Software Package Requirements & Procurement

Anthony Finkelstein & George Spanoudakis
Department of Computer Science, City University, UK
{acwf, gespan@cs.city.ac.uk}

Mark Ryan
School of Computer Science, University of Birmingham, UK
{m.d.ryan@cs.bham.ac.uk}

Abstract

This paper outlines the problems of specifying requirements and deploying these requirements in the procurement of software packages. Despite the fact that software construction de novo is the exception rather than the rule, little or no support for the task of formulating requirements to support assessment and selection among existing software packages has been developed. We analyse the problems arising in this process and review related work. We outline the key components of a programme of research in this area.

Motivation

It is increasingly being recognised that software construction de novo is the exception rather than the rule. The norm for most organisations (including defence procurement agencies, formerly the bastion of purpose-built software) is purchase of commercial off-the-shelf "package" software. In some cases this software is modified or extended, but in most cases the user organisations accept the software "as is". There is an observable trend in business and commerce in which purpose built software is replaced by software packages acquired, often on a pay-by-use basis, over a network and from a diverse supplier base.

Much attention has been given to providing methods and support tools for obtaining purpose built software - acquiring requirements from users, constructing specifications from the requirements, deriving properties of the specifications and ultimately, deriving programs which satisfy the specifications - little or no systematic (let alone soundly based) support exists to help the software user formulate their requirements and procure software packages that meet their requirements.

Our interest in this problem is drawn from experience gained in our work with a small specialist library, with an existing manual catalogue and accession system, who want to obtain software support for their activities - an application close to the heart of specification researchers! Even from this crude sketch it should be immediately obvious that they will not be building their own "library system"; they have neither the skills nor resources to do so. They will undoubtedly purchase an existing library package. In this context how should the requirements document be organised and what guidelines can be provided to the "procurement team"? Existing requirements engineering practice provides little guidance, indeed a substantial field study reported by Edmunson & Jeffery (1984) casts significant doubt on the overall appropriateness of conventional requirements engineering approaches to the acquisition of packaged software.

Software Procurement

In the discussion below we identify four constituent "customer" activities within software procurement and outline some features of these activities that present problems for requirements specification. The activities are (1) acquisition and specification of requirements; (2) understanding the available packages (3) assessment of package compatibility (with respect to the requirements) (4) selection of the "best" available package.

Acquisition and Specification of Customer Requirements

The acquisition of requirements for purchasing software packages is an incremental and iterative process. Customers start from an initial perception of their requirements and the priority they attach to them, normally based on a detailed understanding of their domains (that is the application area of their concern), their existing manual systems and other aspects of their operational environment (for example hardware and other software systems in use). Initial requirements are revised on the basis of advertisements, package descriptions provided by suppliers, demonstrations, use of the packages and comparative studies provided by third parties (trade papers and the like).

At an early stage in this process a distinction has to be made between "core" requirements that is those requirements which we can safely assume are provided by all or most of the available packages (such as logging of borrowing transactions in a library) and those "peripheral" requirements which are very specific to the customer (such as a library containing books with titles having Hebrew or Cyrillic characters). The manner in which this distinction is made shapes the procurement process, core requirements are neglected, as they do not help to distinguish between packages, while peripheral requirements are emphasised.

In general, packages have some features which are relevant to customer's needs and others which are not (for example, the ability of a package to catalogue pictures in a library which only has documents). Of course customers may recognise new requirements in features, which become very attractive once demonstrated but were not perceived as requirements initially (an example might be the ability to add descriptors to the catalogue entries for photographs). Interdependencies between requirements are often only appreciated as the range of features available becomes apparent (in a library system the way in which foreign language text is stored is closely linked to the way in which the thesaurus is constructed).

Understanding Available Packages

Because packages are supplied by different vendors for different, though intersecting, markets they are described differently. The descriptions use a mixture of technical and application domain specific terms, dependent on the speciality or the emphasis of different vendors. Under such circumstances, extracting information about, and understanding the package is a very difficult task.

Understanding a package demands a translation between two types of vocabularies, the vocabularies of various package specifications (Pi) and the vocabulary of the customer's requirements specification (R). These translations may be thought of as morphisms from the vocabularies of Pis to the vocabulary of R, which may not be total, onto and isomorphic. In fact, there may be features of packages wholly irrelevant to customer's requirements, requirements not satisfied by some packages, requirements satisfied through a synergy of more than one package features and more than one requirements satisfied by the same package feature.

In certain cases, especially if the package to be bought is expensive, the translation may result from a cooperation among the customer and the vendors. In this setting the customer issues a request for proposal (RFP), which enables vendors to describe their packages in a uniform manner that is, in terms of the RFP. Of course this process violates some of the "incrementality" discussed above. Further, experience suggests that the common response to an RFP is simply the standard package description! In most cases the onus for understanding packages rests with the prospective customer.

Assessment of Package Compatibility

Customers have to assess the extent to which different packages satisfy their requirements. Although in some cases the satisfaction of a requirement admits a binary answer (for example portability of a package onto a particular platform without alteration), for most requirements, different degrees of satisfiability may be distinguished. For example, the requirement for supporting foreign languages in a library catalogue is satisfied in various degrees by different library system.

Compatibility assessment demands some form of inexact matching between package and requirements specifications. This matching must cover both functional and non-functional requirements and cope with the feature dependencies and the incompleteness of both requirements and packages specifications. An additional complication is to distinguish cases where requirements cannot be satisfied by the package without substantial modification or rewriting and cases where requirements, although not satisfied by the default configuration of a package, could be met after some customisation, that is reconfiguration of the package within a predicted range.

Selection of Best Package

The selection of the "best" among the packages available depends on the assessment of their compatibility with the requirements specification and the prioritisation of these requirements. Selection is not static.

Customers may have to compromise on requirements not satisfied by any of the available packages, even if they had high initial priorities. In this situation, a common assumption underlying much work on specification for software development that "the customer is always right" is no longer valid. In this view the software always changes while the customers practices are fixed. In reality there is a "balance of mutability" customers may be prepared to change their practices in order to fit the best available package.

In many cases customers may decide to re-prioritise requirements. Such a re-prioritisation may be necessary in order to readjust the sensitivity of the selection process in the light of the merits and demerits of the different packages. This in turn requires the customer to have an accurate understanding of the impact of their priorities and

the robustness of selection decisions to changes in those priorities.

Related Work

Though we have argued above that little or no support exists to help the software user formulate their requirements and procure software packages that meet their requirements some mention should be made of information system development methods such as Information Engineering and SSADM (Ashworth & Goodland, 1990) which recognise the make vs buy decision and its importance.

Conventionally software procurement has been viewed as a simple sequential process (Zviran, 1993; Conger, 1994), involving: (1) the identification of potential vendors; (2) the preparation of an RFP and selection of benchmarks for eventual validation; (3) the distribution of the RFP to vendors; (4) the evaluation of vendor's proposals and the selection of the best of them; and, (5) the validation of selected proposals and the final choice among remaining vendors.

The RFP plays a key role in the whole process, since (in principle) it compels vendors to describe their packages in a uniform way, which enables customers understand and compare them. The RFP is a structured form of requirements, response guidelines, contract terms and corporate information (such as vendor size and experience). By convention requirements are distinguished into technical (for example hardware, software and operating system requirements), managerial (for example delivery schedule) and financial (for example cost, payment schemes) as well as into measurable as against qualitative and mandatory as against mutable requirements. Some detailed typologies of requirements have been proposed as a basis for RFPs covering hardware, operating system, financial and managerial requirements.

Each of the requirements or bundles of requirements constitute a selection criterion, weighted by its importance. Weighting of criteria may be based on techniques such as an analytic hierarchy, fixed scale valuation or binary weighting. Packages are evaluated using overall scores, estimated as the weighted sums of their individual scores for each of the selection criteria. Such scores are assigned by the customer. Various refinements of this approach use sophisticated techniques drawn from economics and multi-criteria decision making in supporting the final selection.

In spite of their strength in supporting the evaluation of packages with respect to “non functional” (generally measurable) software requirements (such as security, reliability, performance), traditional approaches are weak in treating qualitative and core functional requirements. In particular, they are weak in supporting multi-valued features, features valued in partially ordered sets and inexact-matching of features with requirements. The weighting techniques they employ generally require a total prioritisation ordering of requirements, an unjustified and misleading simplification, since commonly customers are only able to partially order requirements according to their significance. Finally, the view of procurement as a linear process fails to take into account revision, learning and iterative assessment.

Software selection and comprehension have been investigated as problems within the broader problem of software reuse (Biggerstaff & Richter, 1987; Krueger, 1992; Spanoudakis & Constantopoulos, 1994). However, it may be difficult to transfer the methods and techniques developed in this setting. In particular, the focus of concern in software reuse has either been fine-grain source code components (Diaz, 1991; Constantopoulos et al. 1995), algorithms and abstract data types (see software schemas in Katz, Richter & The, 1989) or software architectures, large grain global structures for software systems designs (Shaw, 1991; Neighbors, 1989). None are directly comparable with software packages. The reuse process generally entails significant further development, involving modification and integration into larger software frameworks as against the more limited customisation associated with software packages. In the case of reuse, research aims at creating a large, distributed and diverse market of software components while for software packages this market already exists and must be shaped, or at any rate, managed. Furthermore, since reuse is generally carried out by software developers knowledge and skills are required which cannot safely be presumed of those engaged in software package procurement.

A common, if misleading, way of fitting “the square peg” of software package procurement into “the round hole” of conventional software engineering practice is to view it as, in essence, a validation problem. Packages are validated against requirements, begging the question of what these requirements are, how they are acquired and structured. Suggestions have included the use of scenarios (Benner et al. 1992) though it is unclear where these scenarios come from and the extent of the assurance they deliver.

Research Programme Components

Some of the issues identified above are obviously not restricted to off-the-shelf software packages and apply to purpose-built software - though not with equal force. We might expect a research programme which looks at software package requirements and procurement to yield

more general insight into the software specification process. Below we describe the key building blocks of such a programme whose objective is to develop a set of organising principles for the software package requirements and procurement process. In large part we believe that the issues we face are fundamental, and require us to pursue a theoretical-logical approach. By taking this approach we place software package requirements and procurement within the mainstream of specification research and establish clear targets in terms of the soundness of the guidelines and principles that may result.

• *Description*

Clearly there needs to be a coarse grain framework within which requirements can be described and to which compatibility assessment makes reference. To line up with package descriptions in terms of "features" we propose specification in terms of "services" with corresponding service descriptions which include the constraints on the provision of those services.

The description of a service is an abstract, independent specification of the services intended behaviour in isolation. It presents a theory whose terms and constraints are expressed in a formal language (Cohen, 1995). An important issue in a language for describing features is the need for feature specifications to make minimal assumptions about the specification they're over. This will allow features over one specification to be applied to another. For example, consider a basic library system package, and define for it the feature of being able to keep accounts of users' fines. We would like this feature to be specified in a way that makes as few as possible assumptions about the library system specification, so that it can be applied to any other library system specification.

• *Matching*

Questions of vocabulary matching have been addressed before in specification, and have given rise to the notion of signature. Much formal work addresses how to translate between signatures and compare specifications in different vocabularies (Goguen & Burstall, 1984; Fiadeiro & Maibaum, 1991; Turski & Maibaum, 1987). This work could potentially be recruited to our setting. The assumptions that a feature makes about a specification may be stated in terms of its signature. The users-fines feature assumes that the underlying signature has the notions of "users" and "overdue returns".

• *Compatibility*

The topic of ranking packages according to the degree to which they satisfy requirements is strongly related to the topic of verisimilitude in logic (Brink, 1989 and Benthem, 1987), where theories are ordered according to how close they are to a notional "truth". But there are several ways in which the package requirements and procurement problem differs from the standard problem of verisimilitude. One difference is that the user is usually not concerned if a particular package extends the requirements in some way, as this is only an extra feature, the concern is only in the case that the package falls short of the requirements. Such a notion of verisimilitude is considered in Ryan & Schobbens, 1995. Another way in which our problem differs from the standard problem of verisimilitude is that there are priority relations among some of the requirements.

• *Revision*

In the case of revision, change and mutability, a theoretical approach could be expected to provide some insight too. The requirements could be represented as an ordered theory, where the ordering is used to specify the entrenchment, or priority, with which the user asserts the requirement. We could potentially exploit work in the topic of theory revision (Gardenfors, 1988; Ryan, 1994) to get a handle on the question of the degree of change required to the package.

The introduction of features on a specification involves specification revision. Returning to the library example, the library system augmented by the user-fines feature will manifest behaviours (such as blocking loans to users who have unpaid fines) which were not shown by the basic library system. Thus, a first approximation to the semantics of feature introduction might use the standard work on belief revision.

However, usually there are constraints within the original specification which should be considered immutable. In a state-based system such as the library, axioms constraining the notion of state should not be revised, but axioms describing the effects of actions should be.

We have worked on the idea that the models of the specification revised by a feature should be computed by taking those models of the feature which are "as close as possible" to the original specification. This requires the notion that interpretations are ordered according to closeness to the specification. Our results on simple specifications suggest that interpretations should be compared only if they agree on action occurrences.

Conclusion

This position paper has given an account of the problem of software package requirements and procurement and briefly reviewed related work. It has

sketched the key components of an approach to this problem. In the final analysis however the purpose of this paper is to situate software package requirements and procurement within specification research and to argue that it is a problem worthy of serious consideration. Further, the issues it raises which if not wholly unfamiliar are differently configured and with significant change of emphasis. They are at any rate fundamental and we cannot expect to simply hand this problem on to management scientists or other research communities.

References

C. Ashworth and M. Goodland. (1990); SSADM: A Practical Approach, McGraw Hill.

K. Benner, M.S. Feather, W. L. Johnson & L. Zorman. (1992) ,Utilizing Scenarios in the Software Development Process; IFIP WG 8.1 Working Conference on Information Systems Development Process, 9 December 1992.

J.van Benthem. (1987); Verisimilitude and conditionals, What is Closer-to-the-Truth, (ed) T.Kuipers, pages 103-128, Rodopi, Amsterdam

T. Biggerstaff, C. Richter. (1987); Reusability Framework, Assessment and Directions, In Frontier Series: Software Reusability, Volume I: Concepts and Models, T. Biggerstaff and A. Perlis(eds), ACM Press, New York

C. Brink. (1989); Verisimilitude: Views and reviews. History and Philosophy of Logic , 10, pp. 181-201, 1989.

B. Cohen. (1995); The Description and Analysis of Services as Required and Provided by their Agents, Department of Computer Science, City University, London

S. Conger. (1994); The New Software Engineering, The Wadsworth Series in Management Information Systems, ISBN 0-534-17143-5, Belmont, California

P. Constantopoulos et al. (1995); The Software Information Base: A Server for Reuse, The VLDB Journal (to appear)

R.P. Diaz. (1991); Implementing Faceted Classification for Software Reuse, Communications of the ACM, 34(5), May 1991

R.H. Edmundson, D.R. Jeffery. (1984); The Impact of Requirements Analysis upon User Satisfaction with Packaged Software, Information & Management 7, Elsevier Science Publishers, North Holland

J. Fiadeiro , T. Maibaum. (1991); Describing, structuring and implementing objects, In Proceedings of REX Workshop on Foundations of Object-Oriented Languages, Springer-Verlag

P. Gardenfors. (1988); Knowledge in Flux: Modelling the Dynamics of Epistemic States, MIT Press

J.A. Goguen, R.M. Burstall. (1984); Introducing institutions, In Proceedings of Workshop on Logics of Programming, (eds) E.Clarke and D.Kozen, Lecture Notes in Computer Science 164, Springer-Verlag

S. Katz, C. Richter, K. The. (1989); PARIS: A System for Reusing Partially Interpreted Schemas, In Frontier Series: Software Reusability, Volume I: Concepts and Models, T. Biggerstaff and A. Perlis(eds), ACM Press, New York

C. Krueger. (1992); Software Reuse, ACM Computing Surveys, 24(2), June 1992

J. Neighbors. (1989); Draco: A Method for Engineering Reusable Software Systems, In Frontier Series: Software Reusability, Volume I: Concepts and Models, T. Biggerstaff and A. Perlis(eds), ACM Press, New York

M.D. Ryan. (1994); Belief revision and ordered theory presentations. In Logic, Action and Information, (eds) A.Fuhrmann and H.Rott, De Gruyter Publishers, 1994 (also in Proceedings of the 8th Amsterdam Colloquium on Logic, University of Amsterdam, 1991)

M. D. Ryan, P.-Y. Schobbens (1995); Belief revision and verisimilitude; Notre Dame Journal of Formal Logic, Vol. 36(1), 1995.

G. Spanoudakis, P. Constantopoulos. (1994); Similarity for Analogical Software Reuse: A Computational Model, In Proceedings of the 11th European Conference on Artificial Intelligence, Amsterdam, The Netherlands, August 1994

M. Shaw. (1991); Heterogeneous Design Idioms for Software Architerctures, In Proceedings of the 6th International Workshop on Software Specification and Design, Como, Italy, October 1991

W.M. Turski, T.S.E. Maibaum. (1987); The Specification of Computer Programs, Addison-Wesley, London

M. Zviran. (1993); A Comprehensive Methodology for Computer Family Selection. Journal of Systems and Software, 22(1), July 1993.