Commercial off-the-shelf software can save development time and money if you can find a package that meets your customer's needs. The authors propose a model for matching COTS product features with user requirements.

Acquiring COTS Software Selection Requirements

Neil A. Maiden and Cornelius Ncube, City University, London

Although procuring off-the-shelf systems from commercial suppliers is increasingly popular in our industry, fitting those systems to customer requirements remains problematic. The London Ambulance Service fiasco in 1992 exemplifies a system failure due, at least in part, to poor commercial off-the-shelf product selection. Following the system's introduction in October 1992, a number of social and technical problems arose. The entire system descended into chaos, was taken off-line, and the LAS reverted to the manual dispatching system. The use of COTS software, also known as component-based software engineering, introduces new problems for requirements engineers, including when to acquire new customer requirements and when to reduce the number of candidate products.

Requirements engineering acquires, models, and validates functional and nonfunctional requirements. System design specifies the functional and physical architectures as well as the hardware and software design to meet these requirements. Systems integration involves product acquisition, prototype integration, and integration testing. The integrated system is then evaluated against criteria that include risk and cost. Direction determines the process's high-level objectives during each iteration.

0740-7459/98/$10.00 © 1998 IEEE

However, most methods and software tools for this process support only systems design[2] and integration,[3] despite the often cited importance of requirements acquisition. Further, the requirements acquisition guidelines given tend to be too general. For example, John Dean and Mark Vidger propose acquiring a small number of high-level requirements prior to iterative and concurrent product evaluation and selection.[4] The lack of methods and software tools is all the more surprising given the new opportunities that component-based software engineering offers the requirements acquisition process. For example, stakeholders often have prior knowledge of candidate products during the requirements acquisition phase, so acquisition can focus on the requirements that can be used to best discriminate between competing COTS products, and products can be rejected as relevant new requirements are acquired.[5]

To support requirements acquisition for selecting commercial off-the-shelf products, we propose a method we used recently for selecting a complex COTS software system that had to comply with over 130 customer requirements. The lessons we learned from that experience refined our design of PORE (procurement-oriented requirements engineering), a template-based method for requirements acquisition. We report 11 of these lessons, with particular focus on the typical problems that arose and solutions to avoid them in the future. These solutions, we believe, extend state-of-the-art requirements acquisition techniques to the component-based software engineering process.

Acquiring COTS Product Requirements

Our customer for the project whose lessons learned helped shape PORE was the UK Ministry of Defence's Procurement Executive. The PE wanted methods and software tools for managing a new naval platform's requirements; the platform would take an estimated 20 years to develop. A team of four members, including both of us, acquired requirements and recommended commercial requirements engineering methods and software tools for trial by the PE.

The team had 11 weeks to make a recommendation. We decided that, where possible, we would adhere to commercial product selection procedures. We did not use research ideas or techniques from disciplines outside the fields of systems or requirements engineering.

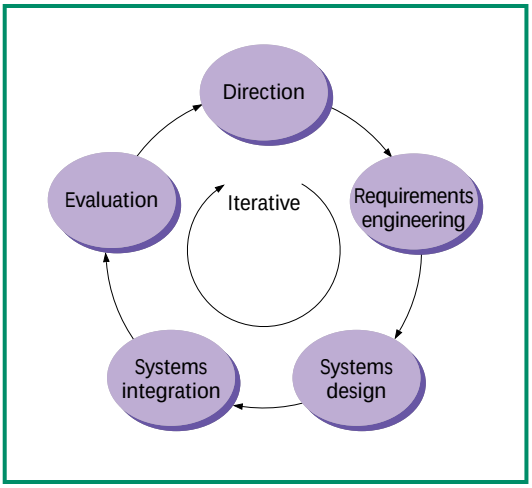


FIGURE 1. The iterative system development process for COTS and custom-made components.

We acquired requirements from both documents and stakeholders. Five meetings with between six and 10 stakeholders took place over a three-week period. We divided each meeting into a review of the current requirements document and acquisition of new requirements from the stakeholders. The final requirements document contained 133 atomic requirement statements.

To save time, in parallel with requirements acquisition we did market research to identify candidate products. We transformed an initial version of the requirements document into a questionnaire and sent it to more than 30 candidate suppliers to determine their products' coverage. Using supplier responses to this questionnaire, we produced a shortlist of six candidate products. Next, for product evaluation we developed a set of 35 complex test cases from the final requirements document. Five of the six shortlisted suppliers demonstrated their product against the 35 test cases. During each evaluation, three members of the team recorded product compliance with each requirement, using both quantitative scores and qualitative comments. After each evaluation, we conferred and agreed on the final product–requirement compliance scores. We then investigated these scores using the weighted scoring method. The main stakeholder weighted each customer requirement to indicate its importance. The team multiplied product–requirement compliance scores by these weightings during the calculations to select the best-fit products. As a result

of this process, we recommended trial use of two requirements engineering software tools.

Lessons Learned

Although the customer was satisfied with our recommendations, we found the requirements acquisition and product selection processes problematic. We encountered 11 major problems and propose here possible solutions to each of them. These solutions, as well as other measures not reported here, guided our design of PORE.

Lesson 1

Acquire in more detail those requirements that enable effective discrimination between products.

Problem. We spent too much time acquiring and modeling the requirements met by all five evaluated products. Most of these requirements did not enable us to discriminate between the products. In contrast, we did not model other requirements in sufficient detail to enable effective product selection. For example, requirements about product compatibility with Microsoft's Word and Access products were less detailed but more important for product selection.

Solutions. Requirements acquisition and product evaluation should be both iterative and concurrent. The selection team can then become familiar with requirements and products at the same time, thus making both requirements acquisition and market research more flexible and responsive. This can be achieved using a range of techniques. Card sorts, for example, are simple to use and can acquire requirements that discriminate between products.[6] The requirements engineer writes candidate product names on 3" × 5" cards and asks stakeholders to use the cards to sort the products into categories. Criteria for these sorts, such as "compatible with Microsoft Word," indicate customer requirements that discriminate between products. Product categories, such as "compatible" and "not compatible," indicate product compliance with these requirements. A useful variation, card sort triage, requires the stakeholder to describe the similarities between two products and their common differences with a third product. Card sort techniques also fit well with laddering,[7] in which the requirements engineer asks the stakeholder to describe common categories and classes of products and their features to discover important but nondiscriminating customer requirements, thus avoiding time-consuming acquisition of less important requirements. Discriminating requirements then provide a starting point for more thorough requirements acquisition using other techniques.[6]

Making requirements measurable is easier said than done.

Lesson 2

Requirements must be as measurable as possible to enable effective product selection.

Problem. We had difficulty measuring product–requirement compliance for most requirements, which made product selection problematic. This difficulty stemmed in part from the requirements being unverifiable because their fit criteria were not expressed as logical expressions or quantifiable tests recommended by current commercial requirements standards.

Solution. Making requirements measurable is easier said than done. We advise predetermining how the requirements will be used during product selection—for example, as questions in questionnaires sent to suppliers or as evaluation test cases during online product demonstrations—then tailoring the verifiable fit criteria for these requirements accordingly. However, you cannot always do this effectively for large numbers of customer requirements, so focus on requirements that enable effective product discrimination. An iterative approach is essential for success: use the techniques reported in Lesson 1 to determine requirements that discriminate between products, define the fit criteria for these requirements, then reevaluate product–requirement compliance using the criteria. Be patient: several iterations might be needed to determine precise and measurable fit criteria for the requirement and evaluate the degree of product compliance to it.

Solution. You can also use contrived acquisition techniques to acquire quantitative scores for product requirement compliance. One such technique is repertory grid analysis, in which you ask stakeholders for attributes applicable to a set of entities and values for cells in an entity-attribute matrix. This technique's advantages include representing requirements in a standardized, quantifiable format. Such a format can be used for statistical analyses that serve as a basis for justifying product selection decisions.

Lesson 3

Use software prototypes to aid generation of test cases for product evaluation.

Problem. Without prior knowledge about candidate products or prior extensive experience generating test cases, we found generation of measurable test cases difficult.

Solution. We used a software prototype to help us generate test cases. Fortunately, we had access to an in-house prototype requirements management tool built with Microsoft Access. We did a mock evaluation of this prototype using first-draft test cases as if the prototype were a candidate product. This led to some refinement of the test cases themselves, in part to make them more measurable. Using the prototype, we could also determine the correct responses to complex database queries used during product evaluation. However, on most projects, such a prototype must be developed. Rather than generate a single, integrated prototype, a more cost-effective approach might be to generate several. These smaller, partial prototypes or concept demonstrators would contain the desirable but discriminating product features with which to develop verifiable fit criteria.

Lesson 4

Structure the requirements so that test case formulation becomes easier.

Problem. The hierarchical structure of the requirements proved incompatible with the sequential structure of the test cases. This made test case generation difficult. For example, we included the requirement that the product be configurable to customer needs in all 35 test cases to evaluate how configurable all of the functions of each product were. However, this weakened the link between requirements and test cases and made test case management more difficult.

Solution. You should acquire requirements using use cases and scenarios that make the requirements more amenable to test case generation. For example, Ian Graham's Somatik approach proposes seamless decomposition of goals into tasks that achieve these goals, then generation of use cases that are equivalence classes of task scripts, and scenarios that are equivalence classes of use cases.[8] Somatik's seamless transformation is both practical and effective. Graham reports its successful use on over 20 projects at Swiss Bank Corporation. It even generates simple software prototypes that can be useful during test case generation, which also provides at least a partial solution to the problems reported in Lesson 3.

Lesson 5

The scope of the product under evaluation is difficult to define.

Problem. Requirements management tools are complex and depend on other products, such as database management systems. We had to assess these other products to undertake a complete evaluation, which in turn made our assignment more complex and time consuming.

Solution. Loral's revision of the spiral process for COTS product selection[9] recognizes the need to select cornerstone products before integrating other products around them. This is a sensible suggestion. However, you still need guidance for detecting and selecting these products. Checklists can tell you if customer requirements are essential, stable, and urgent. They can also track if each candidate product

- will be used for a long period,
- meets essential rather than nonessential requirements,
- needs modification,
- is currently available, and
- adheres to current product standards.

You should determine important dependencies between products: for example, "Can this function be achieved without additional products?" If not, ask follow-on questions such as "Are the additional products available with the core product?" "Where are the additional products available from?" and "How much configuration of these products is needed?" We have since embedded such questions in PORE's method templates to encourage the requirements engineer to ask the right question at the right time.

The hierarchical structure of the requirements proved incompatible with the sequential structure of the test cases.

Lesson 6

Stakeholder representatives should be present during product evaluation.

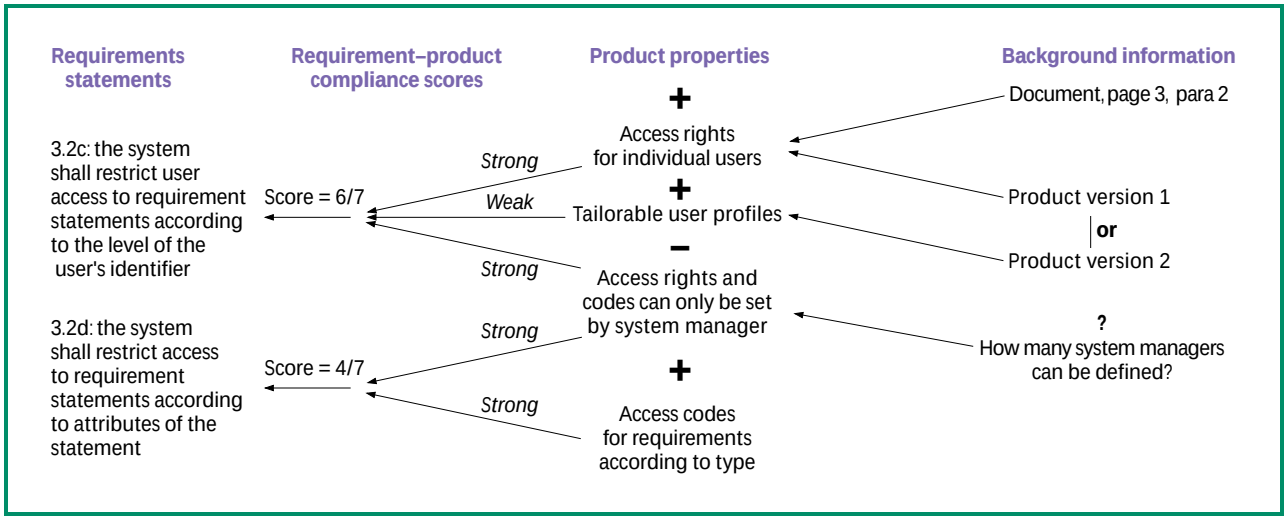


Figure 2. This sample design rationale, tailored for product evaluation, lets the user record product–requirement compliance scores and their reasons, as well as references to other scores, requirements, and product properties. For example, the product received a score of 6 out of 7 for compliance to requirement 3.2c. Positive reasons for this score include the product's provision of access rights for individual users and tailorable user profiles. However, these access rights and codes can only be set by the system manager. Further, different product properties are available from two different versions of the product. The OR link between them indicates that only one of the two versions can be selected. This, in turn, influences both the selection to be made and the techniques used to help make the decision.

Problem. We sometimes needed to ask detailed questions that required information about the problem domain. We did not always acquire such information from stakeholders prior to the evaluation, or recall it during the evaluation.

Solution. To ensure that we could get answers to our more detailed questions, we arranged for a stakeholder representative with a good understanding of the problem domain to be available throughout each evaluation.

Lesson 7

You need techniques to record information during product evaluation.

Problem. Numerous requirements made product–requirement compliance a complex task. We made over 1,500 compliance decisions during 18 hours of product evaluation. We did not, however, use techniques to record the rationale for these decisions. This made it difficult for us to agree on product–requirement compliance scores after each evaluation because we had not recorded all the reasons for all the scores.

Solution. During the evaluation, you should record the rationale for product–requirement compliance scores using the design rationale techniques described by Tom Moran and Jack Carroll.[10] Figure 2 demonstrates these techniques. To save time, before evaluation begins you can enter into the tool each product–requirement compliance test, the properties of each product, and the requirement statements. We also recommend using a scribe who is independent of the evaluation process to record rationale statements during each evaluation. Videotapes of the evaluation session offer another, objective means of recording information. Further, tapes can be linked to design rationale diagrams by including a time reference to when a product requirement evaluation took place.

Solution. Detecting dependencies between product features is essential for effective product selection. You can do this by extending rationale diagrams to show key dependencies. Figure 2 shows that two product features (access rights for individual users, tailorable user profiles) belong to two versions of the same product. Since you often have to choose one version or the other, we added a logical operator OR to the diagram. Such dependencies have an important impact on the selection of decision-making techniques, as we report in Lesson 8.

Solution. You can also use feature analysis tech-

niques from nonsoftware product procurement[11] to obtain product–requirement compliance scores in a more systematic way. The Desmet approach proposes taxonomies of desirable features, derived from Desmet team members' expertise, to draw on during product selection. The Desmet approach treats each product feature as independent of any other; this is not, however, often the case for even simple software products, so use this approach with care.

Lesson 8

Weighting requirements for product selection can be problematic.

Problem. One stakeholder weighted all the customer requirements using simple percentage scores. However, these weightings were sometimes inconsistent and led to confusion about which were the most essential customer requirements.

Solution. Use more sophisticated requirements weighting methods such as multicriteria decision-making techniques. One such technique, the Analytic Hierarchy Process, has received some interest in the requirements engineering and software engineering communities. AHP was developed for multiple-criteria decision-making situations. It supports hierarchical structures that are common when modeling system requirements. You obtain rankings and weightings through paired comparisons of requirement statements that are converted to normalized rankings using the eigenvalue method. This means that the relative rankings of alternatives are presented in ratio scale values that total one.[12]

However, one strong assumption for using AHP is that all criteria are independent. This was not the case in our example, nor is it for most system requirements. Thus results from AHP analyses might be unreliable when dependencies between system requirements exist. Outranking methods offer one alternative solution. Such methods seek to enrich dominance relations between criteria without having to make the strong assumptions needed for the AHP. There are several formal definitions of outranking methods, but all involve building the outranking relation first, then exploiting it with regard to the current problem. We are currently undertaking case studies to explore the effectiveness of the outranking method for software product selection and will incorporate our results into the advice given in the PORE method.

Lesson 9

Weighting requirements for product selection can be time consuming.

Problem. Thorough use of AHP[12] would have required an estimated 42,000+ individual paired-comparison scores. Clearly, time constraints on product selection made this impossible.

Solution. You should use the AHP for specific purposes only. To avoid a combinatorial explosion in the number of individual paired-compliance scores to be calculated, use multicriteria decision-making techniques to weight customer requirements, but not to determine product compliance to these requirements. When doing this, only use the AHP if underlying assumptions for its use are met; that is, when there are few if any interdependencies between customer requirements or between product features. Furthermore, to ensure its cost-effective use, use the AHP when other, simpler decision-making techniques are inappropriate or when you need a more sensitive analysis to resolve disagreements or to support critical decisions about discriminating product features (Lesson 1) or cornerstone products (Lesson 5).

Lesson 10

Product evaluation is a team game, so treat it as such.

Problem. Biases are always possible when scoring product–requirement compliance. One post-evaluation analysis of individual and agreed product–requirement compliance scores revealed a trend toward agreement with one team member more than the other two, due in part to the members' different experiences.

Biases are always possible when scoring product–requirement compliance.

Solution. You can use reference products with which all participants are familiar, such as commercial products or, as in our case, the in-house requirements engineering tool. Such products let the evaluation team calibrate product requirement compliance scores before undertaking each individual evaluation test case. However, no one reference product will always have all of the product features to be evaluated, so be prepared to use several reference products to calibrate

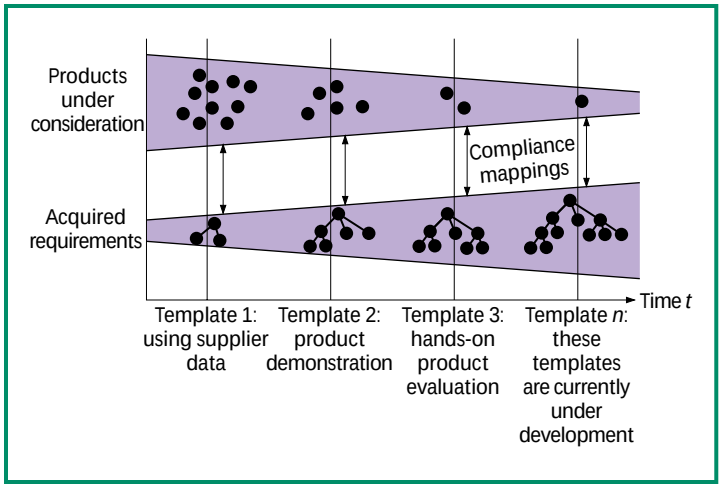


Figure 3. Outline of the PORE process model for product selection.

scores. This does, however, mean that the evaluation will often take more time. Card-sort triage techniques from Lesson 1 can be adapted to ask for quantitative measures of similarity and difference between two candidate products and the reference product. You can also include such reference products in design rationale diagrams (Lesson 7) to record the rationale for product–requirement compliance scores.

Lesson 11

Beware of the supplier's sales pitch and focus on the product.

Problem. A requirements management tool is a complex COTS product. To evaluate it properly, you need to see an effective demonstration by supplier representatives. However, in our experience, the quality of such demonstrations varied considerably across products. Of course, the law of the marketplace suggests that the supplier ultimately gets what it deserves—but this does not ensure that, before that happens, you will be presented with the accurate information necessary to purchase the product most compliant with your needs.

Solution. We chose to stick to the script imposed with the test cases and ask the same questions of all the supplier representatives. Our flexible follow-up questions then coaxed additional information out of these representatives.

The product selection exercise also gave rise to other lessons, problems, and solutions. We brought all the lessons and solutions together in the integrated, template-based PORE method.

The PORE Model

PORE integrates techniques for requirements acquisition and product selection with process guidance for choosing and using each technique. It draws on techniques from several different disciplines, including

- *knowledge engineering* techniques such as card sorting and laddering,[7] which are useful when acquiring information about categories of products, suppliers, and procurement contracts, as well as hierarchical information about product properties and the requirements themselves (see Lessons 1 and 2);
- *feature analysis* techniques[11] to aid when scoring the compliance of each product to each requirement (see Lesson 7);
- *Multicriteria decision making* (MCDM) techniques[12] and outranking methods to aid decision making during the complex product ranking and selection process (see Lesson 8); and
- *design rationale* techniques to record and aid this decision-making process (see Lesson 7).

PORE also includes guidelines for designing product evaluation test cases (see Lessons 3, 4, and 5) and for organizing effective evaluation sessions (see Lessons 6 and 11). PORE presents these techniques and guidelines in a series of templates for requirements acquisition, product modeling, and product selection at different stages in the product selection process.

PORE templates

PORE supports iterative requirements acquisition and product selection and rejection until one or more products comply with a sufficient number of customer requirements. It divides this process into stages and, in the first version, provides the three templates shown in Figure 3 for three key stages of the process:

- Template 1 provides guidance when acquiring the essential customer requirements and product information necessary to select and reject products as a result of supplier-given information.
- Template 2 provides guidance when acquiring the customer requirements and product information necessary to select and reject products from supplier-led demonstrations using test cases for individual requirements.
- Template 3 provides guidance in acquiring customer requirements and product information to select and reject products as a result of customer-led product exploration.

Each template defines the product information and customer requirements to be acquired as well as the techniques for acquiring this information and making decisions about it. At the beginning of the process there are few customer requirements, but many candidate products. As products are rejected, customer requirements increase and candidate products decrease.

The iterative nature of the requirements acquisition and product selection processes means that each template might be used several times during a product selection process. A detailed examination of the first two templates follows. The third template will be available soon at our Web site.

Template 1

You use this template during the early stages of requirements acquisition and product selection, when the evaluation team relies on supplier data in sales brochures, technical documents, telephone conversations, responses to questionnaires, Web site information, and internal or public market analyses. Due to a lack of detailed and accurate data, the requirements engineers should be wary and prepared to sometimes backtrack on selection decisions made earlier if important new information becomes available. PORE recommends the use of simple criteria for product selection, which require quantitative information about products (such as how many concurrent users the product supports) and Boolean responses to product–requirement compliance questions (such as if the product is compatible with Microsoft Word). Figure 4 shows a segment of the template. It outlines the types of product information and customer requirements to acquire, and simple techniques to use. The full template for this stage includes more techniques, guidelines for their use, and simple frames for describing product information and customer requirements of different types.

Template 2

It is common to have supplier-led product demonstrations during product selection. Such demonstrations are often the first chance for the evaluation team to undertake more complex product–requirement compliance tests. This template encourages the requirements engineer to explore product compliance with individual requirements. Effective preparation of the product evaluation tests is critical. The template places more emphasis on technique use than on the information to be acquired. Figure 5 shows part of the template. Again, the full template for this stage is much more complex. We are currently extending the two templates shown with more prescriptive process guidance and developing Template 3 for subsequent hands-on product evaluation.

Developing and Evaluating PORE

Component-based software engineering has yet to exert its full influence on the requirements engineering process, but publications on the topic[4,13] have begun to appear. One consequence of this influence is that complete requirement specification is not always needed, a fact recognized by Anthony Finkelstein and colleagues.[5] Rather, it is sufficient to acquire those requirements that discriminate between the candidate products, then use the selected product as a working prototype for more detailed requirements acquisition. To meet the new intellectual challenges that arise from these changes, academic research must take new directions to underpin the design of new methods and software tools.

PORE proposes a model of product–requirement compliance to provide a sound theoretical basis for technique selection and use.

PORE's product–requirement compliance model

PORE proposes a model of product–requirement compliance to provide a sound theoretical basis for technique selection and use. A prerequisite for effective product selection is compliance between one or more features of each candidate product and one or more customer requirements. This compliance is, in essence, a relationship between a problem and a potential solution to that problem. To do this effectively, the requirements engineer must model not only customer requirements but also each software product. You can choose from many requirements modeling languages to model customer requirements, but few such languages are available for modeling software products. Yet possible exceptions[14] do exist. The PORE model lets software engineers model candidate software products in the same way that requirements models let

INFORMATION AND REQUIREMENTS TO ACQUIRE:

• BASIC PRODUCT AND SUPPLIER

• TECHNICAL PRODUCT FEATURES (e.g. current version number and period since last release);

• TECHNICAL SUPPORT ARRANGEMENTS for the product (e.g. licensing arrangements and cost, cost for technical support, training cost and availability, run-time fees, is source code available, extra vendor support provided and policies on upgrades and fixes);

• HISTORICAL INFORMATION about the product and supplier (e.g. how long the supplier has been in business, how long has the product been available, supplier annual turnover and customer base, number of products sold, number of employees, supplier references, track record in the business sector);

• ESSENTIAL FUNCTIONAL USER REQUIREMENTS, acquired from stakeholders and derived from candidate products.

TECHNIQUE SEQUENCE TO USE:

1. GATHER PRODUCT INFORMATION: read product documentation to gather basic product information;

2. ACQUIRE CUSTOMER REQUIREMENTS: acquire first-pass essential customer requirements using simple techniques such as brainstorming and interviewing (Maiden & Rugg 1996), if possible without reference to candidate products. Acquire functional rather than non-functional requirements, since products are easier to evaluate for functional requirements at this stage;

3. DEVELOP A QUESTIONNAIRE to ask each supplier how much this product is compliant with each of these essential user requirements. Also use this questionnaire to provide basic supplier and product information. Design the questionnaire to elicit sufficient information without it becoming too long and difficult to complete. Distribute it to all suppliers, set a deadline for replies and receive responses. Responses from suppliers should be quantitative, enumerative or boolean so that analysis of responses is simpler;

4. GET TO KNOW THE CANDIDATE PRODUCTS: familiarise yourselves with products using demonstration copies;

5. EVALUATE QUESTIONNAIRE RESPONSES to reject products which are non-compliant with essential customer requirements;

6. DISCOVER MORE CUSTOMER REQUIREMENTS from product information elicited from questionnaire responses. Often candidate products have desirable properties not discovered during requirements acquisition. These requirements should be explored with the customer at this stage. Use techniques such as structured interviews, prototype walkthroughs using product demonstration copies and scenarios of the use of the discovered requirement;

7. ACQUIRE MORE CUSTOMER REQUIREMENTS which enable discrimination between products (lesson 2). The template proposes iterative use of: (i) card sorts to acquire requirements which enable discrimination between products: (ii) rejection of products due to poor product–requirement compliance;

8. ACQUIRE DETAILED CUSTOMER REQUIREMENTS using scenarios as a basis for designing test cases for product evaluation using Template-2 (lesson 4).

DECISION-MAKING TECHNIQUES TO USE:

Decision-making at this stage is straightforward. However record rationale for product acceptance and rejection using simple decision tables and, when more complex, design rationale techniques.

Figure 4. Part of the template for product selection using supplier data.

software engineers model requirements.

Commercial software products often interact with users during routine tasks such as updating a requirement statement. They also have internal system functions such as checking requirement compliance with a standard, and have increasingly complex system architectures to support these tasks and functions. PORE draws on the techniques of task modeling from human-computer interaction, functional modeling from software engineering, and architecture modeling from system design[2] to model a software product at these three levels. However, the model's most important feature is the inclusion of complex interdependencies between user interactions, internal system functions, and architectures. Such a model provides the analytic basis for modeling critical product dependencies, understanding how a software product works, and reasoning about possible changes to this product. The model also provides the theoretical basis for

TO DO BEFORE THE DEMONSTRATION SESSION:

1. DEVELOP SIMPLE WORKING PROTOTYPES OF THE REQUIRED SYSTEM to discover and acquire further customer requirements prior to product evaluation. Use the prototype to improve and design of test cases for product evaluation (lesson 3);

2. HAVE STAKEHOLDER REPRESENTATIVES PRESENT during each demonstration to suggest previously-unforeseen requirements or to provide important domain information (lesson 6);

3. WORK WITH STAKEHOLDERS TO WEIGHT CUSTOMER REQUIREMENTS. If requirements are hierarchical use the AHP (lessons 8 & 9) on small, self-contained clusters of requirements with few dependencies to other requirements. This will avoid an exponential increase in the number of weighting decisions to be made and ensure the suitability of the AHP;

4. MAKE COMMERCIAL SOFTWARE TOOLS AVAILABLE, for example Saaty's Expert Choice and Karlsson & Ryan's tool, to calculate requirement weightings with the AHP (lesson 8);

5. PROVIDE EFFECTIVE UNITS OF MEASURE for product–requirement compliance scores through iterative refinement and evaluation of verifiable fit criteria for requirements which discriminate between products (lesson 2).

DURING EACH DEMONSTRATION SESSION:

6. ASK QUESTIONS ABOUT THE PRODUCT to determine cornerstone products first (lesson 5);

7. ONLY ALLOCATE COMPLIANCE SCORES IF THE PRODUCT PROPERTIES ARE DEMONSTRATED. Do not score unsubstantiated claims about the product (lesson 11);

8. If it is difficult to score for compliance USE REFERENCE MODELS (lesson 11). A reference model describes well-known, prototypical properties of a product and exemplar compliance scores for common, everyday tasks;

9. RECORD DECISIONS BEHIND COMPLIANCE SCORES using video and commercial design rationale software tools (lesson 7). Have an independent scribe record these rationale during the evaluation. Time-stamp each product–requirement compliance feature so that the rationale can be linked to the video record.

AFTER EACH DEMONSTRATION SESSION:

10. ACQUIRE MORE CUSTOMER REQUIREMENTS using different forms of the card sorting technique (lesson 2). One example is to ask stakeholders to grade the degree of compliance of products (cards) to requirements (categories). These grades can be quantitative (e.g. 0-7) or qualitative (good, average or poor fit). Also use triage sorting techniques described in lesson 1;

11. USE LADDERING TECHNIQUES to discover further important but non-discriminating customer requirements.

Figure 5. Part of the template for use during a supplier-driven product demonstration.

choosing the most suitable multicriteria decision-making technique for product selection. We continue to develop and refine PORE's product model.

Once a product model is available, you can define product–requirement compliance. This calls for new advances in similarity-based reasoning techniques. Recent advances in computational mechanisms for analogical reasoning in artificial intelligence[15] should extend existing research in requirements reuse and conflict detection in viewpoints to address problems of product requirement mapping. Research into extending these computational mechanisms is also ongoing. We envisage that such a mechanism will be an important component of a software tool supporting the PORE method.

PORE's practical development

Academic research is progressing in parallel with practical development of the PORE method. The first full version of PORE's templates will soon be made available through the Internet from http://www.soi.city.ac.uk/pore/welcome.html. The site will also include facilities for capturing feedback on the PORE method from practitioners who use it and more templates for supporting different requirements acquisition and product selection activities. We will be updating the PORE model to include lessons learned from using the current partial method to evaluate product selection by two commercial banks. We also envisage software tool support for PORE that integrates requirements acquisition, product modeling, and product selection functions.

The experiences we've reported indicate a paradigm shift with significant implications for software engineering practice and research. First, as component-based software engineering becomes more widespread, stakeholders are more

likely to express customer requirements in the form, "I want one like that product." Software products, and indeed software components and patterns, will provide a basis for a *lingua franca* that can communicate a large number of implicit customer requirements already operationalized in off-the-shelf software products. Consequently, requirement specifications must be sufficient to enable effective product selection rather than complete with respect to the user's needs. This provides one of the greatest challenges for software engineering researchers and vendors in the near future. ❖

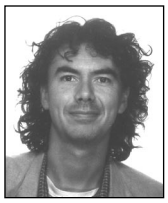
Acknowledgments

We thank Systems Engineering and Assessment Ltd. and, particularly, Brian Baker for his contributions. We also thank Roger Gibb of the Procurement Executive for all his work and time and all the suppliers who participated in the process. Cornelius Ncube is funded by a research studentship from City University's School of Informatics. Some of the material in this article, particularly the Lessons Learned section, appeared in condensed form in the December 1997 issue of *Communications of the ACM*.

References

1. V. Stavridou, "COTS, Integration and Critical Systems," *IEE Digest*, 97/013, Jan. 1997.
2. D. Garlan , R. Allen, and X. Ockerbloom, "Architectural Mismatch or Why It's Hard to Build Systems Out of Existing Parts," *Proc. 17th Int'l Conf. Software Eng.*, IEEE Comp. Soc. Press, Los Alamitos, Calif., 1995, pp. 179-185.
3. M.R. Vidger, M.W. Gentleman, and J. Dean, "COTS Software Integration: State of the Art," NRC-CNRC Report , National Research Council, Canada, Jan. 1996.
4. J. Dean and M.R. Vidger, "System Implementation Using Commercial Off-The-Shelf Software," NRC Report No. 40173, National Research Council, Canada, 1997
5. A.C.W. Finkelstein, G. Spanoudakis, and M. Ryan, "Software Package Requirements and Procurement," *Proc. 8th Int'l Workshop Software Specification and Design,* IEEE Comp. Soc. Press, Los Alamitos, Calif., 1996, pp.141-145.
6. N.A.M. Maiden and G. Rugg, "ACRE: Selecting Methods for Requirements Acquisition," *Software Eng. J.*, Vol. 11, No. 3, 1996, pp. 183-192.
7. G. Rugg and P. McGeorge, "Laddering," *Expert Systems*, Vol. 12, No. 4, 1995, pp. 339-346.
8. I. Graham, "Task Scripts, Use Cases and Scenarios in Object-Oriented Analysis," *Object-Oriented Systems 3*, 1996, pp.123-142.
9. N. Walters, "Systems Architecture and COTS Integration," *Proc. SEI/MCC Symp. Use of COTS in Sys. Int.*, Software Engineering Institute Special Report CMU/SEI-95-SR-007, SEI, Carnegie Mellon University, Mass., 1995.
10. T.P. Moran and J.M. Carroll, *Design Rationale: Concepts, Techniques, and Use*, Lawrence Erlbaum Assoc., Hillsdale, N.J., 1996.
11. B. Kitchenham and L. Jones, "Evaluating Software Engineering Methods and Tools: Part 5, The Influence Of Human Factors," *Software Engineering Notes,* Vol. 22, No. 1, 1997.
12. T.L. Saaty, *The Analytic Hierarchy Process*, McGraw-Hill, New York, 1990.
13. C. Potts, "Invented Requirements and Imagined Customers: Requirements Engineering for Off-The-Shelf Products," *Proc. 2nd Int'l Symp. Req. Eng.*, IEEE Comp. Soc. Press, 1995, pp. 128-130.
14. T.R.G. Green and D. Benyon, "The Skull Beneath the Skin: Entity-Relationship Models of Information Artefacts," *Int'l J. Human-Computer Studies,* Vol. 44, No. 6, 1996, pp. 801-828.
15. R.P. Hall, "Computational Approaches to Analogical Reasoning: A Comparative Analysis," *Artificial Intelligence 39*, 1989, pp. 39-120.

About the Authors



Neil Maiden is a senior lecturer in the Centre for Human-Computer Interface Design, a research center in City University's School of Informatics. His research interests include frameworks for requirements acquisition and negotiation, scenario-based systems development, component-based software engineering, requirements reuse, and more effective transfer of academic research results into software engineering practice. He has numerous journal and conference publications.

Maiden received a BA in business computing from the University of Sunderland and a PhD in computer science from City University, London. He is an affiliate member of the IEEE Computer Society, a member of ACM, and a member of the British Computer Society as well as its Software Reuse Specialist Group treasurer and co-founder and treasurer of its Requirements Engineering Specialist Group.

Cornelius Ncube is a PhD candidate in the Centre for Human-Computer Interaction Design at City University. His research interests include requirements acquisition methods, techniques, and software tools for component-based systems engineering, and in particular for software product evaluation and selection.

Ncube received a BSc in computer science from Brunel University and an MSc in software engineering from City University, London. He is a member of the International Council on Systems Engineering and the British Computer Society's Requirements Engineering Specialist Group.

Address questions about this article to Maiden at Centre for HCI Design, City University, Northampton Square, London EC1V OGB, UK; N.A.M.Maiden@city.ac.uk; http://www.soi.city.ac.uk/~cc559/info.html; or to Ncube at C.Ncube@soi.city.ac.uk; http://www.soi.city.ac.uk/homes/dg571/connie.html.