A Market-driven Requirements Engineering Process - Results from an Industrial Improvement Programme

Regnell, Björn; Beremark, Per; Eklundh, Ola

Published in:
Requirements Engineering

1998

Published in Proceedings of CEIRE'98 (Conference on Europeean Industrial Requirements Engineering, London, 19-20 October 1998).
Accepted for publication in the *Requirements Engineering* journal (Springer Verlag)

A Market-driven Requirements Engineering Process
- Results from an Industrial Process Improvement Programme

Björn Regnell[1], Per Beremark[2], Ola Eklundh[2]
[1]Department of Communication Systems, Lund University, Sweden, bjornr@tts.lth.se
[2]Telelogic AB, Malmö, Sweden, (per.beremark | ola.eklundh)@telelogic.se

Abstract. In market-driven software evolution, the objectives of a requirements engineering process include the envisioning and fostering of new requirements on existing packaged software products in a way that ensures competitiveness in the market place. This paper describes an industrial, market-driven requirements engineering process which incorporates continuous requirements elicitation and prioritisation together with expert cost estimation as a basis for release planning. The company has gained a measurable improvement in delivery precision and product quality of their packaged software. The described process will act as a baseline against which new promising techniques can be evaluated in the continuation of the improvement programme.

1. Introduction

Requirements Engineering (RE) for packaged software is different from RE for bespoke software. In tender projects, the customer is well-defined and the requirements specification often acts as a contract between the developer and the customer. When developing packaged software for a market place, the RE process should be able to invent requirements based on foreseen end-user needs and select a set of requirements resulting in a software product which can compete on the market. A packaged software product, sometimes called COTS (Commercial off-the-shelf) software, is often an integration of components. The product with its components is evolved in releases, with each release including new and improved features that, hopefully, ensure that the vendor stays ahead of competitors.

This paper describes a specific industrial RE process for packaged software, called REPEAT (Requirements Engineering ProcEss At Telelogic), which is enacted by the Swedish CASE-tool vendor Telelogic AB; a fast growing company, currently with 180 employees, more than 600 customers world-wide, and a predicted revenue for 1998 of circa 200 million SEK (increase from 107 million SEK, 1997).

REPEAT is used in-house at Telelogic for eliciting, selecting and managing requirements on a product family called Telelogic Tau; a software development environment for real-time systems, used by many of the world's largest telecommunication systems providers in their software development. Telelogic Tau supports standardised graphical languages, such as SDL [1], MSC [2], TTCN [3], and UML [4], and provides code generators for integration with several real-time operating systems[1].

1. More information on Telelogic Tau can be found at http://www.telelogic.se

Telelogic Tau is an integration of in-house developed COTS components and can be tailored for the specific needs of a customer or a market segment, and is available on UNIX and MS-Windows platforms. It is built using an architecture with an implicit invocation style [5], which enables changes with local impact.

The paper is structured as follows. Section 2 describes the current and first version of the RE process denoted REPEAT-1. The lessons learned from three subsequent enactments are used for the continuation of the RE process improvement program at Telelogic aiming at the definition of REPEAT-2. In this work, REPEAT-1 will act as a *baseline* against which process improvement proposals can be evaluated in case studies. The past experiences of the REPEAT process improvement programme are concluded in Section 3, together with proposals regarding its continuation.

2. REPEAT: A Market-Driven Requirements Engineering Process

REPEAT is an RE process that manages requirements throughout a whole release cycle. It covers typical RE activities [6], such as elicitation, documentation, and validation, and has a strong focus on requirements selection and release planning. Management of requirements changes due to, e.g., new market demands, is an important function.

The actors involved in REPEAT include:

- *Requirements Management Group* (RQMG). This group is responsible for requirements management, and makes decisions on which requirements to implement. It is also responsible for requirements change management. RQMG includes, among others, product and project managers together with the quality manager.
- *Issuer.* Any employee at Telelogic can submit a requirement to RQMG. An issuer is usually a person from marketing & sales or customer support, but can also be e.g. a developer or a tester.
- *Customer*s & *users* provide input and feedback to an issuer regarding user and market needs.
- *Requirements Team.* A team with the responsibility of analysing and specifying a set of requirements. RQMG has several of these teams at their disposal. A requirements team is cross-functional and includes persons participating in implementation, testing, marketing & sales, and customer support.
- *Construction Team.* A team with the responsibility of designing and implementing a set of requirements.
- *Test Team.* A team with the responsibility of verifying a set of requirements.
- *Expert*. A person that is assigned to evaluate a specific requirement in depth, concerned with e.g. cost estimation and impact analysis.
- *Requirements Database* (RQDB). All requirements are stored in this in-house-built database system. RQDB has a web-interface that can be accessed by Telelogic employees from a multi-continent intranet.

Elicitation is continuously active, and a requirement can be issued at any time by an issuer that foresees a market need. Each requirement is stored in RQDB as an entity described in natural language with unique identity. Throughout the continuous enact-

ment of REPEAT process instances, each unique requirement has a life-cycle progressing through specific states as shown in Fig. 1.

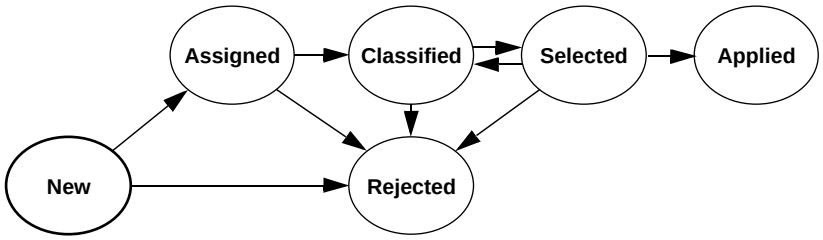

Fig. 1. The states of a requirement in the REPEAT process.

The semantics of the states are explained below. The RQMG with support from experts is responsible for deciding on requirement state transitions.

- *New.* The initial state of a requirement after it has been issued and given an initial priority.
- *Assigned*. The requirement has been assigned to an expert for classification.
- *Classified*. A rough estimate of cost and impact is attached to the requirement. Comments and implementation ideas may also be stated.
- *Rejected*. An end-state indicating that the requirement has been rejected, e.g. because it is a duplicate, already implemented, or it does not comply with the long-term product strategy.
- *Selected*. The requirement has been selected for implementation with a certain priority attached to it combined with results from detailed cost and impact estimations. There is also a more detailed specification of the requirement available. A selected requirement may be de-selected, due to requirements changes, and then re-enters the classification state or gets rejected.
- *Applied*. An end-state indicating that the requirement has been implemented and verified.

REPEAT is instantiated for each release, and each process instance has a fixed duration of 14 months. A new product version is released at fixed dates every sixth month, which implies that different REPEAT instances overlap with at most three simultaneous enactments, as shown in Fig. 2. The REPEAT process instance *n* denotes the *current* release project. Each REPEAT instance consists of five different phases separated by milestones at pre-defined dates. The different phases are described subsequently.

2.1 Elicitation Phase

The elicitation phase includes two activities: *collection* and *classification*. Collection of requirements is made by an issuer that fills in a web-form and submits the requirement for storage in RQDB (see Fig. 3).

Requirements are described using natural language and given a summary name by the issuer. An explanation of *why* the requirement is needed is also given. The issuer

Published in Proceedings of CEIRE'98 (Conference on Europeean Industrial Requirements Engineering, London, 19-20 October 1998).
Accepted for publication in the *Requirements Engineering* journal (Springer Verlag)

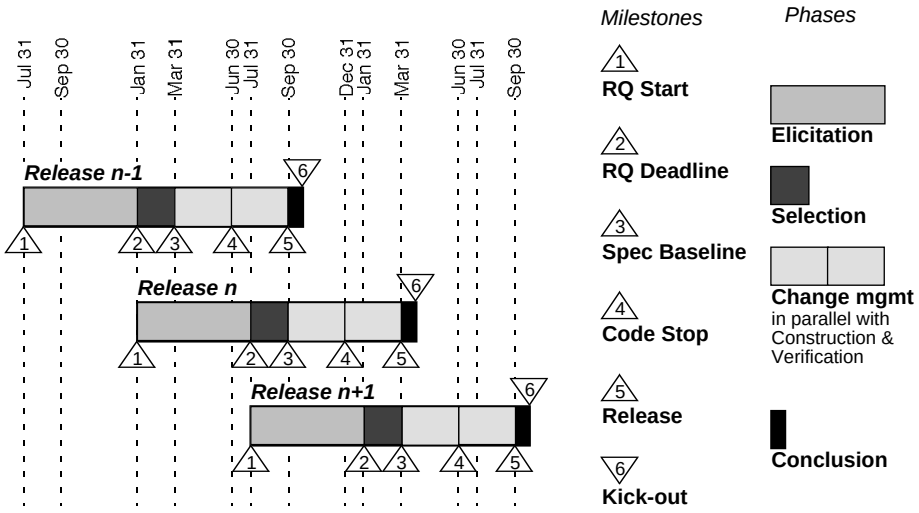

Fig. 2. The milestones, phases, durations and parallelisation of REPEAT process instances.

gives the requirement an initial priority P, which suggests in which release it may be implemented. P is a subjective measure reflecting the view of the issuer, and is measured on an ordinal scale with three levels, as shown in Table 1.

Table 1. The ordinal scale of the priority P.

Level	Semantics
1	The requirement is allowed to impact on-going construction.
2	The requirement is incorporated in the current release planning.
3	The requirement is postponed to a later release.

The requirement is initially in the *new* state, and a first check is made by RQMG to see if it is detailed enough; if not it is returned to the issuer for clarification of its description.

When a new requirement has arrived, RQMG issues a classification of the requirement by assigning it to an expert. The expert classifies the requirement by assigning to it a rough estimate of its cost (C) and impact (I). The cost estimate C is given on an ordinal scale of implementation effort from 1 to 5, as shown in Table 2.

Table 2. The ordinal scale of the cost estimate C.

Level	Semantics
1	Less than 1 day.
2	Less than 5 days.
3	Less than 5 weeks.
4	Less than 3 months.
5	More than 3 months.

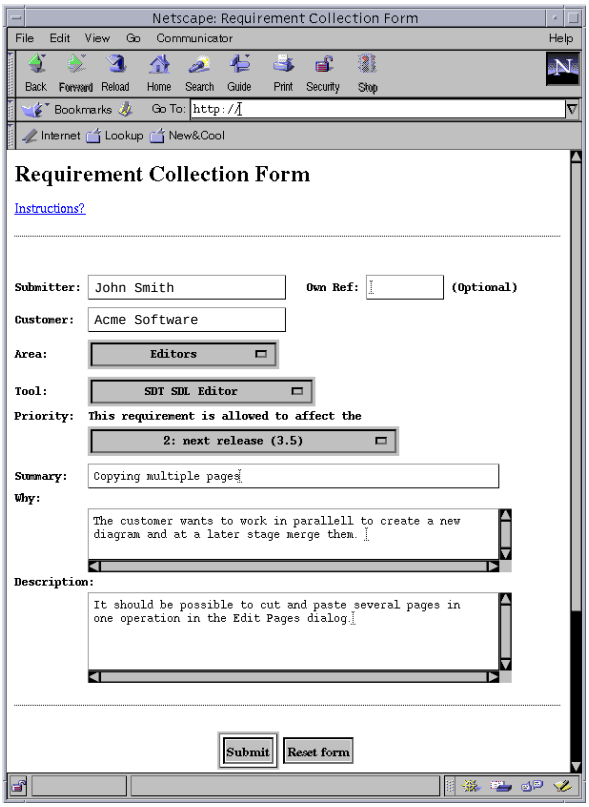

Fig. 3. The web-form for issuing requirements that are stored in the RQDB.

The impact estimate I is given to assess how many architectural components that are affected by the requirement. The I measure is given on an ordinal scale from 1 to 5 as shown in Table 3.

Table 3. The ordinal scale of the impact estimate I.

Level	Semantics
1	Impact is isolated to one component.
2	A few components are impacted.
3	Less than half of all components are impacted.
4	More than half of all components are impacted.
5	Nearly all components are impacted.

The expert also reconsiders the priority P and may recommend RQMG to change P. Further comments and implementation ideas may also be given.

The classification (i.e. estimating C and I, and reconsidering P) should take about 15-30 minutes. If more effort is needed, the expert should recommend the RQMG to

Published in Proceedings of CEIRE'98 (Conference on Europeean Industrial Requirements Engineering, London, 19-20 October 1998).
Accepted for publication in the *Requirements Engineering* journal (Springer Verlag)

issue a pre-study, where the requirement can be decomposed to more fine-grained requirements that are easier to classify.

When the priority P, cost estimate C, and impact estimate I, have been given, the requirement enters the *classified* state, and will be further treated when the subsequent selection phase is started.

2.2 Selection Phase

The goals of this phase are: (1) to select which requirements to implement in the current release, (2) to specify the selected requirements in more detail, and (3) to validate the requirements document.

The output of this phase is a Requirements Document (RD) which includes a selected-list, a detailed specification of all selected requirements, and a not-selected-list including the requirements that are postponed to the next release (see Fig. 4).

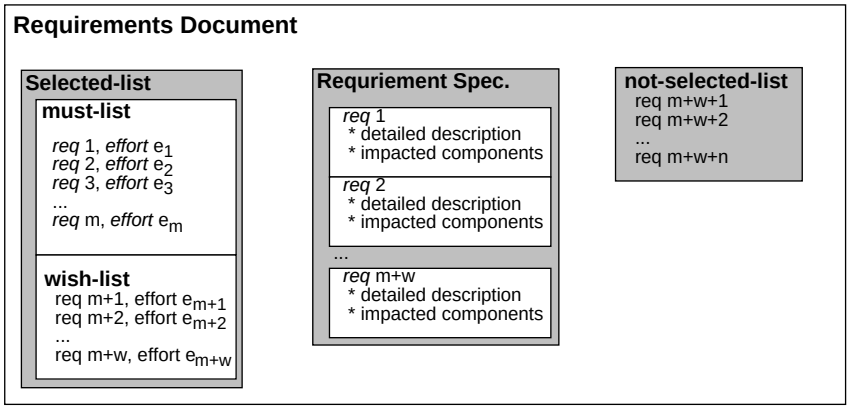

Fig. 4. The requirements document including a list of selected requirements sorted in priority order.

In the RD, there are a total of $m+w$ requirements in a selected-list, and a total of n requirements in a not-selected-list. The selected-list is divided into two parts: m requirements in the must-list and w requirements in the wish-list. The select-list is sorted in priority-order.

For each requirement i on the selected-list, a detailed effort estimation e_i is given, measured on a ratio-scale of hours. Given that there is a total effort of E hours available for implementing the planned release, the *selection-rule* given in Fig. 5 must be fulfilled by the RD.

For all req on the selected-list:	For all req on the must-list:	For all req on the wish-list:
$\left(\sum_{i=1}^{m+w} e_i\right) \le 1.3E$	$\left(\sum_{i=1}^{m} e_i\right) \le 0.7E$	$\left(\sum_{i=m+1}^{m+w} e_i\right) \le 0.6E$

Fig. 5. The selection-rule.

The selected requirements are estimated to take 130% of the available effort *E*. The must-list comprises 70% of *E* (i. e. a 30% risk buffer) and the wish-list comprises 60% of *E*. This implies that up to half of the wish-list will be implemented.

The effort estimation and detailed specification of all selected requirements are made by requirements teams, and more effort is put on specifying the high-priority-requirements. The sorting of the selected requirements in priority order, is made by RQMG with support from the requirements teams, using the P, C, I, and e_i measures and the detailed specifications as input information.

When the RD is completed, it is validated in an inspection before it is put in the Specification Baseline. The not-selected-list is used in the validation of the RD so that no requirements are unintentionally omitted. The not-selected requirements are in state classified, and are normally placed in the selected-list in the RD of the next release.

2.3 Change Management, Construction, Verification and Conclusion

After Specification Baseline, the REPEAT process instance enters the Change Management Phase. When this happens a new REPEAT process instance is started in the Elicitation Phase (see Fig. 2). During change management the RQMG takes decisions on changing the RD caused by new incoming requirements with P=1, i.e. high-priority requirements that are suggested to impact the current *development process* (including construction and verification) running in conjunction with the change management phase of REPEAT.

When the RD is changed, and a new requirement is allowed to enter the must-list, the selection-rule in Fig. 5 must still hold, and a set of requirements amounting to the same effort as the new requirement must be de-selected. The new requirement is inserted by RQMG at a position in the selected-list that reflects its decided priority. Feedback is given to the issuer on the decisions taken to the change request.

The Code Stop milestone separates *construction* from *verification*. Construction is made using an iterative design and implementation process with a weekly build and unit test. In the verification activity, the requirements in the selected-list that where really implemented are verified against the RD using a requirements-based testing method. When the implementation is correct with respect to RD, the new release is delivered to marketing & sales and the implemented requirements enter the applied state. A Conclusion Phase is then entered, where metrics are collected and a final report is written that summarises the lessons learned from this REPEAT enactment.

2.4 Some Process Enactment Scenarios of REPEAT

To further explain how REPEAT is enacted in its different phases, we present a partial process scenario model, using Message Sequence Charts (MSC) [2]. Fig. 6 shows a High-level MSC (HMSC) [8], that describes the events related to *one* requirement.

Assume that we are in the elicitation phase of the current release *n*, and we issue a requirement *req* that is classified according to the classification scheme described in Section 2.1. In Fig. 7, a typical process scenario for an elicitation episode is depicted. (The case where *req* is rejected before it is assigned is not included, c.f. Fig. 1.)

Published in Proceedings of CEIRE'98 (Conference on Europeean Industrial Requirements Engineering, London, 19-20 October 1998).
Accepted for publication in the *Requirements Engineering* journal (Springer Verlag)

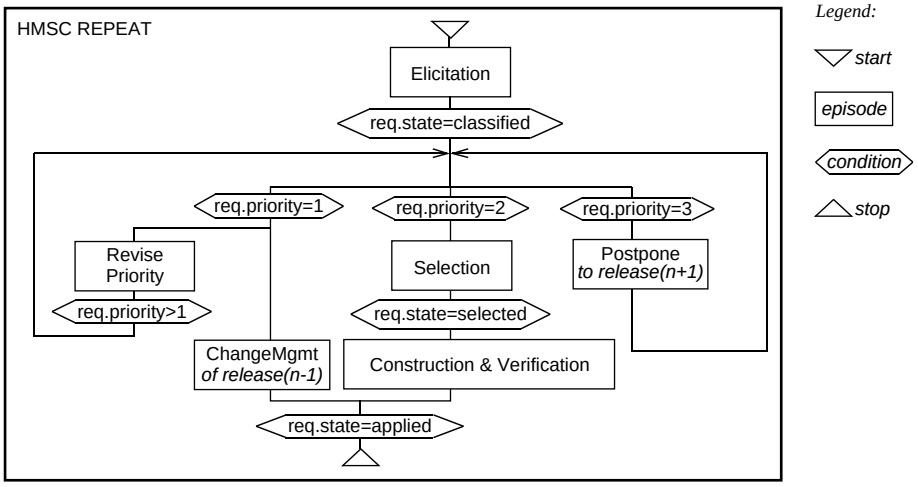

Fig. 6. Different ways of handling a requirement depending on its classification. (Time progresses down-wards.)

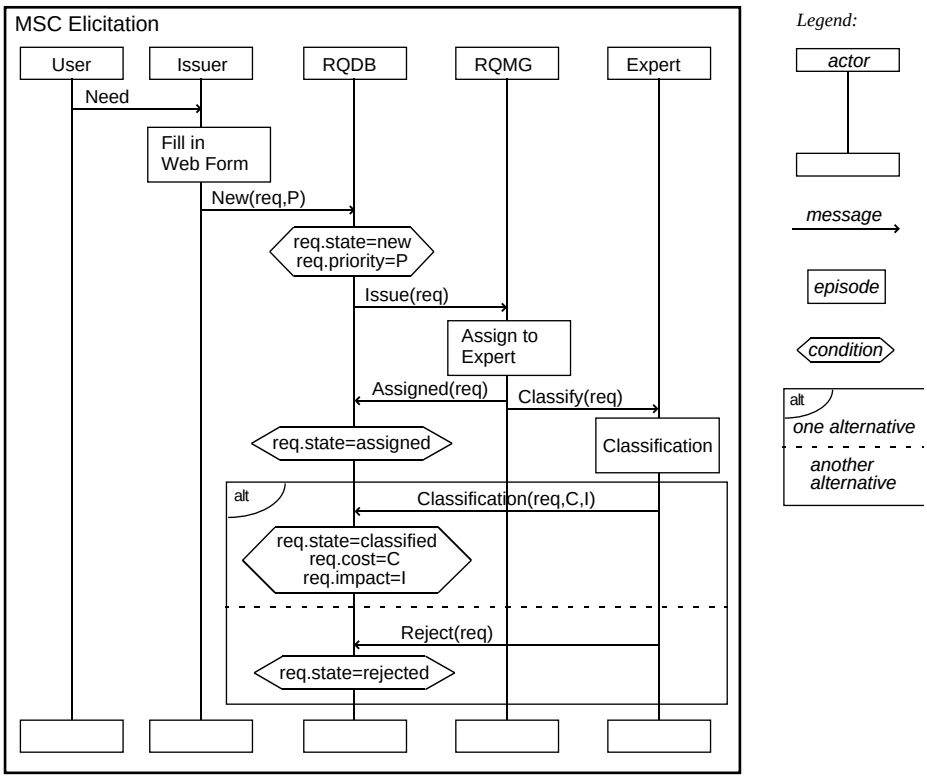

Fig. 7. A partial description of the events in the elicitation phase of REPEAT. (Time progresses down-wards.)

When the RQ deadline is reached (see Fig. 2) and the elicitation phase is ended the priority of *req* determines which release it will affect. Thus, P suggests to which release it should be "routed" for further treatment.

If *req.priority*=1 then release *n*-1 may incorporate *req* in its change management phase. As it is rather expensive to incorporate late changes, it is not unusual to enact the Revise Priority episode, so that ongoing construction is unaffected (see Fig. 6).

If a change management of release *n*-1 is enacted, different actions are taken depending on how far release *n*-1 has reached, as shown in Fig. 8.

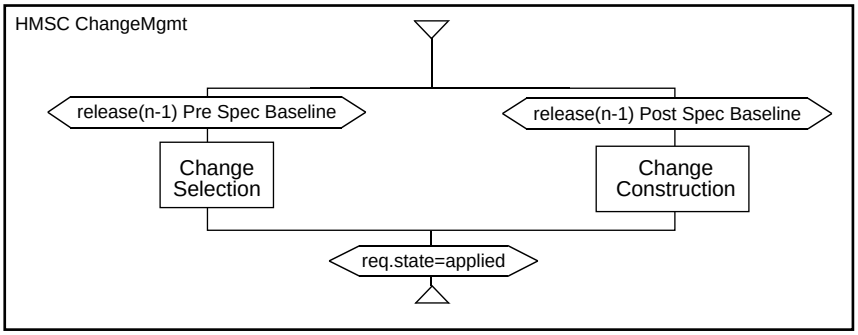

Fig. 8. Different change episodes depending on the advance of release *n*-1.

If the REPEAT process of release *n*-1 is in the selection phase (i.e. pre Specification Baseline), the Change Selection episode is enacted where *req* is allowed to change the selected-list. This change is not so expensive as the other case, where the Specification Deadline milestone is passed. Then *req* implies that a Change Construction episode is enacted, causing expensive re-design and, if Code Stop is reached, re-testing.

If *req.priority*=2 then the treatment of *req* will continue with the selection phase in release *n*, where *req* is specified in more detail and subjected to detailed effort and impact estimation. By the end of the selection phase, the must-list and wish-list are constructed (as described in Section 2.2) including *req* at its decided priority level. When req is on the must- or a wish-list, it is in the *selected* state and when it has been implemented and verified it is in the *applied* state.

If *req.priority*=3 then *req* is kept in the *classified* state and postponed to a later release, where its classification (P, C, and I) is reconsidered. If *req* at this stage is given priority 1 or 2 it will eventually change its state to *selected*, and at some future stage become *applied*.

3. Conclusions from the REPEAT Process Improvement Programme

During 1995, Telelogic realised that they needed a repeatable and defined RE process and the work started on the formulation of an RE Process Improvement Programme resulting in REPEAT-1.

Prior to REPEAT-1, Telelogic had an ad hoc process for managing requirements and faced a number of challenges related to release precision and product quality. Ver-

sion 3.0 of the product family was released 8 months later than planned, and version 3.1 was released with a 3 months delay. Between 3.0 and 3.1, seven intermediate releases were needed in order to mend quality problems and add on extra requirements. In May 1995, a CMM assessment [7], conducted by an external software engineering consultancy, concluded that very few of the Key Processes Areas of CMM-Level 2 were in place.

REPEAT-1 was introduced in January 1996. In February 1998, a second CMM assessment showed that Telelogic had almost all Key Processes Areas of CMM-Level 2 in place. When REPEAT-1 was applied, product version 3.2 was released with a small delay of 15 days, and the subsequent version 3.3 was released three days *ahead* of schedule. The current version 3.4 under construction is to date on schedule, and is predicted to be released on time. The product quality has increased as indicated by the monotonic decrease of reported failures in operation measured from version 3.1 to 3.3. Almost no requirements were unintentionally missed in the latest two versions. The authors are convinced that the introduction of REPEAT-1 is the major explanation for these achievements.

The major elements of REPEAT-1 that are believed to cause the dramatic improvements in release precision and product quality, are the prioritisation of requirements, the effort estimation, the detailed requirements specification, and the continuous change management throughout design, implementation and verification. The classification activity gives experts the opportunity to carefully consider which requirements to be implemented in which release, so that the requirements that are believed to give the highest value to the lowest cost are implemented first. The must-, and wish-lists are strong tools for enforcing that a release project does not take in more requirements than can be achieved within 6 months. Customer support and marketing can easily issue requirements as a reaction to their observation of end-user and market needs.

However, a number of challenges have been identified in the past enactment of REPEAT. Some of these challenges are outlined below:

- *Overload control.* With the web-interfaced requirements database, it is very easy to issue new requirements. Every new requirement has a cost, even if its never implemented. Requirements that are in state classified must eventually be either applied or rejected. Currently, the number of classified requirements in RQDB is increasing for every release, which is about to cause REPEAT-1 to be overloaded. A mechanism is needed to avoid congestion in the RE process.
- *Connecting fragments.* The requirements entities are not related to each other. They are only grouped in relation to implementation components. Requirements fragments need to be packaged into coherent bundles, in order to give them a structure that reflects the functionality as seen by the user. This is necessary for managing dependencies and making prioritisation of sets of requirements.
- *Bridging the chasm between elicitation and selection.* Related to the above challenges, the authors have made the qualitative observation that the performance of REPEAT-1 is low in the gap between elicitation and selection (see Fig. 9), due to congestion caused by too many incoming, unrelated requirements fragments with sometimes poor description quality. The requirements fragments are described at very different levels of abstraction and classification gets difficult.

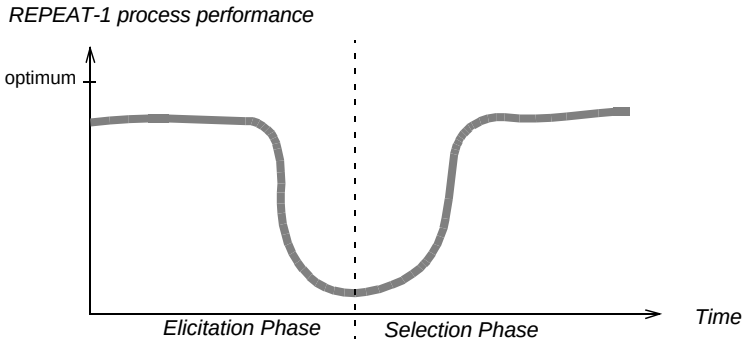

Fig. 9. An informal depiction of the REPEAT process performance challenge of bridging the chasm between elicitation and selection.

- *Long-term product strategy for a diversity of market segments*. As REPEAT-1 triggers on the issuing of new requirements, RE becomes to some extent reactive rather than pro-active. There is a foreseen need of promoting activities related to the existing long term product strategy and prioritisation in relation to a range of market segments.

During the continuation of the REPEAT Process Improvement Programme, REPEAT-1 will act as a *baseline* against which promising techniques, that are believed to meet the above challenges, can be evaluated using expert surveys, case studies and experiments [9]. Two techniques that are candidates for introduction in REPEAT-2 are:

- *Hierarchical use case modelling* [8, 10, 11]. A hierarchy of informal or semi-formal use cases may help to connect requirement fragments and provide an integrated model of the product's "functionality architecture" as seen by its users. Hopefully, a long-term product strategy with the priorities of different market segments can be integrated with such a use case model.
- *Cost-value use case prioritisation*. In order to increase process performance and avoid congestion, a more efficient approach to the sorting of requirements is need. Currently the selection is made using expert judgement. A smart grouping of requirements based on the use cases to which they are related, combined with a systematic cost-value prioritisation approach [12, 13] applied to use cases instead of single requirements, may speed up the selection process.

Published in Proceedings of CEIRE'98 (Conference on Europeean Industrial Requirements Engineering, London, 19-20 October 1998).
Accepted for publication in the *Requirements Engineering* journal (Springer Verlag)

Acknowledgements. The authors would like to give a special acknowledgement to Anders Ek at Telelogic, who participates in the development of REPEAT. Thanks also to Per Runeson, Lars Bratthall, and Claes Wohlin at Dept. of Communication Systems, who have carefully reviewed this paper. The academic participation in this work is partly funded by the National Board of Industrial and Technical Development (NUTEK), Sweden, grant 1K1P-97-09690.

References

1. Specification and Description Language (SDL), *ITU-T Standard Z.100*, International Telecommunication Union, 1992.
2. Message Sequence Chart (MSC), *ITU-T Recommendation Z.120*, International Telecommunication Union, 1996.
3. Tree and Tabular Combined Notation (TTCN), ISO/IEC Recommendation 9646-3, International Standardisation Organisation, 1992.
4. Fowler, M., Scott, K., *UML Distilled - Applying the Standard Object Modelling Language*, Addison-Wesley, 1997.
5. Shaw, M., Garlan, D., *Software Architecture - Perspectives on an Emerging Discipline*, Prentice Hall, 1996.
6. Sommerville, I., Sawyer, P., *Requirements Engineering - A Good Practice Guide*, Wiley, 1997.
7. Humphrey, W. S., *Managing the Software Process*, Addison-Wesley, 1989.
8. Regnell, B., Andersson, M., Bergstrand, J., "A Hierarchical Use Case Model with Graphical Representation", *IEEE International Symposium and Workshop on Engineering of Computer-Based Systems*, Germany, March 1996.
9. Wohlin, C., Gustavsson, A., Höst, M., Mattsson, C., "A Framework for Technology Introduction in Software Organizations", *International Conference on Software Process Improvement*, Brighton, UK, 1996.
10. Regnell, B., Kimbler, K., Wesslén, A., "Improving the Use Case Driven Approach to Requirements Engineering", *IEEE Second International Symposium on Requirements Engineering*, York, UK, March 1995.
11. Regnell, B., *Hierarchical Use Case Modelling for Requirements Engineering*, Licentiate Thesis No. 120, Dept. of Communication Systems, Lund University, Sweden 1996.
12. Karlsson, J., Ryan, K., "A Cost-Value Approach for Prioritizing Requirements", *IEEE Software*, September/October 1997.
13. Karlsson, J., *A Systematic Approach for Prioritizing Software Requirements*, Doctoral Dissertation No. 526, Dept. of Computer and Information Science, Linköping University, Sweden, 1998.