The Effect of Variability Modeling on Requirements Satisfaction for the Configuration and Implementation of Off-The-Shelf Software Packages

Amanda Rubython
City&Guilds Kineo
Brighton, UK
amanda.rubython@kineo.com

Neil Maiden
Centre for Human-Computer Interaction Design
City University London, London, UK
N.A.M.Maiden@city.ac.uk

***Abstract*—An industrial experience of the use of a method for discovering customer requirements with which to configure an off-the-shelf software package for implementation is reported. The method uses an adapted form of product variability model to provide common ground between the customer and supplier about requirements and capabilities. An associated decision support software tool guides the supplier and customer through a model-based walkthrough to discover new requirements, based on equivalent capabilities described in the product variability model. We applied the method in the work processes of the commercial provider of a software-based learning management system, and collected quantitative and qualitative data from supplier-customer interactions. Our first experiences with the method led to an increased exposure and expression of customer requirements in the customer-supplier dialogue, compared to the baseline dialogue during software package demonstrations. The paper also reports some first lessons learned to improve the method and adopt its use with other software supplier organizations.**

***Index Terms*- software packages; requirements-led configurations; product variability modelling; transcript analysis**

I. Requirements-Led Configuration of Software Packages

Whilst methods and software tools are now available to support customers to select off-the-shelf software packages that satisfy their requirements [e.g. 1,2,5], there is relatively little research with which to understand how to support software suppliers to adapt (or otherwise) these same off-the-shelf packages, so that a customer's requirements will be satisfied when the package is implemented. Various researchers [e.g. 7] have reported methods and software tools to extract, analyze and manage new requirements on future releases of software packages, however, the role of requirements in the implementation of a selected software package at each customer's site is less understood, and few methods or software tools are available to support it. In this paper, we report experiences with a method based on software product variability modeling and an associated decision support tool that was developed to provide common ground between a software supplier and customer to articulate and agree requirements in response to possible software package variations.

Although methods and tools now exist, specifying requirements, understanding off-the-shelf software packages and aligning requirements to package features remain difficult tasks for customers to undertake, for at least two reasons. The first is that the information that suppliers often provide about their off-the-shelf software packages is influenced by the need to make sales, rather than to provide comprehensive detail about what the software does and does not do. The second reason, which follows from this first, is that customers often experience problems when determining the extent to which their requirements can be satisfied by a software package's features [1]. One consequence is that more information about the software package needs to be made available, and more guidance about how to use this information needs to be provided. In our work, undertaken on behalf of a supplier of software packages, we sought to develop then evaluate a solution to these problems from the supplier side.

The software supplier, City & Guilds Kineo (CGK), is a global learning and technology company that provides workplace e-learning and learning systems to businesses from industries such as retail and telecoms to travel, electronics and the public sector. The core learning system that CGK offers is called Totara, a configurable, off-the-shelf learning management system. Currently, after a customer selects to purchase or to upgrade the Totara software package, one of CGK's solution architects works closely with customer representatives to understand their requirements so that the package can be configured and/or customized to better satisfy these requirements. The solution architect role combines business analysis and interaction design, and involves liaising with customers to implement Totara into the customer's organization. Such analysis work is increasingly needed to configure and to customize the package's features to satisfy the requirements for the wide range of industries that purchase CGK's products – rarely can one software package satisfy the requirements of such a diverse customer base.

However, CGK's experiences have revealed how little knowledge their customers can sometimes have during the

post-sales process of both Totara's capabilities and their requirements on it. This lack of understanding can lead to unrealistic customer expectations about the system that can cause problems during implementation, and can be attributed to two factors. The first is that some customers fail to have detailed requirements discussions at the purchasing stage due to an over-reliance on Request for Proposals, with the decision to buy based on how CGK responded to a list of customer requirements in writing, rather than a practical assessment of Totara capabilities per se. The second factor is that the dialogue between CGK and the customer at the sales stage is usually supported by software demonstrations that do not illustrate specific system features and configurations to meet customer needs. Moreover, the inevitable time and resource pressures on both supplier and customer imposed by a procurement exercise can exacerbate both of these factors.

Therefore, to improve customer understanding of Totara and to support requirements specification during the CGK sales process, we developed a new method to provide a customer with explicit information about requirements-level capabilities that Totara was originally developed to provide to users. The method is new in that it combines variability modeling from software product line engineering with text-based requirements and use case specification techniques to describe possible variations in the configuration of Totara in a form that non-technical customers can understand. We sought to explore, through studies conducted as part of CGK's sales activities with its own customers, whether the method could:

Q1: Expose more customer requirements on the Totara software package implementation, compared to the current software package demonstration process?

Q2: Improve customer understanding of software package capabilities, compared to the Totara current software package demonstration process?

Q3: Allow supplier and customer to align customer requirements and software package capabilities more effectively, compared to the Totara current software package demonstration process?

The remainder of this experience report is in 4 sections. Section 2 outlines the method, called *ETHER*, introduced into CGK's sales work process, and selected variability modeling developed for the Totara software package used in the method. Section 3 reports experiences from the use of the method with 4 different CGK's customers, as well as a comparative baseline process undertaken with a 5th customer without the *ETHER* method. Analyses of the transcripts of the customer representative-solution architect discussions are reported to seek first answers to the above questions. Section 4 reports lessons learned from the analyses of the experiences, and outlines both future research opportunities and the next steps for CGK.

II. THE *ETHER* APPROACH

Currently, when selecting the Totara software package for their organization, customers ask CGK for information about the package. A sales consultant is responsible for investigating requirements with the customer, providing information about Totara to the customer, and costing the proposal. However, when a customer requests more information about one or more Totara features, a solution architect is called upon to demonstrate features to customer representatives. This normally involves an optional short exploration of the customer requirements, then a detailed presentation of the feature and its functions in the software package that lasts about one hour.

We developed the *ETHER* (*Expose THE Requirements*) method to replace these software-led feature demonstrations with more effective, requirements-level communication about how the system can be configured to meet customer requirements more effectively, as well as to allow CGK to produce more accurate requirements-based costings for customers. The *ETHER* method can be applied either in a face-to-face meeting or in a webinar in which the solution architecture and customer representatives explore requirements, package capabilities and their alignment. The method is composed of a four-step process and a decision support software tool that was developed to support some of these steps. Each of the four steps of the method is outlined in turn.

A. Step 1: Introduction

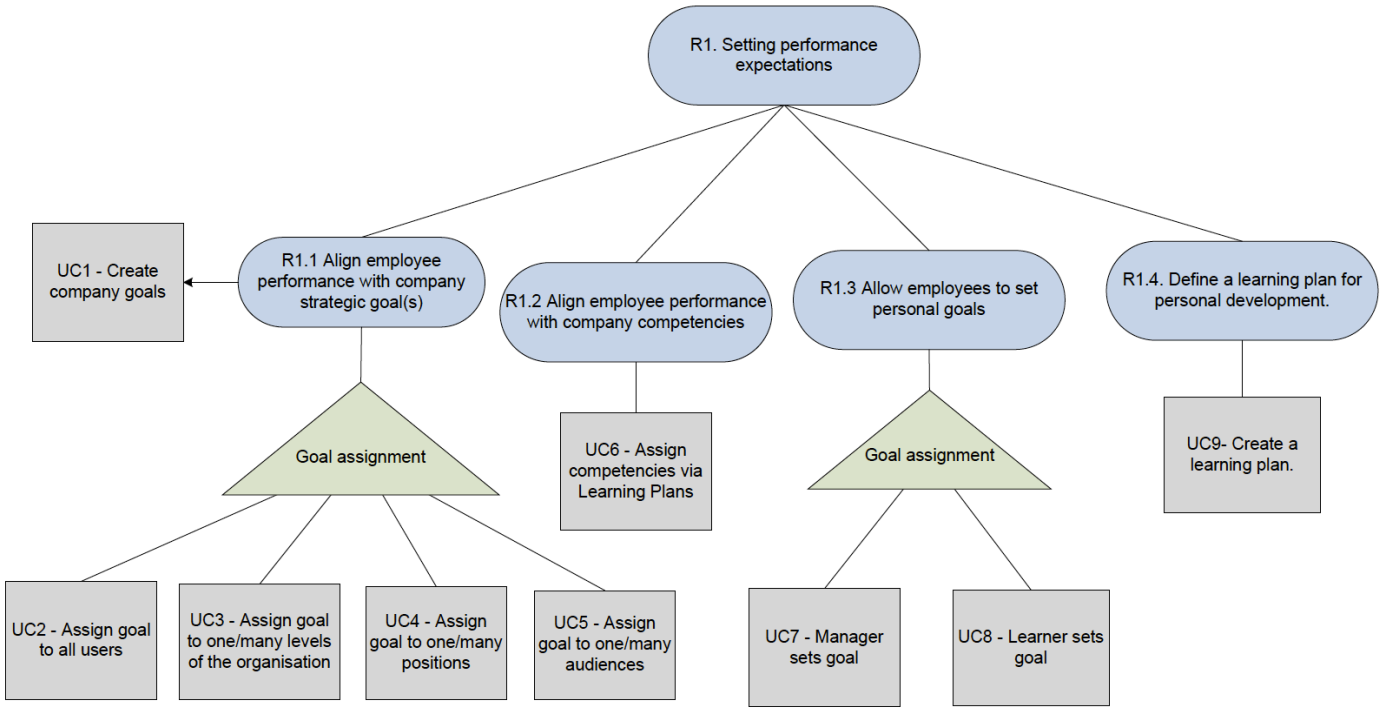
First of all, the solution architect presents the Totara features to be explored, and sets customer expectations – not all customer requirements can be satisfied by a feature, so more expensive software customization and/or business process change might sometimes be needed.

B. Step 2: Build Common Ground

Requirements information is exchanged between customer representatives and the solution architect to build common ground needed for later communication. Common ground is defined as what two or more individuals jointly know or believe [4, p93] – a form of common self-awareness – and communication between people cannot be effective unless all participants share the same perception of knowledge and beliefs [4, p120]. Therefore, the solution architect seeks to build common ground about the customer's requirements and capabilities of Totara to satisfy these requirements, independent of the design and implementation of the software features. To do this, the solution architect first elicits relevant business requirements from the customer representatives, then shares a glossary of relevant Totara terms to avoid potential miscommunication about the software package. Afterwards, the solution architect verbally provides the customer representatives with a high-level description of package capabilities, to prepare the customer for the more in-depth analysis in the next step.

C. Step 3: Requirements that the Product Supports

Then, rather than walk through relevant software features without reference to requirements as undertaken in the current process, the solution architect uses *ETHER* to walk the customer through a bespoke support tool with which to guide decision-making about requirements on the Totara implementation. The decision support tool guides the solution architect and customer representatives through key requirements-based decisions to make about selected Totara features. Support for each decision to make with the tool is based on an underlying goal variability model that was developed as part of the method. This goal variability modeling is based on variability models from software product lines engineering, a paradigm which aims to reduce time and cost in software development by reusing components across a product line [9]. Each variability



model shows where variations can occur in the product line and why, as well as the relationships of dependency and constraint [3]. In *ETHER*, each goal variability model specifies the possible variations in the capabilities of Totara features, independent of their software implementation. The solution architect uses the model and decision support tool to guide the decision-making about requirements that can be satisfied by Totara.

For example, to create a goal variability model for a Totara feature such as *performance management*, a CGK solution architect first defines the requirements achievable using the feature, then maps these requirements to one or more components of the feature, and to a sub-requirement that was a means to achieve the requirements. For each sub-requirement that is not further refined into requirements, a use case was specified of the step-by-step actions using Totara to achieve the requirement. The resulting model for one top-level requirement related to the *performance management* feature of Totara is depicted in Figure 1. The model specifies four variations to achieve the requirement *R1. Setting performance expectations*. One variation is *R1.1 Align employee performance with company strategic goal(s)*, which is in turn achieved by completing the use case *UC1 Create Goals* and assigning goal, which in turn be accomplished in 4 ways represented by 4 use cases.

Whilst it is possible that each goal variability model, on its own, could provide common ground in discussions between the solution architect and customer representatives, first experiences with such models indicated that its form was too difficult for most customer representatives to understand. Therefore, the *ETHER* method was extended with the decision support software tool that presents each variation point in each model as a requirement-based decision to make to the solution architect and customer representatives. During this step, the solution architect and customer representatives walk through decisions about product capabilities, guided by *ETHER's* decision support tool. For each decision point, which is equivalent to an alternative capability in the model achieved by use of the Totara package, the solution architect describes alternative capabilities, and asks which of these align to one or more customer requirements. If the customer's answer is *yes* for any of the capabilities, then the solution architect selects that option in the decision support tool, and explores each set of associated lower-level capabilities using the same process. At the lowest-level node of each hierarchy of capabilities is a description of a use case that specifies the steps needed to perform a task to deliver the capability – steps that the architect could later demonstrate with the software package, beyond the current scope of the method. Examples of alternative capabilities of Totara's *performance management* feature presented in *ETHER's* decision support tool are shown in Figure 2. The left-hand side shows alternative capabilities, and the right-hand side alternative means to achieve one capability.

D. Step 4: Session Review

After all alternative requirements-level capabilities for the feature have been walked through, the solution architect asks the customer representatives if there are any remaining requirements to investigate. Once these other requirements have been considered, the solution architect produces a written summary of the session and sends it to the customer for agreement.

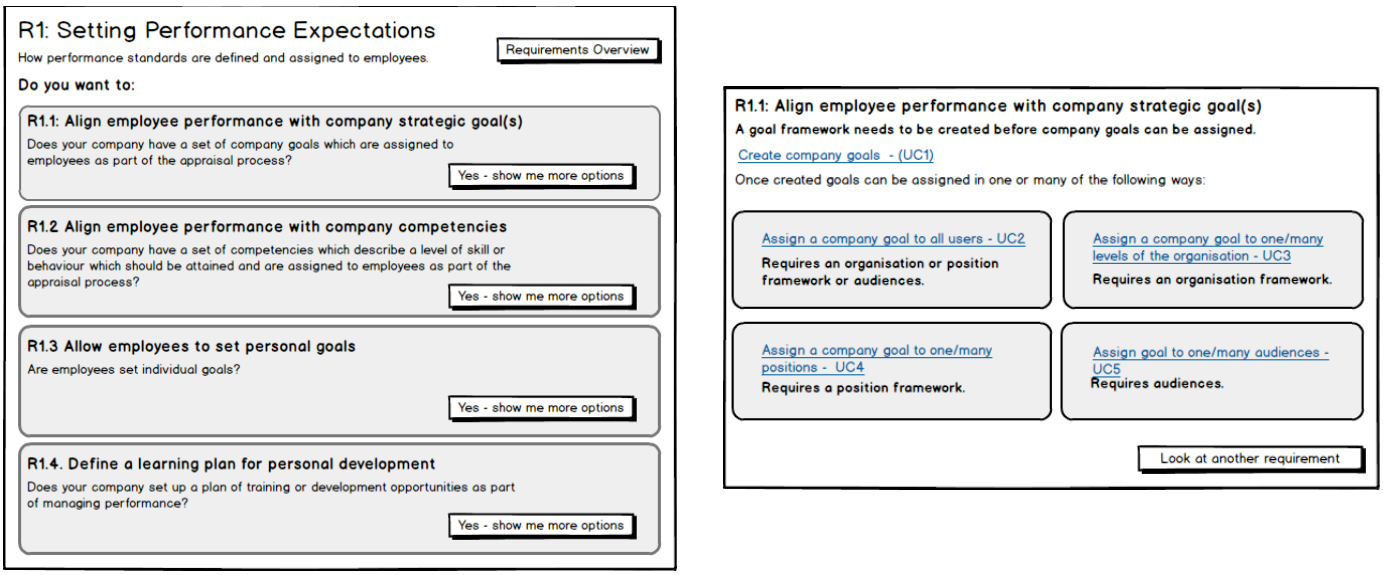


Figure 2: Two screenshots from the *ETHER* decision support tool that replaces demonstration of concrete software features. The left-hand side shows alternative capabilities to achieve *R1: Setting performance expectations*. The right-hand side depicts the lower-level capability *R1.1 Align employee performance* with company strategic goal(s) has been selected

E. Delivering a First Version of the ETHER Method

We developed a first complete version of the *ETHER* method, with descriptions of the steps and a prototype decision support software tool, and made it available to CGK for use by its solution architects. Experiences of use of the method are reported at length in the next section.

III. First Experiences with the *ETHER* Method

In order to evaluate the *ETHER* method, the first author organised sessions with 5 CGK customers to present the *performance management* feature available in Totara version 2.5. At the time of the evaluation, Totara 2.5 was scheduled for release to customers, so each session offered an early opportunity for each customer to preview the new feature and to start to make budgetary decisions regarding whether to upgrade or not their version of Totara. The 5 customers were recruited via the CGK sales team as customers who had previously shown interest in implementing appraisals and performance reviews into their learning management systems. They had very little knowledge of what capabilities the new *performance management* feature consisted of. Each session was run as either a face-to-face meeting or a webinar depending on the customer's preference. One solution architect facilitated each session. The first webinar session was undertaken as a baseline session in order to capture data about the current demonstration process used in CGK that data from the other sessions would be compared to. During the other 4 sessions, the solution architect applied the *ETHER* method. Three of these sessions were webinars, and 1 was conducted in a face-to-face meeting. Three different CGK solution architects undertook the 5 sessions with 5 different customers in 5 different sectors, and the durations of the sessions was from 45 to 143 minutes, see Table 1.

Each of the 5 sessions was successful, in that the solution architect was able to demonstrate software features of Totara in session 1, and step through all 4 steps of the *ETHER* method in the other 4 sessions to investigate alternative requirement-level capabilities. The one failure to apply the *ETHER* method as specified occurred in the first step of all 4 sessions, when the solution architects did not support their verbal summaries of each session with the planned written reports due to lack of resources to complete the tasks.

Interviews undertaken by the first author with solution architects after each session revealed some first perceived advantages and disadvantages of the *ETHER* method over the current software package demonstration process. After session 2, for example, the solution architect reported that: "*Yeah it did help more than doing a demo without it in the sense of I feel that after that amount of time I have a better idea of what they are trying to achieve then when I don't use something that is this structured and then do a Totara demo*". Some of the customer representatives also reported benefits, for example: "*.. opportunity to see the system at a high level in order to think about/initiate discussion around our own business requirements*". That said, after the third session, the solution architect stated his belief that some of the method steps were more useful than others: "*I find the walkthrough really, really useful for the first half but for the second half I feel that you don't have to use it as much, more as a prompt so when I'm like ah okay which bit do I need to focus on next I go back to it….*" The use of the glossary of terms to describe the software package was important during step 2 for building common ground: "*I think internally as they didn't have terminology for some of these things it was more an agreement of OK this is what it means. I'm not certain it made a great deal of difference when I was doing the demo. It is a useful thing to do and also introduces them to the terminology. It introduces them to the characters they will come across in the demo… you know here are a list of people we are dealing with – that sort of thing*". Establishing the glossary of package terms also appeared to improve understanding: "*Yes so, well I ran through the glossary, terminology thingy, they call them what we call them, but that was actually quite useful as they said we don't*

use that, we don't do that so that set my expectations for what I then covered, so what do you call competencies, we don't call them anything, we don't use them".

Table 1: Comparative data about the 5 solution architect-customer representative sessions

Session	Led by	Number of customer attendees	Customer industry	Format	Duration (mins)
Baseline	SA3	3	Training Provider	Webinar	60
Session 1	SA1	2	Catering / Facilities management	Webinar	45
Session 2	SA2	1	Newspaper /Magazine wholesaler	Webinar	143
Session 3	SA1	3	Property Development	Face to face	73
Session 4	SA2	2	Retail	Webinar	75

That said, the solution architect in the first session reported that it: "*.. took longer to do the demo than originally thought as you need to cover areas that they don't know about e.g. learning plans (which leads onto questions about programs) and competencies. These in themselves open further questions so it's easy to be drawn into other areas of Totara….. In general it covered everything they needed to know, but it was quite high level. Will definitely require follow up demos more targeted at their actual requirements*". The comment indicated that the solution architects might have preferred to combine use of the *ETHER* method and tool with the demonstration of software package features.

The use of the decision support tool for the method was perceived to aid the process, but the potential for further features was identified. One solution architect reported that: "*using the tool gives a nice structure to the demo although some parts raise questions about other areas that I covered later, so you need be able to answer questions early, or manage the demo to say it's coming later. Might be worth adding in additional links to jump around the structure to get to related areas so that the demo isn't so prescriptive in its flow and order*". Another solution architect identified the tool as useful in parts of the session: "*so actually probably a good high level comment that I have is this feature – this – this section – performance expectations was really good and really helped, but facilitating the appraisal process I didn't really use it for, the reason why was because this one is more about the consultancy of how we are going to do it, whereas facilitating the appraisal process was more you set up the form, you set up the stages, there is not so much consultancy*."

Overall, the feedback from these interviews revealed both the potential benefits and the limitations of the first version of the *ETHER* method. However, to investigate the effectiveness of the method in more detail, we audio-recorded the solution architect-customer representative dialogue in each session. This dialogue was the primary means of exposing requirements then communicating and aligning them with product capabilities in face-to-face meetings and webinars, therefore a first-hand analysis of the dialogue was expected to reveal new insights into use of the *ETHER* method. Each full audio recording was transcribed, and information from the screen captures from the decision support tool was described and added to the transcript. We then applied a simple thematic coding to each transcript segment. The set of codes generated from the research questions and reported in Table 2 was developed and applied to each transcript from each session.

Table 2: Thematic codes used to analyze the transcripts of the 5 solution architect-customer representative sessions

Code	Description
Requirement	An expression of capability and/or quality needed or wanted by the customer
Requirement-capability alignment	An expression of a requirement's alignment to one or more product capabilities
Requirement-capability non-alignment	An expression of a requirement's explicit non-alignment to one or more product capabilities
Customer understanding	An explicit expression of understanding of one or more package capabilities, features or qualities

To analyze the codified transcripts, the totals of instances of each type of coded segment were computed and compared across the 5 sessions. We then undertook a content analysis for each codified transcript segment to compute the totals of requirements. In the next sections, we report key results from this analysis of the data collected from the 5 sessions.

A. Exposed Requirements

First of all, we applied results from the contents analysis to calculate the totals of different atomic requirements that were articulated in each session. We also calculated the totals of these articulated requirements that were stated to be satisfied by the package capabilities, and the totals stated explicitly to be unsatisfied by these capabilities.

In the baseline session, the customer representatives and solution architect did not articulate any requirements in the one-hour session, a result consistent with previous CGK experiences. In contrast, in the 4 sessions during which the *ETHER* method was used, the customer representatives and solution architects articulated a total of 67 different atomic requirements – an average of almost 17 requirements per session, see Table 3.

Table 3: The totals of different atomic requirements articulated in each of the sessions, and the totals of these requirements explicitly matched to package capabilities, and explicitly not matched to package capabilities

Session	Totals of Articulated Atomic Requirements	Totals of Requirement Capability Alignment	Totals of Requirements Capability Non-Alignment
Baseline	0	0	0
Session 1	18	3	0
Session 2	29	8	4
Session 3	16	3	0
Session 4	4	2	1

That said, the sessions varied – session 2 led to the articulation of 29 requirements, whereas session 4 led to the articulation of only 4 requirements. Both of these sessions were ran by the same solution architect, but session 2 ran for twice as long as session 4. Of the total of 67 requirements, only 16 (24%) were aligned to package capabilities in the sessions, 5 (7%)

were recognised as unaligned to any package capabilities in the sessions, and the remaining 44 (69%) were not associated to any package capability. This result revealed that the sessions with the *ETHER* method did not manage to associate most customer requirements to the package capabilities.

In contrast, the *ETHER* method did appear to increase the number of customer requirements articulated, compared with its current software package demonstration process, even if most customer requirements were not explicitly identified as satisfied or not by the package capabilities. To explore possible reasons for this, we examined which steps of the *ETHER* method each requirement was articulated in.

B. Where and How Requirements were Exposed

The totals of the requirements that were articulated in the 4 steps of the *ETHER* method are reported in Table 4. The majority – 46 (69%) – were articulated in step 2 – building common ground – when the solution architects asked customer representatives to express their requirements. The remaining requirements were articulated first in step 3 – requirements that the product supports – revealing that the role of the decision support tool to explore alternative package capabilities and surface new requirements. No new requirements were articulated during introductions and reviews in steps 1 and 4.

We also undertook a simple content analysis of all of the 67 requirements that were articulated during the 4 sessions, to understand the nature of the requirements exposed by use of the *ETHER* method. To do this, we categorized each of the 67 requirements as a member of one of 3 possible types:

- *An independent requirement*: a customer representative states explicitly that the customer wants a capability, function, feature or quality, independently of any package capabilities, for example *we need to do x*, *if it could do x*, and *they would need x*;
- *A deficient business process*: a customer representative expresses what is needed in terms of deficiencies in the current process. Examples include: *so at the moment we do x*, to describe the process, and *we have x,* to describe an element of the process, such as a form or an element of the current system in place;
- *A requirement that is generated directly from package capabilities*: these types of requirements arise from the dialogue about package capabilities. The customer representative makes a statement about the capability and how they will use it, for example: *ok so I can have x*, to describe what the customer would like, and *so if the package does x, then we can do y.*

Table 4: Totals of different atomic requirements reported in each of the 4 steps of the ETHER method, in each session

Session	Step 1	Step 2	Step 3	Step 4
Session 1	0	12	6	0
Session 2	0	19	10	0
Session 3	0	12	4	0
Session 4	0	3	1	0
Total	**0**	**46**	**21**	**0**

Table 5 reports the total number of occurrences of each type of requirement in the 4 sessions.

Table 5: The totals of different atomic requirements, by type, reported in each session

Session	Independent requirement	Deficient business process	Requirement generated from package capabilities
Session 1	2	14	2
Session 2	6	19	4
Session 3	0	14	2
Session 4	1	3	0
Total	**9**	**50**	**8**

Most of the requirements, 50 of the 67, were expressed as deficiencies in current work processes to resolve, as this extract from session 2 demonstrates: *"At the moment we use a separate system…and basically we have an online scorecard where people can create online goals - the executees create their goals at the start of the financial year which is the 1st September so we're just beginning that process now. Those goals are cascaded to senior team and then they are cascaded down the organization and then individuals are asked to go in and amend the goals to make them more appropriate and applicable to themselves basically".* In this event, 4 requirements were exposed – for executees to create performance objectives for the company and assign objectives to the senior management team, for the senior management team to assign objectives to individual employees, and for individual to amend objectives.

In contrast, the customer representatives expressed many fewer requirements independent of the package capabilities, for example: *"We need to be mindful that that human element is going to be taken away we need some robust logic that says this equals that and therefore you get this document rather than their manager saying I'm not really sure",* and *"I'm wondering whether we could set a learner or a user up to create their own 360 forms so perhaps their manager approves it or something. That might be quite useful."* These new requirements specified the role for learners to create their own 360 forms and for managers to approve these forms.

More surprisingly, only 8 of the 69 requirements were generated directly from package capabilities. For example in session 2, the customer recognizes a requirement when the options for assigning a company goal are shown on screen: *"I think we'd want to do both really depending on who the learner is sometimes it would be learner driven sometimes it would be manager driven. The ideal is that all learners go out and manage objectives but I didn't that's reality so I think flexibility to do both please."* The customer generated this requirement from capabilities of the package, in contrast to software demonstrations in which the customer was not encouraged to compare capabilities directly to their business activities. However, such instances of requirements generation were few and far between during the sessions.

A qualitative analysis of the relevant transcript segments revealed that each statement of satisfaction between a requirement and a package capability emerged in one of two different ways. In the first way, the solution architect or one of the customer representatives would refer back to a requirement articulated in step 2 when common ground was established, then aligned or otherwise that requirement to one or more capabilities of the package, for example in session 1 the solution archi-

tect says: *"So the defining the appraisal is what we just run through, so it setting up those stages, setting up those pages to view, assigning appraisals as I just mentioned you can assign using audiences positions and organization hierarchies so we can get the right things to the right people. As you say you've got multiple forms so what you can do within the system is create multiple appraisals"*. Often the solution architects appeared to recall the customer requirements because they did not make external notes of customer requirements during the sessions.

The second way in which a statement of satisfaction was made was to align a new requirement directly to one or more package capabilities immediately in the session. Both the solution architects and customer representatives articulated compliance statements when recognizing that a new requirement that might align to a capabilities currently being described. For example, one of the customer representatives asked: "*is there a way of completing appraisals off-line at some point? For example they use down time whenever they can they may not always be on the Internet, they may be on a train or somewhere else and maybe not necessarily now, but in the future will there be some sort of functionality to do stuff off-line because some people have teams of about 50 people and it's such a huge task for them"*. The solution architect responded: "*Yeah yeah I know, I can definitely see why. Unfortunately no, as you have said the site is web-based so it does require the Internet…",* thereby articulating an explicit lack of alignment of that package capability to the requirement.

C. Customer Understanding

The coding of the transcript segments to reveal evidence of understanding package capabilities by customer representatives summarized in Table 6 revealed that the customers articulated all but one statement of understanding in step 3 exploring requirements supported by the package. In contrast, the customers did not make such statements during the other 2 steps of the method.

Table 6 - Where understanding occurred

Session	Step 2	Step 3	Steps 1 and 4
Session 1	1	8	0
Session 2	0	1	0
Session 3	0	4	0
Session 4	0	0	0
Total	**1**	**13**	**0**

Moreover, most of the statements of understanding were expressed as questions such as: *"hmmm, so we can have lots of different messages going out at different stages…?"* The statements were often prompted by information displayed on the screen shared by the solution architect and customer. For example, in session 1, the solution architect explained: *"…but what I can do is add a specific, those are the company goals, I can add in a goal from a manager",* to which one of the customer representatives replied: *"so those goals are added based on the audience those goals area added based on the audience automatically?"*. The solution architect then responded: *"yup, I think actually, this is where I may get a little bit vague. I think what it might be is when you… I think on this version when you go in, you have to manually add them in. My understanding is for the release version it will automatically pull them in for you"*.

To conclude, analysis of the dialogue transcripts revealed little direct evidence of customer understanding, although there was also little direct evidence of misunderstanding between each solution architect and customer representatives.

IV. Revisiting the Questions, and Lessons Learned

We reviewed the outcomes from this first use of the *ETHER* method against the 3 questions, presented at the start of the paper, to extract some first lessons learned for both CGK and for requirements research applied to implementing off-the-shelf software packages.

To answer the research question Q1, the results indicated that use of the *ETHER* method did expose more customer requirements on the implementation of the software package compared to CGK's current demonstration process. Both CGK's past experiences and the baseline session revealed that few requirements are exposed and documented during demonstration sessions. In contrast, the 4 *ETHER* sessions exposed a total of 67 customer requirements. In simple terms, the sessions revealed that replacing the software package implementation with decision support about requirements-level capabilities increased the exposure of requirements in the dialogue between supplier and customer. Although perhaps not the most surprising result, the result does indicate that the level of abstraction at which information is presented to customers has the potential to influence the discovery of and communication about requirements. However, most of the exposed requirements were expressed not as direct features or qualities that the implementation should have, but in terms of deficiencies in each customer's current business process that could be enhanced by the Totara package. The customer representatives expressed most of these requirements in a relaxed and natural way, perhaps in part because they expected the CGK solution architects to formalize the requirements afterwards on their behalf. Indeed, one interpretation of this behaviour was that the customer representatives were waiting to understand the package capabilities before seeking to formalize requirements further. For example, the customer who participated in Session 2 requested a follow-up requirements session before finally deciding to buy the *performance management* feature. One broader implication is that the *ETHER* method needs further guidance to extract requirements from deficiencies in the customer's current situation. We explore this implication further in the lessons learned reported at the end of this section.

To answer the research question Q2, the use of *ETHER* revealed only limited evidence of improved understanding of software package capabilities by the customers, compared to CGK's current software package demonstration process. One possible reason for this result is the good level of understanding that the Totara customers already had of these capabilities – customers in all 5 sessions reported that the sessions were '*very effective*' for ensuring understanding of the *performance management* feature.

Finally, to answer the research question Q3, the use of the *ETHER* method did not appear to allow the supplier and customer representatives to align customer requirements and software package capabilities more effectively, compared to the current software package demonstration process. Whilst there was an increase in the number of requirements that were both explicitly aligned to and not aligned to package capabili-

ties in *ETHER* sessions over the baseline session, this increase was not large, and most requirements that were exposed during the sessions with the *ETHER* method were not either explicitly aligned or unaligned with the package capabilities. The original design of the *ETHER* method assumed that customers would provide well defined and clearly stated business requirements at the start of each session. The result indicates that this did not happen. Instead, the method will need to provide more support for aligning requirements and capabilities.

From the results, we drew a number of simple lessons learned that can be applied first by both CGK and researchers seeking to align off-the-shelf software packages and requirements from customers more effectively. The first lesson is that supporting software package demonstrations with equivalent artifacts that describe the requirements-level capabilities of these packages can lead to the exposure of more customer requirements on packages. In this work we report the combined use of an adapted product variability model and explicit software decision support tool to walk customers through the capabilities of a system and the requirements that it supports. Of course, other types of artifact that depict goals and requirements which are reverse-engineered from software package features can also be used, but the focus of goal variability models on decision points made them an effective mechanism with which to focus on requirements-based decisions to make.

The second lesson is one that will be applied to future uses of *ETHER* in CGK, who will extend the method with one of more mechanisms with which to structure and record customer requirements more explicitly. This explicit record of each requirement will be needed, we believe, to facilitate the more effective alignment of requirements to the software package capabilities – one area of method use that was weak in the reported experiences. Existing requirements templates such as VOLERE [8] and writing guidelines such as EARS [6] can be used to guide customers to write and structure key requirements before each session and during step 2 when building common ground, although care will be needed to ensure that this change does not result in a long list of detailed requirements from the customer and a transactional approach similar to the Request for Proposal process. Moreover, the method will need to provide guidance to extract clear requirements statements from descriptions of business process deficiencies. The session dialogue between the solution architect and customer representatives can then extended with an explicit handshake process, in which the solution architect verbally presents new customer requirements back to the customer representatives during the session. Simple tabular or graphical representations can be used to externalize and encourage alignments to be specified between requirements and package capabilities. These representations can be supported digitally to allow solution architects to align emerging requirements with modeled package features.

Finally, a wider possible change is to extend the scope of the *ETHER* method beyond the sessions involving the solution architect and customer representatives. For example, rather than present the software package capabilities to customers during each session with the decision support tool, the goal variability model and/or decision support tool could be provided to customer representatives to familiarize themselves with the key decisions to make, prior to the session. Of course, presenting different package capability options to customers without expert guidance from suppliers would create risks, and necessitate additional documentation about the software package to be delivered.

We are looking forward to implementing these changes, and reporting on their use in the near future.

ACKNOWLEDGMENT

We acknowledge the support of both City&Guilds Kineo and their customers to undertake the evaluation studies reported in this paper.

REFERENCES

[1] Alves, C., Finkelstein, A., 2002. 'Challenges in COTS decision-making: a goal-driven requirements engineering perspective', in: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering. pp. 789–794.

[2] Alves, C., Franch, X., Carvallo, J.P., Finkelstein, A., 2005. 'Using goals and quality models to support the matching analysis during cots selection', in: COTS-Based Software Systems. Springer, pp. 146–156.

[3] Bühne, S., Lauenroth, K., Pohl, K., 2004. 'Why is it not sufficient to model requirements variability with feature models', in: Proceedings of the Workshop: Automotive Requirements Engineering (AURE'04), Co-Located at RE'04, Nagoya, Japan.

[4] Clark, H., 1996. 'Using language',. Cambridge University Press.

[5] Maiden N.A.M. & Ncube C., 1998, 'Acquiring Requirements for Commercial Off-The-Shelf Package Selection', IEEE Software, 15(2), 46-56.

[6] Mavin A., 2009, 'Easy Approach to Requirements Syntax (EARS)', Proceedings 17th IEEE Requirements Engineering Conference, IEEE Computer Society Press, 317-322.

[7] Natt och Dag J., Gervasi V., Brinkkemper S. & Regnell R., 2005, 'A Linguistic-Engineering Approach to Large-Scale Requirements Management', IEEE Software 22(1), 32-39.

[8] Robertson, S., Robertson, J., 2013. 'Mastering the requirements process: getting requirements right'., 3rd ed. ed. Addison-Wesley.

[9] Schmid, K., Santana de Almeida, E., 2013. 'Product Line Engineering'. Softw. IEEE 30, 24–30.