Constraint-Based Recommendation for Software Project Effort Estimation

Bernhard Peischl[a], Markus Zanker[b,c], Mihai Nica[a] and Wolfgang Schmid[b]
[a]Technische Universität Graz, Austria
[b]Universität Klagenfurt, Austria
[c]ConfigWorks Informationssysteme & Consulting GmbH, Austria
[a]Email: {bpeischl,mnica}@ist.tugraz.ac.at
[b]Email:{markus.zanker,wolfgang.schmid}@uni-klu.ac.at
[c]Email:mzanker@configworks.com

Abstract— **Identifying the most appropriate effort estimation methods is an important aspect for software project management. Within the scope of an software industry cluster project an expert system recommending estimation methods that best match the software development project's characteristics and context has been developed. The knowledge-based recommender exploits an explicit knowledge base in order to infer matching items based on the software project's context. The contribution of this article lies in presenting a constraint-based reasoning mechanism for computing recommendable items from a large set of choices and in its application to the domain of software project management. It discusses a recommendation model for effort estimation methods and presents specific extensions like explanation and repair mechanisms that proved exceptionally useful in this application domain. The application was conceptualized and developed in an iterative process and results from two rounds of evaluation are reported.**[1]

Index Terms— **constraint-based recommendation, software project management, recommender applications**

I. Introduction

In today's competitive software development environment it is of utmost importance to deliver products on time, with the desired quality attributes and at the specified cost. Depending on the specific project, estimation of size, effort and schedule can be done in an iterative way, that is by primarily estimating requirements of the next development step. However, ever increasing competition among software companies often results in fixed-price projects. It is thus important to estimate the effort required, size and schedule of a specific project even in the early stages of software development. Thus accurate estimation methods like, for example, the Function Point method, T-Shirt Sizing or Calibration with project-specific data have gained increasing importance [2], [3]. Notably, this also holds for the development of custom software and integration projects, where functionality is primarily extended or substituted.

Although there is a large amount of literature on software estimation methods [2], [3], there is often no detailed knowledge about when to apply which method. To the best of our knowledge, no out of the box knowledge base exists that supports the selection and prioritization of adequate techniques depending on a specific project's characteristics. Consequently, there is a need for personalized advice, taking into account the specific project characteristics, the availability and granularity of comparable data and the current state of development.

In particular, the multitude of potential users, the fact that project managers are solely aware of a small number of relevant techniques and the availability of various techniques for diverse project settings suggests that constructing a recommendation application for software estimation techniques would be worthwhile. Furthermore, harnessing a web-based recommendation system may also considerably contribute to the widespread applicability of software estimation techniques in today's mainstream software engineering projects.

However, the provision of a knowledge base for software effort estimation is particularly challenging as there is no formalized description of the underlying knowledge. Instead the acquired knowledge stems from textbooks and experienced engineers. Further, to establish the necessary trust into an on-line recommendation application for software effort estimation - as confirmed by the survey presented herein - a transparent line of reasoning of the proposed method has to be provided. Moreover, engineers try to get the optimal choice out of the methods being proposed. Consequently, the system has to provide a flexible interaction mechanism that allows the user to explore the solution space and trade-offs between different methods.

Therefore, this article contributes a knowledge-based recommendation application that allows software project managers and engineers to select appropriate methods for effort, size and schedule estimation. In Section II we discuss related work, namely recommendation applications for software engineering, and Section III outlines our novel knowledge model for software estimation techniques. Notably, our model takes the cone of uncertainty [3] into account and thus reflects the accuracy of the different estimation methods at various stages of software development. We further discuss two innovative exten-

[1]The article extends and further develops the work presented in [1].

© 2010 ACADEMY PUBLISHER
doi:10.4304/jetwi.2.4.282-290

sions, an explanation facility for providing a transparent line of reasoning and the application of model-based reasoning techniques in order to allow users to flexibly explore the solution space. Section IV presents our pilot application and its user interface. Moreover, we report on a two phase survey among our industry partners and discuss the most notable remarks triggering improvements of our on-line application. Finally, Section VI concludes our article.

II. Related work

Recommendation systems for software engineering are tools that help developers and managers to better cope with the huge amount of information faced and a variety of different tasks in today's software projects. Typically recommendation applications provide guidance for engineers in a number of activities (e.g., testing, debugging, refactoring, etc.), or to alert stakeholders of potential issues (e.g., redundant code, conflicting requirements, failure-inducing changes, etc.).

To the best of our knowledge no work has been done in trying to implement a recommendation system for software estimation techniques.

The authors of [2] and [3] present more than 30 estimation methods and their specific characteristics. However, it requires lots of expertise for an engineer or project manager to consider all the methods and then decide which one is best suited for the current project.

The authors of [4] present a recommendation system for project planning. This approach mainly supports project planning by taking previous projects into account. It can be used solely when project-specific data is available from previous projects. Additionally, this approach requires information like cost, size and well defined user requirements to be known. In contrast, our technique does not necessitate project specific data, but rather makes use of project specific data if such are available.

The authors of [5] present a survey of different recommendation tools for software maintenance and development. Based on automatic or manual queries, each tool presented in the survey is capable of recommending what should be done in the upcoming stage of ongoing software development efforts.

Other works deal with improvements to developer productivity. The authors of [6] present a recommendation system for identifying the most appropriate functions for implementing a specific software feature. Their approach is based on collaborative filtering algorithms [7]. Moreover the literature reports on successful applications of recommendation systems for detecting software conflicts, as presented in [8], or for focusing software testing on specific modules or components [9].

The authors of [10] provide a prototypical implementation as well and present ChangeCommander, an Eclipse plugin that implements an approach to recommend insertions of particular if-statements before calling a method. ChangeCommander presents context change suggestions by highlighting method invocations in the source code and thus provides automated code adaptation support.

III. Knowledge model for recommendation

Knowledge-based recommender systems exploit a knowledge base that explicitly mediates between user requirements and the characteristics and limitations of different effort estimation methods in order to identify those items that best fulfill them. An overview on knowledge-based recommendation systems can for instance be found in [11]–[14].

According to Adomavicius and Tuzhilin [15] a recommender system in general can be defined as a function $rec(i, u)$ that computes a recommendation score for a given item i and user u, expressing how useful or interesting the item will be.

However, it is common practice in constraint or knowledge-based recommenders to relax queries and identify maximally succeeding sub-queries like done by [16]–[18] that do not emulate a function that computes scores for all items in the catalog but returns only a subset of items. Moreover, in cases where a single catalog query results in multiple recommendations, an additional sorting mechanism is necessary because the recommender only assigns binary recommendation scores: 1 if an item satisfies the query conditions and 0 otherwise. As a result, many systems employ cascading hybridization, using a second algorithm to refine the recommendation list by performing additional filtering and ranking [19]. For instance, the CWAdvisor system [20] uses a scheme based on multi-attribute utility theory (MAUT) [21] and Zanker and Jessenitschnig [22] experimented with cascading knowledge-based and collaborative recommendation algorithms.

However, the hybridization of constraint-based recommendation with another approach comes at the cost of additional engineering as well as knowledge acquisition and maintenance effort. Therefore, we utilize the natural ranking scheme that is produced by comparing items with a set of (soft) constraints on an individual basis. Thus, we define a constraint-based recommender as follows.

Definition 1: A constraint-based recommender is a function rec_γ that computes a continuous value (score) that constitutes the assumed usefulness of a catalog item i for a user u as modeled by a standard finite domain constraint satisfaction problem Γ, i.e. $rec_\gamma(i, u, \Gamma) \mapsto [0 \dots 1]$.

A standard constraint satisfaction problem (CSP) is described by a tuple $\Gamma = (X, D, C)$ where X is a set of variables, D a set of finite domains for the variables in X and C a set of constraint restrictions representing a knowledge base that defines which combinations of values can be simultaneously assigned to variables [23]. The knowledge base was derived by interviews with domain experts and by studying the popular literature on this topic [2], [3]. More concretely, the RS for advising about effort estimation methods elicits the project's context like the development style or the phase the project is currently in via a forms-based dialogue (see Table I for the most important domain characteristics). Variables describe the software project's characteristics such as

© 2010 ACADEMY PUBLISHER

TABLE I.
PROJECT CHARACTERISTICS

Context variable	Description
project type	e.g. *new development*, *maintenance* or *customization project*
development style	*iterative* or *sequential* development
project size	*small*, *medium* or *large* depending on development staff
development stage	early or late phases of the development process
stakeholders	audience addressed by the estimate
countable items	defines what can be counted in the current project (e.g. requirements, GUI elements or Function Points)
available historical data	types of historic data that can be exploited for the current effort estimation task
data granularity	degree of detail of historic data (compare to countable items)

Id	Weight	Preference rule
c_1	10	**If** stage $=$ *'early'* **AND** stakeholder $=$ *'external'* **then** method $=$ *'T-Shirt sizing'*.
c_2	10	**If** stakeholder $=$ *'internal'* **then** method $\neq$ *'T-Shirt sizing'*.
c_3	10	**If** countable $=$ *'fp'* **then** granularity $=$ *'fp'*.
c_4	10	**If** countable $=$ *'fp'* **then** granularity $=$ *'fp'* **OR** granularity $=$ *'dutch'* **OR** granularity $=$ *'feat.'* .
c_5	10	**If** granularity $=$ *'fp'* **then** countable $=$ *'fp'*.

TABLE II.
EXAMPLE CONSTRAINTS

Id	Method	Granularity
i_1	T-Shirt Sizing	none
i_2	Function Points	fp
i_3	Dutch Points	dutch

TABLE III.
EXAMPLE DATABASE OF METHODS

development style and estimation methods' properties like type and granularity level of required historical data in this application domain. Variable domains stem from available instances in the catalog of items and restrictions specified during the requirements elicitation dialogue. The recommendable items and *effort estimation methods* constitute the item catalog I containing a finite set of database instances. Items are characterized by a set of properties that correspond to the variables describing the project context like their applicability to different development styles, project sizes or project phases, the required level of detail for historical data or the level of granularity of this data. In addition, they are grouped into method families and annotated with qualitative aspects like the predictive accuracy of the estimations.

For the purpose of illustration we will discuss the underlying reasoning mechanism using small examples. For this reason we restrict the set of variables $X = X_C \cup X_I$, where $X_C = \{stage, stakeholder, countable\}$ ascribes the project characteristics and $X_I = \{method, granularity\}$ the items.

Constraints can be either hard meaning that they always have to be fulfilled or soft, i.e. $C_{hard} \cup C_{soft} = C$. All $c \in C$ are associated with a penalty value $pen(c)$ for situations when they cannot be fulfilled. Note that we also assign weights to hard constraints, although they are never relaxed, ensuring that they also influence computed scores. Constraints are derived from domain expertise and some examples are given in Table II. For instance, the method *T-Shirt Sizing* is recommended for early development stages and external stakeholders (see constraint c_1) but it is not applicable for internal stakeholders (see c_2) or if Function Points (*'fp'*) can be counted the proposed method should also require the same level of granularity (see c_3). However, in case constraint c_3 can not be fulfilled but has to be relaxed, constraint c_4 ensures that methods exploiting data on the level of granularity of Dutch Points (*'dutch'*) or features (*'feat.'*) are ranked higher than those that do not. The rationale behind constraint c_4 is for instance that Function Points constitute finer granular data that allows to reconstruct coarser granular data like Dutch Points or features (see also Figure 1). Finally, constraint c_5 is the inverse of implication c_3 ensuring that the method's requirement for data granularity is supported by the project's characteristics. For illustration purposes let us also assume an example database of methods I as defined in Table III. Each item $i_k \in I$ in Table III has to be represented by a hard constraint that describes a conjunction of its features, i.e. i_1: *method* $=$ 'T-Shirt sizing' $\wedge$ *granularity* = 'none', i_2: *method* $=$ 'Function Points' $\wedge$ *granularity* $=$ 'fp' and i_3: *method* $=$ 'Dutch Points' $\wedge$ *granularity* = 'dutch'.

Furthermore, the knowledge-based RS takes a set of specific requirements (SRS_u) describing the project context of user u as input. Consequently, items are recommended as follows:

Definition 2: An item $i \in I$ is recommended with maximum score, i.e. $rec_\gamma(i, u, \Gamma) = 1$, iff $SRS_u \cup \{i\} \cup C$ is consistent or satisfiable. If $SRS_u \cup \{i\} \cup C$ is not consistent then the constraint set has to be relaxed and a maximum set $C' \subseteq C$ that needs to encompass all hard constraints ($C_{hard} \subseteq C'$) has to be identified such that $SRS_u \cup \{i\} \cup C'$ is consistent and that there does not exist a set $C'' \subseteq C$ and $C' \subseteq C''$ s.t. $SRS_u \cup \{i\} \cup C''$ is consistent. Consequently, in case of constraint relaxation the recommendation score has to be lowered.

In practice, the recommendations are computed with conjunctive queries on a database. Therefore, we assume the following premises for computing recommendation scores for an item i and user u using relational database queries:

© 2010 ACADEMY PUBLISHER

1) A constraint c is applicable ($app(c)$), *iff* its condition part ($cond(c)$) evaluates to true, i.e. $SRS_u \cup \{i\} \cup \{cond(c)\}$ is consistent.
2) A constraint c is satisfied ($sat(c)$), *iff* it is either not applicable or its consequent part ($cons(c)$) evaluates to true, i.e. $SRS_u \cup \{i\} \cup \{c\}$ is consistent.
3) If all constraints are satisfied, item i will receive the highest possible recommendation score (see Definition 2).
4) If any $c \in C_{hard}$ is not satisfiable in $SRS_u \cup \{i\} \cup \{c\}$ the item i cannot be recommended and thus receives the lowest possible recommendation score.
5) The recommendation score constitutes the relative share of weights of those constraints that are applicable and satisfied compared to all applicable ones.

Formally, a constraint-based recommendation function can be computed as follows:

$$rec_\gamma(i,u,\Gamma) = \begin{cases} score(i,u,\Gamma) & : \forall c \in C_{hard}\ sat(c) \\ 0 & : \text{else} \end{cases}$$

$$score(i,u,\Gamma) = \frac{1}{\sum_{\substack{\forall c \in C \\ app(c)}} pen(c)} \times \sum_{\substack{\forall c \in C \\ app(c) \wedge sat(c)}} pen(c)$$

Additional assumptions can be made to improve the efficiency of score computation. First, the condition part of an implication constraint does not contain variables describing product properties and therefore the applicability of a constraint can be decided on the basis of the user's requirements (i.e. the user model) alone. Second, in most situations the consequent part of a constraint only contains variables describing the catalog of items allowing the satisfaction of constraints to be precomputed at design time. In practical problems, nearly all consequent parts of constraints contain only variables describing the product catalog. Therefore, the satisfaction of a constraint can be computed once using the relational selection operator δ on catalog I. Thus, c is satisfied with respect to a specific item $i \in I$ *iff* $i \in \delta_{[cons(c)]}(I)$. Therefore, our implementation precomputes recommendation scores by executing a query for every $c \in C$ once during the startup of the system and stores a binary constraint/item matrix in memory comparable to [18]. This removes the restriction of having to compute maximally succeeding subqueries in situations where a query fails, allowing us to compute sets of applicable and satisfied constraints in linear time. Recommendation scores are then derived from these sets by traversing over all items and summing up the weights of applicable constraints and as well as those that are both applicable and satisfied.

If the user's specific requirements are $SRS_u = \{r_1, r_2, r_3\}$ with r_1: *stage* = 'early' r_2: *stakeholder* = 'external' r_3: *countable* = 'none' we can compute recommendation scores for all items in I. Based on SRS_u only constraint c_1 is applicable that is only satisfied for i_1 but not for i_2 and i_3. Therefore $rec_\gamma(i_1,u,\Gamma) = 1$, $rec_\gamma(i_2,u,\Gamma) = 0$ and $rec_\gamma(i_3,u,\Gamma) = 0$.

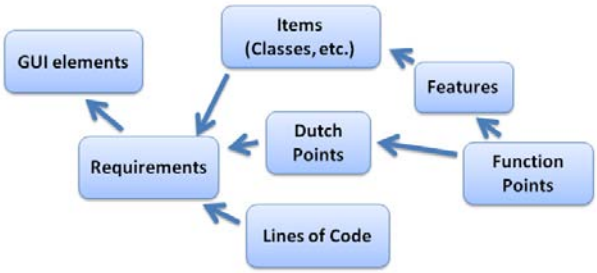

Figure 1. Dependencies among granularity levels of historic data and countable items

IV. Implementation

The knowledge base consists of around 50 different constraints, some of which are rather obvious like *If the project follows development style X then the method should be applicable to development style X*. However, matching the historic data requirements of estimation methods with the granularity of the available historic data and the countable items of the current project data is more complex. Figure 1 sketches the assumed dependencies between different classes of countable elements for deriving software estimates. The directed edges indicate an informal containment hierarchy. For instance if one can compute Function Points (FP) then - in principle - Dutch Points (being only a subset of FP elements) or method-level features may also be computed. Therefore, only estimation methods that are applicable to the figuratively *greatest common divisor* or lowest granularity of the available historic data and the countable data in the current project must be selected.

The implementation is based on a generic recommendation framework [24], [25] that can be instantiated in different application domains. Figure 2 depicts a sample dialogue page that requires the user to specify the current project phase and informs him/her of the variability that can be expected in effort estimation methods at this point. After each user input the system computes and displays the intermediate results and their ranking in order to make the user aware of the impact of the different characteristics of the project context. Figure 3 further depicts the WikiWeb that we used for visualizing the diverse methods. The sidebar lists all available methods - thus the system provides an overview on the available methods. By clicking onto a specific method the WikiWeb provides a description and explanation of this method in a uniform structure. As outlined in Section V, our test users reported that the WikiWeb together with the community portal improved the practical applicability considerably. In addition, the system supports two additional features that we will highlight in the following.

A. Transparent line of reasoning

Figure 4 depicts an exemplary result page of the system that summarizes the project context, recommends a spe-

© 2010 ACADEMY PUBLISHER

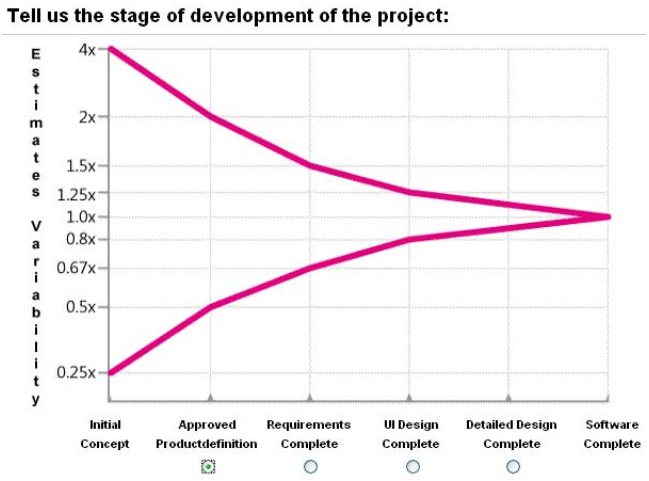

Figure 2. Sample question

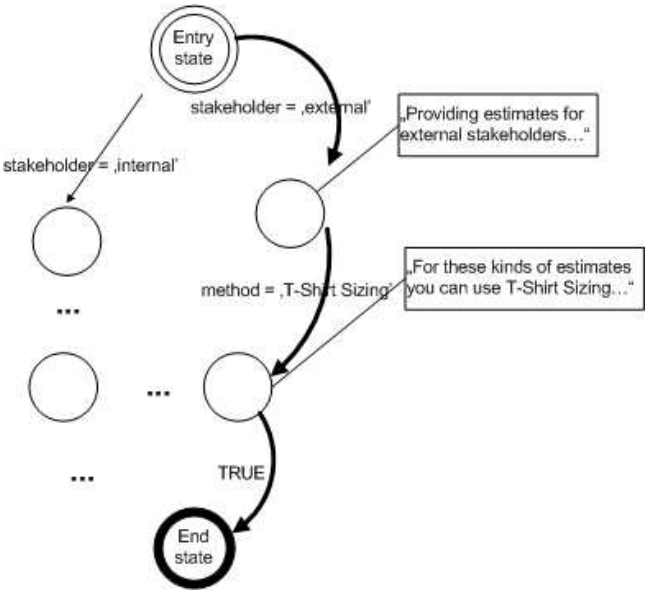

Figure 5. Sketch of PBFSM

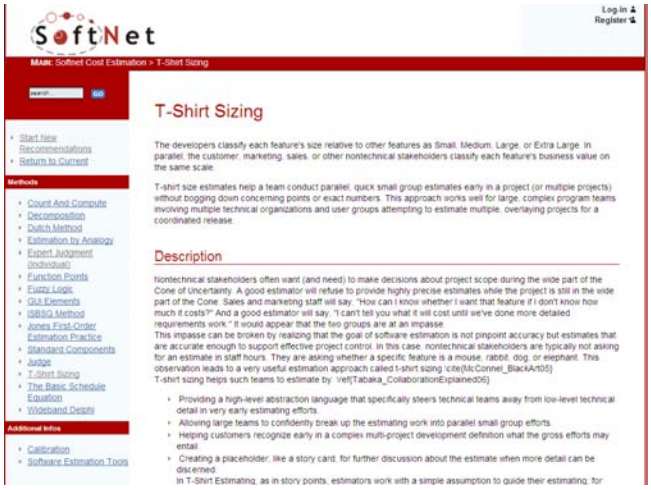

Figure 3. Results page

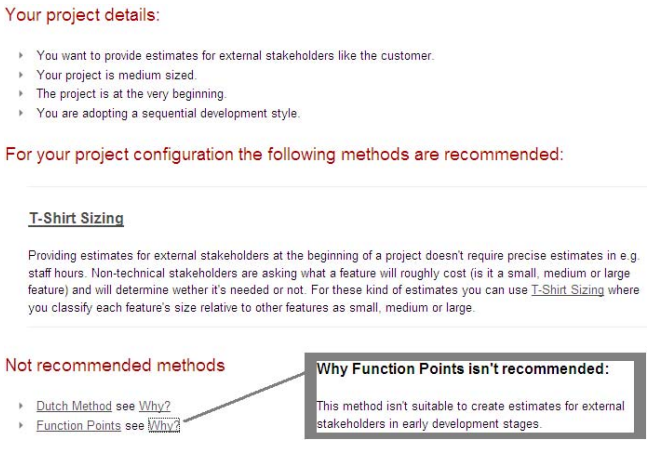

Figure 4. Results page

cific method like *T-Shirt Sizing* including an explanation text and also outlines why another method like Function Points is not applicable in this case. This explanation facility is implemented by a predicate-based finite state machine (PBFSM) comparable to the one presented in [26]. While in [26] states constitute different steps in the interaction process with the user, here each state is attached with a phrase that can become part of the explanation for a recommended item. Transitions between states are associated with logical conditions. Furthermore, there exists a single designated entry state and an end state. Figure 5 depicts an example for the state-graph, where the explanation facility has to compute a valid path connecting the entry and the end state where for a given user and a given recommended item all specified conditions of the transitions have to be satisfied. An explanatory text is therefore sequentially composed from the phrases that are attached to the states along such a valid path. For instance, given that the user searched for a method that is applicable for external stakeholders (*stakeholder* = 'external') and the recommended method is 'T-Shirt Sizing' the valid path in our example state-graph is outlined by bold-faced transitions.

B. Flexible exploration of the solution space

The automated computation of repair alternatives for the formulated project characteristics (SRS) is another feature that promises high utility in this application context. Repair actions for user input to recommender systems in order to avoid empty result sets have been proposed by Felfernig et al. [12]. However, in this article a different application scenario for computing model-based repair actions is sketched that arose from the feedback of users: Flexibly exploring the solution space by requesting a *next better* alternative for a given recommendation. For instance, method A is recommended but the user would

© 2010 ACADEMY PUBLISHER

be interested to use method B because it can deliver more precise estimation results or is already known to the user beforehand. A model-based repair mechanism can consequently propose what input the user would have to modify in order to get a recommendation for method B.

Let's assume the knowledge base consists of the constraints enlisted in Table II, i.e. $KB = \{c_1, c_2, c_3, c_4\}$.

Furthermore, KB has to be consistent with any item from I, i.e. $\forall k \ \{i_k\} \cup KB$ needs to be consistent.

Given a set of initial user requirements $SRS = \{r_1, r_2, r_3\}$ with r_1: *stage* = 'early' r_2: *stakeholder* = 'external' r_3: *countable* = 'none' and a preferred item i_{pref} the repair task is to identify a set of repaired user requirements SRS_{rep} such that $SRS_{rep} \cup \{i_{pref}\} \cup KB$ is consistent [12].

A conflicting subset of user requirements $SRS_{CS} \subseteq SRS$ for a given item preference i_{pref} and KB leads consequently to $SRS_{CS} \cup i_{pref} \cup KB$ being inconsistent. A conflict set SRS_{CS} is minimal iff there does not exist a conflict set $SRS_{CS'}$ such that $SRS_{CS'} \subset SRS_{CS}$.

A diagnosis is a subset of user requirements $\Delta \subseteq SRS$ that has to be removed such that $SRS \backslash \Delta \cup \{i_{pref}\} \cup KB$ is consistent. Similarly, a diagnosis Δ is minimal iff there is no diagnosis Δ' where $\Delta' \subset \Delta$.

In our example domain, SRS lead to item i_1 being recommended, but if the user would like to apply the Function Points method instead, i.e. $i_{pref} = i_2$, the minimal conflict sets $SRS_{CS_1} = \{r_1, r_2\}$ and $SRS_{CS_2} = \{r_3\}$ can be computed for instance by the QUICKXPLAIN algorithm [27]. Applying Reiter's hitting set algorithm [28] will consequently return two minimal diagnoses Δ_k notably $\Delta_1 = \{r_1, r_3\}$ and $\Delta_2 = \{r_2, r_3\}$. The system will therefore propose the user to ensure that $countable = 'fp'$ and that either variable $stage$ has to be unequal to $'early'$ or $stakeholder \neq' external'$. Thus the explicit knowledge representation of a constraint-based recommender allows the application of general problem solving strategies and model-based reasoning techniques.

V. Evaluation

We performed a two step qualitative evaluation phase considering the intermediate and the final version of the system. In order to evaluate the initial version of our recommendation application we presented it's basic rationale and purpose to our industry partners during a workshop. Afterwards we provided them the prototype installation alongside with a qualitative online questionnaire for making their own experience with the system and for collecting their feedback. The panel of evaluators consisted of 7 software project practitioners (project managers, software engineers and IT consultants) from a mix of small and medium-sized as well as big companies. The qualitative evaluation primarily addressed utility and usability issues [29], [30] related to the task of recommending estimation techniques. Table IV outlines the most significant questions that are categorized into four groups: general questions, dialogue related questions, perceived quality of results and usability and usefulness.

The survey's main aim was to evaluate (1) the perceived practical applicability and usefulness of the system for mainstream software engineering projects, (2) the completeness of the catalog of methods and the comprehensiveness of the recommendation approach, (3) the correct characterization of a software project's context, and (4) the plausibility and traceability of the given recommendations. Although, the panel considered the intermediate system basically as useful and usable, they noted several intricacies. The most notable remarks and suggestions for further improvements from our evaluation panelists have been discussed in more detail in [1]. Next, we summarize shortly how we addressed them:

Recommendations are not precise enough: We considered this issue by re-designing parts of our knowledge base. Particularly, we focused on those methods that belong to the same family of methods. For each pair of methods within the same family or group we identified discriminating features and preconditions for their application and updated our knowledge base accordingly.

Lack of concrete examples and Web 2.0 functionality: In order to address this issue, we integrated a WikiWeb with community features such as commenting of methods (see Section IV).

Incomplete and inaccurate characterization of the project: We addressed this shortcoming by extending the scope of questions, e.g., we explicitly considered development from scratch and maintenance of existing software applications.

The system fails to explain why a specific method is chosen: According to our panel evaluators, solely enlisting the applied constraints as done by the first version of the system does not explain why a specific method is the most adequate one. We therefore integrated a newly developed explanation component as described in Subsection IV-A.

Having thoroughly revised and extended the intermediate version of the recommendation system [1] as outlined, we initiated a second round of evaluation with the same evaluation panelists. In the following we summarize the most notable statements of our panelists.

(1) Practical applicability: Table V lists a few comments w.r.t. the applicability of our recommendation application in everyday work and Table VI outlines some respondents' replies to the question on the system's perceived usability and usefulness. For instance, one participant pointed out that - in his opinion - recommendation applications are not adequate for software project effort estimation as one typically applies a couple of methods rather than solely a single method. Admittedly, current recommendation system technology focuses on proposing a ranked list of items and only few works address the issue of bundling recommendations like for instance [31]. However, the most recent revision of the

© 2010 ACADEMY PUBLISHER

Questionnaire	
General questions	What exactly is the subject of estimation (costs, time, resources)? Which effort estimation methods do you know and which of them do you apply? For which kind of company are you working and what is your position?
Dialogue related questions	Are the terms used for the conversation of the recommender system comprehensible? Which of the terms used require further clarification? Is the characterization of the project context complete? If not, which impact factors would you like to be considered in addition?
Perceived quality of obtained results	Do you miss some methods for the specific project context you have characterized? Is the documentation of the recommended methods adequate? In which situations is the number of recommended methods insufficient or too big? Are the proposed recommendations plausible? If not, what would be a plausible explanation?
Perceived usability and usefulness	Would you use the recommendation system for your everyday work? If not, propose what would need to be improved? Which actions would increase the perceived (1) trust and (2) usefulness in the provided recommendation application?

TABLE IV.
THE MOST RELEVANT QUESTIONS FROM THE ONLINE QUESTIONNAIRE FOR THE EVALUATION.[2]

Would you use the recommendation system for your everyday work?
"It is quite interesting to get known to novel methods due to this application. However, most of the time we use a well-known method together with a well established process."
"I will use this recommendation application in any case."
"Probably, recommendation applications are less suited for software effort estimation. Those who do estimations on a regular basis know their methods quite well. Moreover, the number of methods is manageable. However, there is a need to establish conditions when a specific method can be applied."
"In particular, when one is required to estimate large projects with numerous uncertainties, it is a major benefit to apply several effort estimation techniques. In these cases, I plan to use the recommendation application. A further improvement is the traceability: I can see why a specific method has been recommended."
"In my opinion, the recommendation application is asking the 'right' questions and the recommendations are well explained. However, it is not very likely that I will use the recommendation application, because - when applied to our specific project contexts - the system always recommends the expert judgement method."

TABLE V.
EXAMPLE RESPONSES.[2]

Which actions would increase the perceived (1) trust and (2) usefulness of the recommendation application?
"The recommendation application would be more usable if it would support inverse search as well, meaning that for a specific method and a coarsely specified project context the recommendation application should list the exact preconditions for applying this method."
"The overview on the responses being provided could be structured in a better way."
"In order to improve the quality of the recommendations, the recommendation application should also incorporate experience that has been collected over time."

TABLE VI.
EXAMPLE RESPONSES.[2]

effort estimation recommender already proposes a set of at least two suitable methods from two distinct families of methods. Furthermore, although the number of estimation methods is fairly manageable for experienced engineers that perform effort estimation on a regular basis, flexible mechanisms for navigating the potential solution space are important. For instance, understanding what parameters of the project context would have to be changed in order to be able to apply an alternative estimation method is a practical requirement. Notably, the example on applying repair mechanisms (see Subsection IV-B) with constraint-based recommendation addresses this aspect. Another perspective in the context of practical applicability has been pointed out, namely that particularly in large projects the adoption of several estimation methods reduces uncertainties and risks. Therefore this test user confessed to employ the recommendation system in his/her upcoming projects.

(2) Completeness and comprehensiveness: The panel participants reported that the description of the methods for effort estimation in the WikiWeb improved the usability of the recommendation application considerably as some of them have been partly unknown beforehand.

(3) Characterization of project context: Most evaluators considered the characterization of the project context as satisfactory after the revisions. However, one project manager noted the explicit consideration of distributed software development projects, whereas another one raised the (yet unconsidered) issue of re-usability and third party software. Moreover, a test user pointed out to consider the degree of experience of the team.

(4) Traceability of recommendations: According to our test users the provision of the explanation feature strongly supports the traceability of recommendations and thus encourages trust in the provided recommendation and advice.

To summarize the qualitative survey confirmed the utility and usability of the recommendation systems for software project effort estimation. Regarding (2) completeness and comprehensiveness and (4) traceability of recommendations the results showed that evaluators have been completely satisfied. Furthermore, the (3) characterization of the project context is considered to be complete - issues like re-usability and bearing in mind third party software would further enlarge the scope of applicability. Regarding (1) practical applicability, our test users

© 2010 ACADEMY PUBLISHER

explicitly pointed out the usefulness for large software development projects and - in line with our novel repair extension presented herein - suggested a feature that allows for finding out the prerequisites for the next powerful estimation method. [2]

VI. Discussion and Conclusion

This paper contributed a knowledge-based recommendation application to the domain of selecting appropriate effort estimation methods for software project management. Unlike to the mainstream applications in e-commerce domains we applied the idea of online recommendation for supporting software engineers and software project managers in selecting appropriate effort estimation methods. Besides of establishing awareness for the topic of software estimation (which is particularly important for fixed price projects), the applied research presented herein introduces a novel approach for recommendation of software effort estimation methods. Most notably, our online recommendation application reasons on hard- as well as soft constraints representing explicit domain knowledge. Besides the recommendation model itself, this article introduces specific extensions like a transparent line of reasoning by providing explanations for recommended items and flexible interaction opportunities by offering repair mechanisms for initial user input.

As there is no formalized body of knowledge on software effort estimation (we primarily relied on textbooks and experience) we iteratively developed the knowledge base. For ensuring its practical applicability we conducted two primarily qualitative evaluation phases among a panel of industry practitioners. The article finally reports on the essential outcomes from these surveys and thus provides insights to the problem domain of establishing a recommendation system for software engineering professionals.

Acknowledgment

The presented research was conducted within the competence network Softnet Austria (www.soft-net.at) and funded by the Austrian Federal Ministry of Economics (bm:wa), the Province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the City of Vienna via the Center for Innovation and Technology (ZIT). Furthermore, we appreciate the help of Martin Hölbling in implementing parts of the system and the sustainable support of all members of Softnet project no. 2 for participating in the evaluation panel and providing detailed feedback.

References

[1] B. Peischl, M. Nica, M. Zanker, and W. Schmid, "Recommending effort estimation methods for software project management," in *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and International Conference on Intelligent Agent Technology (WI/IAT) - Workshops*, Milan, Italy, 2009, pp. 77–80.

[2]The comments of the panelists have been shortened and translated to English. To our best knowledge, we quote the original intention of the individual comments being provided in German.

[2] M. Bundschuh and C. Dekkers, *The IT Measurement Compendium: Estimating and Benchmarking Success with Functional Size Measurement*. Springer Publishing Company, Incorporated, 2008.

[3] S. McConnell, *Software Estimation: Demystifying the Black Art (Best Practices (Microsoft))*. Redmond, WA, USA: Microsoft Press, 2006.

[4] C.-S. W. Heng-Li Yang, "Recommendation system for it software project planning: a hybrid mining approach for the revised cbr algorithm," in *ICSSSM'08: Proceedings of the 5th International Conference on Service Systems and Service Management*, Melbourne, Australia, 2008, pp. 670 –674.

[5] M. W. Happel Hans-Jörg, "Potentials and challenges of recommendation systems for software development," in *RSSE '08: Proceedings of the 2008 international workshop on Recommendation systems for software engineering*. New York, NY, USA: ACM, 2008, pp. 11–15.

[6] N. Ohsugi, A. Monden, and K. Matsumoto, "A recommendation system for software function discovery," in *Software Engineering Conference, 2002. Ninth Asia-Pacific*, 2002, pp. 248–257.

[7] M. R. W. Hill, L. Stead and G. Furnas, "Recommending and evaluating choices in a virtual community of use," in *In Proc. of the 1995 Conference on Human Factors in Computing Systems (CHI95)*, 1995, pp. 194–201.

[8] D. Prasun, "Dimensions of tools for detecting software conflicts," in *RSSE '08: Proceedings of the 2008 international workshop on Recommendation systems for software engineering*. New York, NY, USA: ACM, 2008, pp. 21–25.

[9] S. Kpodjedo, F. Ricca, P. Galinier, and G. Antoniol, "Not all classes are created equal: toward a recommendation system for focusing testing," in *RSSE '08: Proceedings of the 2008 international workshop on Recommendation systems for software engineering*. New York, NY, USA: ACM, 2008, pp. 6–10.

[10] B. Fluri, J. Zuberbühler, and H. C. Gall, "Recommending method invocation context changes," in *RSSE '08: Proceedings of the 2008 international workshop on Recommendation systems for software engineering*. New York, NY, USA: ACM, 2008, pp. 1–5.

[11] R. Burke, "Knowledge-based recommender systems," *Encyclopedia of Library and Information Systems*, vol. 69(2), pp. 180–200, 2000.

[12] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker, "An integrated environment for the development of knowledge-based recommender applications," *International Journal of Electronic Commerce*, vol. 11, no. 2, pp. 11–34, 2006.

[13] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich, *Recommender Systems - An Introduction*. Cambridge University Press (CUP), 2010.

[14] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker, *Recommender Systems Handbook*. Springer, 2010, ch. Developing Constraint-based Recommenders, pp. 187–215.

[15] G. Adomavicius and A. Tuzhilin, "Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 6, pp. 734–749, 2005.

[16] McSherry, David, "Retrieval Failure and Recovery in Recommender Systems," *Artificial Intelligence Review*, vol. 24, no. 3-4, pp. 319–338, 2005.

[17] N. Mirzadeh, F. Ricci, and M. Bansal, "Supporting user query relaxation in a recommender system," in 5^{th} *International Conference on E-Commerce and Web Technologies (EC-Web)*. Zaragoza, Spain: Springer, 2004, pp. 31–40.

[18] D. Jannach, "Finding preferred query relaxations in content-based recommenders," in *IEEE Intelligent Systems*

© 2010 ACADEMY PUBLISHER

Conference (IS). Westminster, UK: IEEE Press, 2006, pp. 355–360.

[19] R. Burke, *Hybrid Web Recommender Systems*. Heidelberg, Germany: Springer, 2007, pp. 377–408.

[20] M. Zanker, M. Jessenitschnig, D. Jannach, and S. Gordea, "Comparing Recommendation Strategies in a Commercial Context," *IEEE Intelligent Systems*, vol. 22, no. 3, pp. 69–73, 2007.

[21] D. Winterfeldt and W. Edwards, *Decision Analysis and Behavioral Research*. Cambridge, England: Cambridge University Press, 1986.

[22] M. Zanker and M. Jessenitschnig, "Case-studies on exploiting explicit customer requirements in recommender systems," *User Modeling and User-Adapted Interaction: The Journal of Personalization Research, A. Tuzhilin and B. Mobasher (Eds.): Special issue on Data Mining for Personalization*, vol. 19, no. 1-2, pp. 133–166, 2009.

[23] E. Tsang, *Foundations of Constraint Satisfaction*. UK: Academic Press Limited, 1993.

[24] M. Jessenitschnig and M. Zanker, "ISeller: A Flexible Personalization Infrastructure for e-Commerce Applications," in *10th International Conference on Electronic Commerce and Web Technologies (EC-Web)*, Linz, Austria, 2009.

[25] ——, "A generic user modeling component for hybrid recommendation strategies," in 11^{th} *IEEE Conference on Commerce and Enterprise Computing (CEC)*. Vienna, Austria: IEEE Press, 2009, pp. 337–344.

[26] A. Felfernig and K. M. Shchekotykhin, "Debugging user interface descriptions of knowledge-based recommender applications," in *Proceedings of the International Conference on Intelligent User Interfaces*, 2006, pp. 234–241.

[27] U. Junker, "QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems," in *Proceedings of the Nineteenth National Conference on Artificial Intelligence*, 2004, pp. 167–172.

[28] R. Reiter, "A theory of diagnosis from first principles," *Artif. Intell.*, vol. 32, no. 1, pp. 57–95, 1987.

[29] J. Nielsen, *Usability Engineering*. San Diego: Cambridge University Press (CUP), 1994.

[30] J. Rubin, *Handbook of Usability Testing*. New York: John Wiley & Sons, 1994.

[31] M. Zanker, M. Aschinger, and M. Jessenitschnig, "Development of a Collaborative and Constraint-Based Web Configuration System for Personalized Bundling of Products and Services," in 8^{th} *International Conference on Web Information Systems Engineering (WISE)*. Nancy, France: Springer, 2007, pp. 273–284.

Bernhard Peischl is the coordinator of the competence network Softnet Austria. He received a MS in telecommunications engineering (2001) and a Ph.D in computer science (2004) from the Technische Universität Graz, Austria. He is responsible for managing the network's R&D activities and a number of applied research projects dealing with software test, verification and debugging. He has authored and co-authored over 35 scientific articles on peer-reviewed workshops, conferences and in scientific journals.

Markus Zanker Markus Zanker is an assistant professor in the Department for Applied Informatics and the director of the study programme Information Management at the University of Klagenfurt. He is also a cofounder and director of ConfigWorks GmbH, a provider of interactive selling solutions. His research interests focus on knowledge-based systems, particularly in the fields of interactive sales applications such as product configuration and recommendation. He also works on knowledge acquisition and user modeling for personalization.

Mihai Nica is a Ph.D. candidate at the Institute for Software Technology of the Technische Universität Graz. He received a MS from the University of Craiova, Romania.

Wolfgang Schmid is a master student of Computer Science at the University Klagenfurt.

© 2010 ACADEMY PUBLISHER