Towards requirements engineering practice for professional end user developers: a case study.

FAILY, S.

2008

Towards Requirements Engineering Practice for Professional End User Developers : A Case Study

Shamal Faily
Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD, UK
shamal.faily@comlab.ox.ac.uk

Abstract

End-User Development has received a lot of attention in the research community. Despite the importance of Requirements Engineering in the software development lifecycle, comparatively little exists in the way of prescriptive advice or case studies on both Requirements Engineering and End-User Development. This paper argues that end-user developers can obtain practical benefit by adopting professional Requirements Engineering practices. We report on how these practices were fostered within a workplace environment and illustrate that evaluating the effectiveness of teaching such practices can lead to a better understanding of the relationship between End-User Development and Software Engineering in general.

1 Introduction

End-user developers are non-professional software engineers who write software in support of achieving their goals [12]. Examples of such developers are scientists who develop computational models to investigate phenomena [26] and business users who develop and maintain spreadsheets [14]; several other perspectives of End User Development are discussed by Blackwell [2]. Segal [19] has argued that not all end-user developers are the same and has carried out empirical studies focusing on *professional end-user developers*, who are proficient in their programming languages of choice, have a deep knowledge of their chosen problem domain, but are neither trained nor interested in professional software development.

End-User Development (EUD) has received a lot of attention within the research community, as characterised by Nardi [13] and, more recently, Lieberman et al [10]. It is generally agreed that the population of the EUD community is increasing, as is the critical impact of the software being developed. If such software is not dependable, then serious consequences await those who rely on this software working correctly [12].

While a lot of attention has been given to end user development practices in general, comparatively little has been said on the larger issues of software engineering, described by Blackwell [2] as *end-user software engineering*. This dearth may be attributable to what Kelly [8] describes as a chasm between the proponents of domain-independent software engineering and developers working within application domain silos. Although the benefits of what Wing [27] describes as *Computational thinking*, e.g. thinking in terms of abstraction, decomposition and recursion, extend to those beyond the Computer Science community, Wilson [25] suggests that people on the other side of this chasm do not understand these concepts without repeatedly encountering them during the development process. Some research on using software engineering knowledge to inform the end-user programming environment is on-going. For example, Burnett et al. [4] show how aspects of the software lifecycle can support dealing with testing, assertions and fault localisation within the Forms/3 framework [3].

The contribution of Burnett and her colleagues should not be dismissed, but it does illustrate that of the lifecycle phases being considered, little attention appears to be paid to Requirements Engineering (RE) as an EUD activity. Some of the possible consequences of this have been reported in recent work. In a case study of scientific software developers by Sanders & Kelly [18], none of the participants interviewed were aware of requirement specifications being written before the software under specification was developed. Segal & Moore [20] illustrate the cultural conflict between software engineers who require up-front requirements and scientists who, by virtue of their focus on exploring the application domain, are unable to carry out up-front RE as requirements are expected to be emergent.

Lieberman et al. [9] suggest that current EUD research for dealing with the capture of user requirements is based on developing of Domain Specific Languages which allow users to express their desired functionality. As promising as

this line of research may be, it is unlikely to benefit many end-user developers in the short to medium term, who have neither the time nor the inclination to change the fundamental way they develop software. Berti et al. [1] describe how scenarios and sketches can be used to capture informal input from end-user developer stakeholders. While this work facilitates working at different levels of abstraction, there is an implicit assumption that the fundamental difficulty lies only at the beginning of the design process. None of these research exemplars say anything about the pedagogical issues which need to be addressed before requirements can be expressed and modelled, i.e. how to teach end-user developers to elicit, analyse, verify and validate requirements.

Wilson [23, 25] has investigated approaches for educating computational scientists in professional software engineering techniques and concluded that intensive short courses is a less effective method for delivering new tools and techniques than mentorship with an experienced developer. Wilson has also provided general, but practical, guidance on how to gather requirements during the development process [24].

2 Background

2.1 Context

The work described by this paper was carried out by members of the European Space Agency's (ESA) Flight Dynamics Division, based at the European Space Operation Centre (ESOC) in Darmstadt, Germany. The division is responsible for orbit and attitude determination of ESA spacecraft supported by ESOC, and its members develop, maintain and operate Flight Dynamics Systems software running on the agency's Orbit and Attitude Operations System (ORATOS), a mission-independent software and hardware infrastructure.

This study reports the results of teaching and applying RE best practice, i.e. practices commonly used by requirements engineers in industry, to a project developing a cross-section software tool for generating Spacecraft Trajectory Data Messages (STDMs). STDMs consist of a sequence of earth-fixed state vectors, containing components for spacecraft position and velocity. These are transmitted to ground stations to support the tracking of ESA spacecraft [28]. Each section within the Flight Dynamics division used their own software tools for generating STDMs, which had evolved over several years to meet requirements specific to the individual sections.

This work was carried out as part of a programme to build the next generation of ORATOS (ORATOS-NG). Although the STDM generation software would form part of ORATOS-NG, this particular work activity was considered a pilot project. As such, the project was a vehicle for experimenting with different tools and techniques for building ORATOS-NG artifacts. These tools and techniques would include an RE approach, suitable for adoption by the division for future ORATOS-NG projects.

The project team itself consisted of nine on-site ESA and contractor staff. Most of the team members, including the project manager, were professional end-user developers. While only a few of these team members had experience of the STDM generation process, each possessed a general domain knowledge of flight dynamics operations. These users were also competent Fortran programmers and had developed components for flight dynamics systems used by several missions. These users also had a passing familiarity with the production of requirements documentation and would occasionally be required to contribute to the Flight Dynamics requirements compilation for a particular mission, or comment on the requirements of related ground segment systems.

Three of the team members, including the author, were professional software engineers and assigned the role of *process coaches*. The coaches were responsible for critically monitoring the process and identifying areas for improvement. The coaches were also responsible for recording the information exchanged during the process on the Flight Dynamics Division wiki. Although professional software engineers, the coaches also had general experience of the Flight Dynamics domain as developers and maintainers of the software infrastructure used by the Flight Dynamics end-user developers. All of the coaches had received some form of RE training, either as part of an undergraduate degree programme or industrial training, and had varying levels of experience in carrying out professional RE activities.

In conjunction with the RE activities described within the approach, knowledge capture activities were also undertaken by project team members. This involved documenting content describing domain concepts, salient to STDM generation. Although details on this knowledge capture exercise is beyond of the scope of this case study, the resulting content was often used as supporting material for the requirements. This material included an overview of STDMs, as well as more in-depth detail on related infrastructure items, such as Time Formats, Ground Stations and Planetary Ephemerides. This content was added to the Flight Dynamics Division wiki. As this wiki had been in use by the division for some years, it was considered the most effective way of disseminating domain knowledge to the rest of the team.

Due to the critical nature of the day-to-day work carried out by the team members, all the work described by this paper took place on a part-time basis, over a period of 12 months.

2.2 Training Approach

Given the lack of pedagogical guidance from the EUD and RE communities for imparting RE knowledge to end user developers, it was not possible to formulate a clear strategy for delivering the requisite RE knowledge to the ORATOS-NG team at the beginning of the project. Consequently, the process coaches decided to deliver instruction based on RE tools and techniques which had been either been personally tried and tested by the process coaches, or anecdotal evidence suggested their adoption would be successful. As operational commitments made it difficult for users to get time away for dedicated, full-time training, the following approaches were used as vehicles for delivering RE knowledge.

- Team meetings
- Wiki-based guidance
- One-to-one tutorials

Initial team meetings were used as a means to explain core concepts to the entire project team. For example, at an early stage, there was uncertainty within the team about how best to begin a Requirements Engineering process. To remedy this, the author used one of the team meetings to deliver a training session on the use of context modelling. The presentation consisted of a brief overview of Context Models, Use Cases and Problem Frames [7], supplemented with simple domain and non-domain specific examples. Following the presentation, the team informally discussed the pros and cons of each approach before deciding to trial the use of context models and use cases. The meetings which subsequently followed were used to collectively work on an STDM generator context model and project blast-off document, as described in Robertson & Robertson [16] to establish the scope of the project. When the scope of the project had been agreed, the amount of RE instruction provided during the meeting was restricted to dealing with problems which arose in the process of authoring use cases and requirements.

In addition to full team meetings, a splinter group of users, with specific knowledge of STDM generation, would frequently meet to collectively discuss elicited requirements for the STDM generator and its dependent infrastructure elements. As these meetings were led by the users and focused on requirements validation meetings, the role of the process coach was to clarify any issues raised by users relating to the quality of requirements. These included responding to specific queries about authoring requirement text and commenting on the quality of requirements and fit-criteria in specific cases.

Wikis have proven to be useful in software development as a discussion medium [11] and, when appropriate structured, as a framework for supporting RE activities [5, 15]. Nevertheless, it was believed that the wiki may not be robust enough as a requirements management tool. Consequently, the process coaches evaluated a number of commercial and open-source requirements tools, based on the criteria specified within the INCOSE Requirements Management Tool Survey [6]. Following a short trial of two shortlisted applications by the rest of the project team, Telelogic DOORS [22] was selected as the ORATOS-NG Requirements Management tool. The process coaches added material to the wiki, which provided guidance on how to author requirements and use cases. An assumption was made by the process coaches that focusing on techniques for authoring robust, testable requirements statements would force users to consider the thought processes involved in writing good requirements. Although this material was primarily prepared to enable users to read and digest in their own time, an overview of the initial content added was also given during one of the weekly team meetings. Many of the guidelines documented were based on ideas presented in Robertson & Robertson [16], including the use of the VOLERE Requirements template as the basis of a standard template for ORATOS-NG requirements.

One-to-one tutorial sessions were held with a number of users to validate the quality of requirements being currently authored. During the session, the user and the coach would discuss each requirements statement, with respect to its wording, fit criteria and justification/rationale ; particular emphasis was placed on inspecting rather than reviewing requirements. For example, the author would occasionally use Z [21] to formalise requirements text, to demonstrate how poor wording, coupled with loose fit criteria, could conspire to break the system, while still satisfying the statement text and fit criteria. Although the users were mathematicians, the discrete mathematical notation of Z proved to be unfamiliar. As Z was also unfamiliar to the other process coaches, this technique was used exclusively within tutorial sessions led by the author, where the necessary notation could be explained. Notational issues aside, users recognised the value of this technique as a way of animating bad requirements text.

3 Training Experience

3.1 Advantages and Benefits

While the part-time, on-site delivery of RE best practice does not replace the benefits of full-time, off-site training, we did observe a number of benefits from the approach adopted. Over the lifetime of the project, these led to a progressive improvement in the users' ability to analyse and

validate requirements. This improvement in quality was measured by using the ORATOS-NG requirements guidelines to check the conformance of requirements text in several DOORS modules.

Based on the splinter group meetings attended by the process coaches, an improvement was also noted in the users ability to validate requirements, re-evaluate fit criteria and re-analyse supporting material to verify requirements traceability. The use of fit criteria was considered to be especially useful, as it often gave a focal point for discussion. On the strength of these observations, we believe the approach followed proved to be largely effective and users obtained tangible benefits, with respect to improvements in their ability to model and validate requirements. While a general improvement in requirements elicitation was also noted, it is difficult to say whether this improvement was substantial. As the users were expert both in the knowledge of the domain and the technology necessary to ultimately realise the requirements, requirements elicitation activities were primarily based on retrospectively inferring requirements from design documentation, technical notes and source code. As these activities were solitary activities carried out within the users' own offices, any improvement in these techniques could not be measured.

Although the STDM generation software was subsequently never implemented, the team was sufficiently satisfied with the improvement in their ability to carry out RE activities that the approach was adopted for subsequent ORATOS-NG projects. The advantages of each element of the overall approach is described in the following sections.

3.1.1 Team meetings

Early versions of requirements produced by many of the team members resembled a wish-list of language implementation specific features. Additionally, many of the mandated fields in the requirements template were left incomplete. While the spirit of the requirements was understood by their respective authors, team meetings provided a forum for other users to challenge these implicit assumptions. By raising the issue of requirements quality during team meetings, it was possible not only to discuss how this quality could be improved, but also to address any misconceptions held by the users.

3.1.2 Wiki-based guidance

As their experience grew, users also began to take progressive ownership of the process. This was most evident when considering the evolution of guidelines provided for using DOORS and authoring requirements. Users took responsibility for customising the guidelines on the wiki and proposing changes to the DOORS ORATOS-NG requirements template, e.g. the Source and History attributes were found to be realised within DOORS by traceability links and object history respectively.

As well as improving the original guidelines, users also provided several domain-specific examples of how to complete the requirements template, pitfalls which should be avoided and useful hints and tips for using DOORS.

3.1.3 One-to-one tutorials

From the coaches' observations during splinter group meetings, we believe that the main catalyst to this increased process ownership was the emphasis on inspecting, rather than simply reviewing, requirements. Rather than passively discussing a requirements module under review, users would actively challenge the text of each requirement for completeness, relevance and traceability. In many ways, users would implicitly carry out a Quality Gateway process as described by Robertson and Robertson [16]. This is particularly interesting as although users were encouraged to use [16] as a reference guide, no explicit instruction on the use of a Quality Gateway process was ever given.

3.2 Difficulties experienced

Although the approach described proved to be effective, it was not completely without problems. The three factors below categorise particular difficulties experienced by users during the take-up of the prescribed techniques. We believe that these difficulties arose both as a consequence of the approach adopted as well as the context in general.

3.2.1 EUD and Software Engineer dichotomy

Even though the users were receptive of the knowledge being imparted, they would occasionally remind the process coaches that they were not "software engineers" when faced with problems understanding particular aspects of a technique. When faced with continual adversity, users would also express hesitance in investing additional effort, if they did not believe the subsequent pay-off was worth the additional mental workload.

During the preparation of use cases, users took particular issue with the level of software engineering expertise they believed was implicitly assumed by the process coaches. Following the initial drafting of use cases for different STDM generation scenarios, users reported problems understanding the meaning of the constituent parts of the use case template. To illustrate the use of the prescribed template, one of the process coaches examined the same source material and, in collaboration with one of the users, produced an alternative version of one of the use cases. The structure of both use cases differed significantly and, although the user acknowledged his better understanding of

the template, concern was raised about level of expertise believed necessary to author use cases.

3.2.2 Over abstraction of best practice

Users occasionally reported confusion refining what they considered to be over-generalised explanations of use cases and requirements; in some cases, questions were asked about whether some aspects of the template were relevant to their specific problem domain. Over abstraction of use case terminology was illustrated by one user who compared and contrasted a trivial, non Flight-Dynamics specific, use case described in Robertson & Robertson [16], with those produced by the team for the different scenarios for STDM generation. The user could understand the role of actors, events and triggers in the example use case, but reported difficulty mapping this understanding to the STDM problem domain, without making too many assumptions about possible solutions. For example, in a number of use cases, it is necessary to define the Orbit Determination sub-system as an actor invoking the generation of STDMs and assume certain interface operations. However, infrastructure elements within this sub-system were also being defined and users were concerned that they may inadvertently make assumptions about the Orbit Determination sub-system implementation, which may eventually prejudice any implementation arising from the STDM Generator requirements and vice-versa. Users believed the guidance available to them for authoring use cases was too generic and didn't factor in practical considerations of working with interleaving levels of abstraction.

In the case of requirements, it was also necessary to make customisations to the VOLERE based requirements templates to make the attributes more relevant to the team. For example, the *Business Rules* column was proposed for removal by some users, as it was not considered relevant. When this attribute of the shell was explained as rules for the context of the system's operation rather than a commercial context per se, this attribute was changed to *Operational Rules* so members of the division would better be able to relate to its meaning.

3.2.3 Nomenclature differences

Although ESOC is an international environment and all non-British team members were fluent in English, confusion occasionally occurred when jargon used in Software Engineering literature was interpreted literally. An example of this occurred towards the end of the requirements capture phase when, during a presentation on Software Architecture and UML, one of the users complained that the grammatical incorrectness of the definition for architecture in the UML Reference Manual, i.e. the organizational structure of a system, including its decomposition into parts, their connectivity, interaction mechanisms, and the guiding principles that inform the design of a system [17], conflicted with the meaning the definition was trying to express.

This issue was compounded by the fact that end-user developers had their own nomenclature which didn't always match that used in general software engineering community. One reason for this is that many of the software development practices in the division can trace their origins back to the late 1960s and appear to have evolved independently from the rest of the software engineering community.

4 Discussion

The results of placing emphasis on inspecting rather than reviewing requirements appear to concur with Wilson's finding that users can transfer their ability to think methodically to areas outside of their natural domain [26]. Further studies would be needed to determine whether this premise holds for other classes of professional end-user developers and how much stake-holder ownership of the process contributes to successful adoption of the techniques described.

While the approach taken was an economical means of delivering a discrete set of usable RE best practice, problems with the take-up of use cases led a loss of confidence in a potentially useful technique, which was never regained. These problems also illustrate that end-user developers can be equally intolerant of aspects of Requirements Engineering, as well as Software Engineering, best practice if the perceived pay-off is dubious. In this instance, more up-front-training in this technique and more examples of where use cases are appropriate may have made a difference with respect to their adoption. Creating supplementary artefacts may have also helped. For example, using a team meeting to create a use case model to supplement the context model, as well have collectively drafting a number of use cases, may have helped remove any confusion about actors and levels of abstraction.

The results also indicate that professional end-user developers are particularly sensitive to nomenclature differences and can fall foul of over-generalised prescriptive guidelines, especially when English is not their first language. RE practitioners may be desensitised to these problems, not only because of their familiarity of the literature, but also because of their implicit confidence in the techniques prescribed. Such confidence is typically achieved by practical awareness of the techniques' nuances, having applied them professionally on previous occasions. End-user developers, who draw experience exclusively from the application domain they work in, are not usually able to draw on such personal experiences. In turn, this makes such developers less likely to tolerate any approach which is dismissive of the associated cognitive load they have to bear. One solution to this problem might be to focus on estab-

lishing a shared understanding of terms and terminology at an early stage and reinforcing this in the wiki-based guidance. However, it is possible that the disconnect between the domain independent and domain specific communities is sufficiently well established that the observations reflect conflicts between the respective norms and values. A discussion on how to reconcile these cultural differences is beyond the scope of this paper. Suffice it to say, we believe these concerns can be best addressed by finding which RE approaches scale between these communities. Only then can a pedagogical discussion on how best to teach these techniques have real value.

5 Conclusions

This paper has described how professional end-user developers can gain practical benefit from adopting RE best-practice. We have also confirmed previous work alluding to the applicability of an end-user developer's methodical nature to software engineering. The bullet points below summarise the findings we believe to be of most interest to professionals and researchers, with regard to professional end-user developer RE education.

- RE techniques can be delivered in an on-site, part-time context in lieu of classroom instruction, provided sessions are exemplar driven and timely with respect to identified problems.
- Emphasising inspection over review of requirements not only increases sensitivity of users to practicalities of expressing requirements, it also fosters activities towards a quality gateway process by leveraging the pre-existing ability to think methodically.
- Cultural conflicts between instructors and pupils can manifest themselves as confusion over nomenclature and complaints about the relevance of techniques. These can be initially reconciled by providing additional training for techniques which may be novel to end-user developers without any formal software engineering education, and being prepared to customise, where possible, standardised phraseology to fit within the context of development and operation. However, these issues may be indicative of deeper cultural divisions, in which case alternative techniques should be explored.

These claims do come with the caveat that the study was carried out using a small team of on-site staff with very specific domain knowledge. Consequently, generalising these results for the entire End-User Development community would be a fallacy, especially when users are spread over multiple sites. Nonetheless, not only do practices long enjoyed by RE practitioners appear to be useful additions to the end-user developer's arsenal, they can also be conveyed with comparatively little pedagogical ceremony.

6 Acknowledgements

The data used within this case study was gathered while the author was working for Logica at ESOC. The author would like to thank the ORATOS-NG team at ESOC, for their assistance and support during the pilot project, and their comments during the preparation of this paper. The author would also like to thank Ivan Fléchais, the OUCL Security Reading Group and the anonymous referees for their valuable review comments, and Judith Segal, for her helpful pointers to related work.

References

[1] S. Berti, F. Paternò, and C. Santoro. Natural development of ubiquitous interfaces. *Communications of the ACM*, 47(9):63–64, 2004.

[2] A. F. Blackwell. Psychological issues in end-user programming. In H. Lieberman, F. Paternò, and V. Wulf, editors, *End-user Development*, volume 9 of *Human-Computer Interaction Series*, pages 9–30. Springer, 2006.

[3] M. Burnett, J. Atwood, R. W. Djang, J. Reichwein, H. Gottfried, and S. Yang. Forms/3: A first-order visual language to explore the boundaries of the spreadsheet paradigm. *Journal of Functional Programming*, 11(2):155–206, 2001.

[4] M. Burnett, G. Rothermel, and C. Cook. An integrated software engineering approach for end-user programmers. In H. Lieberman, F. Paterno, and V. Wulf, editors, *End-User Development*, volume 9 of *Human-Computer Interaction Series*, chapter 5, pages 87–113. Springer, 2006.

[5] B. Decker, E. Ras, J. Rech, P. Jaubert, and M. Rieth. Wiki-based stakeholder participation in requirements engineering. *IEEE Software*, 24(2):28–35, March-April 2007.

[6] INCOSE. INCOSE requirements management tools survey. *http://www.paper-review.com/tools/rms/read.php*, 2007.

[7] M. A. Jackson. *Problem frames : analysing and structuring software development problems*. Addison-Wesley/ACM Press, Harlow, England, 2001.

[8] D. Kelly. A software chasm: Software engineering and scientific computing. *IEEE Software*, 24(6):120–119, Nov.-Dec. 2007.

[9] H. Lieberman, F. Paternò, M. Klann, and V. Wulf. End-user development: An emerging paradigm. In H. Lieberman, F. Paternò, and V. Wulf, editors, *End-user Development*, volume 9 of *Human-Computer Interaction Series*, chapter 1, pages 1–8. Springer, 2006.

[10] H. Lieberman, F. Paternò, and V. Wulf. *End-user Development*, volume 9 of *Human-Computer Interaction Series*. Springer, Dordrecht, 2006.

[11] P. Louridas. Using wikis in software development. *IEEE Software*, 23(2):88–91, March-April 2006.

[12] B. A. Myers, M. M. Burnett, S. Wiedenbeck, and A. J. Ko. End user software engineering: CHI 2007 special interest group meeting. In *CHI '07 extended abstracts on Human factors in computing systems*, 2007.

[13] B. Nardi. *A Small Matter of Programming: Perspectives on End-User Computing*. MIT Press, Cambridge, MA, 1993.

[14] R. R. Panko. What we know about spreadsheet errors. *Journal of End User Computing*, 10(2):15–21, 1998.

[15] E. Ras, R. Carbon, B. Decker, and J. Rech. Experience management wikis for reflective practice in software capstone projects. *IEEE Transactions on Education*, 50(4):312–320, Nov. 2007.

[16] S. Robertson and J. Robertson. *Mastering the requirements process*. Addison-Wesley, Upper Saddle River, NJ, 2nd ed edition, 2006.

[17] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language Reference Manual*. Addison-Wesley, Boston, 2nd edition, 2005.

[18] R. Sanders and D. Kelly. Dealing with risk in scientific software development. *IEEE Software*, 25(4):21–28, July-Aug. 2008.

[19] J. Segal. When software engineers met research scientists: A case study. *Empirical Software Engineering*, 10(4):517–536, 2005.

[20] J. Segal and C. Morris. Developing scientific software. *IEEE Software*, 25(4):18–20, July-Aug. 2008.

[21] J. M. Spivey. *The Z notation: a reference manual*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1992.

[22] Telelogic AB. *Telelogic DOORS*.

[23] G. Wilson. Software carpentry: Getting scientists to write better code by making them more productive. *Computing in Science and Engineering*, 8(6):66–69, 2006.

[24] G. Wilson. Software Carpentry : Version 1122. *http://swc.scipy.org*, 2007.

[25] G. Wilson. Those Who Will Not Learn From History... *Computing in Science & Engineering*, 10(3):5–6, May-June 2008.

[26] G. V. Wilson. Where's the real bottleneck in scientific computing? *American Scientist*, 94:5, 2005.

[27] J. M. Wing. Computational thinking. *Communications of the ACM*, 49(3):33–35, 2006.

[28] G. Ziegler. ESA/ESOC First Acquisition Strategies. In *17th International Symposium on Space Flight Dynamics*, 2003.