How to Select a Suitable Tool for a Software Development Project: Three Case Studies and the Lessons Learned

Mark Kibanov, Dominik J. Erdmann, Martin Atzmueller
Knowledge and Data Engineering Group, University of Kassel, Germany
{kibanov, erdmann, atzmueller}@cs.uni-kassel.de

Abstract: This paper describes a framework for evaluating and selecting *suitable* software tools for a software project, which is easily extendable depending on needs of the project. For an evaluation, we applied the presented framework in three different projects. These projects use different software development methods (from classical models to Scrum) in different environments (industry and academia). We discuss our experiences and the lessons learned.

1 Introduction

With the growth of the software industry the number of software products (programs, tools, frameworks) with similar functions has also increased. Therefore, the process of selection of the required software has also become more complex. In this paper, we introduce a general three-step framework for selecting suitable software for the current project and environment. Furthermore, we describe the application of the framework to three different software projects: These use different software development methods and environments. All three case studies show promising results and indicate the possibility to apply the suggested framework for a wide range of different projects. In these contexts, we discuss our experiences and the lessons learned.

The rest of the paper is structured as follows: Section 2 discusses related work. After that, Section 3 presents the framework, its advantages, disadvantages and the three distinct steps of selecting the software tools. Section 4 describes three case studies where we applied the framework and the results we obtained during these case studies.

2 Related Work

Starting in 1980 with [Saa80], Thomas L. Saaty developed the AHP (Analytic Hierarchy Process): This method makes it easier for teams to determine the relevant criteria. The Cost-Benefit Analysis is a widely used approach in different disciplines, cf. [Sen00]. General IT-management approaches such as the Capability Maturity Model (CMM) [coo02] and Control Objectives for Information and Related Technology (COBIT) framework [IT 07] describe software acquisition, among other processes. Kneupe [Kne09] investigates the optimization of software acquisition when using the CMMI model - an extension of CMM. Wiese [Wie98] describes the software selection process from an economical point of view. Searching for standard software Wiese uses benefit analysis and describes its efficiency. Nelson et al. [NRS96] describe the software acquisition mainly as a process of making a choice between custom and package software and between insource and outsource acquisition team. Gronau [Gro01] discusses how software is researched, evaluated and finally integrated in large companies. Using two case scenarios, Gronau gives a realistic overview of critical factors for success [Gro12] .

All these approaches are conceptual and describe mainly the process of software acquisition for enterprise, not concentrating on practical issues of selecting particular software for smaller teams (7 - 15 persons). Litke and Pelletier [LP02] describe the Spreadsheet Analysis method for selecting between two alternatives – whether to buy or to build the software. Laakso and Niemi [LN08] use this approach to evaluate different AJAX-enabled Java-frameworks. The method is extended to a scenario-based evaluation similar to our proposed approach.

In comparison to the approaches discussed above, the main difference between methods and the suggested framework is the possibility to apply the framework for application cases without implementing big IT governance system frameworks, which are often rather hard to implement for small teams. The suggested framework can be seen as a modification of the Analytic Hierarchy Process adopted for environments mentioned above.

3 Framework

The proposed framework consists of three main steps:

1. *Identify the software according to minimal and desired requirements*: This results in the list of requirements and software which should be considered in the next steps.
2. *Quantify the requirements*: In order to produce a list of granulated requirements ranked according to their their importance. This list describes the most suitable software product for the project.
3. *Evaluate the software according to the requirements*, i. e., evaluate if the considered software tools meet the requirements.

The result of execution of these three steps is the ranking which shows which software fits to the current project and environment best.

3.1 Step 1: Minimal and Desired Requirements

First, the minimal and desired requirements for the software should be defined by project stakeholders and technical managers so both functional and technical requirements are taken into consideration. Also the users of the software should be interviewed. The person(s) who lead(s) the evaluation may also analyze and add requirements. Stakeholders include project managers, executives and customers. In this context, the stakeholders make the final decision about the choice of a tool. Minimal requirements may be categorized as operational (which support existing workflows and processes) and technical (which enable the use of software in the current environment). The desired requirements are the features and properties which can optimize the processes but are not usually critical for the team.

Second, the software type should be defined. This task is not as trivial as it seems. The main challenge here is to detect which parameters of the software may be important. Assume, for example, that the project management tool should selected. A specification like "IT project management tool" does not help much as most of the tools support IT project management. Moreover, it is unclear what is specific about IT projects. But if the project uses Scrum [TN86] it gives a clear image of which software type is needed.

Third, the software for further evaluation according to the minimal requirements needs to be selected. The desired requirements may also be used for selecting the software if the minimal requirements are not enough. The person who makes evaluation needs to keep in mind that the selected software will be tested with the selected scenarios.

3.2 Step 2: Quantification of requirements

The aim of the second step is to rank the requirements by their importance. Also some additional requirements may be found in this step, which should not be critical. In the case that the additional requirements are critical, the first step should be executed once again. First, all the future users should rank how important different qualities and requirements of the software are. Each team member gets a list of all collected requirements (in the first step). He or she should then weight the requirements by their importance. During our case studies we figured out that it is easier for users to estimate how important each single feature is, not considering them in the context of the whole system. For example, it is much easier for the user to estimate how important the security of the software is on a scale of 0 to 10 than estimating "which part does the security play in this software?" where the possible answer could be 18%.

Afterwards, the importance scores of the software are normalized in order to get the weight of each requirement, using the following formula:

$$w(r_i) = \frac{a(r_i)}{\sum_{k=1..n} a(r_k)},$$

where $w(r_i)$ is the normalized weight of the requirement i, and $a(r_i)$ is the average weight of the requirement i.

Sometimes a second iteration of this procedure is necessary since some of the users (as mentioned below) may add new requirements which should be estimated by all other users. The result of this step is the ranked list ofall requirements, which can be grouped according to functional or organizational role of these requirements for better understanding of the priorities of the project.

3.3 Step 3: Evaluation of Software

The third step is the evaluation of the software and the application of the spreadsheet-analysis method for the final ranking. The main challenge of this part of the evaluation is getting objective information about particular software tools. Of course, the level of this analysis depends on the quantity and accessibility of the software. It is possible to retrieve the information from independent (online- and offline) sources and from the software documentation; information from commercials should be avoided, instead specific technical documentation can be requested from the software vendors. However, it is hard to compare software and the quality of feature implementations relying only on documentation.

We suggest performing a scenario-based analysis to estimate the quality of the implementation of the software features. First, the individual scenarios need to be defined. The scenarios should enable an evaluation of the features that users marked as important and should be done in the environment similar to the environment where the software will be used. One scenario should help to evaluate many aspects of the software, e.g. "Initial installation of the software" may check quality of software support, usability, operating system compatibility. On the other hand, important requirements should be tested with more than one scenario. E.g. if the support is a critical issue, the tester should try to contact vendors and ask some questions during the scenarios.

Second, the scenarios should be executed and the degree of suitability of the software according to the requirements defined in the steps 1 and 2. Then, a decision analysis spreadsheet suggests to give each software the score from -1.0 to 1.0 where 1.0 means "Alternative fully satisfies business requirement or decision criterion" and -1.0 "Alternative fully dissatisfies business requirement or decision criterion." [LP02] However, also other scales may be applied if they have two main properties the person who evaluates the software should not have difficulties giving scores to the software and vice versa: it should be understandable what the different scores mean.

Third, according to the user rating and software evaluation the results of spreadsheet analysis should be calculated and evaluated. The estimated rankings depend on the chosen scale, the number of requirements and the users who estimated the importance of different requirements. Usually about $2-3$ alternatives should be considered further for making the final decision. The framework supports decision making but cannot replace the whole process of reaching the decision. The framework is focused on the technical factors and does not consider economical issues such as software license models or integration costs.

4 Case studies

We applied the proposed framework in three different projects. In the following, we introduce each of the projects, the goals of the software evaluation, and discuss the implications of the introduced method.

4.1 Industry Project/Scrum

Capgemini Deutschland GmbH[1] develops the Manufacturing Execution System (MES) for a german car producer company.[2] MES manages the production of the company and is used as an interface between rather slow business processes, e. g., accounting, which are managed in the Enterprise Resource Planning (ERP-) System, and rather quick local production processes, i. e., local devices in the production, cf. [SRSS09]. The team developing the graphical user interface (GUI) of the system decided to move from the current version control system (VCS) SVN[3] to another modern system in order to optimize their internal processes and the collaboration with the IT department of the customer. We were not only concerned with the control version system: Additionally, we expected the whole software development environment to be changed, as VCS plays very an important role.

The first step involved the minimum requirements. They were defined by observing the day-to-day workflows and interviews with developers and project managers. The team used Scrum – a typical agile development method. The two special points about the project were:

1. Each team uses its own tools, utilities and methods for their module development. It was unclear if we have to offer an optimal solution for the GUI team only or for the all MES product teams.
2. The GUI team collaborates with the IT department of the customer. The customer has some requirements for the software development process and has control over the source code of the system. The challenge was to balance the requirements of the team and those of the customer.

We decided to concentrate on the development processes of the team as it was unclear if the unification of the processes of different teams would be possible. Furthermore, the analysis of all requirements and processes of different teams would be very complex and not possible with the existing resources. We also decided to collect the aggregated requirements from the Capgemini employees to minimize contradictions.

In total, we identified 31 systems which are currently developed. Considering these systems and minimal/desired requirements we left only 10 options to select from. In addition, we identified about 20 different requirements and asked all the team members to weight these requirements and to add their own, if necessary. 25 requirements were identified and afterwards estimated by the team members. All the features were divided into five groups:

- Software quality
- Software features
- Integrability to the project environment
- Integrability to the project processes
- Other project relevant properties

The rank of the requirement groups (Table 1) shows that the project-specific features are more important: They obtained an overall score of 65% against 35 % of basic qualities

[1]`http://www.capgemini.de/`
[2]Due to legal restrictions the name of the system and the car producer company cannot be named in this paper.
[3]`http://subversion.apache.org`

Criterion	relative Weight
Basic Qualities	18%
Basic Features	17%
Integrability to the project Environment	22%
Integrability to the development process	34%
Other features relevant for the team	9%

VCS	DAS-Score
Mercurial	77%
Bazaar	74%
Plastic SCM	69%
Git	67%
Synergy	63,5%
Perforce	50%
PureCM	43%
Integrity	40%
AccuRev SCM	39,5%
SVN	38%

Table 1: Weights of different requirements and decision analysis spreadsheet scores for the top 10 version control systems selected for the integration into Capgemini MES Project.

and features. Next, we developed and executed five scenarios, so each of the ten tools was evaluated. The top four tools (Table 1) were considered by the stakeholders of the project. The stakeholders decided to make the transition to Git[4]. The two main factors influenced this decision: some members of the team had previous experience with Git, thus the whole integration seemed to be easier and the customer who could also influence a decision opted for a more "prominent" system like Git. Further details and a full description of this case study can be found in [Kib12].

4.2 Ubiquitous Platform/VENUS

UBICON[5] is the platform and framework for different ubiquitous applications which allows the observation of different social and physical activities [ABD+12]. The platform is developed by Knowledge and Data Engineering group (KDE) at the University of Kassel. The platform is currently hosting the following applications:

- Conferator [ABD+11] – a social conference management system.[6]
- MyGroup – a social system for supporting members of working group.[7]
- WideNoise – the web application for aggregation and illustration of noise-related data collected by the WideNoise smartphone application.[8]
- AirProbe – the web application for case studies measuring air pollution.[9]

The developer team of the UBICON project partially applies the Venus-method for software development [GLRS12] which lets scientists and experts from different areas (e. g.,

[4]http://git-scm.com
[5]http://ubicon.eu/
[6]http://www.conferator.org
[7]http://ubicon.eu/about/mygroup
[8]http://cs.everyaware.eu/event/widenoise
[9]http://cs.everyaware.eu/event/airprobe

Criterion	**relative Weight** UBICON	**relative Weight** BibSonomy
Issue Tracker	16,2%	15,8%
Continuous Integration Interface	8,1%	7,3%
User Administration	5,9%	5,6%
Software Reliability	12,5%	17,8%
Version Control System	27,8%	31,7%
Project Management	6%	3,2%
Developer Support	23,5%	18,6%

Table 2: Weight of different requirement types in the UBICON and BibSonomy cases.

System	**UBICON DAS-Score**	**BibSonomy DAS-Score**
Redmine	80%	88%
Jira	85%	90%
Trac	57%	66&
FusionForge	45%	53%

Table 3: Decision analysis spreadsheet scores for the top 4 project management systems selected for the UBICON and BibSonomy projects.

information systems, law, usability) work together to create the social acceptable ubiquitous applications. The developers currently use Fusion-Forge for the development of UBICON. Suffering from many different bugs, complicated handling and insecurity of the software the KDE unit aims to replace the Fusion-Forge system. During the first step of the evaluation we performed interviews, where we discussed common use of Fusion-Forge to identify the requirements for the system and later asked the researchers to rate them.

We executed the second step, where the scientists and students estimated how important different requirements are. The results show how the current (and future) software is used and point to some interesting findings. Seven groups of requirements were identified (cf. Table 2). The three biggest groups (Issue Tracker, Version Control System and Developers Support) constitute together 67,5 % of all requirements. Software Reliability (receiving 12,5 %) is also very important for the developers. This can be explained by the significant reliance on the versioning system and the importance during the software development process. Also, the researchers demanded support of new version control systems which can be a signal for transition to Git or Mercurial[10].

Redmine[11] and Jira[12] obtain the two top scores (Table 3). Compared to the BibSonomy case study, the different systems obtained slightly lower scores. Possible explanations, include for example, that different applications are hosted on UBICON. Additionally, some of the applications are developed by international projects and the developers have different locations. Furthermore, UBICON uses a new development approach which implies new requirements for the project management.

[10] `http://mercurial.selenic.com`
[11] `http://www.redmine.org`
[12] `http://www.atlassian.com/de/software/jira/overview`

4.3 Research Projects/Waterfall Model

Benz et al. [BHJ$^+$10] present BibSonomy[13] as a social bookmark and publication system. It aims at helping a researcher in finding literature for his daily work. For this purpose, it uses a growing database of bookmarks and literature references. Since it is simple to use, it has a large number of user, building up a tag-related cloud of papers, links and literature for scientific work.

The development model of BibSonomy is a waterfall model which can be modified dependent on current projects and aims of research. The BibSonomy team currently uses the same instance of Fusion Forge as the UBICON team.

After interviews of the researchers we determined, that the needs of the BibSonomy team are very similar to those of the UBICON team, but the second step of analysis could detect two main differences:

- The version control system is more important for the BibSonomy team;
- The developer support is not as much required by BibSonomy team as by the UBICON team;

Despite these differences and the fact that all of examined systems were ranked lower by the UBICON team, the tendences are the same. As the license costs of Jira are to high, Redmine was selected as the new system for project management of both projects.

4.4 Discussion

All three studies show the possibility to apply the framework in rather different environments. We noticed that the first step was rather easy to execute, but the results were not complete (in all three studies new requirements were discovered in the second step). The second step reveals interesting project properties regarding utility of some features (it is especially interesting to find currently lacking but wished features). The third step usually clearly shows the need (or no need) of the new technology or product: if currently used product is ranked high, the transition may be not necessary. On the other hand, the users tend to rank missing features higher, and thus rank current software lower: In the UBICON and BibSonomy case studies, for example, users ranked interface for version control systems which are currently not used (Git and Mercurial) higher than the systems which are currently used (CVS and SVN). The same tendency can be found in the Capgemini case study, where current system was ranked as last. In all three cases the interview partners were highly motivated to change the current software because they missed some important features. They tended to rank these features high, so the software containing these features got better ranking and thus were ranked higher than current tools.

[13]http://www.bibsonomy.org/

5 Conclusions and Future Work

We presented a three step framework for evaluating and selecting software based on the user's utility and workflows on the one hand and (partially) tested software on the other hand. The presented framework (based on the presented case studies) has the following advantages:

- It is quite flexible (the steps maybe expanded or reduced – depending on special issues of the project and of the software we need to choose from).
- As experienced in the case studies, the framework is easy to use.
- The framework provides a quantified result.

However, the framework has also the following limitations:

- The person who performs the evaluation needs to be able to understand complex technical relations, analyze business processes, and evaluate the software.
- We assume that the framework is easy to apply only in IT teams – as the step two needs technical background (e. g., users need to estimate the importance of security of the software or be able to list additional required software features).
- The framework is focused on the technical requirements and so does not consider such economical factors as software cost or different risks. These should be analyzed afterwards individually for each alternative.

In summary, the framework proved to be an easy and useful tool in the three case studies.

For future work, the framework can be extended to consider not only technical and utility factors but also costs of the software and different risks. In this context the "fair" price of software product may be estimated. Another question is if and how the framework can be extended for special types of software and/or environment. The main challenge is keeping the simplicity of the framework and the transparency during the whole evaluation process.

Acknowledgement

We wish to thank Capgemini Deutschland GmbH and Dr. Mirko Streckenbach and Lukas Birn who helped to organize the first case study, and Dr. Michael Ritzschke from the Humboldt University of Berlin who supervised the Capgemini case study.

This work was supported by the VENUS research cluster Research Center for Information System Design (ITeG) at Kassel University and by the EU-project EveryAware.

References

[ABD+11] Martin Atzmueller, Dominik Benz, Stephan Doerfel, Andreas Hotho, Robert Jäschke, Bjoern Elmar Macek, Folke Mitzlaff, Christoph Scholz, and Gerd Stumme. Enhancing Social Interactions at Conferences. *it – Information Technology*, 53(3):101–107, May 2011.

[ABD$^+$12] Martin Atzmueller, Martin Becker, Stephan Doerfel, Mark Kibanov, Andreas Hotho, Björn-Elmar Macek, Folke Mitzlaff, Juergen Mueller, Christoph Scholz, and Gerd Stumme. Ubicon: Observing Social and Physical Activities. In *Proc. 4th IEEE Intl. Conf. on Cyber, Physical and Social Computing (CPSCom 2012)*, 2012.

[BHJ$^+$10] Dominik Benz, Andreas Hotho, Robert Jäschke, Beate Krause, Folke Mitzlaff, Christoph Schmitz, and Gerd Stumme. The Social Bookmark and Publication Management System BibSonomy. *VLDB*, 19(6):849–875, 2010.

[coo02] Software Acquisition Capability Maturity Model (SA-CMM) Version 1.03. Technical report, 2002.

[GLRS12] Kurt Geihs, Jan Marco Leimeister, Alexander Roßnagel, and Ludger Schmidt. On Socio-technical Enablers for Ubiquitous Computing Applications. In *SAINT*, pages 405–408. IEEE, 2012.

[Gro01] Norbert Gronau. *Industrielle Standardsoftware-Auswahl und Einführung*. München, 2001.

[Gro12] Norbert Gronau. *Handbuch der ERP-Auswahl [mit Mustervorlagen auf CD]*. GITO, Berlin, 2012.

[IT 07] IT Governance Institute, editor. *CobiT 4.1: Framework, Control Objectives, Management Guidelines, Maturity Models*. IT Governance Institute, Rolling Meadows, 2007.

[Kib12] Mark Kibanov. Untersuchung von Versionsverwaltungssystemen mit Zielsetzung der Optimierung der kollaborativen Entwicklung. Master's thesis, Humboldt-University of Berlin, 2012.

[Kne09] Ralf Kneuper. Verbesserung der Beschaffung von Produkten und Leistungen auf Basis des CMMI für Akquisition (CMMI-ACQ), 2009.

[LN08] Tuukka Laakso and Joni Niemi. An evaluation of AJAX-enabled java-based web application frameworks. In Gabriele Kotsis, David Taniar, Eric Pardede, and Ismail Khalil Ibrahim, editors, *MoMM*, pages 431–437. ACM, 2008.

[LP02] Christian Litke and Michael Pelletier. Build it or buy it? - For CEOs Tech Cell, 2002.

[NRS96] Paul Nelson, William B. Richmond, and Abraham Seidmann. Two Dimensions of Software Acquisition. *Commun. ACM*, 39(7):29–35, 1996.

[Saa80] Thomas L. Saaty. *The analytic hierarchy process : planning, priority setting, resource allocation*. McGraw-Hill International Book Co., New York; London, 1980.

[Sen00] Amartya Sen. The Discipline of Cost-Benefit Analysis. *The University of Chicago Press*, 29, 2000.

[SRSS09] Michael Schäfer, Jens Reimann, Christoph Schmidtbauer, and Peter Schoner. *MES: Anforderungen, Architektur und Design mit Java, Spring & Co*. Software + Support Verlag, 2009.

[TN86] Hirotaka Takeuchi and Ikujiro Nonaka. The New New Product Development Game. *Harvard Business Review*, 1986.

[Wie98] Jens Wiese. Ein Entscheidungsmodell für die Auswahl von Standardanwendungssoftware am Beispiel von Warenwirtschaftssystemen. Technical report, 1998.