What is essential? – A pilot survey on views about the requirements metamodel of reqT

Björn Regnell

Dept. of Computer Science, Lund University, Sweden
bjorn.regnell@cs.lth.se

Abstract. [**Context and motivation**] This research preview presents ongoing work on the metamodel of a free software requirements modeling tool called reqT that is developed in an educational context. [**Question/problem**] The work aims to make an initial validation of a survey instrument that elicits views on the metamodel of the reqT tool, which seek to engage computer science students in Requirements Engineering (RE) through an open source requirements engineering DSL embedded in the Scala programming language. The research question underpinning the presented work is: Which RE concepts are essential to include in the metamodel for a requirements engineering tool in an educational context? [**Principal ideas**] A survey instrument is developed with a list of 92 concepts (49 entities, 15 relations and 28 attributes) and a set of questions for each concept that elicit the respondents' views on the usage and interpretation of each concept. [**Contribution**] The survey is initially validated in a pilot study involving 14 Swedish RE scholars as subjects. The survey results indicate that the survey is feasible if the respondents is willing to invest around 30 minutes of their time. The analysis of the responses suggest that many of the concepts in the metamodel are used frequently by the respondents and there is a large degree of agreement among the respondents about the meaning of the concepts. The results are encouraging for future work on empirical validation of the relevance of the reqT metamodel.

Keywords: requirements engineering, requirements metamodel, CASE tool, requirements engineering education, embedded domain-specific language, empirical software engineering

1 Introduction

There are many challenges in teaching Requirements Engineering (RE) [6, 9], including conveying requirements modeling skills [1]. Given a wide-spread attention on agile methods with less emphasis on extra-code artifacts [8], it may be particularly challenging to motivate coding-focused engineering students (and software practitioners) to spend serious effort on requirements modeling. One way to inspire software engineers to learn more about and do more RE may be to offer an interesting software tool. There are nowadays numerous commercial RE tools available, but many are expensive, complex and not sufficiently open [2].

This paper presents on-going work on a tool named reqT that aims to provide a small but scalable, semi-formal and free software package for an educational setting

that (hopefully) can inspire code lovers to learn more about requirements modeling. A long-term goal of reqT is to offer an open platform for RE research prototypes, e.g. for feature modeling and release planning research. The tool development started in 2011 at Lund University, where reqT is used in RE teaching at MSc level in the Computer Science & Engineering program. In 2012 reqT was rebuilt from scratch based on student feedback. The tool can be downloaded from: `http://reqT.org`

The paper is organized as follows. Section 2 states the objectives and motivates the design strategy of reqT. Section 3 presents the metamodel of reqT and some example reqT models. Section 4 discusses limitations and some initial experiences from using reqT in teaching and concludes the paper with a sketch of future research directions.

2 Goals, Design Strategy and Rationale

The main objective behind reqT is to establish a set of essential RE concepts and capture them in an expressive, extensible and executable language appealing to computer science students (and eventually practitioners). This general objective is accompanied by the following main goals and associated design strategies:

1. **Semi-formal.** *Goal:* Provide a semi-formal representation of typical requirements modeling constructs that can illustrate a flexible combination of expressive natural language-style requirements with type-safe formalisms allowing static checks. *Design:* Use graph structures based on typed nodes representing typical requirement entities and attributes, and typed edges representing typical requirements relations, and implement the graph as an associative array (map). *Why?* Graphs are well-known to many CS students. Maps are efficient from an implementation perspective and may be less complex to master compared to e.g. SQL databases.
2. **Open.** *Goal:* Provide a platform-independent requirements tool that is free of charge. *Design:* Use Java Virtual Machine technology and release the code under an open source license. Use tab-separated, tabular text-files for import and export. Use HTML for document generation. *Why?* There are many free libraries available that runs on a JVM. Tab-sep and HTML support interoperability.
3. **Scalable.** *Goal:* Provide an extensible requirements modeling language that can scale from small, concise models to large families of models with thousands of requirements entities and relations. *Design:* Implement reqT as an internal DSL (Domain-Specific Language) in the Scala programming language [7]. Use Map and Set from Scala collections to represent requirements graphs. *Why?* Scala is a modern, statically typed language with an efficient collections library. Scala offers scripting abilities that provide general extensibility without re-compilation. Integrated development environments [11], as well as interactive scripting tools are available [3].

These goals, design strategies and rationale are directing the on-going work, and it remains to be investigated to what extent the main objective and goals can be met. A critical issue is how to interpret what are ”essential” RE concepts and ”typical” modeling constructs. The reqT tool is used in a course based on a specific text book [4] and

a specific student project concept [5], and the concepts of the reqT requirements metamodel (see Fig. **??**) reflect that context. However, the reqT architecture is prepared for extensions of new concepts in the metamodel to cater for different educational contexts.

3 Modeling requirements with reqT

A reqT model includes sequences of graph parts `<Entity><Edge><NodeSet>` separated by comma and wrapped inside a `Model( )` construct. A small reqT Model with three `Feature` entities and one `Stakeholder` entity is shown below:

```
Model(
  Feature("f1") has (Spec("A good spec."), Status(SPECIFIED)),
  Feature("f1") requires (Feature("f2"), Feature("f3")),
  Stakeholder("s1") assigns(Prio(1)) to Feature("f2")
)
```

4 Discussion and Conclusion

The results of the on-going work with reqT remains to be further investigated and a validation of reqT as a RE learning tool and research experimentation platform is subject to future work. This section discusses some preliminary experiences, limitations, relation to state-of-the-art and future research directions.

Preliminary proof-of-concept. The first version of reqT was tried on a voluntary basis by 12 students working in groups of 6 students each during fall 2011. Statements from course evaluations indicate that the students found reqT useful in their learning. One group used a configuration management tool for reqT models to manage their parallel work, while one group used a cloud service and tab-sep export/import to collaborate over the Internet. The group with the largest requirements model produced 64 features, 18 tasks, 12 functions, 30 data requirements and 33 quality requirements, in total 157 requirements entities.

Several students appreciated that reqT can mix informal text with a graph-oriented formalism, but some requested more elaborated functionality for document generation, as well as linking to external images. Some students also requested more modeling examples that show how the text book techniques could be transferred to reqT models.

Based on student feedback, reqT was rebuilt from scratch during 2012 with a new architecture and a new version of the meta model (see Fig. **??**), as well as a revised Scala-internal DSL. The template-controlled HTML generation was implemented based on student suggestions. The teaching material was complemented with more example models directly related to the textbook. The second version of reqT is currently tested by students in a new course instance and a post-course evaluation of reqT is planned in spring 2013.

Our preliminary experiences from applying reqT in teaching suggest that reqT, if used in a suitable teaching context, may encourage students with a code-focused mind set to learn and practice RE in the following ways: (1) A free and platform-independent

software tool that is implemented using a modern programming language with interactive scripting facilities can attracts the interest of code-focused students. (2) Requirements can be processed, queried, transformed or exported using Scala scripts, and the open-ended nature of reqT that allows students to code their own scripts to both manage requirements models and to adapt reqT to fit their RE needs in the course project was appreciated by several coding-literate students. (3) By turning requirements models into executable code, students can use programming tools such as a console command line interpreter (the Scala REPL) as well as a source code version control system (e.g. git-scm.com) to branch and merge their collaborative work on requirements in ways they are used to from their previous collaborative software implementation courses, including issue tracking systems and code review support.

Relation to state-of-the-art. To the best of our knowledge there is no other RE tool that allows semi-formal requirement models to become executable programs through an internal Scala DSL, and thus letting coding, testing and requirements engineering share the same media. In the RE tool survey by Carrillo de Gea et al. [2] it is pointed out that "many expensive tools aren't sufficiently open". The reqT technology aims to be completely free and open to facilitate academic usage, collaborative evolution and incorporation of new RE concepts in different teaching and research contexts. Many of the existing tools have proprietary representations [2], while users of reqT can extend the reqT metamodel with new entities and attributes simply by adding case classes with a few lines of code. However, reqT cannot compete with versatile commercial RE tools [2] in terms of e.g. features completeness and graphical user interface.

Limitations. In its current version, reqT has a number of limitations: (1) As the user interface is text based and depends on the command line interface of the Scala REPL or a script editor environment [3, 11], students that only are prepared to use graphical user interfaces may be discouraged. Some of our students preferred to work in a GUI spreadsheet application using tab-separated exports from reqT that was generated by other team members assigned by the student group to be reqT experts. (2) It requires some knowledge of Scala to tailor reqT exports and there is a need for a more comprehensive API for adaptable document generation. (3) The embedded DSL requires some learning efforts and it remains to be investigated if the effort is justified by the knowledge gained. (4) To support scalability to large families of reqT models there is a need for modularization concepts and overview visualizations. (5) The explicit typing of entities with keywords such Feature and Stakeholder can be perceived as verbose compared to more concise but potentially cryptic abbreviations (e.g. Fe, Sh). This may be addressed by DSL-specific editor support, such as code-completion, code folding and code templates.

Future work. Further directions of research include (1) incorporation of constraints on models for support of prioritization and release planning [10], (2) more elaborate semantic checks to better guide requirements modelers, and (3) graphical visualization of requirements graph models. (4) Natural Language Processing technology including e.g. ambiguity risk detection may be interesting to combine with reqT. (5) It is also important to further investigate the pedagogic advantages and limitations of the approach.

A major objective of this research preview paper is to expose the latest version of reqT to the community of RE scholars and to invite discussions and contributions.

References

1. Callele, D., Makaroff, D.: Teaching requirements engineering to an unsuspecting audience. In: Proceedings of the 37th SIGCSE technical symposium on Computer science education. pp. 433–437. SIGCSE '06 (2006)
2. Carrillo de Gea, J., Nicolas, J., Aleman, J., Toval, A., Ebert, C., Vizcaino, A.: Requirements engineering tools. Software, IEEE 28(4), 86 –91 (july-aug 2011)
3. Kogics: Kojo, `http://www.kogics.net/kojo`, visited Nov 2012.
4. Lauesen, S.: Software Requirements - Styles and Techniques. Addison-Wesley (2002)
5. Lund University: `http://cs.lth.se/ets170`, visited Nov 2012.
6. Memon, R.N., Ahmad, R., Salim, S.S.: Problems in requirements engineering education: a survey. In: Proceedings of the 8th International Conference on Frontiers of Information Technology. pp. 5:1–5:6. FIT '10, ACM (2010)
7. Odersky, M.e.a.: An overview of the Scala programming language. Tech. rep. (2004), `http://lampwww.epfl.ch/~odersky/papers/ScalaOverview.html`
8. Ramesh, B., Lan, C., Baskerville, R.: Agile requirements engineering practices and challenges: an empirical study. Information Systems Journal 20(5), 449 – 480 (2010)
9. Regev, G., Gause, D.C., Wegmann, A.: Experiential learning approach for requirements engineering education. Requirements Engineering 14(4), 269 – 287 (2009)
10. Regnell, B., Kuchcinski, K.: Exploring software product management decision problems with constraint solving - opportunities for prioritization and release planning. In: Software Product Management (IWSPM), 2011 Fifth International Workshop on. pp. 47 –56 (2011)
11. Scala Eclipse IDE: `http://scala-ide.org/`, visited Nov 2012.

Table 1. Definitions of concepts.

Entity	*Definition*
Actor	A human or machine that communicates with a system.
App	A computer program, or group of programs designed for end users, normally with a graphical user interface. Short for application.
Barrier	Something that makes it difficult to achieve a goal or a higher quality level.
Breakpoint	A point of change. An important aspect of a (non-linear) relation between quality and benefit.
Class	An extensible template for creating objects. A set of objects with certain attributes in common. A category.
Component	A composable part of a system. A reusable, interchangeable system unit or functionality.
Configuration	A specific combination of variants.
Data	Information stored in a system.
Design	A specific realization or high-level implementation description (of a system part).
Domain	The application area of a product with its surrounding entities.
Epic	A large user story or a collection of stories.
Event	Something that can happen in the domain and/or in the system.
Feature	A releasable characteristic of a product. A (high-level, coherent) bundle of requirements.
Function	A description of how input data is mapped to output data. A capability of a system to do something specific.
Goal	An intention of a stakeholder or desired system property.
Idea	A concept or thought (potentially interesting).
Interface	A defined way to interact with a system.
Issue	Something needed to be fixed.
Item	An article in a collection, enumeration, or series.
Label	A descriptive name used to identify something.
Member	An entity that is part of another entity, eg. a field in a in a class.
Meta	A prefix used on a concept to mean beyond or about its own concept, e.g. metadata is data about data.
MockUp	A prototype with limited functionality used to demonstrate a design idea.
Module	A collection of coherent functions and interfaces.
Product	Something offered to a market.
Quality	A distinguishing characteristic or degree of goodness.
Relationship	A specific way that entities are connected.
Release	A specific version of a system offered at a specific time to end users.
Req	Something needed or wanted. An abstract term denoting any type of information relevant to the (specification of) intentions behind system development. Short for requirement.
Resource	A capability of, or support for development.
Risk	Something negative that may happen.
Scenario	A (vivid) description of a (possible future) system usage.
Screen	A design of (a part of) a user interface.
Section	A part of a (requirements) document.
Service	Actions performed by systems and/or humans to provide results to stakeholders.
Stakeholder	Someone with a stake in the system development or usage.
State	A mode or condition of something in the domain and/or in the system. A configuration of data.
Story	A short description of what a user does or needs. Short for user story.
System	A set of interacting software and/or hardware components.
Target	A desired quality level or goal .
Task	A piece of work (that users do, maybe supported by a system).
Term	A word or group of words having a particular meaning.
Test	A procedure to check if requirements are met.
Ticket	(Development) work awaiting to be completed.
UseCase	A list of steps defining interactions between actors and a system to achieve a goal.
User	A human interacting with a system.
Variant	An object or system property that can be chosen from a set of options.
VariationPoint	An opportunity of choice among variants.
WorkPackage	A collection of (development) work tasks.

Attribute	*Definition*
Benefit	A characterisation of a good or helpful result or effect (e.g. of a feature).
Capacity	The largest amount that can be held or contained (e.g. by a resource).
Code	A collection of (textual) computer instructions in some programming language, e.g. Scala. Short for source code.
Comment	A note that explains or discusses some entity.
Constraints	A collection of propositions that restrict the possible values of a set of variables.
Cost	The expenditure of something, such as time or effort, necessary for the implementation of an entity.
Damage	A characterisation of the negative consequences if some entity (e.g. a risk) occurs.
Deprecated	A description of why an entity should be avoided, often because it is superseded by another entity, as indicated by a 'deprecates' relation.
Example	A note that illustrates some entity by a typical instance.
Expectation	The required output of a test in order to be counted as passed.
FileName	The name of a storage of serialized, persistent data.
Frequency	The rate of occurrence of some entity.
Gist	A short and simple description of an entity, e.g. a function or a test.
Image	(The name of) a picture of an entity.
Input	Data consumed by an entity,
Max	The maximum estimated or assigned (relative) value.
Min	The minimum estimated or assigned (relative) value.
Order	The ordinal number of an entity (1st, 2nd, ...).
Output	Data produced by an entity, e.g. a function or a test.
Prio	The level of importance of an entity. Short for priority.
Probability	The likelihood that something (e.g. a risk) occurs.
Profit	The gain or return of some entity, e.g. in monetary terms.
Spec	A (detailed) definition of an entity. Short for specification
Status	A level of refinement of an entity (e.g. a feature) in the development process.
Text	A sequence of words (in natural language).
Title	A general or descriptive heading.
Value	An amount. An estimate of worth.
Why	A description of intention. Rationale.

Relation	*Definition*
binds	Ties a value to an option. A configuration binds a variation point.
deprecates	Makes outdated. An entity deprecates (supersedes) another entity.
excludes	Prevents a combination. An entity excludes another entity.
has	Expresses containment, substructure. An entity contains another entity.
helps	Positive influence. A goal helps to fulfil another goal.
hurts	Negative influence. A goal hinders another goal.
impacts	Some influence. A new feature impacts an existing component.
implements	Realisation of. A module implements a feature.
interactsWith	Communication. A user interacts with an interface.
is	Sub-typing, specialization, part of another, more general entity.
precedes	Temporal ordering. A feature precedes (is implemented before) another feature.
relatesTo	General relation. An entity is related to another entity.
requires	Requested combination. An entity is required (or wished) by another entity.
superOf	Super-typing, generalization, includes another, more specific entity.
verifies	Gives evidence of correctness. A test verifies the implementation of a feature.

Acknowledgments. This work is partly funded by VINNOVA within the EASE project.