit of functionality in the requirements system [1]. User stories are expressed in short phrases and should be measurable and testable. This artifact is used in a planning game session of eXtreme Programming.

User story consists as a simple statement regarding the feature that requested to the system. For example, "A customer detail is shown by selecting it from a list" [9]. Some of the research modified the user story to provide also the estimation number. Pelrine [13] add the estimation value with the estimation point which have a scale from 1 (sure about this feature) to 4 (not idea about the feature). The estimation is multiplied by the load factor. Load factor is a multiply factor that used to show the uncertainty of the feature. Load factor has a range from 1.0 (certain) to 3.0 (agile). The result of multiplication between the load factor and the estimation point provide time that needed by the team to solve the problem.

Another research about user story estimation is provided by Woit [14]. Woit states that user story simple provide a simple statement (between 1 – 3 statements), time to estimate for each user story, progress, and some note about the urgency or additional info in the user story. Figure 1 provides a user story illustration which is written on story card / index card [1].

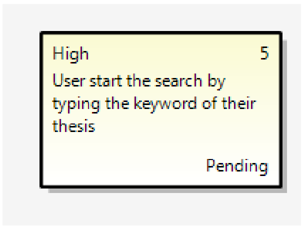

Figure 2. User Story

Cohn [3] in his research provides a formal way to estimate the user story as a software complexity asset. The estimation step is provided as a five simple steps which are displayed in figure below.

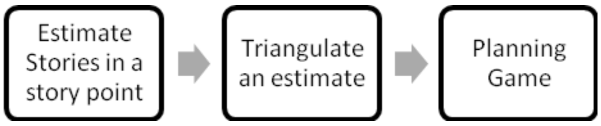

Figure 3.User Story Estimation

1. Estimate Stories in a story points

Story point is defined as complexity estimation, efforts, or duration of a story. Therefore, it can be as a man-days or another numerical representative like integer value that discussed through the team. For example if a story has a 5 points its might be solved ion five man-hours or five man-days, it's depend on the agreement on the team.

2. Triangulate an estimate

Triangulate an estimate is grouping the entire user story regarding of their points. Grouping makes the team aware the complexity of the story and preparing the team to create an iteration planning.

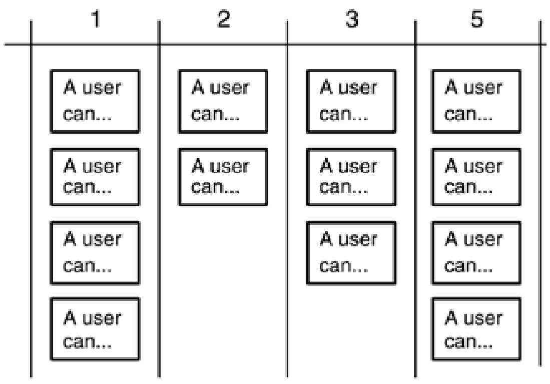

Figure 4. User Story Triangulation

3. Planning game

Planning game addressed two main questions in the agile development, which are about the iteration plan, and the product development roadmap. Planning game is done by the team and the client, in distributed extreme programming planning game is proposed by the team and adjusted by the client [16].

Iteration planning provides information about how many story points in iteration. For example, if iteration has 2 week length, and 2 week length is equal with 40 story points then the user story that included in the iteration is not more that 40 story points.

Product development roadmap or also known as project planning is provides an agreement about how many milestone (or iteration) that should be exist to provide functional product. Project planning provides detail information what will be delivered in iteration including amount of time to deliver the feature.

2.3. eXtreme Programming Estimation Model at Practices

Keefe [2004] shows when applying XP, there are some circumstances where the estimation is far from accurate. There were two reasons for the inaccurate estimations

- The complexities of the task at hand
- The lack of experience the team had in creating estimates for themselves.

Those reasons remembered us, the use case points which also includes the complexities (technical factor) and experience (experience factor). In software development both of those challenge also called as risks. Li et al. [12] proposed a risk driven XP development, and the interesting point in their research is a fact that risks in XP is categorized into four main risks which are requirements risks, estimation risk, technology risk, and personnel risk. In their research, those risks are described qualitatively in a range low, medium, and high. They consideration using qualitatively rather than quantitatively is because quantitative estimation requires a lot of time and cost, and sometime it is difficult for developers to collect enough data for quantitative analysis.

Based on the previous research that we have learned we found some opportunity that illustrated from the table 4.

Those opportunities are described formally as a distributed extreme programming (DXP) estimation technique, which are simply as a selective integration between use case point estimation and user story point estimation.

Table 4. Use Case Points, User Story Points, and Its Opportunity

issue	use case point	User story points	Opportunity
risk identifica-tion	yes	partially yes, (qualitative only)	make it quantitatively
standard	yes, UML	no, free style	make formal estimation model through reference
distributed support	yes, counting distributed as technical factor	no, there are no additional info	make the estimation model support distributed project
efforts and cost estimation	yes	implicitly derived	make formal cost estimation

3. DXP ESTIMATION TECHNIQUE

DXP estimation technique is proposed to fulfill some following gap that happen in the existing XP estimation technique, which are.

- Estimate the user story with other risk estimation like technical and experience factor
- Provide a quantitative analysis with more strength in formal and numerical method.
- Provide a quantitative analysis for cost and efforts estimation,
- Support estimation technique for a distributed software development model.

DXP estimation provides three basic steps to calculate the estimation efforts which are.

- Estimate the unadjusted user story points (UUSP). Unadjusted user story points is a user story points which are not including risk estimation
- Estimate the adjusted use story points (USP) which extends the UUSP among risk estimation.
- Estimate the man-hours and effort needed for the following USP.

3.1. Unadjusted user story points

User story which are used in this estimation technique is consisted with three main components which are.

- The name of the user story including the short description about it.
- The estimate point, which are integer range value started from 1.
- The estimate priority, which are integer point started from 1 (nice to have), 2 (added business value), and 3 (essentially must have).

The estimate point is agreed point that subjectively proposed by the coach in XP team. The problem is some of the team is to narrow in make the estimation, for example, the hard is 3 point and the easy one is one point (only 2 point different). When the team selects the estimate range that is too narrow, the most probably problem that happens is biased value, like answering how hard that features if it's only have a 2 point differentiate. In order to avoid those kinds of situation, we encourage the team to estimate using Fibonacci number. Fibonacci number started from 1, but exponentially increases. Since the Fibonacci is also unlimited in term of value, we are using 7 level of Fibonacci (starting from 0). Therefore we have a sequence range started from 1, 1, 2, 3, 5, 8, 13, and 21. Table V provides the Fibonacci number as estimation points and the means of estimation.

Table 5. Fibonacci Estimation Points and Estimation Meaning

Estimation Points	estimation meaning
1	Simple, just replicate the code.
1	Simple, looking and using codes reference from the API document (Application Programming Interface).
2	Simple, having seen the working codes
3	Medium, creating from the scratch or finding the existing code that need to be converted (like different programming language)
5	Medium, creating from scratch without reference logically can be implemented
8	Medium, interoperability and circular dependency to build such features.
13	Challenging, complex business process and knowledge domain dependency.
21	Challenging, not sure that it can be implemented, and never seen the working example.

Those numbers is defined by the developer. McConnell [7] in his research provides that early estimation can make +40% or -40% than it should. Novice developer will estimate wider than it should, and experienced developer will estimate narrower than it should. In heterogenic team member, we encourage the team to estimate by combining a novice and expert in a pair.

Each story is estimated and triangulated in a blackboard or case tools. There are others benefit when we are using a Fibonacci estimation point in term of triangulation. The triangulation is more concentrated and not too wide. After doing a tribulation the unadjusted user story can be calculated with a sum of all user story point.

$$UUSP = \sum user\ story\ estimation\ po \qquad (9)$$

In the next step, those user story points are arranged based on the priority in planning game session.

3.2. Adjusted User Story Points

Adjusted user story points or simply user story points are unadjusted user story points with additional refinement of risk like technical factor and experience factor. Both technical factor and experience factor is described by adopting Carroll result in use case points. However we add additional experience factor (E9) which tells that the software is developed remotely or distributed.

Technical factor is calculated to get the size of user story (szUS). The DXP estimation technique follows the size of user story by using the formula. The size of user story here can be also identified as software complexity / software size.

$$szUS = TCF * UU \quad (10)$$

TCF variable is derived from the technical complexity calculation factor just like when we calculate the complexity in Carroll use case points.

The user story points are derived from multiplication between ECF and the size of user story (szUS). The ECF variables also derived from Carroll use case points.

$$USP = szUS * E \quad (11)$$

The user story points then can be calculated as man-days effort.

3.3. Estimate the man-days effort

The man-days effort can easily calculated by adopting the work in sustainable pace extreme programming values. In that value a team member can only work effectively not more than 8 hours. Therefore, when calculating man days we calculate using this formula.

$$man - days = \frac{ER*}{} \quad (12)$$

Effort rate (ER) in user story is following the Carroll effort rate. The man-days value then converted using a standard rate that agreed both client and the team.

4. PUTTING IT ALL TOGETHER

Based on the DXP estimation technique that derived in the above, we make an effort to implement it into a real project. This project is developing accounting and product distribution system for manufacturing company. The project is developing in distributed development model. The client and the team are geographically separated. We called this project as Code-Named: Sidik.

The first step is calculated the UUSP based on user story estimation points. The system has 406 user stories that equal with 687 unadjusted user story points. Those user stories have a wide distribution between 1 through 8 Fibonacci number.

In order to calculate user story size or software complexity, we calculate the technical complexity factor by doing table reference like below.

Table 6. Calculating the Complexity Factor

Tech. Factor	Factor Descriptions	Weight Factor	rating	TFactor
T1	Distributed solution	2	5	10
T2	Specific performance objectives	1	3	3
T3	meet end-user efficiency desires	1	1	1
T4	complex internal processing	1	5	5
T5	code must be reusable	1	1	1
T6	must be easy to install	0.5	1	0.5
T7	must be easy to use	0.5	3	1.5
T8	must be portable	2	0	0
T9	must be easy to change	1	5	5
T10	must allow concurrent user	1	5	5
T11	special security features	1	5	5
T12	provides interoperability for 3rd parties	1	3	3
T13	special user training	1	3	3
Total TFactor				43

Technical Complexity Factor (TCF) for Sidik system is:

$$TCF = (0.01 * 43) + 0.6 = 1.03 \quad (13)$$

As a result, we can calculate the software size or user story size by:

$$szUS = 1.03 * 687 = 707.61 \quad (14)$$

What is the meaning of 707.61 in software complexity? Is the software is complex or simple enough? By simply seeing the differentiation between total UUSP and szUS, we intuited that the software is more complex than expected since szUS > UUSP.

To calculate the user story points, we calculate the Experience factor by doing table reference like below.

Table 7. Calculating the Experience Factor

Exp. Factor	Factor Descriptions	Weight Factor	rating	EFactor
E1	familiar with software process	1	3	3
E2	application experience	0.5	0	0
E3	paradigm experience (OO)	1	5	5
E4	lead analysis capability	0.5	3	1.5
E5	motivation	0	1	0
E6	stable requirements	2	5	10
E7	part time workers	-1	3	-3
E8	difficulty of programming language	-1	0	0
E9	distributed development	-1	3	-3
Total EFactor				13.5

Experience complexity factor (ECF) for Sidik system is:

$$ECF = (-0.03 * 13.5) + 1.4 = 0.995 \quad (15)$$

As a result, we can calculate the adjusted user story point by:

$$USP = 0.995 * 707.61 = 704 \quad (16)$$

User story point than can be used to calculate the man-days effort. By seeing the effort rate rules, we can get 20 man-hours per USP. Therefore the man-days can be calculated.

$$\text{Man-days} = (20 * 704) / 8 = 1760 \text{ (rounded)} \quad (17)$$

That number can be easily converted as project length by seeing the maximum expected time from client or team member that exist in the team. In example if the teams have 7 members the project will run smoothly in 251 work-days or if the client need to be done in 6 month (120 work-days) the teams need to be aligned at least 14 members.

By seeing the example we can estimate that the project is

- Sidik project is in high complexity since szUS > UUSP
- The team is in sufficient experience to do the project since USP < UUSP.
- Ideally this project will be finished in 251 work-days with the 7 members or 120 work days with the 14 members.

5. CONCLUSION

This paper main contribution is estimating the software quantitatively. This paper proposed a DXP estimation technique, which is an improvement of the user story point estimation and use case points estimation. This estimation technique can estimate the complexity of the software, and the man-days effort to build the software.

This paper is limited in theoretical background without sufficient empirical research. Therefore we see an opportunity to do empirical research and comparison this technique with the others agile estimation technique.

6. REFERENCES

[1] Beck, K. 1999. **eXtreme Programming Explained**. Boston : Addison-Wesley.

[2] Boehm, B.W. 1981. **Software Engineering Economics**. Englewood Cliffs, NJ : Prentice Hall.

[3] Cohn, M. 2004. **User Stories Applied: For Agile Software Development**. Boston : Addison Wesley.

[4] Conte, S.D., Dunsmore, H.E. and Shen, V.Y. 1986. **Software Engineering Metrics and Models**. Menlo Park, CA : Banjamin/Cummings.

[5] Fenton, N. and Pfleeger, S.L. 1997. **Software Metrics : A rigorous and Practical Approach**. Boston : MA : PWS Publishing.

[6] Jones, C. 1998. **Estimating Software Costs**. New York NY : McGraw-Hill.

[7] McConnell, S. 2006. **Software Estimation : Demystifying the black Art**. Washington : Microsoft Press.

[8] Afsharian, S. Giacomobono, M, and Inverardi Paola. 2008. “A framework for software project estimation based on cosmic, dsm and rework characterization”. **International Conference on Software Engineering 2008**. ACM.

[9] Bostrom, G., Wayrynen, J. and Boden, M. 2006. “Extending XP Practices to Support Security Requirements Engineering”. **Software Engineering Educators Symposium 2006**. May 20-21. Shanghai, China, ACM.

[10] Carrol, R.E. 2005. “Estimating Software Based in Use Case Points”. **The International Conference on Object Oriented Programming Systems Languages 2005**. October 16–20. San Diego, California, USA : ACM.

[11] Lavesque, G., Bevo, V. and Tran Cao, D. 2008. “Estimating Software Size with UML Models”. **Canadian Conference on Computer Science & Software Engineering 2008**. May 12-13. Montreal : Canada, ACM.

[12] Li, M., Huang M., Shu, F. and Li. J. 2006. "A Risk-Driven Method for eXtreme Programming Release Planning". **International Conference on Software Engineering 2006**. May 20-28. Shanghai, China. ACM.

[13] Pelrine, J. 2000. "Modelling Infection Scenarios - A Fixed-price eXtreme Programming Success Story". **The International Conference on Object Oriented Programming Systems Languages 2000**. Companion, Minneapolis, Minnesota, ACM.

[14] Woit, D.M. 2005. "Requirements Interaction Management in an eXtreme Programming Environment: A Case Study". **International Conference on Software Engineering 2005**. May 15-21. St. Louis, Missouri, USA. ACM.

[15] Albrecht, A.J. and Gaffney, J.E. 1983. "Software Function, Source Lines of Codes, and Development Prediction : A Software Science Validation". **IEEE Transaction on Software Engineering** 9, 6:639-647.

[16] Ferdiana, R. 2009. "Distributed eXtreme Programming". **The Architecture Journal** 20, Washington. Microsoft.

[17] McCabe, T.J. 1976. "A Complexity measure". **IEEE Transaction on Software Engineering** 2, 4:308-320.

[18] Symons, C.R. 1988. "Function Point Analysis : Difficulties and Improvements". **IEEE Transaction on Software Engineering** 14, 1:2-11.

[19] Waguespack, J.L. and Badlani, S. 1987. "Software Complexity Assessment: An Introduction and Annotated Bibliography". **Software Engineering Notes** 12, 4:52-71.