International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)
Web Site: www.ijettcs.org Email: editor@ijettcs.org
Volume 6, Issue 3, May- June 2017 ISSN 2278-6856

Development of Software Lifecycle Management

Deepak Kumar

Assistant Professor, Department Of Computer Science And Engineering, Guru Gobind Singh Educational Society's Technical Campus, Bokaro Steel City, India

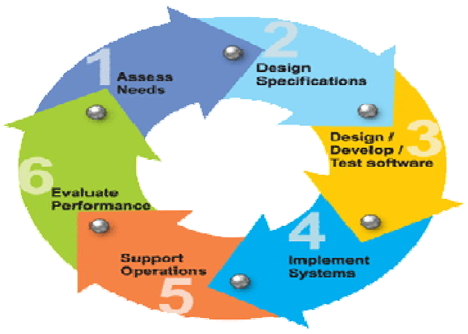
Abstract

Software Life Cycle Development (SLCD) is a step by step highly structured technique employed for development of any software. This enables cost savings as organizations can reuse and tailor SW products developed by other organizations free of charge. SLCD allows project leaders to configure and supervise the whole development process of any software. There are various SLCD models widely accepted and employed for developing software. The state-of-the-art and state-of-the-practice are well established in open-source development and SW product lifecycle management but they need to be applied for public sector. SLCD models give a theoretical guide line regarding development of the software. Employing proper SLCD allows the managers to regulate whole development strategy of the software. Each SLCD has its advantages and disadvantages making it suitable for use under specific condition and constraints for specified type of software only. Developers employ SLCD models for analyzing, coding, testing and deployment of software system. Software developed by employing the suitable SLCD models is better performers in the market when compared with their competitors. However, without coordination SW products will evolve in an uncontrolled manner. This article argues the importance of application lifecycle management during the evolution of SW products in the public sector's open-source communities. A model which guarantees the development and delivery (release) teams engaged in some project have strong co-ordination and collaboration leading to enhanced productivity, efficiency, effectiveness and longer market life is developed. This can be achieved by incorporating concept of Release Management with basic SLCD phases. This article is also targeted at IT research organizations and software companies that operate with public sector organizations.

Keywords: Software Life Cycle management Development system, software system, open source, application life cycle management public sector and software configuration management.

1.INTRODUCTION

The System Development Life Cycle framework provides a sequence of activities for system designers and developers to follow. The ideas about the software life cycle development (SLCD) have been around for a long time and many variations exist, such as the waterfall, spiral, prototype and rapid application development model. These variations have many versions varying from those which are just guiding principles, to rigid systems of development complete with processes, paperwork and people roles. It consists of a set of steps or phases in which each phase of the SLCD uses the results of the previous one. A Systems Life Cycle Development (SLCD) adheres to important phases that are essential for developers, such as planning, analysis, design and implementation. Currently Software is increasingly used in the private and public sector. Similarities in public sector organizations allow the reuse of SW products using open source like characteristics. This enables cost savings as organizations can reuse and tailor SW products developed by other organizations free of charge. A number of system life cycle development (SLCD) models have been created: waterfall, spiral, prototype and rapid application development model. Various software life cycle development models are suitable for specific project related conditions which include organization, requirements stability, risks, budget and duration of project. One life cycle model theoretical may suite particular conditions and at the same time other model may also looks fitting into the requirements but one should consider trade-off while deciding which model to choose. A software life cycle model is either a descriptive or prescriptive characterization of how software is or should be developed. A descriptive model describes the history of how a particular software system was developed. Descriptive models may be used as the basis for understanding and improving software development processes. A prescriptive model prescribes how a new software system should be developed. Prescriptive models are used as guidelines or frameworks to organize and structure how software development activities should be performed and in what order. However, without coordination these SW products will evolve in an uncontrolled manner and they may not evolve in the direction that best serves the needs of public sector organizations. To enable this coordination, the Ministry of Finance has started a research effort to define models for the management of open-source like SW products' evolution in the public sector (i.e. lifecycle management). To present an abstract definition of software life cycle development model (SLCD) incorporated with release management. There are various SLCD models widely accepted and employed for developing software. SLCD models give a theoretical guide line regarding development of the software. Employing proper SLCD allows the managers to regulate whole development strategy of the software. Each SLCD has its advantages and disadvantages making it suitable for use under specific condition and constraints for specified type of software only. We need to understand which SLCD would generate most successful result when employed for software development. There are various SLCD models such as Waterfall, Spiral, Prototype, Incremental and model etc. The purpose of this article is to describe the models and discuss in more detail the model that was identified as having the most potential for piloting SW product lifecycle management in the public sector in future. Furthermore, the article aims to encourage scientific and professional discussion about the community-based management of open-source software in public sector.



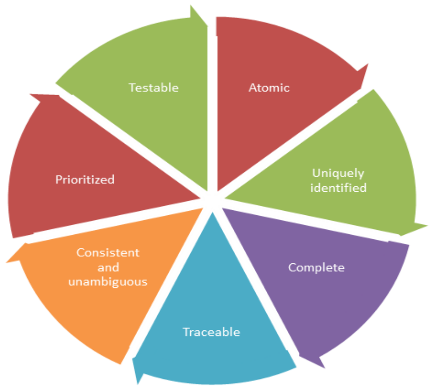
2.Software System Related Work

WHAT IS SLCD

The systems life cycle development (SLCD),also referred to as the application development life-cycle, is a term used in systems engineering, information systems and software engineering to describe a process for planning, creating, testing, and deploying an information system.

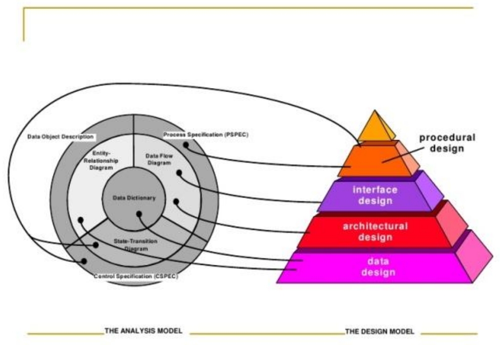
Requirement Analysis

Requirements analysis, also called requirements engineering, is the process of determining user expectations for a new or modified product. These features, called requirements, must be quantifiable, relevant and detailed. In software engineering, such requirements are often called functional specifications. Requirement analysis involves frequent communication with system users to determine specific feature expectations, resolution of conflict or ambiguity in requirements as demanded by the various users or groups of users, avoidance of feature creep and documentation of all aspects of the project development process from start to finish. Energy should be directed towards ensuring that the final system or product conforms to client needs rather than attempting to mold user expectations to fit the requirements. Requirements analysis is a team effort that demands a combination of hardware, software and human factors engineering expertise as well as skills in dealing with people.



Design

Software design is the process of implementing software solutions to one or more sets of problems. One of the main components of software design is the software requirements analysis (SRA). SRA is a part of the software development process that lists specifications used in software engineering. It is the first step to move from the problem domain towards the solution domain. It is the most creative phase in Software Development Life Cycle. The goal of this phase is to transform the requirement specification into structure. The output of this phase is Software Design Document (SDD).



Coding

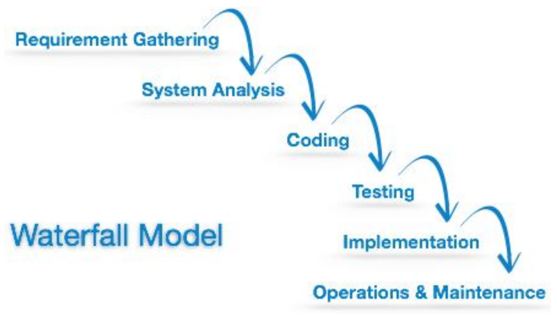
Coding this phase Software Design Document (SDD) is converted into code by using some programming language. Coding conventions are a set of guidelines for a specific programming language that recommend programming style, practices, and methods for each aspect of a program written in that language. Coding conventions are only applicable to the human maintainers and peer reviewers of a software project. It is the logical phase of the Software Development Life Cycle. The output of this phase is program code.

Testing

Software testing is the process of evaluation a software item to detect differences between given input and expected output. Also to assess the feature of A software item. Testing assesses the quality of the product. Software testing is a process that should be done during the development process. In other words software testing is a verification and validation process. Effective testing will contribute to the delivery of high quality software products, more satisfied users, lower maintenance costs, and more accurate and reliable results.

The Waterfall Model

The Waterfall Model was first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed fully before the next phase can begin. This type of software development model is basically used for the for the project which is small and there are no uncertain requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In this model software testing starts only after the development is complete. In waterfall model phases do not overlap.



ADVANTAGES OF WATERFALL MODEL:

- This model is simple and easy to understand and use.
- It is easy to manage due to the rigidity of the model – each phase has specific deliverables and a review process.
- In this model phases are processed and completed one at a time. Phases do not overlap.
- Waterfall model works well for smaller projects where requirements are very well understood.

DISADVANTAGES OF WATERFALL MODEL:

- Once an application is in the testing stage, it is very difficult to go back and change something that was not well-thought out in the concept stage.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

WHEN TO USE THE WATERFALL MODEL:

- This model is used only when the requirements are very well known, clear and fixed.
- Product definition is stable.
- Technology is understood.
- There are no ambiguous requirements Ample resources with required expertise are available freely The project is short.
- Very less customer interaction is involved during the development of the product. Once the product is ready then only it can be demoed to the end users. Once the product is developed and if any failure occurs then the cost of fixing such issues are very high, because we need to update everywhere from document till the logic.

REQUIREMENT ANALYSIS AND DEFINITION

All possible requirements of the system to be developed are captured in this phase. Requirements are a set of functions and constraints that the end user (who will be using the system) expects from the system. The requirements are gathered from the end user at the start of the software development phase. These requirements are analyzed for their validity, and the possibility of incorporating the requirements in the system to be developed is also studied. Finally, a requirement specification document is created which serves the purpose of guideline for the next phase of the model.

ADVANTAGES

- Simple to understand and use.
- Easy to arrange tasks.
- Process and results are well documented.
- Each phase has specific deliverable and a review.
- Works well for projects where requirements are well understood.
- Works well when quality is more important than cost/schedule.

DISADVANTAGES

- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- No working software is produced until late in the life cycle.
- Risk and uncertainty is high with this process model.
- Adjusting scope during the life cycle can end a project

OPEN-SOURCE SOFTWARE

Open source software is computer software that has a source code available to the general public for use as is or with modifications. This software typically does not require a license fee. There are open source software applications for a variety of different uses such as office automation, web design, content management, operating systems, and communications. The key fact that makes open source software (OSS) different from proprietary software is its license. As copyright material, software is almost always licensed. The license indicates how the software may be used. OSS is unique in that it is always released under a license that has been certified to meet the criteria of the Open Source Definition. These criteria include the right to:

- Redistribute the software without restriction;
- Access the source code;
- Modify the source code;
- Distribute the modified version of the software.

In contrast, creators of proprietary software usually do not make their source code available to others to modify. When considering the advantages of open source software you should consider the open source product itself. Open source products vary in quality. OSS software does not come with phone support or personalized e-mail support. However, there are commercial service providers who will provide support. If you need a lot of support, consider whether the overall costs of using an open source product will be higher than that of a proprietary product. Open source software is unique in that it is always released under a license that allows users to access, modify and redistribute the source code. Source code is a specialized language that allows software developers to create and modify computer programs. If you do not have legal access to the source code, then the program cannot be changed or moved to a different kind of computer.

The term "open source" (OS) encompasses both the intellectual property strategy and the development methodology for open-source software (OSS). Community-driven OSS has also gained a market share in computer software, such as operating systems (with the Linux operating system), professional software (MySQL, Apache, etc.) and desktop software like web browsers (Firefox, Chromium), office suites (Libre Office) and graphics software (GIMP). In these market segments and others, OSS competes with proprietary software as a free alternative. Although OSS can be acquired, usually with a lower price or no price at all, the costs of using and adopting OSS are not necessarily lower than those of proprietary software. OSS is developed within communities that consist of individuals (volunteers) and/or company members (or individuals from participating organizations). OSS development projects can be categorized into two subcategories: community-founded projects, launched by individuals, and sponsored projects, launched by commercial companies, government agencies or nonprofit-organizations.

Information and support are spread among the people in the community and more importantly; innovations are shared within the community [8]. The community-based software production and innovation method has also provided business opportunities, as companies have successfully formed business models around OSS.

The community-driven development method has also received some interest from organizations to be used as an internal development method, Wessel us provides a case where an internal open source (ISS) method is used in an organization, with some success. The business drivers of normal open-source software are however missing in this kind of activity, hence adopting an OSS community inside an organization is not a straight forward process.

APPLICATION LIFE CYCLE MANAGEMENT PUBLIC SECTOR

SERVER-APPLICATION-SUPPORT ENGINEER

Application lifecycle management (ALM) is the supervision of a software application from its initial planning through retirement. It also refers to how changes to an application are documented and tracked. ALM is a very broad term that reflects a change in attitude towards software development that is also expressed in the term Dev Ops. Dev Ops blends the tasks performed by a company's application development and systems operations teams. In the past, a development team might work independently using a waterfall development model and hand off the completed software application to an operations team for deployment and maintenance. Today, it is more likely that developers will use an agile model and remain involved with the application after deployment, working with business owners and operations to make incremental changes as needed.

There are many ALM tools available for tracking application changes. These range from dedicated ALM products that monitor an application from inception to completion, automatically sorting files into logical buckets as changes are noted, to simple wikis that require team members to record changes manually. There might be many issues that are likely to occur and will lead to reduce functionality of the software being developed. If these futuristic problems are identified and a help desk is stood against them then the efficiency and effectiveness of the software will increase manifolds. Thus server application support engineers' responsibility increases in early stages of SDLC as most of the requirements both functional and technical are identified and most likely problems that might associate with them are identified. Following are the two main functions being performed by the server application support engineers, Analyze Release Policy: Release policies are high-level statements of how releases are to be managed, organized, and performed in your environment. Policies include management goals, objectives, beliefs, and responsibilities. Use these topics to learn more about release policies and to learn how to view, create, and edit the policies.

Release policy purpose: Release policies are typically written at an overall strategy level. The management directives contained in a policy determine the way in which activities and tasks within a given release work-flow are performed.

View release policy: Use this procedure to view release policies that have been written and posted to the database. These policies provide guidelines for carrying out a release work-flow.

Create/Edit release policy: Use this topic to specify a link to a release policy that you have created or edited. After you create the link, the policy name and description are displayed in the Release Policies table. The policy document can then be opened and reviewed by any user in your environment who is involved with IBM Tivoli Release Process Manager.

Analyze Release Plan

A release planning meeting is used to create a release plan which lays out the overall project. The release plan is then used to create iteration plan for each iteration release. It is important for technical people to make the technical decisions and business people to make the business decisions. Release planning has a set of rules that allows everyone involved with the project to make their own decisions. The rules define a method to negotiate a schedule everyone can commit to.

SOFTWARE PRODUCT LIFECYCLE MANAGEMENT

Continuous innovation, global collaboration, risk management in complex projects, and rapid technological changes are challenges that compel large and small enterprises to react by focusing on core competence, collaborating with partners in product design, engineering and production, or shifting part of the activities in low labor cost countries. Producing complex products in this scenario requires that information about product and process is accessible to the several actors in the value network such as partners, suppliers, and customers. The tendency is to use a PLM strategy to integrate people, processes, business systems, and information in order to manage the product development and support its lifecycle. PLM means product lifecycle management, and its value is increasing, especially for manufacturing, high technology, and service industries. In fact, today PLM is widely recognized as a business necessity for companies to become more innovative in order to meet current challenges such as product customization and traceability, growing competition, shorter product development and delivery times, globalization, tighter regulations, and legislation. Being an innovative business, it does not simply mean creating innovative products, but it also means improving the processes a company uses to realize its products and how it supports them using innovative approaches for a complete product lifecycle. In fact, the aim of PLM is to trace and manage all the activities and flows of data and information during the product development process and also during the actions of maintenance and support in order to identify a new business model that integrates engineering processes and different ICT tools. Working in this direction, PLM enables companies to satisfy the innovative needs of their business. Different ICT systems contain knowledge about the products (e.g., CAD, CAM, PDM, NC, and CM), and the PLM ones allow to integrate all of them. PLM systems are the enabling technology for PLM; they serve as a central hub for product data supporting the collaborative product design and development and the use and management of information in the whole network of actors (i.e., in an extended enterprise) involved in the realization of the product. Therefore, PLM is a holistic business concept; it is both a business approach and a software solution, which during the last years has evolved from a set of engineering oriented tools into an enterprise-level solution. Looking at the literature and web sources, several definitions of PLM are actually available. These definitions have been designed in different contexts; they come from global consulting and research firms, online PLM communities, government agencies, technology and software vendors, universities and academic communities, worldwide PLM experts, and companies from various industries that have implemented a PLM project. All these definitions can be easily shared among industrial and academic definitions or depending on the emphasized perspective: some focused on technological applications, others on processes or strategies, but many of them are very similar.

SOFTWARE CONFIGURATION MANAGEMENT

Software configuration management (SCM) is a software engineering discipline consisting of standard processes and techniques often used by organizations to manage the changes introduced to its software products. SCM helps in identifying individual elements and configurations, tracking changes, and version selection, control, and base lining.

SCM is also known as software control management. SCM aims to control changes introduced to large complex software systems through reliable version selection and version control.

Software configuration management (SCM or S/W CM) is the task of tracking and controlling changes in the software, part of the larger cross-disciplinary field of configuration management. SCM practices include revision control and the establishment of baselines. If something goes wrong, SCM can determine what was changed and who changed it. If a configuration is working well, SCM can determine how to replicate it across many hosts.
The acronym "SCM" is also expanded as source configuration management process and software change and configuration management. However, "configuration" is generally understood to cover changes typically made by a system administrator.

EXPLAINS SOFTWARE CONFIGURATION MANAGEMENT (SCM)

SCM defines a mechanism to deal with different technical difficulties of a project plan. In a software organization, effective implementation of software configuration management can improve productivity by increased coordination among the programmers in a team. SCM helps to eliminate the confusion often caused by miscommunication among team members. The SCM system controls the basic components such as software objects, program code, test data, test output, design documents, and user manuals.

THE SCM SYSTEM HAS THE FOLLOWING ADVANTAGES:

- Reduced redundant work.
- Effective management of simultaneous updates.
- Avoids configuration-related problems.
- Facilitates team coordination.
- Helps in building management; managing tools used in builds.
- Defect tracking: It ensures that every defect has traceability back to its source.

PURPOSES

The goals of SCM are generally:

- Configuration identification - Identifying configurations, configuration items and baselines.
- Configuration control - Implementing a controlled change process. This is usually achieved by setting up a change control board whose primary function is to approve or reject all change requests that are sent against any baseline.
- Configuration status accounting - Recording and reporting all the necessary information on the status of the development process.
- Configuration auditing - Ensuring that configurations contain all their intended parts and are sound with respect to their specifying documents, including requirements, architectural specifications and user manuals.
- Build management - Managing the process and tools used for builds.
- Process management - Ensuring adherence to the organization's development process.
- Environment management - Managing the software and hardware that host the system.
- Teamwork - Facilitate team interactions related to the process.
- Defect tracking - Making sure every defect has traceability back to the source.

With the introduction of cloud computing the purposes of SCM tools have become merged in some cases. The SCM tools themselves have become virtual appliances that can be instantiated as virtual machines and saved with state and version. The tools can model and manage cloud-based virtual resources, including virtual appliances, storage units, and software bundles. The roles and responsibilities of the actors have become merged as well with developers now being able to dynamically instantiate virtual servers and related resources.

REFERENCES:

[1] Ian Sommerville, Software Engineering, Addison Wesley, 9th ed., 2010.
[2] K., Sutherland, J., & Thomas, D. (2001). Manifesto for Agile software development. Retrieved from http://agilemanifesto.org
[3] Bhalerao, S., Puntambekar, D., & Ingle, M. (2009). Generalizing Agile software development life cycle. International Journal on Computer Science and Engineering, 1(3), 222–226.
[4] Boehm, B. (1988). A Spiral model of software development and enhancement. IEEE Computer, May 1988, pp. 61–72.
[5] Boehm, B. (2000). Spiral development: experience, principles and refinements (Special Report CMU/SEI-2000-SR-008). Carnegie Mellon University.
[6] Carr, M., & Verner, J. (1997). Prototyping and software development approaches. Department of Information Systems, Hong Kong: City University of Hong Kong.
[7] Cockburn, A. (2008). Using both incremental and iterative development. STSC Cross Talk, 21(5), 27–30.
[8] Cohen, S., Dori, D., & de Uzi Haan, A. (2010). A software system development Life Cycle model for improved stakeholders' communication and collaboration. International Journal of Computers Communications & Control, 1, 23–44.
[9] Stone, D., Jarrett, C., Woodroffe, M., & Minocha, S. (2005). Introducing user interface design. In D. Stone, C. Jarrett, M. Woodroffe, & S. Minocha (Eds.), User interface design and evaluation (pp. 3–24). San Francisco: Elsevier.
[10] Durrani, Q. S., & Qureshi, S. A. (2012). Usability engineering practices in SDLC. Proceedings of the 2012 International Conference on Communications and Information Technology (ICCT) (pp. 319–324).
[11] Mathur, S., & Malik, S. (2010). Advancements in the V-Model. International Journal of Computer Applications IJCA, 1(12), 30–35.
[12] Munassar, N., & Govardhan, A. (2010a). Comparison between five models of software engineering. International Journal of Computer Science Issues, 7(5), 94–101.
[13] Rahmany, N. (2012). The differences between life cycle models—Advantages and disadvantages. Retrieved from http://narbit.wordpress.com/2012/06/10/the-differences-between-life-cyclemodels-advantages-and-disadvantages/
[14] Richard H. Thayer, and Barry W. Boehm, "software engineering project management", Computer Society Press of the IEEE, pp.130, 1986.
[15] Craig Larman and Victor Basili, "Iterative and Incremental Development: A Brief History", IEEE Computer, 2003.

[16] N. Munassar and A. Govardhan, “A Comparison Between Five Models Of Software Engineering”, IJCSI International Journal of Computer Science Issues, vol. 7, no. 5, 2010.
[17] P.Humphreys,”Extending Ourselves: Computational Science, Empiricism, and Scientific Method”, Oxford University Press, 2004.
[18] Royce, W., “Managing the Development of Large Software Systems”, Proceedings of IEEE WESCON 26, pp.1-9, 1970.
[19] IEEE-STD-610, “A Compilation of IEEE Standard Computer Glossaries”, IEEE Standard Computer Dictionary, 1991.
[20] Andrew Stellman, Jennifer Greene, “Applied Software Project Management”, O'Reilly Media, 2005.
[21] Jim Ledin, “Simulation Planning” PE, Ledin Engineering, 2000.
[22] Software Methodologies Advantages & disadvantages of various SDLC models.mht No.1, January 2010, “Evolving A New sModel (SDLC Model-2010) For Software Development Life Cycle (SDLC)”
[23] PK.Ragunath, S.Velmourougan, P. Davachelvan, S.Kayalvizhi, R.Ravimohan. Hoek, A. van der, Wolf, A. L. (2003) “Software release management for component-based software. Software—Practice & Experience”. Vol. 33, Issue 1, pp. 77–98. John Wiley & Sons, Inc. New York, NY, USA.
[24] Lerner, J. and Tirole, J. 2002. Some Simple Economics of Open Source. Journal of Industrial Economics, 50(2), 197-234.
[25] Bonaccorsi, A. Rossi Lamastra, C. and Giannangeli, S. 2004. Adaptive Entry Strategies under Dominant Standards - Hybrid Business Models in the Open Source Software Industry, SSRN Electronic Journal, pp. 1-23, 2004.
[26] Hecker, F. 1999. Setting Up Shop: The Business of Open-Source Software, IEEE Software, Volume 16 Issue 1, pp. 45-51.
[27] Moreira, M. 2004. Software configuration management implementation roadmap, John Wiley & Sons, 244p.
[28] Jonassen Hass, A. 2003. Configuration Management Principles and Practice, Addison Wesley.
[29] Bersoff, E., Henderson, V. and Siegel, S. 1980. Software Configuration Management - An Investment in Product Integrity, Prentice Hall.
[30] IEEE Std-828. 2005. IEEE Standard For Software Configuration Management Plans. IEEE Std 8282005 (Revision of IEEE Std 828-1998).
[31] Eskeli, J., Heinonen, S., Matinmikko, T., Parviainen, P. and Pussinen, P. 2010. Challenges and Alternative solutions for ERP's, VTT, Oulu. 55 p. Research report : VTT-R-05936-10 , http://www.vtt.fi/inf/julkaisut/muut/2010/VTT-R-05936-10.pdf
[32] Asklund, U. and Bendix, L. 2002. A study of configuration management in open source software projects. IEE Proceedings of Software Engineering 149(1): pp. 40-46.
[33] Erenkrantz, J. 2003. Release Management Within Open Source Projects, 3rd Workshop on Open Source Software Engineering, ICSE’03, International Conference on Software Engineering, Portland, Oregon, May 3-11, 2003.
[34] Michlmayr, M., Hunt F. and Probert D. 2007. Release Management in Free Software Projects: Practices and Problems, OSS2007: Open Source Development, Adoption and Innovation (IFIP 2.13), Springer, pp295 – 300
[35] Doyle, C. 2007. The importance of ALM for aerospace and defence (A&D), Embedded System Engineering, Vol. 15, No. 5, June, pp. 28 - 29.
[36] Doyle, C. and Lloyd, R. 2007. Application lifecycle management in embedded systems engineering, Embedded System Engineering, Vol. 15, No. 2, March, pp.24 – 25.
[37] Chappell, D. 2008. What is application lifecycle management, Chappell & Associates, White paper. Murta, L., Werner, C. and Estublier, J. 2010. The Configuration Management Role in Collaborative Software Engineering, In: Mistrík et al.,
[38] Collaborative Software Engineering, SpringerVerlag, Berlin, Heidelberg, pp. 179 – 194.
[39] Rajlich, V. and Bennett, K. 2000. A staged model for the software life cycle, Computer, July/2000.
Capiluppi, A., González-Barahona, J., Herraiz, I. and Robles, G. 2007. Adapting the “Staged Model for Software Evolution” to Free/Libre/Open Source