Patterns of Software Development Process

Sandro Javier Bolaños Castro[1], Rubén González Crespo[2], Víctor Hugo Medina García[1]
[1]District University "Francisco José de Caldas" - Bogotá (Colombia)
[2]Pontificial University of Salamanca - Madrid (Spain)

***Abstract* - This article presents a set of patterns that can be found to perform best practices in software processes that are directly related to the problem of implementing the activities of the process, the roles involved, the knowledge generated and the inputs and outputs belonging to the process. In this work, a definition of the architecture is encouraged by using different recurrent configurations that strengthen the process and yield efficient results for the development of a software project. The patterns presented constitute a catalog, which serves as a vocabulary for communication among project participants [1], [2], and also can be implemented through software tools, thus facilitating patterns implementation [3]. Additionally, a tool that can be obtained under GPL (General Public license) is provided for this purpose.**

***Keywords* - Software Process, patterns**.

I. INTRODUCTION

The goal of software development is to generate products, with high levels of productivity and efficiency, that ensure good levels of quality. To achieve this, it is necessary to use different strategies. Among such set of strategies, the use of patterns stands out as one of the most popular one. There is already an important and recognized work about patterns in different areas of software engineering, such as design patterns [4], [5], architectural patterns [6], [7], [8], patterns analysis [9] and others. However, to our knowledge, there is no work addressing software process patterns. There is a wide range of software processes and methodologies, and some concepts that can be compiled to promote recurrent usage have been adopted, which may represent a sort of process patterns. This article proposes a set of patterns, which can be found when we use different software processes and methodologies, which can be evidenced in their conceptual cores.

II. UNDERSTANDING PATTERNS

A pattern has a recognizable structure that makes it special and general. Using a software pattern prevents developers from "reinventing the wheel", also preventing out-of-date reinventions that create more problems than what they really solve (e.g. reinventing a square wheel). The pattern has another key ingredient, namely the ease it provides when communicating an idea, because the pattern itself turns into a vocabulary, not only accepted but also widely recognized. Nevertheless, the most important contribution of patterns might lie in their predictive power. A pattern is a sort of time machine, it is like the Rosetta Stone that allows the understanding of the way an appliction was created, initially employing a language that is difficult to follow – namely, programming language, design language, the language of architecture, language processes – and translates it into the original idea behind such implementation through the vocabulary of patterns. This means you can see the past of software applications and the intention with which they were developed; similarly, you can see into the future of the applications and attain one of the most desirable software features, namely scalability. It is possible to structure future applications to maintain a predictable behavior, and allow for evolution. One can say that the patterns become more powerful because beyond being good practice, patterns become a widely-accepted, spoken vocabulary used by a whole community that has found a simple and effective way of communication [10].

III. SOFTWARE DEVELOPMENT PATTERNS

Software processes have produced a framework of concepts that originate familiar, widespread recurrent structures that are used over and over again when developing software projects. These patterns can be categorized as process workflow between the different activities that constitute the process architecture. Such categories involve the participants and stakeholders within the software process according to the inputs and outputs that affect software development and also according to the knowledge involved. The form [11] of the process pattern is given by its definition, consequences, its advantages and disadvantages. The definition establishes the concept of pattern, the consequences define the effects that the pattern causes; the advantages establish all positive contributions of the pattern, while the disadvantages set unfavorable situations caused by the pattern. It is important to note that, unlike other forms of patterns, these patterns indicate their own potential problems and thus alert developers so as to be aware. This is because in the application of software process patterns, it is important to note the potential risks entailed.

A. Workflow Patterns

With regard to the desired process architecture, there are three possible configurations for the activities, namely parallel, linear or cyclic. Out of these configurations, combinations that produce more complex processes can be obtained in turn.

1. Linear Workflow Pattern

The first and most important antecedent of linearity was in Royce's proposal, about his waterfall model [12]. The fact of the matter is that, despite much progress, the waterfall model isn't quite dead yet [13]. Linear Workflow pattern suggests that software processes should focus on a linear workflow [14], that is, a set of activities is clearly identified and linked so that each link is used to configure a chain. Under these conditions there is a permanent pre and post activity for almost every activity, except for the initial and final activities (Figure 1).

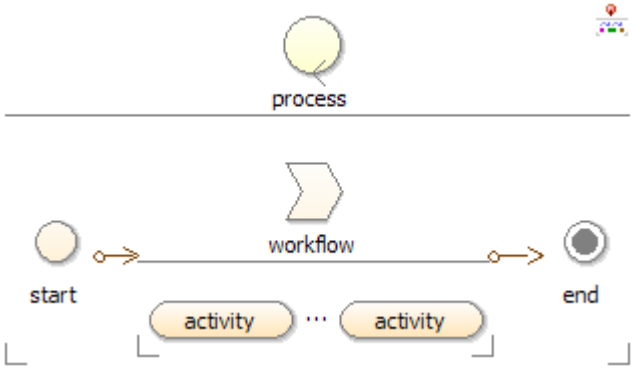

Figure 1. Linear Workflow

Consequences:

The main effect of this linear configuration is that the activities are executed sequentially; therefore, an activity i occurs after the i-1 activity and prior to an *i+1* activity only.

Advantages:

- There is order and control over the activities
- It is possible to budget resources for activities.
- There is clarity in both roles and disciplines involved.

Disadvantages:

- Activity i is highly dependent on activity i-1 and is also the starting point for an activity i +1 in such a way that a domino effect is created between activities whenever there is a problem.
- Time is the most difficult resource to estimate, due to the holistic effect resulting from the integration of activities.
- There are overloaded times for a the roles associated to the activities in execution; when such activities finish, there is uderutilization, wasting a considerable amount of human resources.
- The fall of an activity produces a fall in activity i +1, i +2, ..., i + n, which makes requirements engineering the most critical activity.

2. Parallel Workflow Pattern

Process models, such as the model V [15] propose to mitigate the problem of linearity through the simultaneous development of activities. In the case of V model, activities are confronted with tests during the process. Parallel Workflow pattern suggests that software processes focus on a parallel workflow. The particularity of this model is the execution of parallel activities, at least two activities. The results of the activities that are performed in parallel add their outputs to the next tuple of activities, and so on, until reaching a final result (Figure 2).

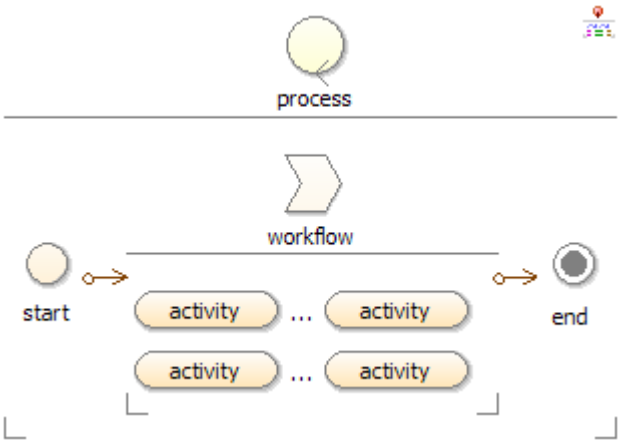

Figure 2. Parallel workflow

Consequences:

Running parallel activities may produce effects such as the need for activity synchronization and the addition of results to configure unified inputs.

Advantages:

- Clear identification of activities.
- Optimization of resources by the simultaneous execution of activities.
- Creation and integration of roles society.

Disadvantages:

- Proper activity synchronization is a difficult task.
- Collaboration between roles requires prior training (as its nurturing factor), which is something most teams lack.

3. Cyclic Workflow Pattern

The cyclical nature of the software process is a recurring concept in different software process proposals [16], [17], [18], [19]. Process models such as the spiral model [20], offer feedback processes which suggest the cyclical nature present not only in this model. Cyclic Workflow pattern suggests that software development processes focus on a workflow with feedback, that is, the pattern clearly identifies a set of interlinked activities and closes a loop with them. It is clear that there is an initial activity that can recycle the product of a final activity (Figure 3).

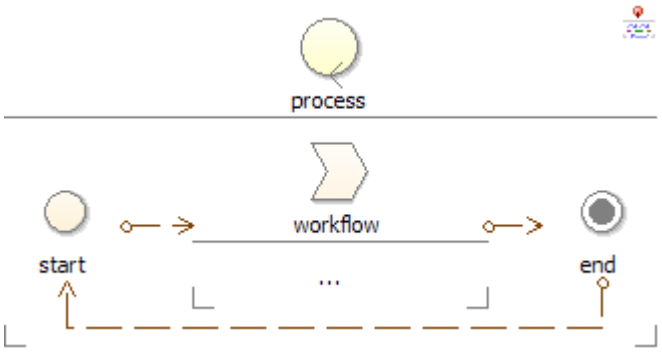

Figure. 3. Cyclic Workflow

Consequences:

This cyclic configuration identifies a loop in which the output of an activity is the input of another previous activity; such a feedback can even occur with respect to the same activity.

Advantages:

- Settings of product refinement cycles are possible.
- Allows activity repair whenever mistakes are made.
- Enhances knowledge of the process as a result of repetition.

Disadvantages:

- Developers may fall into indefinite cycles and therefore lose control of the process.
- It is necessary to make a big efforts and large investments.

B. Patterns according to Participants

Modeling of participants allows to reflect the most important resource in a software process [21]. Individuals and interactions over processes and tools [22]. According to the stakeholders of the process, developers, their communication and role rotation can be modeled.

1. Doer Pattern

Stakeholder theory is an area of strategic management that defines a stakeholder as someone who affects or is affected by the actions of the organization [23], [24], [25], then the processes of the organization are reflected in the software process and these in turn, by the individuals according to Conway's law. Doer pattern allows clear identification of the key parts present when in the execution of the process, namely the doers of the software process and also the consumers along the process. A doer of the software process directly executes an activity of the process and is responsible for carrying it out, while a consumer is the one who benefits from the products of the process without directly affecting the corresponding activities, except for the activities perception. In other words, while a director is an active performer, a consumer is passive. (Figure 4).

Figure 4. Doers

Consequences:

Defining who executes the software development process is essential to define roles and responsibilities and clear roles. Roles also isolate the posibility of noise that is caused by ghost roles (non-doers), who appear in the process due to poor scope estimation.

Advantages:

- It is clearly identified who will perform the process, allowing a good estimate of resources.
- This pattern encourages the identification of roles, functions and responsibilities.
- Ghost roles, which generate noise in the process, are removed.

Disadvantages:

- The identification of roles should be consistent with their own process-specific creation.
- Ensuring the existence of doers for a particular software process means a considerable investment in skilled labor.
- Organizations do not have the wide range of doers that may arise in the process.

2. Communication Pattern

Communication, or rather lack thereof, leads to tremendous problems in the workplace and in software [26], the quality of communication within the development team and between the development team and external entities impacts on the performance of the software project [27]. Communication undoubtedly impact the software development [28]; this development may flow better if the interactions of the participants allow effective exchange that is regulated by good communication mechanisms. Communication pattern enhances the communication structure within the process. It identifies the communication transmitter and receiver as well as the channel and the message. Communication is the key to maintaining synergy in the process; it is also the best mechanism to maintain integrity within a project. See Figure 5.

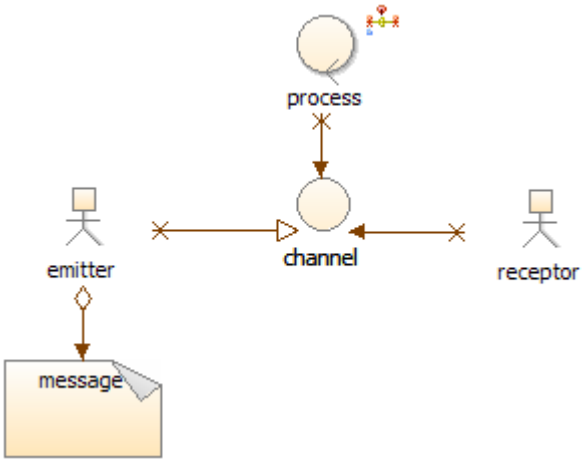

Figure 5. Communication

Consequences:

Keeping a communication scheme facilitates the monitoring and the process control. It eliminates the noise generated by information islands and promotes strengthening knowledge management processes

Advantages:

- It facilitates the exchange of experiences and insights within the project.
- There is elimination of the noise generated by information islands.
- It promotes role societies and the creation of collaborative communities.

Disadvantages:

- Difficulties arise when there are particular interests.
- It is difficult when developers do not speak within the same knowledge domain and there is no attempt to use interfaces in such cases.

3. *Role Rotation Pattern*

Roles are very useful in modeling the authority, responsibility, functions, and interactions associated with managerial positions within organizations [29]. Roles in software projects should be similar to a surgical team where there is clarity in the responsibilities with a hierarchical collaboration [30]. There have been studies that show the importance of the dynamics that should have the roles within the organization and the possibility of change as a way of empowering their activities [31]. Role Rotation pattern defines the impact one role may have on the transformation of another role, motivated by doers engaging in a new activity that is different form their previous activity. To take this step, it must be taken into account that the role is competent to assume such a responsibility. This can be seen in two ways. In the first approach, a role produces certain qualities that will be used by a role that accepts them as input. A role played by the same author who assumes another role. (Figure 6).

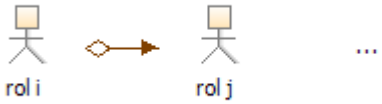

Figure 6. Role Rotation

Consequences:

For an organization, stepping from one role to another within the mind of one actor is essential so that the actor has a broader view of the process he is running. This results in maintaining continuity and consistency in the development process.

Advantages:

- The actor who rotates activities maintains a more complete understanding of the process.
- It is possible to further extend the human resource.

Disadvantages

- Actors get overloaded.
- There is partial and collaborative invasion on actors

C. Patterns based on Inputs - Outputs

In a process, some settings may appear, such as the input-output setting, the document-management setting and the traceability setting, each emphasizes in some features related to the input-output of the process.

1. *Input-Output Pattern*

The classical Leontief model on the correlation of the economy in different industries with respect to their inputs and outputs [32], is also a generalized model of software processes, for example in testing [33], [34], requirements engineering [35], programming [36], among other activities of the development process. Input-Output pattern allows clear identification of the process inputs that are required for its execution and that will be transformed to achieve the expected outputs (Figure 7).

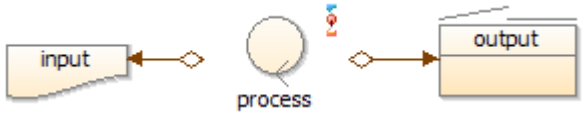

Figure 7. Input - Output

Consequences:

Defining inputs and outputs allows estimating the resources that will be transformed by the process into products and / or services to be consumed. It is important to understand what enters the process and what exits in order to plan how the process will behave and so determine the needs and outcomes.

Advantages:

- It clearly identifies the resources used for the process and the results to be obtained after completion of the process.
- It encourages the planning of the software process since the available resources as wel as the desired outcomes are known.
- It promotes the structuring of the process to transform inputs into outputs, represented in products and / or services.

Disadvantages:

- It is not always easy to identify inputs and outputs.
- Traceability of an input into a product is a wasteful and costly task.

2. Document Management Pattern

Documentation is a factor to consider when you want to succeed in a software project [37], people within a development process tend to have a shared understanding of the software documentation [38], creating the channel through which communication flows and provides support for the project; there are patterns of documentation which detail problems related to intensive use and interaction between the documents [39], the pattern proposed in this paper is a general pattern present in the software development process from the perspective of its use and generation. Document Management pattern clearly identifies the documentation required for the process and the resulting documents after execution of the process (Figure 8).

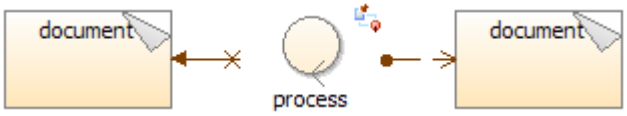

Figure 8. Document Management

Consequences:

Defining the documentation used and also the resulting documentation in the software process is critical to support each of the activities carried out and to support the decisions that are defined as the process unfolds. A document sets the source and history that deals with the project management. When you lose a role, that role represented the mind of an expert. A document that builds a good description of the tasks performed by this role and also of his decisions and experiences in a single record, is are a key asset for the organization.

Advantages:

- It supports decisión making.
- It storages requirements, contracts and agreements.
- It allows retaking actions based on decisions previously recorded.
- Corrections and defect tracking are recorded.

Disadvantages:

- The cost is too high for its realization.
- Updating and maintenance is wasteful.
- The documentation becomes another project that is parallel to obtaining the code. Documents and code should match and support each other 100%.

3. Traceability Pattern

Traceability plays an important role in facilitating software evolution [40], in software maintenance and reengineering [41]; in general, traceability is critical to maintain consistency between business processes and system software [42]. Traceability pattern establishes the way artifacts are linked in a process to illustrate how an idea can be transformed into a product resulting from a concept that starts from an abstraction until getting a concrete product (Figure 9).

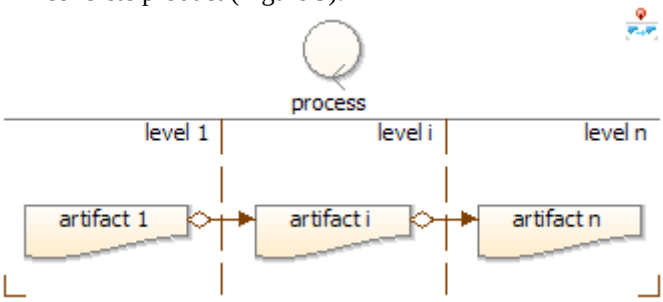

Figure 9. Traceability

Consequences:

Traceability enables actors to etablish a road map of the different elements that are developed within the software process. Each concept sets a milestone that can be woven together with the others to ceate a consistent tissue that is visible and understandable.

Advantages:

- It allows displaying the process.
- It establishes a roadmap, based on the tissue, which forms traceable items.
- There is support to repair errors due to the easy identification of their causes.

Disadvantages:

- There is difficulty in the necessary traceability that must be carried out from the idea itself to its realization.
- There is a lack of inclusive language in the different layers of abstraction.

D. Knowledge Patterns

Software companies can decrease the time and cost for development, increase quality, and make better decisions if they manage their knowledge better[43]. There is a direct implication of knowledge management as a theory support for many aspects of software engineering, one of these trends is the school of engineering toward the process [44]. Software process from the perspective of knowledge, can produce knowledge, change their states, and reside in process participants, this is reflected as patterns.

1. Knowledge Production Pattern

Knowledge is produced by the interactions of participants through the processes they run [45]. If we see software engineering as a process it is clear that their results are also knowledge. Knowledge Production pattern provides the knowledge used within a process and the knowledge that is produced: On one hand, the knowledge used is the conceptual framework necessary for carrying out the process, while the knowledge gained is the result of empirical experimentation, resulting in the execution of the process. See Figure 10.

Figure 10. Knowledge Production Pattern

Consequences:

Defining the conceptual framework to be enlarged in a process is critical because it establishes the characteristics that qualify the process and that will be the input to obtain a wealth of experience, which ultimately forms the generated knowledge.

Advantages:

- This pattern promotes clear identification of the concepts to substantiate and characterize the process.
- It encourages the establishment of the roles that contain knowledge.
- It generates new knowledge from the experience gained when implementing the process.
- Knowledge sets the differentials in the use and performance of processes.

Disadvantages:

- Knowledge is difficult to appropriate by the organization.
- Knowledge is volatile when those responsible for assuming the roles are also volatile in the process.
- Knowledge and experience are not easily transferable to new scenarios and projects.

2. Knowledge States Pattern

With this pattern, the possible states of knowledge are set, and the processes that affect such states are formed by states and transactions. In the software process, this is a valuable resource for monitoring the development from the perspective of the artifacts produced, as it is typically done, but adding a description of the knowledge involved to obtain such results (Figure 11).

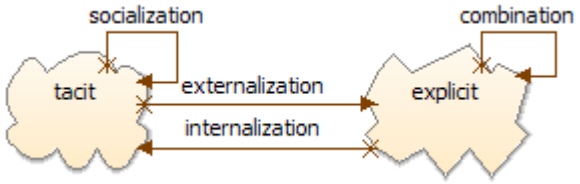

Figure 11. Knowledge States

Consequences:

A definition from the perspective of knowledge management proposed by Nonaka [9]. Socialization processes between aspects such as: tacit knowledge, processes of combination between explicit knowledge, externalization processes from tacit to explicit knowledge, and internalization from the tacit to explicit knowledge; form an important conceptual framework for managing organizational knowledge.

Advantages:

- Defining the states knowledge passes through allows identifying not only the principles that affect it, but also the conditions that surround such knowledge within the software process.
- It is clear that distinguishing tacit knowledge from explicit knowledge allows locating the origin of knowledge, provided it is possible to encrypt it through computer systems, or else, provided it resides in people.

Disadvantages:

- Managing knowledge is not always clear and requires greater effort on the process.
- An additional knowledge-management expert role is necessary for the process.

3. Knowledge Bowl Pattern

The software crisis is due to a knowledge gap resulting from the discrepancy between the knowledge integrated in software systems and the knowledge owned by organizational actors [47], people involved play a major role, because upon them rests the knowledge. Knowledge Bowl pattern establishes who the source of knowledge is. Such a source may reside in an author or a role, and knowledge can be soft or hard knowledge (Figure 12).

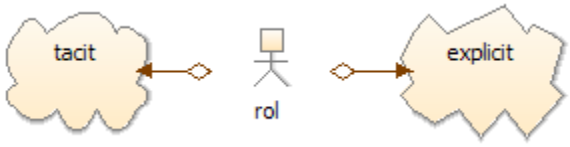

Figure 12. Knowledge Bowl

Consequences:

Defining the source of knowledge present in a software process is critical because it allows identifying both the actors and the important roles within the process, like for example knowledge systems and networks that are being developed during the management of the process.

Advantages:

- Prioritizing the roles and actors who posess valuable knowledge allows managing the knowledge that resides in them.
- The process is guided in its implementation by using the knowledge being generated.
- A knowledge-based process is more reliable and robust.

Disadvantages:

- It is not easy to identify the knowledge repository.
- Knowledge management is a task that goes beyond the engineering discipline.

IV. PROCESS PATTERNS SUPPORT THROUGH SOFTWARE

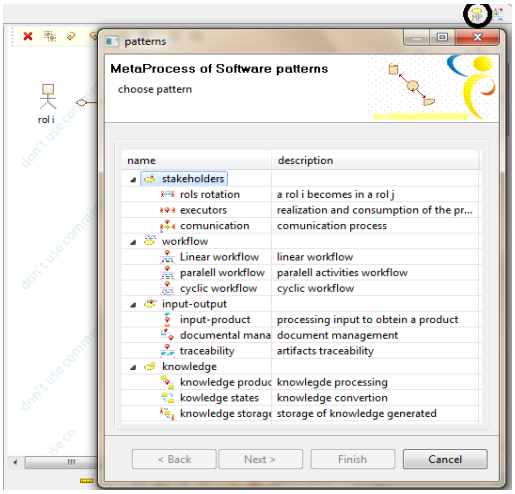

Figure 13. Coloso Software, www.colosoft.com.co

One of the advantages of having a catalog of patterns for software processes is the power to implement the catalog using automated tools, in this case, a process patterns component has been developed for the Coloso platform [10] - (Figure 13) . Patterns implemented on the Coloso PSEE [49] - Process-centered Software Engineering Environments - are modeled with the Process Modeling Language PML [50] that is not matter in this paper but can be used in the tool.

V. CONCLUSION

In the same way software patterns have been proposed in other engineering disciplines such as software design, software architecture, analysis, and so on; in the area of software processes, a proposal that compiles a set of best practices that occur repeatedly in the software development process is also needed. The patterns proposed in this article, constitute a valuable vocabulary to facilitate communication among the participants in a software process, who will be able to quickly and accurately identify the way a software development process has been structured.

The proposed patterns, compile common elements that are present in the software development processes and methodologies. Additionally, these patterns clearly outline aspects such as the structure, the participants, the knowledge and the input-output, which constitute a software development process and through which it is possible to trace a path towards good practice and implementation of a software process.

REFERENCES

[1] Jessop, A., Pattern language: A framework for learning. European Journal of Operational Research, Volume 153, Issue 2, 1 March 2004, pp. 457-46.

[2] Seffah, A., The evolution of design patterns in HCI: from pattern languages to pattern-oriented design. ACM PEICS '10: Proceedings of the 1st International Workshop on Pattern-Driven Engineering of Interactive Computing Systems. 2010.

[3] Christensen. H. B., Frameworks: putting design patterns into perspective. ACM ITiCSE '04: Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education. 2004.

[4] Gamma, E., Helm, R., Ralph, J., & Vlissides, J. *Design Patterns.* Addison Wesley. 1994.

[5] Larman, C. *UML y Patrones.* Pearson. 2003.

[6] Dikel, D., Kane, D., & Wilson, J. *Software Architecture.* Prentice Hall. 2001.

[7] Shaw, M., & David, G. *Software Architecture.* Prentice Hall. 1996.

[8] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. *Pattern-Oriented, Software Architecture.* John Wiley & Sons. 2002.

[9] Fowler, M. *Analysis Patterns.* Addison Wesley. 1997.

[10] Abdullah, N. N. B., Honiden, S., Sharp, H., Nuseibeh, B., Notkin, D., Communication patterns of agile requirements engineering. Proceedings of the 1st Agile Requirements Engineering Workshop, AREW'11 - In Conjunction with ECOOP'11 , art. no. 1. 2011.

[11] Fowler, M. (2011, octubre). *Martin Fowler.* Disponible en http://martinfowler.com/articles/writingPatterns.html

[12] Royce, W. W., Managing the development of large software systems. Proc. of IEEE WESCON . 1970. pp. l-9

[13] Laplante, P., Neill, C., The Demise of the Waterfall Model Is Imminent. ACM Queue, Volume 1 Issue 10. 2004.

[14] Pressman, R. *Ingenieri de Software.* Mc Graw Hill. 2002.

[15] Forsberg, K., Mooz, H., The relationship of system engineering to the project cycle. At NCOSE, Chattanooga, Tennessee. 1991.

[16] Ruparelia N. B., Software development lifecycle models. ACM SIGSOFT Software Engineering Notes , Volume 35 Issue 3. 2010.

[17] Emami, M. S., Ithnin, N. B., Ibrahim, O., Software process engineering: Strengths, weaknesses, opportunities and threats. Networked Computing (INC), 2010 6th International Conference on. 2010.

[18] Graham., DR. Incremental development: review of nonmonolithic life-cycle development models. Information and Software Technology, Volume 31, Issue 1, January-February 1989, pp. 7-20.

[19] Madhavji, N. H., The Process Cycle. *Software Engineering Journal*, September 1991.

[20] Boehm, B. W., A spiral model of software development and enhancement. IEEE Computer. Vol 21 No 5. 1988. pp. 61-72

[21] Voinov, A., Bousquet, F., Modelling with stakeholders. Environmental Modelling & Software, Volume 25, Issue 11, November 2010. pp. 1268-1281.

[22] Manifesto for Agile Software Development http://www.agilemanifesto.org/.

[23] Power, K., Stakeholder Identification in Agile Software Product Development Organizations: A Model for Understanding Who and What Really Counts. AGILE Conference, 2010. 2010. pp 87 – 94.

[24] Kulkarni, V., A Conceptual Model for Capturing Stakeholders' Wish List. Computer Science and Software Engineering, 2008 International Conference on. 2008. pp. 275 – 278.

[25] Williams, C., Wagstrom, P., Ehrlich, K., Gabriel, D., Klinger, T., Martino, J., Tarr, P. Supporting enterprise stakeholders in software

projects. ACM CHASE '10: Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering. 2010.

[26] Huffman Hayes, J., Do you like Pina Coladas? How improved communication can improve software quality. Software, IEEE. 2003. pp 90 – 92.

[27] Hall, T., Wilson, D., Rainer, A., Jagielska, D., Communication: the neglected technical skill?. SIGMIS CPR '07**:** Proceedings of the 2007 ACM SIGMIS CPR conference on Computer personnel research: The global information technology workforce. 2007.

[28] Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., and Still, J., The impact of agile practices on communication in software development. Empirical Software Engineering, Volume 13, Number 3. 2008. Pp 303-307.

[29] Haibin Zhu, MengChu Zhou, Seguin, P., Supporting Software Development With Roles. Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on. 2006. pp 1110 – 1123.

[30] Brooks, F. P. Jr., The Mythical Man-Month. Addison - Wesley, 1995.

[31] Biddle B. J. and Thomas E. J., Role Theory: Concepts and Research, Wiley, 1966.

[32] Leontief, W., Input-output economics. Oxford University Press. 1966-1986.

[33] Cheng, C., Dumitrescu, A., Schroeder, P., Generating small combinatorial test suites to cover input-output relationships. Proceedings. Third International Conference on Quality Software. 2003. Pp 76-82.

[34] Hoare C. A. R., An axiomatic basis for computer programming. Communications of the ACM. 1969.

[35] Kermarrec, Y., Zein, O., Le Pors, E., Grisvard, O., Towards the use of requirements as valuable inputs for complex system design. Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on. 2008.

[36] Osogami, M., Yamanishi, T., Uosaki, K., A method of input-output conditions for automatic program generation using Petri nets. SICE Annual Conference (SICE), 2011 Proceedings of. 2011. pp. 2415 – 2420.

[37] Nasution, M. F. F., Weistroffer, H. R., Documentation in Systems Development: A Significant Criterion for Project Success. System Sciences, 2009. HICSS '09. 42nd Hawaii International Conference on. 2009. pp 1-9.

[38] de Boer, R. C., van Vliet, H., Writing and Reading Software Documentation: How the development process may affect understanding. Cooperative and Human Aspects on Software Engineering, 2009. CHASE '09. ICSE Workshop on. 2009. pp. 40-47.

[39] Correia, F. F., Aguiar, A., Sereno, H., Flores, N., Patterns for consistent software documentation. ACM PLoP '09**:** Proceedings of the 16th Conference on Pattern Languages of Programs. 2009.

[40] Rochimah, S., Kadir, W. M. N., Abdullah, A. H., An Evaluation of Traceability Approaches to Support Software Evolution. Software Engineering Advances, 2007. ICSEA 2007. International Conference on. 2007.

[41] Schwarz, H., Towards a Comprehensive Traceability Approach in the Context of Software Maintenance. Software Maintenance and Reengineering, 2009. CSMR '09. 13th European Conference on. 2009. pp 339-342.

[42] Aversano, L., Marulli, F., Tortorella, M., Recovering Traceability Links between Business Process and Software System Components. Program Comprehension (ICPC), 2010 IEEE 18th International Conference on. 2010. pp. 52-53.

[43] Rus, I., and Lindvall, M., Knowledge Management in Software Engineering. *IEEE Software*, vol. 19, no. 3, 2002, pp. 26–38.

[44] Dingsoyr, T., Bjornson, F.O., Shull, F., What Do We Know about Knowledge Management? Practical Implications for Software Engineering. Software, IEEE . 2009. pp. 100-103.

[45] Reihlen, M., Nikolova, N., Knowledge production in consulting teams. Scandinavian Journal of Management, Volume 26, Issue 3, September 2010. pp. 279-28.

[46] Nonaka, I. A Dynamic Theory of Organizational Knowledge Creation. Aroganization Science, 1994. pp. 14- 37.

[47] Dakhli, S., Ben Chouikha, M., The knowledge-gap reduction in software engineering. Research Challenges in Information Science, 2009. RCIS 2009. Third International Conference on. 2009. pp. 287-294.

[48] Bolaños, S. "SPML", Registro de Soporte lógico – software, Libro-tomo-partida 13-28-279, 4-marzo-2011.

[49] Engels, G., Schäfer, W., Balzer, R., Gruhn, V., Process-centered software engineering environments: academic and industrial perspectives. ICSE '01**:** Proceedings of the 23rd International Conference on Software Engineering. 2001.

[50] Derniame, J. C., Kaba, B. A., Wastell, D., Software process: principles, methodology, and technology. Springer-Verlag Berlin Heidelberg. 1999.