e-Informatica Software Engineering Journal, Volume 8, Issue 1, 2014, pages: 53–64, DOI 10.5277/e-Inf140104

Supporting Analogy-based Effort Estimation with the Use of Ontologies

Joanna Kowalska*, Mirosław Ochodek*
*Faculty of Computing, Institute of Computing Science, Poznan University of Technology
miroslaw.ochodek@cs.put.poznan.pl

Abstract

The paper concerns effort estimation of software development projects, in particular, at the level of product delivery stages. It proposes a new approach to model project data to support expert-supervised analogy-based effort estimation. The data is modeled using Semantic Web technologies, such as Resource Description Framework (RDF) and Ontology Language for the Web (OWL). Moreover, in the paper, we define a method of supervised case-based reasoning. The method enables to search for similar projects' tasks at different levels of abstraction. For instance, instead of searching for a task performed by a specific person, one could look for tasks performed by people with similar capabilities. The proposed method relies on ontology that defines the core concepts and relationships. However, it is possible to introduce new classes and relationships, without the need of altering the search mechanisms. Finally, we implemented a prototype tool that was used to preliminary validate the proposed approach. We observed that the proposed approach could potentially help experts in estimating non-trivial tasks that are often underestimated.

1. Introduction

Accurate effort estimate is invaluable at every stage of software development. At early stages, it helps to assess feasibility of a project and negotiate the contract, whereas during product delivery stages, it helps to establish achievable deadlines and to reasonably allocate project resources.

Unfortunately, the unique nature of effort estimation at different stages of software development makes it difficult to establish a single, coherent method of collecting data for the purpose of effort prediction. The main reason of that is because the required level of details visibly differs between the levels of tasks. At the level of software development project we usually collect some of its general properties. For instance, in the ISBSG database [1] one can find information such as customer's domain, type of application, level of programming language, etc. This data is usually sufficient to identify and indicate the values of so-called cost drivers used in most of the model-based effort estimation methods (e.g., a well-known COCOMO II [2] defines 22 such factors—17 cost-drivers and 5 scale-drivers) or to use analogy-based methods such as ACE [3], ANGEL [4], Estor [5]. However, such general data becomes less usable if one would like to estimate smaller tasks performed within short development cycles advocated by agile software development methods, like Scrum [6] or eXtreme Programming [7]. This is mainly because the contexts of such small tasks are more diverse, what makes definition of a universal set of cost drivers a cumbersome task. For instance, let us consider how contextually different could be these two tasks: conducting a meeting with a customer and implementing a login function in a web application.

This at least partially explains why estimation of low-level tasks is usually performed with the use of expert-judgment methods (e.g., group methods such as Planning Poker [8–11]) and why there are almost no model-based methods to es-

timate effort of such tasks. However, it is important to mention that the expert-based judgment methods are far from being perfect, because they frequently involve a high degree of wishful thinking and inconsistency. In addition, their results could be biased by business pressure [12, 13]. According to Jørgensen [12] the organizations that have had the most success at meeting cost and schedule commitments use a mix of model-based and expert-judgment methods.

Therefore, the question arises whether it is possible to collect and store project data in such a way that it would enable to combine expert-based and model-based methods of project tasks estimation.

In the paper, we address this question by proposing a new approach to model information regarding projects tasks. Our ultimate goal is to combine expert-based and analogy-based effort estimation methods. The proposed approach is based on Semantic Web technologies, such as Resource Description Framework (RDF) and Ontology Language for the Web (OWL) and has the following features:

- it enables to model and store information regarding project tasks and allows to dynamically extend the ontology by introducing new concepts and relationships (Section 2),
- it supports supervised case-based reasoning—allows to dynamically change the abstraction level of search criteria (Section 3),
- it can be potentially applied to support expert-based effort estimation at the level of product delivery stage (Section 4).

2. Modeling Projects Tasks

Semantic Web technologies in their simplest form offer means to express and store facts in the form of triples (subject, predicate, object) using Resource Description Framework (RDF). Each piece of information is uniquely identified by its Uniform Resource Identifier (URI). This representation of information can be augmented with ontologies expressed in one of the variants of Ontology Language for the Web (OWL). It is also possible to use reasoners and rules engines.

The *ontology* forms an information domain model. It uses a predefined, reserved vocabulary of terms to define *concepts* and the *relationships* between them for a specific area of interest, or domain [14]. Although ontologies are developed and studied for many years, we have recently observed rapid evolution of technologies that support ontology modeling.

An example of a simple knowledge base in a form of semantic network is presented in Figure 1. It states that there are two *individuals*: John and Simon. Each of them is uniquely identified by its URI, e.g., my_data:John[1]. Both John and Simon belong to the class my_onto:Person (Person has a type of owl:Class). Because they are people, they have property my_onto:hasName, which represents person's name. In addition, there is a relationship between both of them stating that John knows Simon.

The great advantage of using Semantic Web technologies to store information is that the data model can be easily extended. It is easy to introduce new individuals, classes, properties and constraints—usually, without the need of modifying the source code of a computer program.

2.1. Projects Tasks Ontology

Assuming that the contexts can differ visibly between project tasks, we would like to propose an ontology that defines the most important concepts and relationships to enable modeling project tasks for the purpose of effort estimation. We also assume that the ontology can be extended by definitions of new classes and relationships that are characteristic for a specific context.

The proposed knowledge model will focus on modeling five types of facts regarding project task:

- **Who?** — it represents information about the one that performed the task. It could be either an individual or a group of people.

[1] We are going to omit the namespace part of URI (e.g., my_data:) unless there is a collision between names.

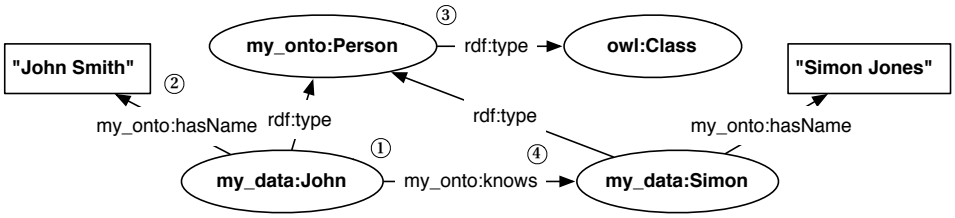

Figure 1. An example of knowledge representation in RDF and OWL (① an individual that belongs to the class Person; ② a data property stating that John has a name John Smith; ③ Person is an OWL class; ④ an object property stating that John knows Simon).

- **Did what?** — it corresponds to both the type of activity and inputs / outputs of the task.
- **How?** — it regards any tools, methods, technologies that were used to complete the task.
- **When?** — it relates to the actual effort and timespan of the task completion.

An exemplary knowledge base storing information about a project task called Task1 is presented in Figure 2. It shows the usage of classes (ellipses with dashed lines) and relationships defined in the proposed ontology. A project task is represented by an individual that belongs to the class *Task*. Each task can have a number of properties corresponding to the aforementioned questions—Who?, Did what?, How? and When?:

- *hasPerformer* — it relates to individuals belonging to the class *Performer* (or its subclasses) that were involved in the completion of the task. Performers can have different capabilities indicated by the property *hasCapability*. A capability has its *level* and the property *in* referring to the subject the capability concerns. In the example, Task1 was performed by John Smith, who is highly skilled Java developer.
- *hasInput* — this property describes all the prerequisites of the task, e.g., requirements, constraints. In the example, Task1 has a single input. It is a use case (UC1) describing user functional requirements to be implemented. We do not restrict the types of inputs to any classes. However, an input can poses a property *hasSize* that is recognized and interpreted by the case-based reasoning algorithm. For instance, the size of the use case UC1 is expressed using the number-of-steps measure. We would also like to emphasize that the presented ontology could be dynamically extended or merged with existing domain ontologies to precisely model the inputs. For instance, UC1 belongs to the class *Creation Use Case* that is not a part of the proposed ontology, however, it still can be used to support effort estimation.
- *hasType* — it represents the type of activity being performed. The taxonomy of types has a hierarchical structure.
- *hasMeans* — the property determines all the means that were used to complete the task. In the example, Java was used to implement UC1.
- *hasOutput* — it represents the artifacts that need to be produced.
- *hasSource* — it provides information about the entity that proposed the task. For instance, it could be a person or company.
- *actualEffortInHours* and *estimatedEffortInHours* — the properties correspond to the actual effort of the tasks, and if available, its estimated effort.
- *from* and *to* — properties defining a timespan when the task was performed.

Tasks can be composed into hierarchies using the *subTaskOf* relationship. This relationship is transitive, which means that if a task has sub-tasks defined, it automatically poses all their features. For instance, in the showed example, Task1 is a sub-task of Task2. This means that Task2 poses all the properties of Task1. For instance, one could conclude that John Smith also participated in completion of Task2. In addition, one of the goals of Task2 was to implement UC1. The composition of tasks enables to compare tasks at different levels.

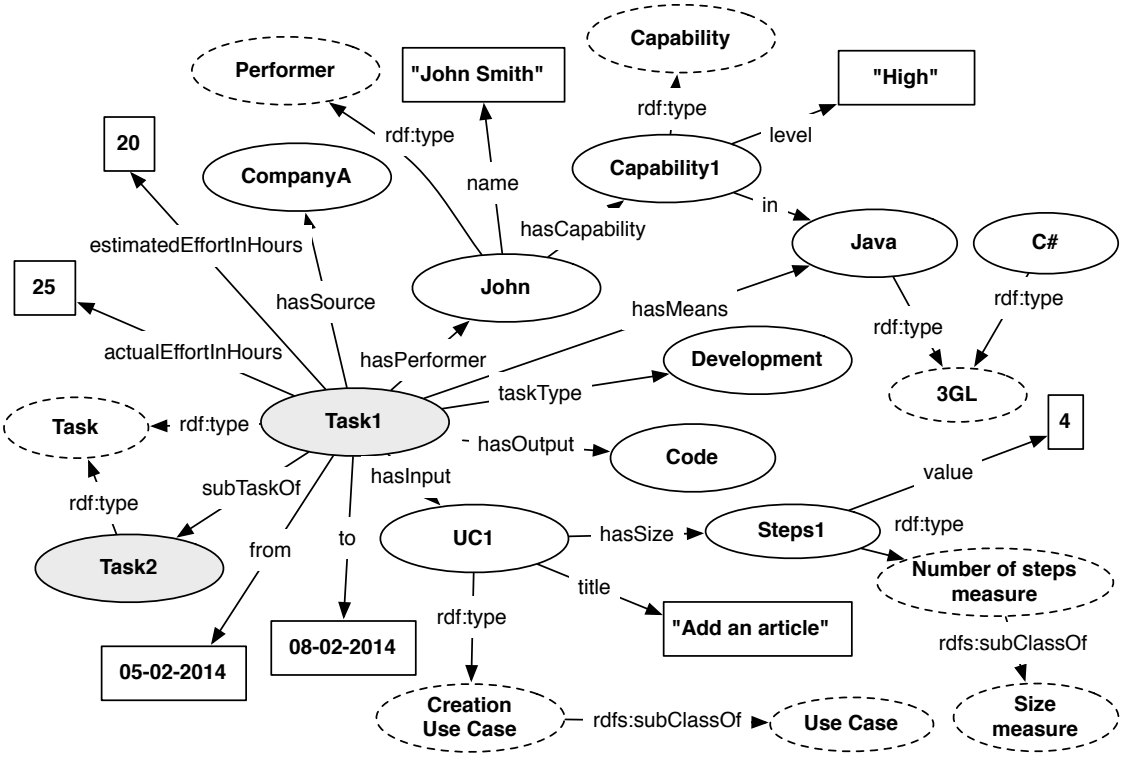

Figure 2. An example of project-task model using the proposed ontology.

3. Supervised Case-based Reasoning

In order to perform case-based reasoning using the proposed ontology we need a method to navigate through semantic network. As the individuals in the ontology form a complicated graph of relationships, we decided to introduce *x-level* notation to indicate the depth of graph exploration. For instance, 1-level of navigation means that the exploration starts at the given node (RDF resource or OWL class) and finishes at the node's direct neighbors. The 2-level navigation implies traversing through all nodes available on 1-level and recursive invocation of 1-level navigation for each of them.

The main goal of the proposed case-based reasoning method is to give the expert possibility to dynamically adjust the demanded level of similarity between tasks. We defined five levels of similarity:

– *Near-exact similarity* — only tasks which have exactly the same values of all properties at 1-level would be classified as similar. For instance, if two tasks are being compared that have almost the same values of all properties, but the sets of performers are different, then, these tasks will not meet conditions to classify them as similar.
– *Similarity after generalization to a given class* — generalization can be defined as navigating up in the hierarchical taxonomy of classes. If two tasks were connected to individuals belonging to the same, given class, then these two tasks would be classified as similar. For instance, let us assume that there are two tasks: the first one was implemented in Java and the second one was implemented in C#. If one considers their similarity after generalizing them to the class *3GL programming language*, then the tasks would be considered similar.
– *Similarity after generalization to classes on a given level* — this approach is more general than the generalization to a given class, because the process of navigating up in class hierarchy is not based on a single class, but it is performed for all classes on a given level. For instance, if a task has individuals that

directly belong to both 3GL programming language and Web Framework classes, a similar task will also have to be connected to individuals that belong to these classes.

– *Similarity when values of a given property are equal* — tasks in the project ontology are not only connected with individuals, but also with plain values. Generalizations work only for class instances, so there is a need to introduce a mechanism of comparing tasks based on so-called *datatype properties* (e.g., integers, strings, etc.). Tasks are considered similar if they have the same values of a given property.
– *Similarity when values of properties on a given level are equal* — it is a more general version of similarity based on equality of properties. This time, all datatype properties at a given level need to be the same to conclude that the tasks are similar.

The proposed approach makes it possible to give the expert opportunity to select which levels of similarity should be selected in a given context. The decision is made by invoking one of the following commands:

– ExactSearch() — it performs a search using *near-exact similarity* comparison,
– Generalize(Relation, Class) — it alters search criteria by introducing the similarity after generalization to the *Class*.
– Generalize(Relation, Level) — it alters search criteria by introducing the similarity after generalization to the classes on the given *Level*.
– SameProperties(Relation, Property) — it alters search criteria by introducing the similarity when values of the given *Property* are equal.
– SameProperties(Relation, Level) — it alters search criteria by introducing the similarity when values of the properties on the given *Level* are equal.

The *Relation* parameter shows in which direction the mechanism should work. It is important to emphasize, that if an expert decides to execute command on the specific relation, then the remaining relations still have to match the previously defined criteria. In addition, all the previously applied commands might be reverted.

The method always starts from the *ExactSearch* command, because it finds the tasks that are the most similar. Afterwards, an expert has possibility to execute different commands and observe the results. The results could be any means supporting expert-based effort estimation, e.g., cumulative density function plots, regression-based models, description of similar tasks.

An example of supervised search session is presented in Figure 3. It presents how the similarity assessment of Task1 and Task2 changes due to execution of commands. Initially, the tasks cannot be classified as similar, because on the 1-level only the *Development* and *Java* is connected to both of them. When the expert executes the SameProperties(hasPerformer, 2) command, *John* and *Anna* become similar, because they both share the same node *HighJava* on the 2-level. Finally, the expert executes Generalize(hasInput, 1) that generalizes UC1 and UC3 to the same class *Creation Use Case*. As a result, Task1 and Task2 are classified as similar.

4. Preliminary Empirical Evaluation

In order to preliminary evaluate the potential usefulness of the proposed approach, we decided to perform a post-mortem analysis of a software development project. In particular, we wanted to investigate whether estimates provided by the proposed method could potentially prevent experts from making the most significant estimation errors (especially prevent them from underestimating effort of tasks).

For the purpose of the method evaluation we implemented a prototype tool on the top of Apache Jena Framework. At current stage of development, the tool cannot be used on-line by an expert, because it lacks easy-to-use user interface. Therefore, instead of conducting the action research study, we decided to perform analysis of existing data. This, however, visibly limits the conclusions we could draw from the study.

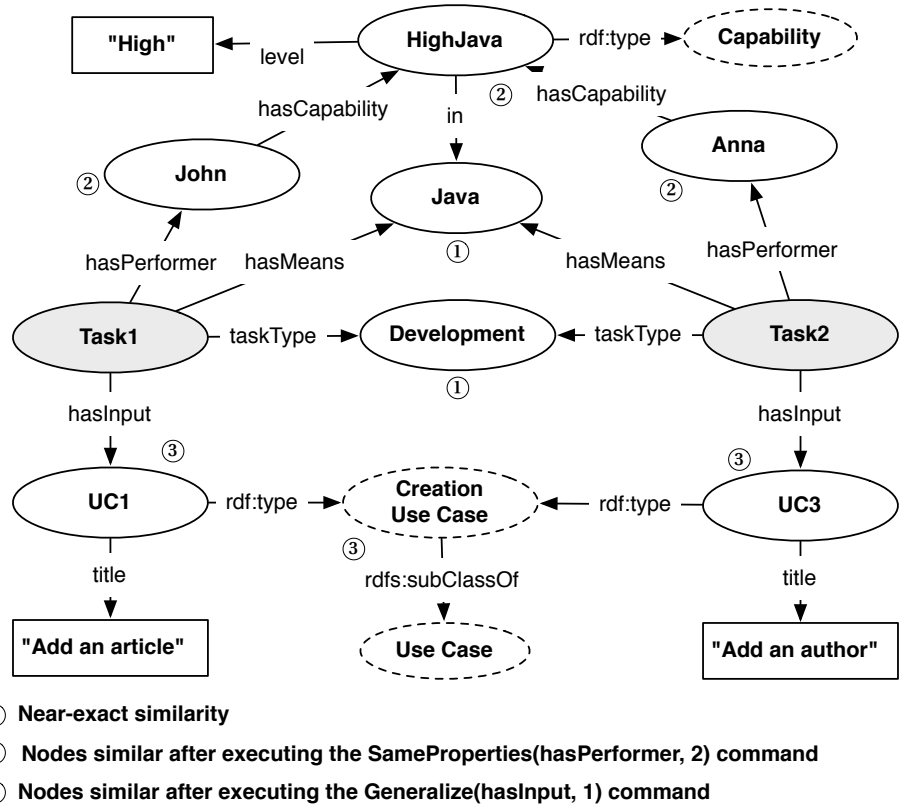

Figure 3. An example of supervised search.

4.1. The Project under Study

The selected project (eProto3) was an in-house software development project conducted at Poznan University of Technology (PUT) in 2011-2012. Its main goal was to enhance the existing system used to collect students' final grades. The development of the new version of the system was one of the steps taken by the University to fully eliminate the need of paper students' record books.

The project was conducted according to the XPrince methodology [15], which combines PRINCE2 [16] at organization level and eXtreme Programming [7] at the product delivery level.

The project team consisted of PUT employees, 3rd and 4th year students. The total reported effort in the project was around 1600 man-hours.

The lifecycle of the project was convergent with PRINCE2 recommendations. During the Initiating a Project stage (IP) non-functional and functional requirements in form of use cases were elicited. The prepared software requirements specification (SRS) served as a product backlog. XPrince assumes that delivery stages are organized similarly to releases in most of agile software development methods. The scope of each delivery stage was agreed during a Planning Game session [7]. Therefore, it was possible that the project would not deliver whole functionality that was defined in SRS.

The analysis was performed based on the tasks recorded in the project's issue tracker (Redmine) during the IP stage, and three delivery stages (the distributions of the tasks' actual effort are presented in Figure 4). Tasks contained information about the estimated effort by the project team members (we would refer to them as expert estimates) and actual effort. The recorded tasks related to large variety of activities, e.g., meetings, requirements engineering, implementation, testing, etc. Many of these tasks had hierarchical structure, especially ones defined during the delivery stages. For instance, each delivery stage had a corresponding task, which was decomposed into set of smaller tasks. For

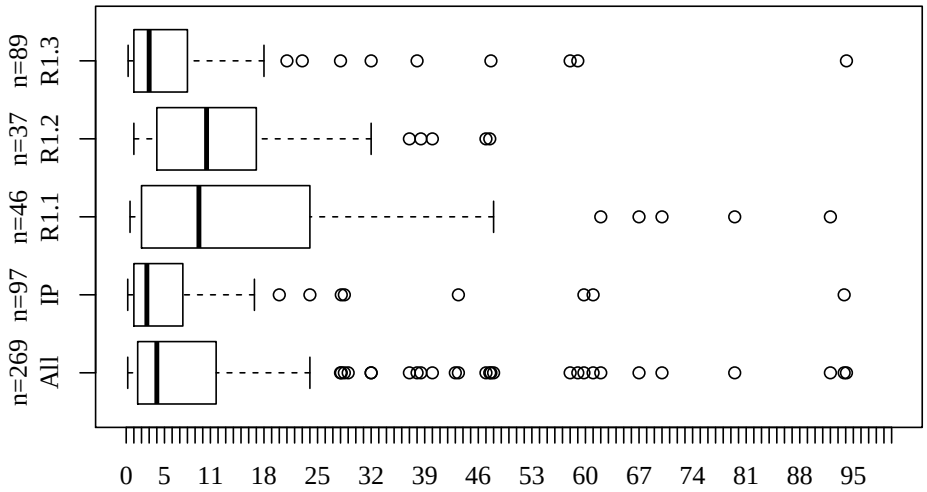

Figure 4. Box-plots presenting actual effort of the tasks in man-hours for all stages (All), Initiating a Project stage (IP), and three releases (R1.1-3).

example, some of the sub-tasks were concerning implementation of use cases. These tasks were further decomposed to tasks which goals were to implement use-case steps.

We were able to automatically retrieve most of the data from the Redmine instance and missing information, e.g., performers capabilities, was added manually (in this particular case, we surveyed team members about their capabilities during the project).

4.2. Evaluation Methodology

We wanted to compare the accuracy of tasks' estimates obtained from three sources:

- (ES) analogy-based effort estimates based on the results of *exact search*,
- (SS) analogy-based effort estimates based on *supervised search*,
- (Exp) project team members' estimates (*experts' estimates*).

In order to compare the accuracy of estimates we used a prediction error metric called balanced measure of relative error (BRE)[2], which is calculated as:

$$BRE = \frac{|actual\ effort - estimated\ effort|}{min(actual\ effort, estimated\ effort)}$$

We also wanted to investigate if the prediction method provided unbiased results. For this purpose, we used slightly modified version of BRE measure, called BRE_bias that is defined in the following way:

$$BRE\_bias = \frac{actual\ effort - estimated\ effort}{min(actual\ effort, estimated\ effort)}$$

If the value of BRE_bias measure is greater than zero it means that the effort was underestimated. Negative value indicates overestimation.

We decided to analyze the accuracy of effort prediction approaches using the k-fold cross-validation method. In the first step we randomly divided the set of tasks into k=10 exclusive subsets with possibly equal cardinality. The validation process took k iterations. During each of the iterations, a single set T became a testing set while the remaining k -1 sets were treated as historical database. Each task from the set T was estimated using a given effort estimation approach. The obtained estimate was compared with the actual effort to calculate prediction error measures.

By definition, the supervised search approach should be used by an expert, who executes the

[2] We decided to use the BRE error measure instead of MRE (Magnitude of Relative Error), which was more frequently used in the past, because the latter one was recently criticized by many researchers, mainly for being unbalanced. [17–20]

commands in order to search for similar tasks. The choice of command that expert executes is determined by the results obtained in the previous step. Therefore, the expert search strategy can differ depending on the task being estimated and content of historical database. Unfortunately, we were not able to simulate such complex behavior. Thus, in the analysis we defined a simple strategy that our virtual expert used to supervise the search. The will of the expert to refine the search was based on the number of similar tasks found in the previous iteration. If the previous steps did not provide a single similar task, expert performed the following steps (after each step verifying if there are any similar tasks found)[3]:

1. ExactSearch — the search started from finding nearly-the-same tasks.
2. Generalize(hasInput, 1) — we decided that the first refinement should concern making inputs more abstract (e.g., instead of UC1 we could have a use case with the main theme of creating an object in the system).
3. SameProperties(hasPerformer, 2) — instead of finding exactly the same performers, we would like to find performers with the same capabilities (e.g., highly skillful Java programmers instead of John Smith)[4].
4. Generalize(hasMeans, 1) — we searched for similar tasks that were performed with the use of similar tools, programming language or technologies.
5. SameProperties(hasType, 1) — finally, we try to look for the tasks similar tasks that have little bit more general type.

The second stage of the case-base reasoning is to predict effort of the task based on found similar tasks. In the study, we used the following strategy to estimate effort. If size was available both estimated and similar historical tasks, we constructed a linear regression model. If the size was not measured for the inputs, we selected mean actual effort of similar tasks as the task estimate.

4.3. Data Analysis

During the analysis of the eProto3 project data it turned out that the experts' estimates were provided for 132 out of 269 tasks (Exp). In addition, the exact search approach was able to estimate 100 tasks (ES) and the supervised search provided estimates for 199 tasks (SS). Therefore, in order to compare the prediction accuracy of approaches we decided to analyze the following sets of tasks: A = Exp ∩ ES ∩ SS (51 tasks), B = Exp ∩ SS (100 tasks) and C = ES ∩ SS (100 tasks).

The first observation was that when all tasks are considered, expert-based estimated are the most accurate (error measures are presented in Table 1). The average values of BRE ranged from 0.33 to 1.06 (depending on the measure of central tendency and set of tasks). They also seemed to be median-unbiased, while for mean-bias we observed a tendency to underestimate. The exact and supervised search approaches on average performed visibly worse than experts—average BRE ranged from 0.76 to 2.53. The estimates seemed to be median-unbiased and contrary to experts' estimates we observe a tendency to overestimate for mean-bias (which from practical point of view is favorable).

The second, not surprising, observation was that the experts performed almost perfect when it comes to small tasks (e.g., 1 man-hour or less). Therefore, it seems that for such tasks no support is necessary. However, taking into account how short iteration-cycles are planned in agile software development, it is rarely observed that tasks are decomposed to such a level. In eProto3 project, during the Planning Game sessions the negotiation between the customer representative and development team was usually at the level of use cases (and rarely at the level of use-case steps). Team members often added the estimates of smaller tasks during the development.

As a result, we decided to filter out tasks having actual effort lesser than 1 man-day (8

[3] As it was presented in Section 4, the execution of commands Generalize and SameProperties does not redefine the search criteria, but refine the existing ones.

[4] During the analysis, it turned out that eProto3 team members had exclusive sets of capabilities, therefore, this step did not have any effect on the results.

		BRE			BRE_bias		
	Tasks set	median	mean	SD	median	mean	SD
All tasks:							
Experts	A=51	0.33	1.06	2.17	0.00	0.43	2.38
Exact Search	A=51	1.00	1.91	3.85	0.00	-0.33	4.29
Supervised Search	A=51	1.00	1.91	3.85	0.00	-0.33	4.29
Experts	B=100	0.45	1.05	1.80	0.00	0.39	2.05
Supervised Search	B=100	1.19	2.53	3.96	-0.06	-1.25	4.53
Exact Search	C=100	0.76	1.78	3.05	-0.01	-0.52	3.49
Supervised Search	C=100	0.76	1.78	3.05	-0.01	-0.52	3.49
Actual effort >= 8h:							
Experts	A'=8	0.85	2.56	4.25	0.79	2.40	4.35
Exact Search	A'=8	0.00	0.38	0.63	0.00	0.03	0.75
Supervised Search	A"=8	0.00	0.38	0.63	0.00	0.03	0.75
Experts	B'=31	0.70	1.49	2.43	0.60	1.29	2.55
Supervised Search	B'=31	0.70	0.95	0.93	0.00	-0.23	1.32
Exact Search	C'=10	0.00	0.33	0.57	0.00	0.00	0.67
Supervised Search	C'=10	0.00	0.33	0.57	0.00	0.00	0.67
All tasks and BRE Experts > 2: (the results for C would be the same as for A)							
Experts	A"=6	4.81	6.07	3.32	4.06	2.91	6.72
Exact Search	A"=6	1.69	2.82	3.88	0.22	-1.24	4.77
Supervised Search	A"=6	1.69	2.82	3.88	0.22	-1.24	4.77
Experts	B"=14	4.00	4.75	2.47	3.65	2.39	4.92
Supervised Search	B"=14	1.35	3.16	3.84	-0.64	-2.27	4.47

Table 1. Effort estimation errors (BRE and BRE_bias).

man-hours) and repeat the analysis. This time the on average values of BRE for experts' estimates ranged from 0.70 to 2.56. We also observed a visible tendency to underestimate effort of bigger tasks by the experts. The exact and supervised search approaches on average performed a little bit better than experts—the average BRE ranged from 0.00 to 0.95 (the most important comparison, based on the set B' indicated difference in median BRE between experts and the supervised search approach at the level of 0.00 and for mean BRE at the level of 0.54). Again the proposed approaches seemed almost unbiased (in one case a minor tendency to overestimate was observed).

The goal of the proposed analogy-based effort estimation method is to support, not eliminate, expert in effort estimation. Therefore, we decided to investigate if the proposed approaches could potentially prevent experts from making most harmful errors in their estimations. The idea was to select tasks that had BRE for expert-based effort estimates greater than 2.00 and observe their corresponding estimates suggested by the tool. The first observation was that both experts and proposed approaches were not able to provide accurate estimates. The average BRE for experts ranged from 4.00 to 6.07 with major tendency to underestimate. The proposed approaches performed better—the average BRE ranged from

1.35 to 3.16. With a single exception, the proposed approaches had tendency to overestimate.

4.4. Discussion of the Results

First of all, we want to emphasize that the goal of the study was not to prove that the method provides estimates with higher accuracy than experts. Instead we treated it as a preliminary study that would show us further directions for improvements.

To sum up the results, we have to admit that generally team members were able to provide accurate effort estimates. The estimates provided by tool, especially for small tasks, were less accurate. However, when the size of the task increased, the accuracy of the tool was comparable, or taking into account its tendency to overestimate even practically favorable.

We observed that the main reason of poor performance of the proposed approaches was the lack of quantitative complexity measured for most of the tasks. We observed that accuracy of the tool could be increased either by providing these kind of measures (e.g., even simple measure such as number of pages of documentation to be produced, etc.) or to precisely describe tasks using the ontology. We believe that the problem could be mitigated if a true human expert was supervising the tool. From our investigation many of the tasks were correctly classified as similar, taking into account available information, however, after reading their titles, the difference between them became obvious.

We also observed that the tool was able to provide better estimates for the tasks that were poorly estimated by experts, which in our opinion is a promising finding. However, still the question arises if the feedback provided by the system would have strong-enough impact on experts' decisions to prevent them from making significant mistakes in their estimates.

4.5. Limitations and Threats to Validity of the Study

There are limitations and threats to validity of the study that needs to discussed. The main threat to construct validity relates to the fact that the proposed study assessed only some aspects of the proposed approaches. We believe that the approaches should support expert-based effort estimation, e.g., as an external voice in group-based effort estimation methods like Planning Poker. However, in the study we simulated behavior of an expert, who always performed in the same way (even if it was unreasonable in a given context).

The main threats to internal validity relate mainly to the project data we obtained from the Redmine system. We suspect that for smaller tasks experts could record actual effort in such a way that it fit the estimated effort (e.g., if one completes a task that was estimated for 30 minutes in 20 minutes he/she very often will just copy the estimated effort as actual). From the practical point of view the difference is not so visible, but when it comes to calculating BRE measure its impact becomes visible.

The threats to external validity regard the ability to generalize the findings. The goal of the study was to collect first observations regarding the method. Therefore, this group of threats does not affect the results too much. The most important threats in this category refer to the size of the sample and software development methodology that was used. For instance, the requirements were documented in the form of use cases rather than in the form of user stories.

5. Related Work

There are three categories of related works that we would like to discuss, namely, analogy-based effort estimation, supporting effort estimation during release planning in agile software development and usage of ontologies for effort estimation.

Analogy-based effort estimation has been developed for many years. Probably the most recognized methods of this type are ACE [3], ANGEL [4], Estor [5].

The main challenge of analogy-based effort estimation is the construction of a mechanism that will enable us to find similar cases (projects)

to the target one that is estimated. Most of the methods tackle with this problem by representing software projects in vector spaces (each feature is represented by a single dimension). Then various techniques are used to find similar projects, e.g., based on different similarity distance measures, e.g., Euclidean, Manhattan, Minkowski.

Another, important problem is that the accuracy of analogy-based methods strongly dependent on the precision of historical data. Recently, Azzeh et al. [21] proposed to use Fuzzy numbers to mitigate this problem. The advantage of this solution is that it can be applied when not all requirements are known. The main drawback is that it is only usable on the project level.

Our approached differs visibly from the previous works in the area, because it enhances case-based reasoning process with semantics. Giving the analogy to the approaches using vector spaces, we could say that we are able to dynamically transform the vector space that is used to describe the projects and to find similarities between them.

Still, the main aim of our approach is to support effort estimation during release planning activities (especially, in agile software development). Majority of related works in this area focus on expert-based (and particularly grouped-based) effort estimation methods. For instance, the Planning Poker method has been recently frequently studied [8–11]. However, there are some works concerning usage of model-based effort estimation methods at the release level. For instance, Hearty et al. [22] proposed the method to predict Project Velocity using Bayesian Nets (BNs); Miranda et al. [23] proposed an approach to support sizing of user stories based on paired comparison.

The usage of ontologies to effort estimation was considered by Hamadan et al. [24]. They identified the importance of organizational and cultural factors and project leadership for improving effort estimates by analogy. The authors created a project ontology, which focuses on the environmental factors. Distance between projects was calculated and used to assess their similarity. However, this approach could be used only at the project level and requires a large number of similar projects in the database.

6. Conclusions

We proposed a new approach to model project data to support expert-supervised analogy-based effort estimation. The data is modeled using Semantic Web technologies, such as Resource Description Framework (RDF) and Ontology Language for the Web (OWL).

In addition, we defined a method of supervised case-based reasoning. The method enables to search for similar project tasks at different levels of abstraction. For instance, instead of searching for a task performed by a specific person, one could look for tasks performed by people with similar capabilities.

The proposed method relies on ontology that defines the core concepts and relationships. However, it is possible to introduce new classes and relationships, without the need of altering the search mechanisms.

Finally, we implemented a prototype tool that was used to preliminary validate the proposed approach. We observed that the proposed approach could potentially help experts in estimating non-trivial tasks that are often underestimated.

References

[1] P. R. Hill, *Practical Software Project Estimation: A Toolkit for Estimating Software Development Effort & Duration.* McGraw-Hill, 2011.

[2] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby, "Cost models for future software life cycle processes: COCOMO 2.0," *Annals of Software Engineering*, Vol. 1, No. 1, 1995, pp. 57–94.

[3] F. Walkerden and R. Jeffery, "An empirical study of analogy-based software effort estimation," *Empirical Software Engineering*, Vol. 4, No. 2, 1999, pp. 135–158.

[4] M. Shepperd, C. Schofield, and B. Kitchenham, "Effort estimation using analogy," in *Proceedings of the 18th International Conference on Software Engineering, Berlin, 1996.* IEEE, 1996, pp. 170–178.

[5] T. Mukhopadhyay, S. Vicinanza, and M. Prietula, "Examining the feasibility of a case-based reasoning model for software effort estimation," *MIS Quarterly*, Vol. 16, No. 2, 1992, pp. 155–171.

[6] K. Schwaber and M. Beedle, *Agile software development with Scrum*. Prentice Hall, 2002.

[7] K. Beck and C. Andres, *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2004.

[8] J. Grenning, "Planning poker or how to avoid analysis paralysis while release planning," 2002.

[9] K. Moløkken-Østvold, N. C. Haugen, and H. C. Benestad, "Using planning poker for combining expert estimates in software projects," *Journal of Systems and Software*, Vol. 81, No. 12, 2008, pp. 2106–2117.

[10] V. Mahnic, "A case study on agile estimating and planning using scrum," *Electronics and Electrical Engineering*, Vol. 111, No. 5, 2011, pp. 123–128.

[11] V. Mahnič and T. Hovelja, "On using planning poker for estimating user stories," *Journal of Systems and Software*, Vol. 85, No. 9, 2012, pp. 2086–2095.

[12] M. Jorgensen, B. Boehm, and S. Rifkin, "Software development effort estimation: Formal models or expert judgment?" *Software, IEEE*, Vol. 26, No. 2, March 2009, pp. 14–19.

[13] R. Popli and N. Chauhan, "Cost and effort estimation in agile software development," in *Optimization, Reliabilty, and Information Technology (ICROIT), 2014 International Conference on*. IEEE, 2014, pp. 57–61.

[14] J. Hebeler, M. Fisher, R. Blace, and A. Perez-Lopez, *Semantic web programming*. John Wiley & Sons, 2011.

[15] J. Nawrocki, L. Olek, M. Jasinski, B. Paliświat, B. Walter, B. Pietrzak, and P. Godek, "Balancing agility and discipline with xprince," in *Rapid integration of software engineering techniques*. Springer, 2006, pp. 266–277.

[16] O. of Government Commerce, *Managing Successful Projects with PRINCE2*. TSO, 2009.

[17] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, "A simulation study of the model evaluation criterion mmre," *Software Engineering, IEEE Transactions on*, Vol. 29, No. 11, 2003, pp. 985–995.

[18] M. Jørgensen, "A critique of how we measure and interpret the accuracy of software development effort estimation," in *First International Workshop on Software Productivity Analysis and Cost Estimation. Information Processing Society of Japan, Nagoya*. Citeseer, 2007.

[19] B. A. Kitchenham, L. M. Pickard, S. G. MacDonell, and M. J. Shepperd, "What accuracy statistics really measure [software estimation]," in *Software, IEE Proceedings-*, Vol. 148, No. 3. IET, 2001, pp. 81–85.

[20] M. Shepperd, M. Cartwright, and G. Kadoda, "On building prediction systems for software engineers," *Empirical Software Engineering*, Vol. 5, No. 3, 2000, pp. 175–182.

[21] M. Azzeh, D. Neagu, and P. I. Cowling, "Analogy-based software effort estimation using fuzzy numbers," *Journal of Systems and Software*, Vol. 84, No. 2, 2011, pp. 270–284.

[22] P. Hearty, N. Fenton, D. Marquez, and M. Neil, "Predicting project velocity in xp using a learning dynamic bayesian network model," *Software Engineering, IEEE Transactions on*, Vol. 35, No. 1, 2009, pp. 124–137.

[23] E. Miranda, P. Bourque, and A. Abran, "Sizing user stories using paired comparisons," *Information and Software Technology*, Vol. 51, No. 9, 2009, pp. 1327–1337.

[24] K. Hamdan, H. El Khatib, J. Moses, and P. Smith, "A software cost ontology system for assisting estimation of software project effort for use with case-based reasoning," in *Innovations in Information Technology, 2006*. IEEE, 2006, pp. 1–5.