Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 302 (2014) 95–109

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Semi-automated Tool Recommender for Software Development Processes

Marina Pilar, Jocelyn Simmonds and Hernán Astudillo

Departamento de Informática
Universidad Técnica Federico Santa María
Santiago, Chile
Email: {mpilar,jsimmond,hernan}@inf.utfsm.cl

Abstract

Application life-cycle management (ALM) tools are key for streamlining software development processes. However, small and medium development companies (SMBs) cannot afford to carry out time- and people-intensive tool evaluations for each project, and instead adopt fixed toolsets, thus losing flexibility. To simplify the tool selection process, this article proposes formalizing tool selection as a set of Multiple-Criteria Decision-Making (MCDM) problem, one for each ALM domain. Our domain-parametric recommender takes as inputs a domain, a process definition, and a set of tool evaluation criteria, and yields a ranked list of tools. The approach has been prototyped with the Testing domain and evaluated using a real process and project; the recommendations generated by our approach were quite similar to those of three Testing experts. Pending further evaluation, these results suggest that our approach can generate project-specific tool recommendations with results comparable to those of experts, but at a fraction of the cost.

Keywords: software development process; taxonomy; testing tools; multi-criteria decision-making

1 Introduction

The increased availability of IT tools supporting a wide range of activities in areas like government and health-care, has created a new problem for these organizations: they must now evaluate and compare an ever-growing set of tools – commercial, academic and open source – in order to determine which ones better suit their needs [7]. This issue is even more critical at software development companies, which have to take into account more criteria when evaluating tools, like their current technological ecosystem, tool integration capabilities, ease of use, user training, etc.; but most importantly, the tool must meet the needs of the project and/or organization.

This has made tool selection a complex and expensive task, one to which Small and Medium Businesses (SMBs) can only allocate limited resources to. As a result, these companies choose tools without much prior research [21, 26], and eventually, these tools either fail to meet the needs of the project and/or the development team

http://dx.doi.org/10.1016/j.entcs.2014.01.022

finds them too difficult or cumbersome to use. Others make ad-hoc use of non-specific tools like MS Office, for example, using a spreadsheet for bug-tracking, but this approach does not usually scale well.

One way to improve this process is to create and maintain a tool catalog that can be used by teams when deciding which tools to evaluate for future use. However, since many different criteria must be taken into account when deciding which tools to evaluate, and it is not clear (a-priori) if these criteria interact/interfere with each other, this approach does not completely solve the problem.

Thus, we propose using Multi-Criteria Decision-Making (MCDM) techniques [5] to semi-automate the tool recommendation process. Our framework takes as input a team's development process and their preference levels for various tool selection criteria and uses MCDM to produce a ranked list of tools that support the input process' tasks. These tools are selected from an extensible tool catalog, which is built on top of a set of tool and task taxonomies, one pair for each ALM domain.

The advantage of this approach is that any company that has formalized its software development processes can easily filter through a large amount of tools quite quickly (using a reduced set of criteria). Moreover, if a company evolves or tailors their development process [9], it is easy to check whether the same tools are recommended for the new process. Another advantage of this approach is that the tool and task taxonomies can be built incrementally.

In this article, we describe our framework as applied to the Testing domain. We have validated our prototype by using it to recommend tools for a real, previously documented process and project; the recommendations we obtained using our prototype were quite similar to those of three Testing experts. This article makes the following contributions: (1) We propose a domain-parametric, semi-automated tool recommendation framework that takes into account the project context and development process; (2) We have developed a testing tool catalog and its corresponding taxonomies; (3) We pose tool recommendation as a MCDM problem, which allows us to control the tool recommendation process through the specification of criteria preference levels.

The rest of this article is organized as follows. We give an overview MCDM and our approach in Sections 2 and 3. The Testing domain taxonomies are described in Section 4, and our expert study is presented in Section 5. After comparing our work with related approaches in Section 6, we conclude in Section 7 with a summary of the article and suggestions for future work.

2 Multicriteria Decision-Making

Decision making has become a mathematical science, where the various aspects involved in the decision making process have been formalized [5]. The key aspects in the decision making process are the problem definition, determining minimum requirements, specifying goals, defining selection criteria (tangible or intangible), as well as identifying possible alternatives. This process requires a significant amount of time, and we hope to reduce the amount of input required from the user. For

these reasons, we have decided to use Multicriteria Decision-Making techniques in order to rank tools by selection criteria. Concretely, we use Analytic Hierarchy Process (AHP) [23] and Multiattribute Utility Theory (MAUT) [29].

2.1 Analytic Hierarchy Process

AHP is a decision making technique that uses pairwise comparison, as well as the judgments of experts to derive priority scales between selection criteria. The comparisons are made using a scale of absolute judgments that represents, how much more, one element dominates another with respect to a given attribute. By creating a criteria comparison matrix, this technique generates a set of weights, which represent the grade of preference between criteria. Later, a comparison matrix must be made for each criteria [23]. Since this technique is highly dependent on the individual user's input, and the number and size of the matrices used grow depending on the number of alternative and criteria, we only use this technique to generate the weights between the selection criteria.

2.2 Multiattribute Utility Theory

MAUT tries to assign a utility value to each action. This utility is a real number representing the preferability of the considered action. Very often the utility is the sum of the marginal utilities that each criterion assigns to the considered action [5]. This utility is obtained using a utility function, which transforms value with different units into a unique dimensional scale. We use this technique to complement AHP – the weights indicating the selection criteria preference scale is obtained using AHP, we then evaluate the alternatives using a sample utility function like the one shown in Eq. 1, where the main elements are:

- Set of alternatives $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$
- Set of selection criteria $\mathcal{C} = \{c_1, c_2, \ldots, c_m\}$
- Set of weights $\mathcal{W} = \{w_1, w_2, \ldots, w_m\}$, where $\sum_{i=1}^{m} w_i = 1$

$$X_j = \sum_{i=1}^{m} w_i A_{ij} \tag{1}$$

$$A_{ij} = \frac{(x - x_i^-)}{x_i^+ - x_i^-} \tag{2}$$

Each alternative is evaluated using Eq. 1 and is assigned a score, which will be used to generate a alternative ranking. In this equation, X_j denotes the utility of alternative j, w_i denotes the weight of the criteria i, and A_{ij} denotes the utility value of criteria i with respect to alternative j. A_{ij} is calculated according the the formula shown in Eq. 2, where x_i^- is worst value associated to alternative i when evaluating criteria j, and x_i^+ is the best value.

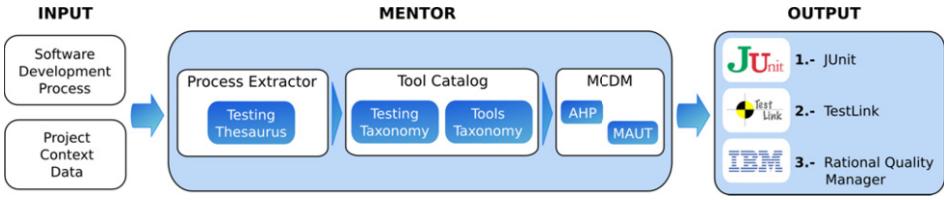

Fig. 1. MENTOR architecture, applied to the Testing domain

2.3 Example

We now use an example to show how AHP and MAUT can be combined to rank alternatives. In this example, we are deciding whether to buy a Toyota Prius, an Audi A3 or a Hyundai Accent (alternatives), taking into account cost and comfort (selection criteria). Comfort is an intangible criteria, so we use a scale from 1 to 5 to evaluate it. First, using AHP, we obtain the weights associated to cost and comfort. Table 1 shows the values of cost and comfort for this example, as well as the weights calculated using AHP.

Table 1
Alternative values and weights according to AHP

	Prius	**A3**	**Accent**	**Weight**
Cost	1000	2000	600	**0.6**
Comfort	3	5	1	**0.4**

As seen in Table 1, the worst value for cost is 2000 and the best one is 600, while the worst value for comfort is 1 and the best one 5. We can now apply MAUT, using Eq. 2 to get that the value of cost for the Prius, which is $(1000-2000)/(600-2000) = 0.71$. The rest of the values are show in Table 2. The last row shows the scores obtained using Eq. 1: according to these values, the Toyota Prius is the alternative that best satisfies the user's needs (taking into account the given weights).

Table 2
Utility Values and Ranking using MAUT

Utility	**Prius**	**A3**	**Accent**	**Weight**
Cost	0.71	0	1	0.6
Comfort	0.5	1	0	0.4
Score	**0.626**	**0.4**	**0.6**	

3 Our Approach

A knowledge base provides a means for information to be collected, organized, shared, searched and used [10]. By creating process and tool knowledge bases, one for each ALM domain, we can reduce some of the complexity of the tool recommendation process, since we can explicitly model the relations between process tasks and tool capabilities, and through these relationships determine how well a tool supports a specific development process. We can then semi-automate the recommendation

process by using MCDM techniques: the development team specifies their preference levels for various selection criteria; these preferences are used to rank the tools that are best suited to the project's needs. As such, our tool recommender is taxonomy-based.

Figure 1 shows the architecture of MENTOR, our **M**ulticrit**E**ria decisio**N**-making **TO**ol **R**ecommender. MENTOR takes as input a software development process specified in SPEM 2.0 [16]. SPEM 2.0 is the OMG standard for modeling processes, we have chosen it as the input format because various Chilean small and medium software development companies have already formalized their processes in SPEM 2.0 [28]. The *Project Context Data*, also input to the recommendation process, is entered directly into our tool. The project context data includes information about the type and size of project being developed, budget, available technology, etc., and is the source of constraints that must be taken into account during the recommendation generation process.

Given a previously defined *Software Development Process*, the *Process Extractor* component first extracts the ALM domain-specific activities, tasks, roles and work products from the process specification, and stores it in a database for further use. Since different organizations give different names to the different process components [2], we have also included a *Thesaurus* component, which we use to normalize terminology. The equivalence relationships between ALM domain activities, tasks and concepts must be maintained by a domain expert.

In order to make tool recommendations, we need to know which tools support the different ALM domain tasks and activities. To this end, we have created a *Tool Catalog* (one per ALM domain), which collects information about available tools. This catalog is based on two taxonomies: a *Domain-specific* taxonomy, which describes the ALM domain; and a *Tools* taxonomy, which describes tool characteristics that are relevant to the tool recommendation process for that ALM domain. This separation of concerns makes our approach extensible, as we can recommend tools for additional domains like requirements analysis, by adding more domain-specific taxonomies (and thesaurus).

At this point, we can use the tool catalog to determine which tools can be used to carry out the activities and tasks of the input *Software Development Process*. This is not a simple task, and if done manually, the team would have to install and evaluate each available tool individually. Instead, we rely on selection criteria to automate this step: the team indicates their preferences with respect to a limited set of selection criteria, and we use the process described in Section 2.3 to rank the tools.

We decided to use a combination of AHP and MAUT in order to reduce the amount of information that had to be entered by the development team. It is relatively easy to assign weights to each tool evaluation criteria using AHP, since the user must only fill in the superior triangle of the criteria comparison matrix, requiring less user input than, for example, the SMART or MACBETH [5] methods, that require values for all pairwise comparisons.

However, AHP also tries to manage inconsistent selection criteria preference

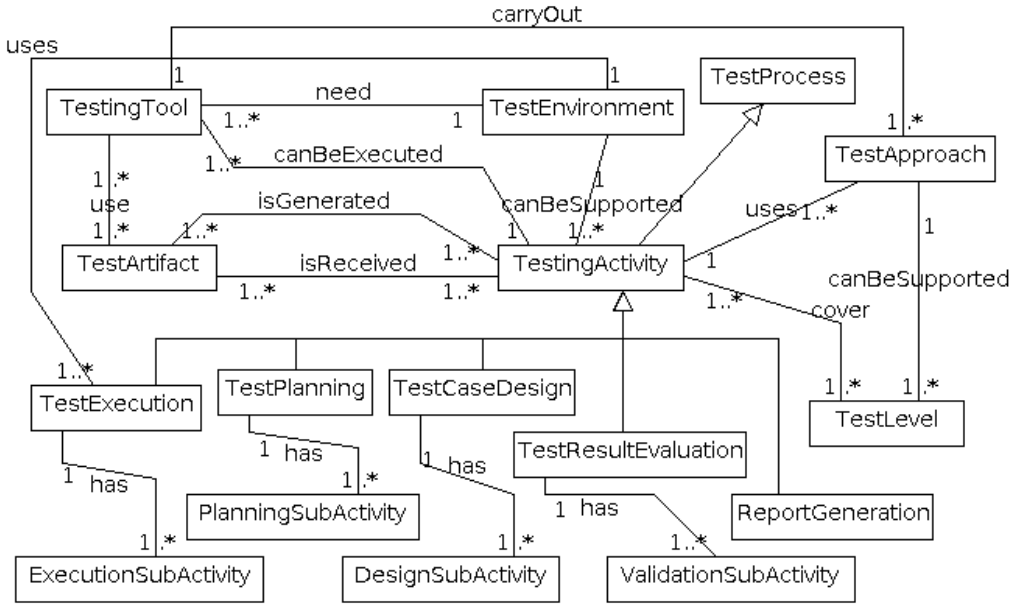

Fig. 2. Testing taxonomy

levels. For example, a user may answer that criterion A is more important than criterion B, and that criterion B is more important than criterion C, but also that criterion C is more important than criterion A. To remedy this, AHP requires additional information after the weight generation process, in order to make sure that the preference levels entered by the user are consistent. For this reason we also use the MAUT technique family, that relies on utility functions. In our prototype we have used one of the sample functions provided by the methodology because our target users do not know of all the tools being considered, whereas in PROMETHEE [5], the preference function must be defined by user, who sets indifference thresholds and strict preferences for each criteria considered.

4 The Testing Domain

Software testing is a complex process. There is a wide range of possible testing activities, and which activities are carried out (and how) depends on the testing approach and techniques used, as well as the type of software being developed. These factors make automated tool selection a difficult task; however, additional tool-specific factors like which testing activities are supported, usability, and compatibility with other tools, makes the tool selection process even more difficult in practice. In this section, we describe the taxonomies used to define the Testing Tool Catalog.

4.1 Testing taxonomy

The Testing taxonomy proposed in this article is based on the ontology presented by Barbosa et al. in [1]. Both Barbosa's ontology and our taxonomy model the

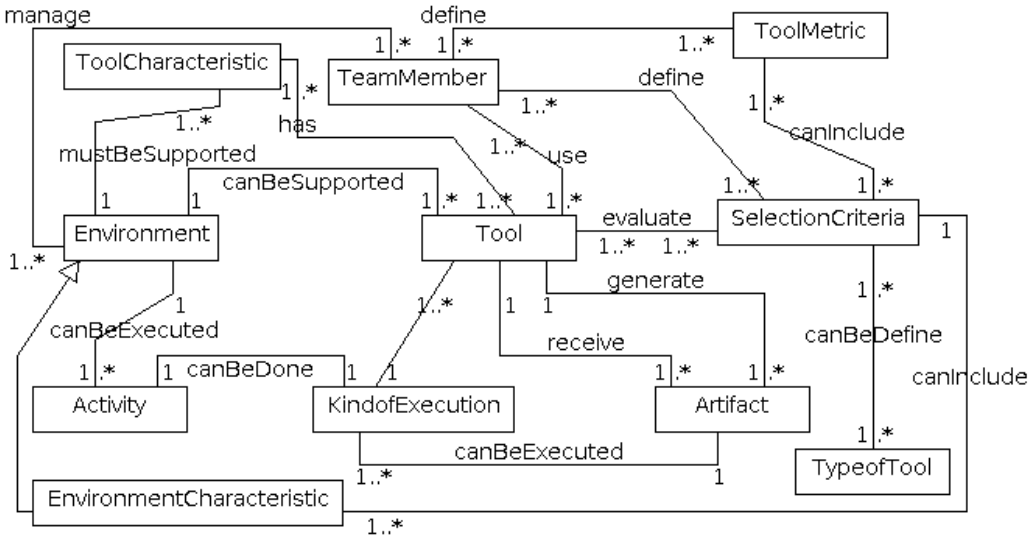

Fig. 3. Tools taxonomy

testing domain; however, with different goals in mind. Barbosa's ontology provides support for testing tool development, where a common vocabulary can increase tool interoperability by providing a common vocabulary. On the other hand, our taxonomy serves as a knowledge base of available tools, and models tool-specific concepts like runtime environment and other technical constraints (inspired by the work in [3,8,13,30]).

Figure 2 shows the key concepts and relations of our Testing taxonomy, presented as a UML class diagram. We now describe these concepts:

- TestingActivity: a TestingProcess consists of various TestingActivities, each one defining a set of actions that need to be performed. TestingActivity is the core concept of this taxonomy, and its five subclasses represent the main phases of the overall testing life cycle [3,8].
  - TestPlanning: groups test management activities, like the development of testing plans. Planning subactivities are modeled as instances of the PlanningSubActivity class, grouping subactivities according to predefined factors like available resources, quality and objectives.
  - TestCaseDesign: groups test case design activities. Subactivities are further organized into DesignSubActivities, which separates subactivities into test suite generation and design techniques. Test design techniques are further divided by testing strategies, e.g., white vs. black box testing.
  - TestExecution: groups activities that deal with test suite execution. The ExecutionSubActivity includes oracle design, artifact inspection, scaffolding design, and test suite execution (manual, record & replay, script-based).
  - TestResultEvaluation: groups activities that validate test suite execution results. The ValidationSubActivity includes subactivities like determining test suite coverage, comparing test suites, and computing metrics.
  - ReportGeneration: groups activities that produce testing reports at any step of

the testing process, as well as final report generation (which includes testing and validation results).

- TestingProcess: the testing part of a previously defined software development process. A process can be broken down into activities, roles, tasks and work products [16].
- TestEnvironment: testing tools run tests under different runtime environment configurations. A TestEnvironment includes information about the installed software and hardware, environment variables, installed code base and required support tasks.
- TestingTool: models tools that help manage and support the testing process and the artifacts its activities produces. Tool descriptions include basic information like: technical requirements and constraints, tool specifications and which testing activities are supported.
- TestArtifact: TestingActivities produce and consume test artifacts like source code, analysis and design diagram, plans, etc. Artifacts can be classified by type and format, and whether they are created or used by a tool.
- TestApproach: various methods can be used to carry out an activity. TestMethods can be divided into two subclasses: technique (error-based, fault-based, combinatorial, functional, etc.) and approach (specification-based and program-based).
- TestLevel: testing activities can be performed at different levels: unit testing, integration testing, system testing, and acceptance testing.

4.2 Tools taxonomy

The Tools taxonomy was developed using an iterative process: the tool selection criteria and templates defined in [6, 12, 18, 25] served as a starting point, and we added concepts modeling tool characteristics, human resources, and existing tool metrics.

Figure 3 shows the key concepts and relations of our Tools taxonomy, presented as a UML class diagram. We now describe these concepts:

- Tool: is the core concept of this taxonomy, and its attributes model basic tool characteristics like description, version, author, etc. A Tool produces and uses Artifacts, and supports different Activities, which can be executed in different ways (see KindofExecution).
- Artifact: these can be classified by type and format, and whether they are created or used by a tool. Typical examples from the testing domain include source code, analysis and design diagram, test plans, etc.
- ToolMetric: groups user-defined tool selection metrics, based on what metrics tools offer, e.g., does the tool support procedural or object-oriented code in the testing context.
- TeamMember: models individual team member's tool-use capabilities, as well as their ability to choose relevant selection criteria and tool metrics. Team members

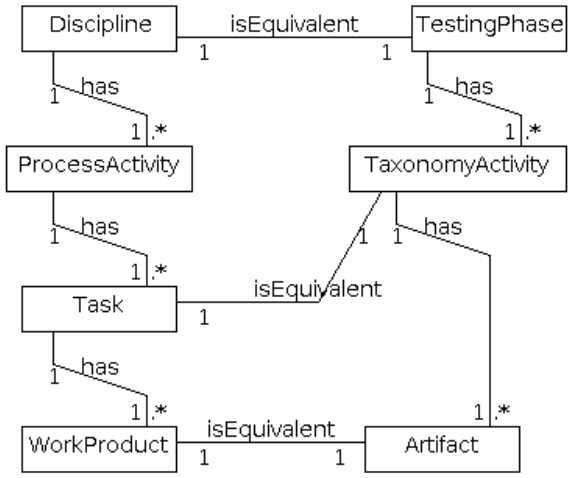

Fig. 4. Thesaurus

are assigned roles within any development process, and depending on their capabilities, tool users can be classified as technical, semi-technical or non-technical.

- KindofExecution: Activities can be performed in more than one manner, e.g., manually or automatically.
- TypeofTool: testing tools can be classified according to tool type, e.g., desktop, open-source, embedded, system, language-specific, library, API, etc.
- SelectionCriteria: People can define tool selection criteria, like cost, risk, technology, complexity, learning curve, etc. More specialized criteria can be defined, for example, by taking into account the type of tool (see TypeofTool).
- ToolCharacteristic: groups specific tool characteristics that must be supported by the testing environment, e.g., script creation, information sharing, script execution, data testing, life cycle integration, etc.
- Environment: Tools are executed under different runtime environment configurations. An Environment includes information about the installed software and hardware, environment variables, installed code base and required support tasks.

4.3 Thesaurus

Figure 4 shows the main equivalences in terminology between SPEM process assets and our Testing taxonomy. This model was created by a domain expert, who indicated which elements represent the same concepts in the two domains.

5 Evaluation

We contacted three Chilean testing experts in order to validate our prototype. These experts all work in software development at SMBs, specifically in Software Quality Assurance. In this section, we first describe the case study that was presented to our three experts and we then discuss the results of this study.

Table 3
Project 1: Tool Recommendations

Rank	Expert 1	Expert 2	Expert 3	MENTOR
1	Tarantula	Testlink	Testlink	Testlink
2	Testlink	Xstudio	Squish Central	Xstudio
3	Xstudio	Silk C T.M.	Rational. Q.M.	Rational. Q.M.
4	E. Tester	Rational Q. M.		E. Tester
5	Silk C T.M.	Squish Central		Silk C T.M.
6	Rational. Q.M.	Selenium-IDE		

5.1 Case Study: Point of Sale Systems

The process presented in [20] has been used by a Chilean SMB to develop Point of Sale (POS) systems, and includes the description of some projects developed using this process. Since the original presentation format is quite extended, we prepared a summary for our experts, highlighting the key characteristics of the SMB and its development process, including information like the type and size of projects that were developed by this SMB, the approximate number of employees, and a characterization of hardware being used.

The experts were presented with two projects for which they had to recommend testing tools:

Project 1: the SMB has been tasked with creating a new inventory management system using PHP and JavaScript. The SMB is in the process of planning its testing process, and is deciding on how to manage the relation between test cases and requirements. The SMB also needs help with bug-tracking, since right now bugs are not being formally assigned to team members, so no one takes responsibility for fixing these bugs. Also, there is no record of which test cases have been executed, which have failed, etc.

Project 2: the SMB has been tasked with adding some new features (like processing credit card payments) to an existing web product catalog, which has also been developed in PHP and JavaScript. In this case, the new features have already been added and the development team is currently testing the GUI. However, the new features were directly added to a live system, so the development team wants to automate the testing process as much as possible in order to wrap-up the testing process as fast as possible.

This information was distributed to the experts via email. We also sent them the full list of the testing tools available in our testing tool catalog. The experts were then given a week to make their testing tool recommendations for these projects.

5.2 Results

Tables 3 and 4 lists the recommendations made by our experts, as well as those made by MENTOR. In both cases, MENTOR was limited to generating the top-5 tool recommendations, as in practice, we do not expect users to evaluate more tools. In the case of Project 1, two of the three experts recommended the same tool as

Table 4
Project 2: Tool Recommendations

Rank	Expert 1	Expert 2	Expert 3	MENTOR
1	Selenium-IDE	Selenium-IDE	Selenium-IDE	Selenium-IDE
2	Testerman	Rational F.T.	Apache JMeter	Apache JMeter
3	Xstudio	Apache JMeter	Squish GUI T.	Silk C T.M.
4	Squish GUI T.	Testlink	Rational F.T.	Xstudio
5		Junit		Squash-TA
6		Silk C T.M.		
7		Squish GUI T.		

MENTOR in first place (Testlink). Expert 1 selected Tarantula as the best suited tool for this project, and listed Testlink in second place. Note however that this expert failed to read the handout carefully: Tarantula could not be used in this case because of the limited hardware available to the SMB. Furthermore, Xstudio appears in second or third position in most of the rankings.

With respect to Project 2, we see that there is an agreement with respect to the best suited tool (Selenium-IDE). Additionally, Apache JMeter appears second or third in most of the rankings. There is less agreement between the rankings as we analyze further recommendations; we believe that this is because the experts attempted to make their recommendations as complete as possible and included tools that can be used to somewhat carry out the tasks described in the handout, whereas MENTOR only included tools that were described as being able to carry out the tasks.

In practice, we expect that development teams will focus on the evaluation of the first two o three tools (where our results are more precise) before analyzing further tools. We are now conducting a study with non-experts users to further validate our approach.

6 Related Work

Recent years have seen a growing interest in the definition of knowledge domains as a way of sharing information and standardizing domains. These domains can be used to avoid differences in concept definitions, which facilitates tool integration, as well as the creation of new tools, so there has been a renewed interest in ontology definition [1]. An ontology defines the common vocabulary for a specific domain, providing a formal specifications of the domain concepts and the relationships between them [15].

Several software engineering-specific ontologies have been defined in the literature. Falbo et al. [4] presents a software process ontology that supports the acquisition and organization of software processes, as well as process knowledge sharing and reuse. Barbosa et al. [1] propose a testing ontology based on the ISO/IEC 12207 standard, which is used to aid in the definition of support tools, as well as enhance interoperability between these types of tools. As this ontology models tool

support in a simplistic manner, in its current state, it cannot be used to automate the testing tool selection process.

In the software testing domain, Nakagawa et al. [14] have focused on tool standarization by proposing a reference architecture for software testing tools. Zhu et al. [8,31] have implemented a prototype multi-agent system that focuses on testing web-based applications. This system does not try to automate the testing process, nor does it recommend testing tools; however, we have used their testing concepts taxonomy as the basis of our testing taxonomy.

Checklists are a popular mechanism for selecting tools,for example, the IEEE 1175 standard defines a checklist template that can be used to assess support tools. Poston et al. [18] extend this template, taking into account not only team requirements, but also any additional information about the tools being considered (if available), like quality and productivity metrics. Additional criteria can be defined, like available platforms and communication interfaces, etc.

Another way to improve the tool selection process is to use tool classifications. For example, Mustafa et al. [12] presents a taxonomy of testing tools based on the testing methodologies that they support, as well as their classification according to intended use. Other work has focused on defining criteria for specific contexts. For example, Tilley et al. [25] define criteria for selecting software visualization tools based on project-specific requirements, like cost, hardware platform, integration with other tools, etc.

Other testing tool selection methods can be website search [24], that can list some testing tools, but do not always cover the SMB needs.

In others domains, Patel et al. [17] present a metodology that describes the issues and factors that should be taken into consideration for select Knowledge Management tools. While Vafaie et al. [27] show a commercial-off-the-shelf (COTS) y government-off-the-shelf (GOTS) selection methodology, where vendors make a survey about tools characteristics. On the other hand, Maxville et al. [11] present a process for selecting COTS from repositories using evaluation techniques like goal/question/metric (GQM), *Analytic Hierarchy Process* (AHP) and *Weighted Score Method* (WSM).

Rivas et al. [21, 22] propose a selection model aimed at supporting software developing SMBs in the selection of project management tools based in ISO/IEC 14102 and Goal/Question/Metric (GQM) approach. Tran et al. [26] propose other tool selection process for SMBs, which consist in three phases: research, vendors communication and tools installation. Theses proposals need a lot manual work to implementation.

In summary, there is plenty of work on defining criteria for tool selection; however, it is the development team's responsibility to determine how well each tool satisfies these criteria. As such, the tool selection process is currently a manual process.

7 Conclusion and Future Work

In this article, we present MENTOR, a domain-based tool recommendation system, which uses a tool catalog to semi-automate the tool recommendation process. It takes as input a formally specified ALM domain, a process definition, the development teams' tool selection criteria preference levels, and using MCDM techniques produces a ranked list of tools that support the process' activities and tasks.

In order to validate our approach, we created tool and task taxonomies for the Testing domain, and we used these taxonomies to create a testing tool catalog. We asked three experts to recommend testing tools for a real, previously documented process and project. Our prototype was able to generate recommendations similar to those made by the experts, and in much less time that if we had manually researched each of the tools available in the tool catalog.

As a result of using this framework, we expect to see an improvement in software quality, since the testing activities of the process will be better supported. We also expect that the testing team will have more time to do actual testing, since they will no longer manually evaluate available support tools. Testing teams may even become aware of the existence of tool support for activities that they had never considered using tools for.

As future work, we will extend this proposal with additional taxonomies, so that MENTOR can make tool recommendations for additional ALM domains. We are also evaluating the use of other MCDM techniques, like Fuzzy [19], in order to improve the recommendation process, as well as improving the selection criteria evaluation method. Finally, we are populating the catalog with more tools, which will be made available on line.

Acknowledgment

This work has been partly funded by the following research grants: project ADAPTE (FONDEF D09i1171), grants UTFSM-DGIP 24.12.50, 24.11.15, and Basal Project CCTVal (FB0821).

References

[1] E. Barbosa, E. Nakagawa, and J. Maldonado. Towards the Establishment of an Ontology of Software Testing. In *Proceedings of the 18th International Conference on Software Engineering and Knowledge Engineering (SEKE'06)*, pages 522–525, July 2006.

[2] R. De Almeida Falbo and G. Bertollo. Establishing a Common Vocabulary for Software Organizations Understand Software Processes. In *EDOC International Workshop on Vocabularies, Ontologies and Rules for The Enterprise, VORTE*, 2005.

[3] N. Eickelmann and D. Richardson. An Evaluation of Software Test Environment Architectures. In *Proceedings of the 18th International Conference on Software Engineering (ICSE'96)*, pages 353–364, 1996.

[4] R. Falbo, C. Menezes, and A. Rocha. A Systematic Approach for Building Ontologies. In *Proceedings of the 6th Ibero-American Conference on AI: Progress in Artificial Intelligence (IBERAMIA'98)*, pages 349–360, 1998.

[5] J. Figueira, S. Greco, and M. Ehrgott. *Multiple criteria decision analysis: state of the art surveys*, volume 78. Springer Verlag, 2005.

[6] R. Firth, V. Mosley, R. Pethia, L. Roberts Gold, and W. Wood. A Guide to the Classification and assessment of Software Engineering Tools. Technical Report CMU/SEI-87-TR-010, Software Engineering Institute, Carnegie Mellon University, August 1987.

[7] W. Guo, X. Fu, and J. Feng. A Data-Driven Software Testing Tools Integration System. In *Proceedings of the International Conference on Computational Intelligence and Software Engineering (CiSE'10)*, pages 1–4, December 2010.

[8] Q. Huo, H. Zhu, and S. Greenwood. A Multi-Agent Software Environment for Testing Web-based Applications. In *Proceedings of the 27th Annual International Conference on Computer Software and Applications (COMPSAC'03)*, pages 210–215, 2003.

[9] J. A. Hurtado Alegría, M. C. Bastarrica, S. F. Ochoa, and J. Simmonds. MDE software process lines in small companies. *Journal of Systems and Software*, 86(5):1153–1171, 2013.

[10] A. Maedche, B. Motik, N. Silva, and R. Volz. MAFRA - A MApping FRAmework for Distributed Ontologies. In *Proceedings of the 13th International Conference on Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web (EKAW'02)*, pages 235–250, 2002.

[11] V. Maxville, J. Armarego, and C. Lam. Applying a reusable framework for software selection. *Software, IET*, 3(5):369 –380, oct 2009.

[12] K. Mustafa, R. Al-Qutaish, and M. Muhairat. Classification of Software Testing Tools Based on the Software Testing Methods. In *Proceedings of the International Conference on Computer and Electrical Engineering (ICCEE'09)*, pages 229–233, 2009.

[13] G. Myers. *The Art of Software Testing.* John Wiley & Sons, Inc., 2004.

[14] E. Nakagawa, A. Simão, F. Ferrari, and J. Maldonado. Towards a Reference Architecture for Software Testing Tools. In *Proceedings of the 19th International Conference on Software Engineering and Knowledge Engineering (SEKE'07)*, pages 157–162, 2007.

[15] N. Noy and D. McGuinness. Ontology Development 101: A Guide to Creating Your First Ontology. Technical report, Stanford Knowledge Systems Laboratory and Stanford Medical Informatics, 2001.

[16] OMG. Software Process Engineering Metamodel SPEM 2.0 OMG Beta Specification. Technical Report ptc/07-11-01, OMG, 2007.

[17] N. Patel and V. Hlupic. A Methodology for the Selection of Knowledge Management Tools. In *Proceedings of the 24th International Conference on Information Technology Interfaces (ITI'02)*, pages 369–374, June 2002.

[18] R. Poston and M. Sexton. Evaluating and Selecting Testing Tools. *IEEE Software*, 9:33–42, 1992.

[19] Y. Ren, Q. Quan, T. Xing, and X. Chen. Fuzzy Decision Analysis of the Software Configuration Management Tools Selection. In *Proceedings of the 2010 Third International Symposium on Information Science and Engineering (ISISE '10)*, pages 295–297, 2010.

[20] M. A. Ribó. Metologia de Desarrollo de Software para PyMEs de Retail. Master's thesis, Universidad de Chile, 2009.

[21] L. Rivas, M. Perez, L. Mendoza, and A. Griman. Towards a Selection Model for Software Engineering Tools in Small and Medium Enterprises (SMEs). In *Proceedings of the Third International Conference on Software Engineering Advances (ICSEA'08)*, pages 264 –269, oct 2008.

[22] L. Rivas, M. Perez, L. Mendoza, and A. Griman. Selection model for software project management tools in smes. In *Proceedings of the 2nd International Conference on Software Technology and Engineering (ICSTE'10)*, volume 1, pages 92–96, oct 2010.

[23] T. Saaty. Decision making with the analytic hierarchy process. *International Journal of Services Sciences*, 1:83–98, nov 2008.

[24] Technology Evaluation Center. Software test tool evaluation center. `http://test-tools.technologyevaluation.com/`, Accessed July 2013.

[25] S. Tilley and S. Huang. On Selecting Software Visualization Tools for Program Understanding in an Industrial Context. In *Proceedings of the 10th International Workshop on Program Comprehension (IWPC'02)*, pages 285–288, 2002.

[26] X. Tran, T. Huynh, S. Shoval, and T. Ferris. Tool selection process and its management for small and medium enterprises in defence projects. In *Proceedings of the IEEE International Conference on System of Systems Engineering (SoSE'08)*, pages 1–7, jun 2008.

[27] H. Vafaie, N. Brown, and L. Truong. Methodology for the Selection of Intelligence Analysis Tools. In *Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI '06)*, pages 55–62, nov 2006.

[28] G. Valdés, H. Astudillo, M. Visconti, and C. López. The Tutelkán SPI Framework for Small Settings: A Methodology Transfer Vehicle. In *Proceedings of the 17th European System, Software & Service Process Improvement & Innovation Conference (EuroSPI'10)*, pages 142–152, Sept 2010.

[29] M. Wang, S.-J. Lin, and Y.-C. Lo. The comparison between MAUT and PROMETHEE. In *Proceeding of the International Conference on Industrial Engineering and Engineering Management (IEEM'10)*, pages 753–757, dec 2010.

[30] M. Young and M. Pezze. *Software Testing and Analysis: Process, Principles and Techniques.* John Wiley & Sons, 2005.

[31] H. Zhu and Q. Huo. Developing A Software Testing Ontology in UML for A Software Growth Environment of Web-Based Applications. In H. Yang, editor, *Software Evolution with UML and XML*, pages 263–295. Idea Group, 2005.