Design Paradigm and Risk Assessment of Hybrid Re-engineering with an approach for development of Re-engineering Metrics

Sandhya Tarar[1], Dr. Ela Kumar[2]

Research Associate[1], Professor and Dean[2]
School of ICT, Gautam Buddha University, Greater Noida, India[1, 2]

tarar.sandhya@gmail.com[1], ela_kumar@gbu.ac.in[2]

Abstract:

Software re-engineering, a recent research area includes reverse engineering & forward engineering while Hybrid re-engineering incorporates both the engineering processes where reverse engineering applies to existing system code to extract design & requirements, although this is often used as means to mitigate risks & reduced costs of operation and maintaining the software system. This paper briefly describes traditional re-engineering then discusses the emerging process of hybrid-reengineering which is often used as means to simplify the cumbersome tasks. The paper represents how maintenance is going to be effect with the help of given software engineering approaches. An analysis of various possible risks, their impact and mapping with various attributes is correspondingly depicted. This paper is also presenting the way to reduce the impact of most of these risks by using hybrid re-engineering.

***Keyword Used*:**

Hybrid re-engineering, commercial off the Shelf (COTS), Service Oriented Architecture (SOA), Re-engineering, Model Driven Engineering (MDE), Defect Removal Efficiency (DRE) and Maintenance, Mean Time To Repair (MTTR), Mean Time to Failure (MTTF), Engineering (Engg.).

1. Introduction

Re-engineering is the examination, analysis and alteration of an existing software system to reconstitute it in a new form, and the subsequent implementation of the new form [1].Hybrid Re-engineering "is a re-engineering process that uses not just a single, but a combination of abstraction levels and alteration methods to transition an existing system to a target system [4]. Though the task of hybrid-reengineering is quite appropriate than normal re-engineering but there are plenty of risks associated with various principles such as Re-think & Re-specify, Re-Design, Re-Code, Re-Test i.e. Technical risk, known risk & project risk belonging to various sub categories of risk such as selection of code translation, operational, line by line translation is not possible, quality, reliability, external, interfacing of COTS with legacy system, schedule, COTS will

perform up to the standards. The process typically encompasses a combination of other processes such as reverse engineering, re-documentation, restructuring, translation and forward engineering. As the software industry moves to a new era, many new software design methodologies are developed, improving software reusability and maintainability, and decreasing development and maintenance time. But most companies have legacy systems that are out of data and costly to maintain. These systems cannot just be replaced with new systems. They contain corporate information and implied decisions that would be lost. They also are an investment, and were too costly to develop and evolve just to discard.

For these purposes, re-engineering becomes a useful tool to convert old, obsolete systems to more efficient, streamlined systems. But project development is always short on time and money, making the need to look at alternatives necessary. The use of COTS packages is seen as a way to increase reliability while decreasing development and test time. Translation of code is a means of decreasing time and cost. This has resulted in a combination of the development methods into a form of hybrid re-engineering. Hybrid re-engineering is a reengineering process that uses not just a single, but a combination of abstraction levels and alteration methods to transition an existing system to a target system.

1.1 **Re-engineering Model:** The goal is to understand the existing software (specification, design and implementation) and then to re-implement it to improve the functionality, performance or implementation of the system and it is used to maintain the existing functionality and prepare for functionality to be added later, achieving greater reliability, preparation for functional enhancement, improve maintainability & migration [1]. Software re-engineering was described by Chikofsky, E. and Cross [18].

1.2 **System Re-engineering:** Re-structuring or re-writing part or all of a legacy system without changing its functionality considered as system re-engineering [2]. It applies where sub-system of a larger system require frequent maintenance [3]. Re-engineering involves adding effort to make functionality of those systems easier to maintain. The system may be re-structured and re-documented [13]. Generally, there are following phases of software re-engineering : Source code translation, Reverse engineering, Program structure improvement, Program modularisation and Data re-engineering.

2. Related Work:

Re-engineering is generally discussed as "business process change". Such change imposes new requirements on systems. Pooley et. al. include re-engineering in business process change not only changes over time within one organization but also the situation presenting many of the same problems in which a system developed in one organization and to be used in another[15]. Expert in re-engineering are much rarer than are experts in design and most of the engineers do not have much research experience in this area[13]. The problems with legacy systems had posed everywhere in the world. Brodie et. al. define a legacy system as one that significantly resists modification and evolution to meet new and constantly changing business requirements regardless of the technology used to design it[17]. The legacy system is replaced by a new system with the same or improved functionality [13].

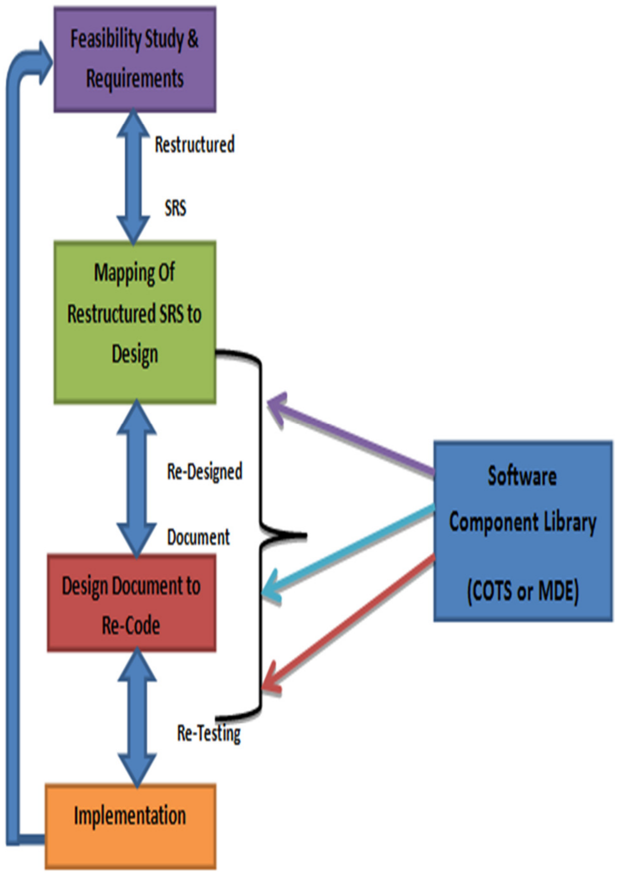

Feasibility Study & Requirements
Restructured
SRS
Mapping Of Restructured SRS to Design
Re-Designed
Document
Design Document to Re-Code
Re-Testing
Implementation
Software Component Library
(COTS or MDE)

Figure 1- Hybrid Re-Engineering Model

3. Hybrid Re-engineering

Hybrid re-engineering is a re-engineering process that uses not just a single, but a combination of abstraction levels and alteration methods to transition an existing system to a target system.

As shown in the figure 1, reverse engineering is the process in which we start from the implementation phase and moving towards the coding, design and requirement phase. Forward engineering is the process which provides downward abstraction from requirement to the implementation phase. In above figure reverse engineering & forward engineering both are represented simultaneously. At first step, the feasibility study is being done i.e. checking system compatibility, after the feasibility is done requirements are re-specify as per the need of the user. The Software Requirements Specification, an output of the requirement phase matters a lot as it consists of all the requirements in written form and is a legal document. In order to re-specify the requirements we need to map these with the SRS. Then after first phase we move towards the next phase i.e. mapping of restructured SRS to the design. The backtracking from one phase to other phase is possible in the figure and is denoted by a bi-directional arrow. In this phase the mapping of restructured SRS to the Design document is being done i.e. integration of new SRS to design in order to get the redesigned document which is the output of this phase. As SRS changes the design structure consisting of DFD/ ER diagrams/UML diagrams need to be changed as per the extent of changed requirements. After the second phase a redesigned document as an output we move to the next phase i.e. Design Document to recode wherein the coding part is being modified as per the changes being made in the design document to make effective the changed requirements. Figure 1 is depicting the way how hybrid re-engineering is working. We can backtrack from this phase to second phase and vice versa if there is such need. The output of this phase leads to retesting & reintegration of the various software modules to perform the exact functionality. As soon as the recode is being done or being modified by the experts so there is the need for testing team to retest the recoded software wherein they check for the bugs, errors & faults and if there is such thing then there is a need to go back to the previous phase or on the first phase of requirements. If everything is perfect so integration is being done among various modules or part of software system to act as a single system. After the integration of various modules are being done, there is a need to implement the modified system and get the target system which is required by the user.

The whole process or the phases as described above tends to the term ***re-engineering*** where in the existing system is being taken off performing the reverse engineering, forward engineering and thereby the target system after the modification being done at every phase. The main advantage of the re-engineering is that it reduces time, effort & money as software is not developed from scratch. Besides the advantages of re-engineering, there are some limitations as there are no metrics available for quantitative measurement. A more sophisticated and efficient approach of re-engineering exists termed as hybrid re-engineering. Hybrid reengineering is an approach in which the part of system which is causing problem is structured again and remaining is kept as it. We can also say that there are two parts of a system i.e. stable & unstable. A stable part is that part that is not causing any problem to the software and need not be touched although the unstable part is that part which is causing problem in the system and hence needs to be reengineered.

In order to do the hybrid reengineering the reengineering model is mapped on with the software component library that can be Commercial off the shelf (COTS) or Math description engineering (MDE). Because hybrid reengineering uses COTS or MDE libraries through which design and requirements can be specified much faster thereby reducing effort, time and increasing reliability. COTS have some risk associated with it i.e. A package will not perform an anticipated or advertised, or that will be unreliable, immature or incomplete. Also COTS product may limit further enhancements to the system because changes in COTS provided functions may not be possible due to legal or contractual issues. All the main principles of hybrid reengineering approach namely re-specify, redesign, recode & retest the main task remains of integrating all the work of these principles together to make an effective model or system. Any specific principle is useless unless there is no integration with the other. **For Example:** If software developer is developing software by following various SDLC phases and works on every phase but did not integrate with each other so one cannot move further to achieve the ultimate objective. So, **custom glue** provides an interface between all these phases to integrate with each other is applied on.

3.1 Benefits of Hybrid Re-engineering:

Hybrid re-engineering has the advantage that it reduces development schedule and hence reduced costs. The development schedule is shortened first by minimizing the amount of reverse engineering. The

use of COTS decreases forward engineering development and test time and thus costs. The use of properly selected COTS also increases reliability because these packages have been extensively tested.

3.2 Limitations of Hybrid Re-engineering:

Since the hybrid re-engineering is a new approach in the re-engineering although a very cost efficient approach as it reduces development time & cost but there are no metrics available for this in order to measure the scalability & performance.

4. Risk Assessment

One method for identifying risks is to create a risk item checklist [6]. Figure 2 is represented different risks associated with software development life cycle phases.

S NO	PHASES	RISKS
1.	Requirements	The package will not perform an anticipated or advertised, or that will be unreliable, immature or incomplete.
2.	Design	Custom code has the same inherent risks as any software developed: Quality, Reliability, and Schedule.
3.	Coding	Availability of experienced personnel.
4.	Testing & Integration	External & Tool Support

Figure 2- Risks Associated With Hybrid Re-Engineering Model

This is the broad classification while as shown in Figure 3, the categorization can be used for risk identification and focuses on some subset of technical, known and project risks in the following generic subcategories:

i. Development Environment (DE)—risks associated the availability and quality of the COTS/MDE tools.
ii. Product Size (PS)—risks impact is directly proportional to the overall product size that has to be hybrid re-engineered/re-build/re-modified.
iii. Size and experience (ST)—risks associated with the overall technical and project experience of the software engineers who will do the task of hybrid re-engineering.
iv. Customer characteristics (CU)—risks associated with the sophistication of the customer and the developer's ability to communicate with the customer in a timely manner.
v. Technology to be built (TE)—risks associated with the complexity of the system to be re-engineered and the "newness" of the technology that is packaged by the system.
vi. Process definition (PR)—risks associated with respect to the process followed by the organization to perform hybrid re-engineering.
vii. Business impact (BU)—risks associated with constraints imposed by management or the marketplace and also the Domain of the business causes risks for performing hybrid re-engineering.

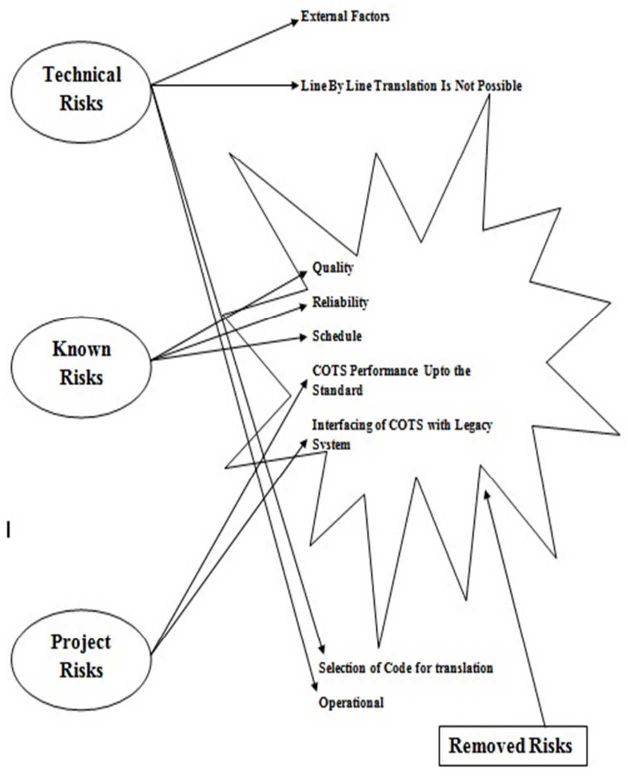

Technical Risks
External Factors
Line By Line Translation Is Not Possible
Known Risks
Quality
Reliability
Schedule
COTS Performance Upto the Standard
Interfacing of COTS with Legacy System
Project Risks
Selection of Code for translation
Operational
Removed Risks

Figure 3- Classification of risks with their category

The cloud in figure 3 is representing the risks those are reduced with the help of hybrid re-engineering. Quality, reliability, Schedule, COTS performance up to the standard and interfacing of COTS with legacy software, these risks are reduced because the development schedule is shortened first by minimizing the amount of reverse engineering and the use of COTS decreases forward engineering development and test time and thus costs. The use of properly selected COTS also increases reliability because these packages have been extensively tested[3][11].

The mapping diagram can be organized in different ways. Questions relevant to each of the topics can be answered for each software project. The answers to these questions allow the planner to estimate the impact of risk. A different risk item checklist format simply lists characteristics that are relevant to each generic subcategory. Finally, a set of "risk components and drivers" are listed. A mapping of various risks and impact on various attributes of software project has been proposed in the paper. This provides useful insight into generic risks for software projects and should be used whenever risk analysis and management is instituted. However, a relatively short list of questions can be used to provide a preliminary indication of whether a project is "at risk". These risks may be catastrophic, critical, marginal or negligible depends on the impact value [10]. In figure 4 all underlined risks are reduced /eliminated by hybrid re-engineering. As shown in below figure 4 various risks associated with the different type of principles are mapped in or linked with each other by direction. Technical risk, threaten the quality and timeliness of the software to be produced. If the technical risk becomes a reality, implementation may become difficult or impossible. Known risks are those risk that can be uncovered after careful evaluation of the project plan the business and technical environment in which the project is being developed. Project risks; threaten the project plan i.e. if the project risks become real it is likely that the project schedule will slip and that costs will increase.

5. Software Metrics for Hybrid Re-Engg.

Metrics is a quantitative measurement of its attributes. Various parameters such as Cost, Reliability, Quality, Object-Oriented Measures, Reusability, Complexity, and Portability can be used to measure the software re-engineering in quantitative form. As shown in figure. 5, corresponding metrics leads to various factors responsible for the measurement of the software re-engineering.

6. Software Maintenance

Software maintenance is the process of modifying a software system or component after delivery to correct faults, improve performances or other attributes, or adapt to a changed environment. In case of hybrid reengineering Corrective & Perfective maintenance is applicable. Here in case of corrective maintenance the developed software is considered and as per the user requirements forward engineering followed by reverse engineering in order to get the changed requirements & finally modify or reengineer

the software system. While in case of perfective maintenance, we apply reengineering processes in order to modify the software system in terms of performance.

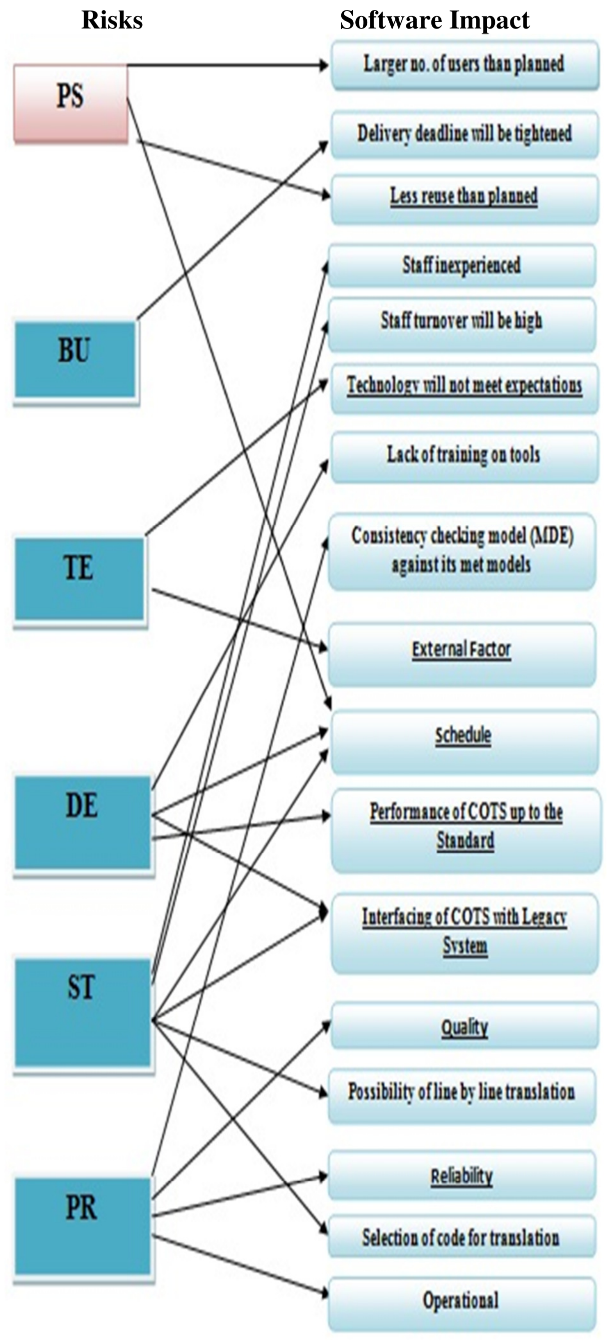
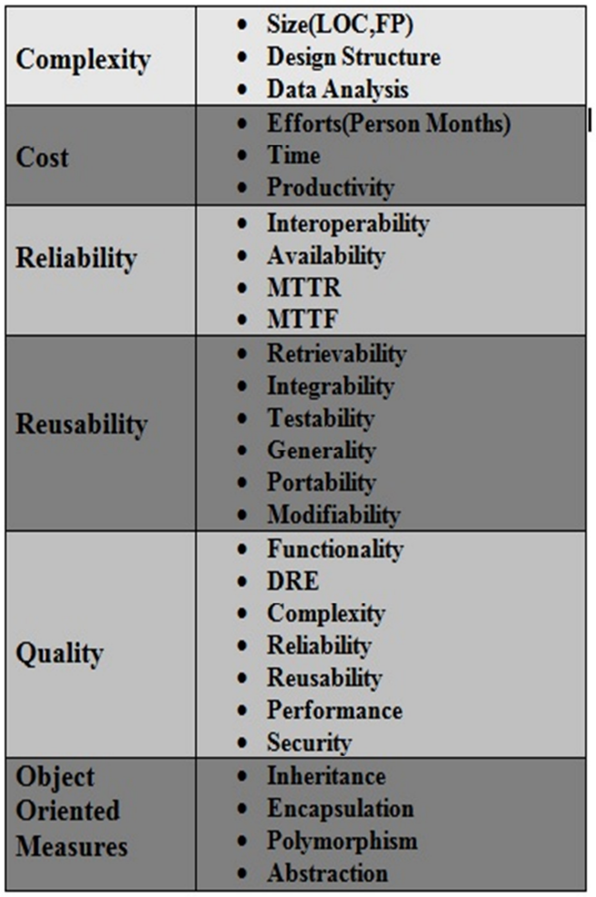

Risks
Software Impact
PS
BU
TE
DE
ST
PR
Larger no. of users than planned
Delivery deadline will be tightened
Less reuse than planned
Staff inexperienced
Staff turnover will be high
Technology will not meet expectations
Lack of training on tools
Consistency checking model (MDE) against its met models
External Factor
Schedule
Performance of COTS up to the Standard
Interfacing of COTS with Legacy System
Quality
Possibility of line by line translation
Reliability
Selection of code for translation
Operational

Figure 4- Mapping of Risks with the impact on software system



Complexity	• **Size(LOC,FP)** • **Design Structure** • **Data Analysis**
Cost	• **Efforts(Person Months)** • **Time** • **Productivity**
Reliability	• **Interoperability** • **Availability** • **MTTR** • **MTTF**
Reusability	• **Retrievability** • **Integrability** • **Testability** • **Generality** • **Portability** • **Modifiability**
Quality	• **Functionality** • **DRE** • **Complexity** • **Reliability** • **Reusability** • **Performance** • **Security**
Object Oriented Measures	• **Inheritance** • **Encapsulation** • **Polymorphism** • **Abstraction**

Figure 5- Hybrid Engineering Metrics

7. Conclusion

Rapid changes in the computer industry continually introduce new hardware and software, making older systems obsolescent and difficult to maintain. Software re-engineering provide reduced risk level. There is a high risk in new software development. There may be development problems, staffing problems and specification problems which are reduced by the use of re-engineering process. Another advantage of software re-engineering is reduced cost. The cost of re-engineering is often significantly less than the costs of developing new software because some part i.e. sub-systems are used as it is and some sub-systems are changed. Based on previous history, project development is usually short on time and money, making it necessary to look at alternatives. Hybrid Re-engineering plays a vital role in this scenario. The use of re-engineering approaches provides a way to increase reliability and quality of the system while decreasing development efforts. Translation of code is a means of decreasing time and cost. This results in a combination of development methods into a form of hybrid re-engineering.

References:

1. Manzella. Mutafelija.: Concept of re-engineering Life Cycle, ICSI Second International Conference On System Integration. IEEE. 1992.
2. Linda H. Rosen Berg, Lawrence E.Haytt. : Hybrid Re-engineering, Software technology conference. Utah, April 1997.
3. MindTree Ltd.: Rehosting & Re-engineering from Mainframe to Wintel, Case Study, 2010.
4. Adolph. W. Stephen. : Cash Cow in the Tar Pit: Reengineering a Legacy System. IEEE Software. May 1996.
5. Wong. Kenny. Tilley. Scott R.. Müller. Hausi A. and Storey. Margaret-Anne D.: Structural Redocumentation: A Case Study. IEEE Software January 1995. 46-54.
6. Ruhl. Mary K. Gunn. Mary T., Software Reengineering: A Case Study and Lessons Learned, NIST, September 2001.
7. Sittenauer, Chris. Olsem. Michael. Balaban. John, Software Reengineering at STSC. Software Technology Support Center. February 1994.
8. Sneed. Harry M., Economics of Software Re-Engineering, Journal of Software Maintenance, September 1991.
9. Sneed. Harry M., Planning the reengineering of Legacy systems, IEEE Software. January 1995.
10. Byrne. Eric J., A Software Re-engineering Process Model. 2nd International Conference On System Integration. 1992.
11. Roger S. Pressman, A practitioner approach of Software Engineering, 6th edition. 2003.
12. Humphrey. W.S. and M. Kellner. Software Process Modeling: Principles of Entity Process Models. Proc. 11th. Intern. Conf. Software Engineering. IEEE Computer Society. Pittsburgh. PA. 331-342. 2002.
13. Shekhar Singh, Significant role of COTS to design Software Reengineering Patterns, International Conference on Software Engineering and Applications(ICSEA),2009.
14. Pooley R., Stevens P., Software Engineering Patterns, SEBPC workshop, 2008.
15. Pooley R., Stevens P., Systems Reengineering Patterns, CSG internal report, 1998.
16. Stevens P., Pooley R., Systems Reengineering Patterns, Proceedings ACM-SIGSOFT, 6th International Symposium on the Foundations of Software Engineering, 2003, pp.17-23.
17. Brodie, Michael L., and Stonebraker, Michael, "Migrating Legacy Systems: Gateways, Interfaces and the Incremental Approach", Morgan-Kaufman Publishers, 1995.
18. Chikofsky, E. and Cross, J., "Reverse Engineering and Design Recovery: A Taxonomy". IEEE Software, 1990, 7(1):13-18.
19. Dewar R., Lloyd A.D., Pooley R., Stevens P., Identifying and communicating expertise in systems reengineering: a patterns approach , IEE Proceedings - Software, vol.146, no.3, 2001, pp.145-152.
20. Lloyd A.D., Dewar R., Pooley R., Legacy Information Systems and Business Process Change: A Patterns Perspective, Communications of the Association for Information Systems, 2001.
21. Dewar R., Characteristics of Legacy System Reengineering, Presented to the writing workshop at EuroPloP'99, Germany, 2002.