2016 IEEE 40th Annual Computer Software and Applications Conference

Knowledge Management Framework for Software Reuse

Simone Maccanti
Robert Morris University
6001 University Blvd, Moon USA
simonemacc@gmail.com

Jameela Al-Jaroodi
Robert Morris University,
6001 University Blvd, Moon USA
aljaroodi@rmu.edu

Arif Sirinterlikci
Robert Morris University,
6001 University Blvd, Moon USA
sirinterlikci@rmu.edu

***Abstract*— In the world of software development, reuse is an important, yet highly underutilized concept. The prospects of saving time, efforts and costs are very high with software reuse. However, the practical issues and challenges imposed are difficult to manage in ways that help optimize the reuse process and reap the benefits. Many papers discussing "knowledge" are focusing on using tools to manage it only on a business prospective. Big companies like AT&T, Wal-Mart, and American Airlines implemented some form of a knowledgebase in past years, yet they are still working on it. They aim to build a complex systems, called Data Warehouse, to aggregate in one 'place' all possible pieces of data and information regarding their business. Apparently the ability to query such systems, gave them the possibility to improve their business, in terms of cost reduction, customers fidelity and appreciation and so on. With such methodology in mind, we are proposing to store the technical knowledge gathered from different software development projects in a specialized knowledgebase to give high tech and software development companies the same benefits. In this paper we propose a methodology to store, organize and manage knowledge to support reuse in a technical field. The framework proposed is a proof of concept to investigate the relevant techniques and possible methods to design such framework. The main issues and components are discussed; however, we have not implemented the full framework at this stage.**

Keywords—Knowledge, knowledgebase, software, software engineering, reuse, framework, methodology

I. INTRODUCTION

High Tech companies generally develop highly sophisticated systems. The development of these systems usually takes several years. Systems are built from scratch one time and then evolve in several and different ways because of new contracts. Every new contract the company uses the same system (let us call it backbone) and applies a set of changes due to the new environment, context, and new set of requirements. The core of the system theoretically does not change however, in reality it does due to these differences, especially for the new requirements.

The continuous changes and adjustments due to the new scope, makes it very difficult reuse a product created for another project. The amount of work needed to adjust the system for the new environment is incredibly high due to the nature of the needed changes. Let us take for example "Project A," a project already done with success. We have all the information, requirements, plans and the software product. We need to develop "Project B," which is in some ways similar to Project A, where we roughly estimate that 80% of technology, software and hardware, can be reused. The two projects, from the customer prospective, have nothing in common, and the requirements for Project B have to be gathered independently as if it was the first time (see Figure 1).

Thus although we plan to reuse a significant part of the original project, there are many changes to make. A big part of the changes are naturally technical. The source code needs to be adjusted or revisited to cover the new requirements, the hardware has to be changed, the interfaces are different plus we need to fix all the issues that are showing up because the core system used as backbone does not fit the new requirements. In other word there is a Merging Effort that can be significantly high.

Things can be made easier to support this type of reuse, if more information is available and this information is organized in better ways. Thus we could create a better way to access and reuse the different components of earlier projects to build new ones. As a result we propose a knowledgebase framework that helps in the process by allowing for better recording, storing and access of project information and documentation.

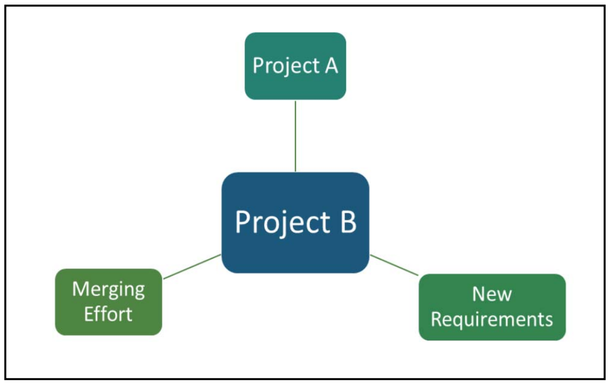

Fig 1 - Project Merging Effort

In the following sections we will first discuss the inadequacies of current practices in Section II. Then we discuss some background work on knowledge management and reuse and briefly introduce our proposed framework in Section III. In Section IV we discuss the proposed knowledgebase and finally in Section V we conclude the paper.

II. INADEQUANCIES

There is not a clear relationship between a design object and the design process [1] and that explains the issues we have even when a reuse methodology is in place. As a result there are several issues to be addressed to be able to facilitate better reuse of software components in multiple projects.

A. Gap Analysis Inadequnacy

One big chunk of the merging effort is something usually called "Gap analysis." During the Bid, a salesman naturally tries to come up with an estimate of the effort needed to cover the gap between the backbone project and the new project we are working with. This gap analysis should include all the aspects that differ between the two projects, the backbone and the actual. This analysis is often dramatically inadequate. Hardware and third party software details are not known at that time, and could take months of unplanned work and countless issues to make the proper adjustments. As project numbers grow and project complexities increase, this process gets even harder and more difficult. As a result, the analysis process takes longer and may lead to inaccurate results.

B. Gap Analysis Schedule Inconsistency

The gap analysis for a project represents all of the gaps from the contract versus the backbone project. The work, fruit of this gap analysis, should be managed through the schedules, but often it is not. This is mainly due to the uncertainties of the process and the lack of adequate knowledge about the relationship between the backbone and the new project.

C. Requirements Discrepancies

Another common issue is the discrepancy between the SRS (Software Requirements Specifications) and the system installed by the software development team. At the beginning the requirements are usually vague and incomplete and matching them with design details is not easy. Not having a clear and stable SRS, it is very difficult to demonstrate the compliance. It is not possible to univocally trace contractual requirements on one or more project requirement.

D. Reuse Fallacy Risks

Another crucial issue, and probably the most important, is the "reuse" of past projects. When a project shows commonalities to previous projects, reuse is the most natural approach mainly for cost reasons. Usually the Engineering Department, based on their deep knowledge of the systems, knows which system can be implemented using previous projects. This philosophy is a fallacy because if on one hand it saves much work, time, and money, on the other, if not done properly, it can lead to problems that will be detected only at a later stage of the projects. Which will result in increased costs and efforts to fix these problems. In fact, there is the tendency to reuse something done in the past which is not entirely applicable to a new requirement, and unfortunately, the "reuse" of the old project is too often used as an excuse when we make design errors.

E. Hardware/Software Management

Having terabytes of source code, analysis and design documents, hardware designs, implementation and testing documents and changes records is a huge patrimony. However, the companies do not find this patrimony as an asset. Empirical studies [2] indicate that Software Engineering, used as a practice to develop cheaper, faster and better software systems, is still immature and companies are still driven by cost reduction strategies, rather than developing useful software engineering practices or techniques. There is not a way to map in details this patrimony and how it evolved, linking for example a software change to what caused it. As a result, it is a lengthy and costly process to go through this information to find some useful links.

III. KNOWLEDGE MANAGEMENT AND REUSE

A lot of work has been done on knowledge management and data warehousing. However, a lot of it is within the business context and does not cover the technical and software development context. In addition, the concept of software reuse has been discussed extensively from various angles. Yet, there is yet much to be done from the software practitioners' perspective. The following is a brief review of some current concepts and research in this area.

A. Reviews

Design reuse is still a developing area. There are ways, for example the "Features model" (a way to find the variation points in the software) [3] or the distribution of commonality and variability in the source code [4], that claim to be effective for the scope of reuse. However, research effort is still needed to better understand the methodology to apply to knowledge users and producers [5]. It is no surprise that around the 20% of the designer's time is spent searching for information, and around 40% of all design information requirements are met by personal stores [6]. It does not even matter if a more appropriate source of knowledge is available.

Business leaders view the knowledge as an asset and the key to sustain a competitive advantage. For this reason companies have started to focus more on what they know and less on what they own. In software engineering, people have been recognized as key holders of valuable knowledge content. Therefore, identifying and capturing what is known by individuals as critical for the business survival [7]. Capturing this knowledge is not an easy task, but Knowledgebase systems (KBS) can offer methodologies to solve "ill-structured" problems (problems that do not have a clearly defined algorithmic solution), where an experienced engineer deals with these ill-structured problems using his/her judgment and experience [8].

What we need is a useful and usable/re-usable source of knowledge: it has to provide exactly what the users' needs and in the forms they need.

B. Proposed Framework

The reuse of code, or technical knowledge in general, is a central aspect of our framework. We are not viewing knowledge as a means for a competitive advantage, but as a vehicle to deliver better and more effective reuse process. By nurturing knowledge, enabling its sharing and use, getting out the knowledge from individual minds, software firms can ensure their long-term advancement and business success [7] .

The concept of "Reuse" is possible when we have precise information about the distribution of commonality and variability [4]. This information is often not available because a methodology for set explicit variation points does not exist.

To address the inadequacies and support the software reuse process, we are proposing a knowledgebase framework. This framework will help acquire and organize available information in ways that will help developers and engineers in identifying and efficiently sing available components in new projects. The first step is to create the framework such that we can:

1. Aggregate all the technical information available about the business and the projects done in one place.
2. Retain the technical knowledge and its variance in accessible forms.
3. Facilitate mining this information/knowledge and make it useful to our business reducing the resources needs for new projects.

Let us introduce a generally standard design process model composed of System Design Planning, System Architecture Design, Preliminary Design, and Detailed Design. Consider the first phase of this model, the "System Design Planning."

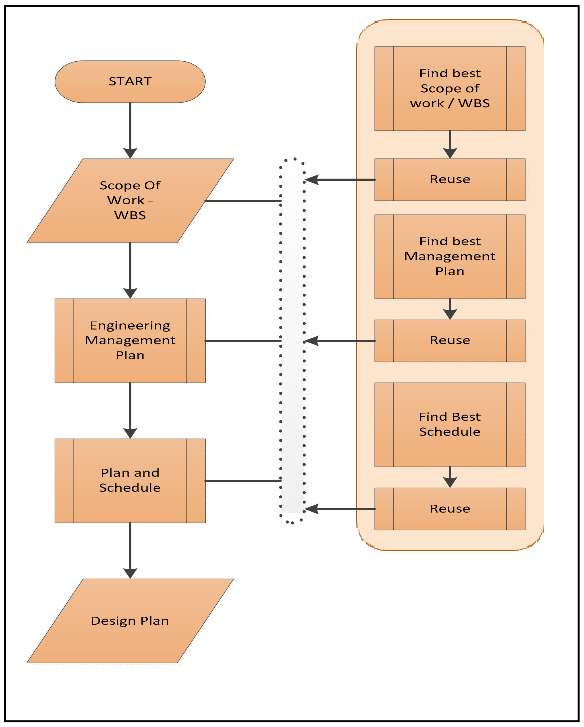

Fig 2 - Process with hidden reuse

Typically a Design Planning framework is composed of the Scope of work, the Engineering management plan, the Schedule, and the Design plan, as shown on the left of Figure 2. What are we missing here? The concept of "reuse" is not included. Usually a simple feedback chain is more than enough to clarify that there will be a re-work, without explaining how and why.

Modern companies with some projects history are not following this flowchart for a simple reason: they already have a Scope of Work, WBS, Engineering Plan, and so on for past projects, which will be reused, or most likely copied. A real Flowchart, for a modern company, has a hidden framework for reuse as shown in Figure 2.

The problem of the reuse approach starts to become clear now. The reuse paradigm is hidden and there is no effective control on what is reused, how, and why. This limits the ability to reuse available components because is not controllable, unfeasible, and in most cases even unrealistic. Users tend to do what they know regardless of the limitations imposed on them. This is the reason why the "reuse" needs to be included in this system and controlled appropriately. However, this is not an easy task as the concept of reuse is linked to the knowledge. The knowledge is composed of organized data or bits of "simple measurement" and patterns that surround us [9], we can reuse what we know (patterns), and to do that we need to find the knowledge which will be our base material for the final product. The core problem is finding what we need, and as we know this paradigm involves "the search," in a database, or more generally in a Knowledgebase to find the desirable results.

IV. KNOWLEDGE BASE

Usually companies have a huge patrimony in documents, software, hardware, design and more, which means a Knowledge Base (KB) needs to be created to enable the search on it. This KB needs be constituted by all the deliverables of all the projects. Naturally the deliverables that have a physical state like a piece of hardware cannot be included in the KB, however, its design documents, functions and other relevant information do.

As we know all the engineering activities are linked to a company's departments to assure the system delivery. There are links that connect the departments with the activities, or most simply provide the needed support to them. The support is the knowledge, which can be collected in a KB. There is no limit to the information that can be collected. Our goal is use this information efficiently to reuse what we did in the past. Most likely the information is not well organized, made up of unstructured and structured data in the form of documents (text), images, diagrams, algorithms, formulas, rules and code.

The problem is organizing all this information in a way that is easy to access and search. There is no magic tool for this, and there are many possible ways to do that. This step depends dramatically on the company and how it organizes the information infrastructure in all of its departments. Companies have to realize that additional resources are needed to create the knowledge and make it accessible and useful. Simply reorganizing the information is not a solution. Reorganizing unstructured information is quite demanding, and that is why we need to build a secondary "Infrastructure" that hosts the KB. The information and raw data from all departments need to be loaded in a separate entity. The data is heterogeneous that is why we need a system able to read and transform the data into a new form. Extraction Transform and Load (ETL) processes are responsible for the extraction of the appropriate data from heterogeneous sources, their transportation to special purpose areas of the data warehouse for processing, and the transformation of the data into a new form to obey the structure of the data warehouse relation to which they are targeted [10]. The most flexible form for such data warehouse is naturally a markup language. This process is shown in Figure 3.

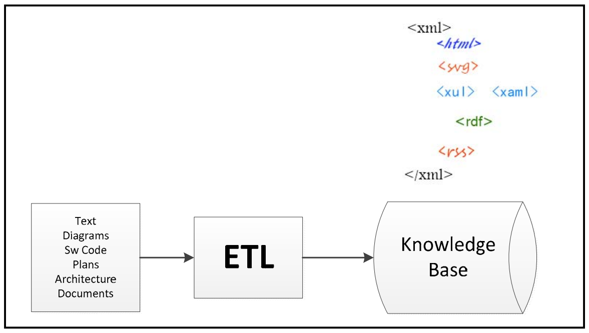

Fig 3 - ETL Knowledge Base

Building the KB is a continuous process. It is not something we build in one go, rather the process is incremental. It relies heavily on accumulating information from various projects as they are designed and built. The KB is also continuously changing and evolving due to the fact that we are dealing with dynamic systems that are constantly changing and evolving. New projects are added, new data, new information and so new knowledge.

To transform data into information and information into knowledge we need a learning paradigm. This paradigm will provide the mechanisms to organize and transform our data, thus allowing us to have more usable access to the final results. The learning paradigm we introduce is distinguished by:

Patterns and Rules: Knowing why something is happening or have happened through the discovery of patterns and rules. The patterns are derived from the various pieces of information gathered over time. The rules are used to distinguish different types of information and decide what patterns they belong to.

Applications: Knowing when to apply procedures or tools and how that relates to the knowledge gathered in the KB. When new projects are initiated, it will be important to identify what areas of knowledge are needed or relevant and search the KB effectively to find this information.

These aspects mentioned above fundamentally answer the questions of what, why and how. Let us take another look at the details of a Design Planning Phase and add the learning component as part of that phase as shown in Figure 4. As we can see the chain of reuse is substituted by a parallel phase called Learning, where we define what, why, and how. The learning process we propose is composed of:

- Extracting information. The extraction process described in the ETL is when the information is not present in the Knowledge Base. In this case the information needs to be searched in other databases. If the information is present in the Knowledge Base it will be extracted from there.
- Rules and patterns. Here where we apply the rules we define to extract information. Patterns are the "why" the information is important, based to a certain rule. The recognition of the patterns has to be done using machine learning: monitoring the user activities.
- Knowledge. When the rules are applied successfully and we are satisfied with the results, knowledge is created, properly marked, and uploaded in the KB

See the Case Study for a practical example.

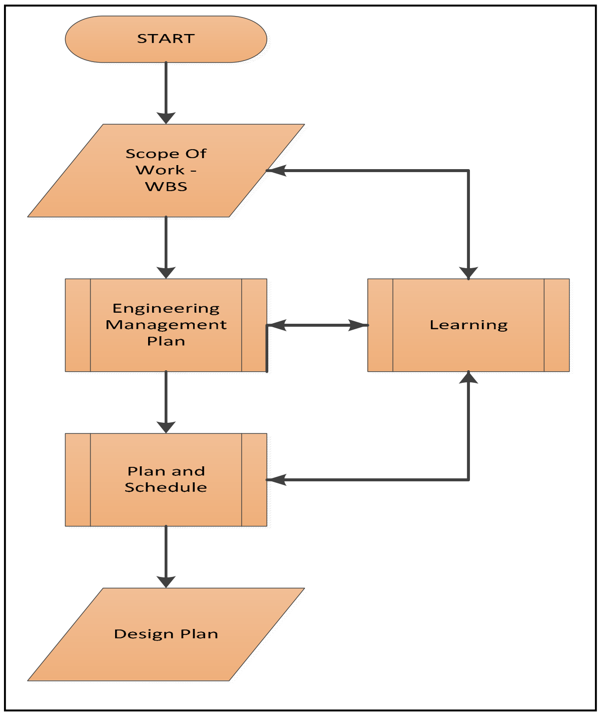

Fig 4 - Learning chain.

This Learning process is shown in Figure 5. The result of this process is to get a satisfactory Δ: the difference between what we have and what we need now to create what needs to be done reusing efficiently our past experience.

The ability to learn from past experiences is a fundamental requirement for the KB Intelligence. Computers, like humans, can be programmed to learn from experience [11]. This learning phase is important because in the near future there could be technologies able to supply reasoning and learning processes. Knowledge-based system, composed by Artificial intelligence and databases of expert knowledge with utilities designed for the knowledge retrieval, can be better than humans in solving problems [12]. This reasoning could be also used for automatic programming [13], however artificial reasoning is needed and so is an efficient knowledge base.

V. CASE STUDY

A software engineer is hired and his/her first task is to develop a new module "M" for functioning software "X." The module will use a system interface "I" already defined. He/she has meetings with a senior engineer that explain what needs to be done. The engineer will query the system defining the rules (what information is needed). For this test case the software engineer, to do the job, will query the system using the following rules:

- Rule 1: "Requisites"

- Rule 2: “The system shall do”
- Rule 3: “Bits for interface”
- Rule 4: “Test plans”
- Rule 5: “Source code”

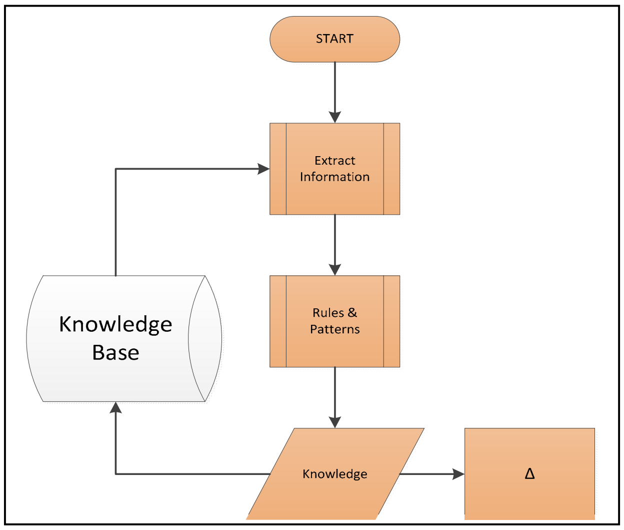

Fig 5 –Learning Process

The software engineer naturally will refine the search, applying more query rules:

1. Requisites for the system “X”
2. Bits of interfaces “I” for the System “X”
3. Test plans for the system “X”
4. Source Code for the System “X”

After the rules are applied and the information is extracted, the engineer will use the information he/she recognize as useful and discard the information he/she does not need. This will create the patterns, unique links between the information extracted and the usefulness. For this test case this could be:

1. Pattern 1: “The requisites 1, 2, 5, 10, 15, and 20 are not useful for the development of module M.”
2. Pattern 2: “Bits for the interface ‘I2” are needed for the development of module M.
3. Pattern 3: “The Software Code for the Interface I2 is required.”

The system recognizes these patterns based to the activity of the user, if this is the first time such queries are used. If this is not the first time this queries are run, other information could be extracted and incorporated with the existing patterns for example as how-to.

VI. Conclusion

In this paper we analyze how the concept of reuse fits in a standard design process model. It is a common practice to reuse what was done before, however adequate controls and suitable and effective methodologies do not exist. Reuse and knowledge are very tight concepts, for this reason a knowledgebase needs to be created to determine efficiently the “Delta” (difference between what we have and what we need now). This Delta helps us create what needs to be done to effective reuse our past experience. Building the knowledgebase is a challenging task and a learning paradigm is needed. The framework discussed in this paper is composed essentially of Rules and Patterns. Defining the rules and patterns well give us the knowledge, and thus the reusable items associated with this knowledge. In addition, the knowledgebase will facilitate better sharing and access to project data for new projects and across multiple teams. The definition of the rules and patterns, especially applied to documentation and software code, is a fundamental aspect. The framework proposed is not implemented yet. Out plan is to design and implement the framework and evaluate it using various test case scenarios as future work.

References

[1] D. Baxter, J. Gao, K. Case and J. Hardnin, "Aa Engineering Design Knowledge Reuse Methodology Using Process Modelling," *Research in engineering design,* 2007.

[2] D. Šmite, C. Wohlin, T. Gorschek and R. F., "Empirical evidence in global software engineering: a systematic review," *Empirical Software Engineering,* 2009.

[3] J. Favaro and M. Morisio, Safe and Secure Software Reuse, Pisa: Springer, 2013.

[4] S. Duszynski, J. Knodel and M. Becker, "Analyzing the Source Code of Multiple Software Variants for Reuse Potential," *Reverse Engineering (WCRE), 2011 18th Working Conference on ,* 2011.

[5] M. L. Markus, "Toward A Theory of Knowledge Reuse: Types of Knowledge Reuse Situations and Factors in Reuse Success," *Journal of Management Information Systems,* p. 40, 2001.

[6] A. Lowe, C. McMahon and S. Culley, "Information access, storage and use by engineering designers, part 1," *The Journal of the Institution of Engineering Designers,* 2004.

[7] A. Aurum, J. Ross, W. Claes and H. Meliha, Managing Software Engineering Knowledge, Springer Science & Business Media, 2013, p. 382.

[8] R. D. Sriram, Intelligent Systems for Engineering: A Knowledge-based Approach, Springer, 2012.

[9] A. Debons, Information Science 101, 2008.

[10] P. Vassiliadis, "A survey of Extract-Trasform-Load technology," *International Journal of Data Warehousing & Mining,* p. 27, 2009.

[11] S. Minton, Learning Search Control Knowledge: An Explanation-Based Approach, Kluwer Academic Publishers, 2012.

[12] A. Rajendra and S. Priti, Knowledge-Based Systems, Jones & Bartlett Publishers, 2010.

[13] C. Rich, R. C. Waters and M. Kaufmann, Readings in Artificial Intelligence and Software Engineering, 2014.