Construction by Configuration: Challenges for Software Engineering Research and Practice

Ian Sommerville
School of Computer Science, University of St Andrews, St Andrews, Scotland
ifs@cs.st-andrews.ac.uk

Abstract

The past ten years have seen a radical shift in business application software development. Rather than developing software from scratch using a conventional programming language, the majority of commercial software is now developed through reuse – the adaptation and configuration of existing software systems to meet specific organizational requirements. The most widespread form of reuse is through the use of generic systems, such as ERP and COTS systems, that are configured to meet specific organizational requirements. In this paper, I discuss the implications of software construction by configuration (CbC) for software engineering. Based on our experience with systems for medical records and university administration, I highlight some of the issues and problems that can arise in 'construction by configuration'. I discuss problems that arise in CbC projects and identify a number of challenges for research and practice to improve this approach to software engineering.

1. Introduction

Software engineering in industry has changed dramatically since the mid-1990s. Prompted by factors such as the need to upgrade legacy systems to cope with the Year 2000 problem, business pressures for faster system deployment and dissatisfaction with existing business processes, many companies and government organizations have switched their primary systems development strategy from 'specify and construct' to 'buy and configure'.

Rather than defining the requirements for a new systems and then developing that system using a conventional programming language such as Java or C#, systems are developed by buying an off-the-shelf generic application system and configuring this system to create a specific organizational application. This may involve buying a general-purpose ERP system and configuring appropriate modules. Alternatively, a generic application (COTS) designed for a specific application area (such as library management) may be bought and adapted to the customer's business.

We know that this approach to application development has had a major impact over the past 10 years. While specific figures are hard to come by, a Gartner report suggested that 95% of the top 2000 US companies use ERP systems. In a study published in 2000, Everdingen et al. [1] predicted that more than 50% of European midsize companies would use ERP systems by 2000 and the figure is almost certainly much higher now. Penetration is significantly less in smaller organizations because of the high initial costs of these systems but, in these organizations, COTS systems have had a significant effect. My own experience suggests that the preferred procurement strategy for information systems in most organizations is now to 'buy and configure' rather than 'specify and program'.

Academic research has reflected this change in a limited way. There is an international conference series on COTS-based software (ICCBSS) but the majority of software engineering research remains focused on 'specify and construct' development. For example, in the proceedings of the 2007 International Conference on Software Engineering (ICSE), there was a single paper on COTS-based systems. In ASWEC 2007, there were 2 papers on COTS-based development.

I believe that this is an area where there is a pressing need for research. The initial promise of rapid system deployment for a relatively low cost is often unfulfilled. There are horror stories of problems with ERP systems development. It seems that delays in delivery and problems in use are as common as with systems developed using a 'specify and construct' approach. Deployed systems often fail to deliver the cost reductions expected and users are frequently unhappy with the system performance and functionality.

In this paper, I will focus on the problems of 'programming' COTS systems where 'programming' means configuring generic modules using the support provided by the system. I call this 'construction by configuration' (CbC). I will discuss some of the problems that we observed during long-term, detailed studies of configuration and deployment of 2 large-scale information systems, as well less detailed analyses of several other systems. I propose a research agenda to extend our understanding of and support for this mode of systems development. I draw on examples from a patient information system for mental health care installed in a large hospital and on a university-wide student administration system being deployed in a large UK university with more than 20, 000 students.

2. Configurable systems

While all systems require some configuration when they are deployed, I am concerned here with software that is designed to be generic but which can be tailored and adapted to specific organizational requirements. The *configurable generic systems* that are intended to be the basis for business-specific applications are of three kinds:

1. Single PC-based applications with built-in programming capability. The best-known example of such an application is probably MS Excel. Starting with basic spreadsheet functionality, Excel can be extensively reprogrammed to create specific applications. Applications that can be extended using plug-in capabilities (e.g. the Firefox browser) can also be thought of as configurable systems.
2. Modular generic systems. These are systems, such as ERP systems, where the supplier has a range of functional modules (manufacturing, CRM, etc.) and the purchaser decides which modules they will include in their system. These individual modules are then integrated and configured.
3. COTS assemblies. In this case, the application is constructed by integrating several off-the-shelf systems. For example, we have studied a bank's e-procurement system, which was created by integrating an invoicing system and a web-based e-commerce system.

In all cases, the developed system may also make use of other desktop applications such as MS Word or MS Excel to provide additional functionality.

2.1 System architecture

Our work on configurable systems has been mostly concerned with organizational information systems, with a shared database and multiple users from different backgrounds. These systems may be constructed using either modular generic systems or COTS assemblies. They may be organised as 'thin client' systems where all processing takes place on servers and access to applications is through a web browser. Alternatively, a 'thick client' approach may be used where applications are installed on client computers and communicate with a server-based database. Each of these architectures results in different types of problem.

Thin-client systems, where all functionality is implemented as services running on a server, simplifies system management. The system servers are responsible for all processing, with functionality accessed through a web browser. This approach avoids the problems of user re-configuration discussed below and reduces installation costs. However, it brings with it its own set of problems:

1. Significant investment in the server infrastructure may be required to cope with periods of peak loading and to provide redundancy in the event of server failure. This is a particular problem when periods of peak loading are relatively short but critical.
2. There has to be close cooperation between staff responsible for configuration and staff responsible for infrastructure management. Large organizations often have heterogeneous computing infrastructures with different operating systems and browsers in use in different parts of the organization. Changes to the infrastructure (browers and operating systems) may lead to problems and incompatibilities.
3. Security problems with browsers are well-known. Where the information maintained is critical (e.g. medical records), the risks involved in using a brower-based systems may be considered to be too high.
4. The system is inflexible. The server-based system cannot easily be tailored to groups of users within an organization who have non-standard needs, such as emergency care or maternity in a hospital.

The thick-client approach avoids some of these problems. In this approach, the server is solely responsible for data management and a special-purpose application which is responsible for local processing is installed on each client. The benefits of this approach are that server load and network traffic are significantly reduced and that off-line operation is possible when servers or network access are not available. This approach, of course, has its own problems:

1. Higher management overhead as the applications deployed on each client have to be updated to reflect changing requirements.

2. Excessive flexibility allowing individuals and groups to reconfigure the system locally thus increasing the costs and problems of system evolution.
3. Security problems as sensitive data is cached on local machines with possible access to that data without conformance to the system security procedures.

Of course, the architecture of the system will largely depend on design decisions made by the developers of the COTS system that is being configured. If a different architecture is required, serious problems can arise. We observed this in the university administrative system where, to force process conformance, it was required that a thin-client architecture should be used.

However, there were no COTS systems available with this architecture so, after a decision was made on a system, the system provider was paid to implement a web-based interface. After these changes were made and the browser interface implemented, it was discovered that the performance of the delivered system was so poor that critical organizational deadlines could not be met.

3. System configuration

The deployment phase for all software systems inevitably involves some system configuration. This may simply involve providing some data about the operating environment or its users. However, for generic COTS systems, this configuration is much more extensive and may involve adapting the system to reflect some or all of the following:

1. The specific needs of a customer (e.g. a hospital).
2. The requirements of a group of users within the organization (e.g. a maternity unit).
3. The required interactions with other systems in its environment.
4. The characteristics of the platform on which the system operates.

To meet these needs, many different configuration activities may be involved in adapting a COTS system to its environment. These include:

- Selecting the required system modules
- Defining a data model
- Defining business rules
- Defining workflows
- Defining external interactions
- Defining the user interface
- Defining the reports to be produced
- Setting platform parameters
- Re-defining business processes

Business process configuration is particularly important. All ERP systems and COTS embed their own generic model of business processes. For these systems to work effectively, it is essential that business processes are adapted to conform to this model [2]. Otherwise, it is extremely difficult to make effective use of these systems.

While the requirement to change process might be seen as a barrier to the adoption of these systems, the opportunity for change is actually welcomed by many organizations. They see this is a way of standardizing processes across the organization. Control of the process is removed from individual units and retained by central management. I discuss some of the difficulties that this can cause later in the paper.

3. The CbC process

The CbC process obviously varies depending on the type of system being developed and the facilities offered by the COTS system that is being configured. In the systems that we have observed, the following process activities have been common:

1. *System selection*. A system is procured based on high-level business rather than operational requirements.
2. *Requirements analysis*. Existing business process stakeholders are consulted and their 'requirements' are elicited. However, these are seen as providing advice to the development team rather than real system requirements.
3. *Business process redesign*. The business processes are redesigned to fit the processes assumed by the COTS or ERP system.
4. *System configuration*. The system is configured to reflect the new business processes and, as far as possible, the user requirements.
5. *System testing*. The configured system is tested to some extent although, as I discuss below, this is a particularly problematic stage of the process.
6. *Deployment and evolution*. The system is deployed, put into use and evolved rapidly to reflect user problems and difficulties.

In the system developments that we observed, there were several process issues that led to problems with the deployed system:

1. *A 2-stage requirements process*. In the first stage, there was minimal user involvement and management defined the requirements that were used to select the COTS product to be purchased. The 2nd stage took place after the procurement decision had been made and the requirements were constrained by the facilities available in the purchased system.

2. *Co-design of software and process*. The ability of rapidly reconfigure the system meant there was scope for user involvement in the development of both the process and the software. Therefore, the requirements evolved during the development and the system at any one time reflected the requirements of the most influential group of system users.
3. *Ad hoc testing*. There were no systematic approaches adopted to testing, partly because the system rarely 'failed'. Each configuration setting led to a working system. However, the lack of detailed requirements made it very difficult to test the system for suitability before deployment.
4. *Lack of supporting processes*. Conventional software engineering includes supporting processes such as configuration and quality management. It was found to be impossible to import these processes into a CbC development process.

Morisio et al. [3] found comparable problems in studies of COTS-based development processes in NASA.

The 2-stage requirements process led to the selection of a system based on the high-level requirements of managers who did not have sufficient knowledge of current processes in their organization. While a systematic approach was used for software selection, the systems that were ultimately procured were found to be lacking in a number of important respects when a more detailed understanding of the requirements emerged.

These problems largely arose because the system providers had 'hard-wired' assumptions into their generic software, with limited ability to reconfigure these assumptions. The buyers of the system did not question these assumptions and only discovered problems when the system was being configured.

An example of an area where this can arise is in assumptions about the legal and regulatory framework in which the system will be used. Many systems have to conform to regulatory requirements (e.g. in the US the Sarbanes-Oxley accounting rules). Typically, a system will first be developed and sold in a home market and the regulations that apply to that market will be implemented in the generic system. However, when these systems are sold in another country with different laws and regulations, it may be difficult to reconfigure the system to conform to these regulations.

When one important reason for procuring a system is to reduce the costs of compliance, this can be particularly problematic. The patient information system we observed was used to manage the information about patients with mental health problems. It is sometimes necessary to detain such patients against their will because they constitute a danger to themselves and others. However, there are strict legal safeguards associated with such detention and often time limits as to when assessments are carried out. It was observed that the purchased system was being used effectively to support this process elsewhere and this was an important factor in the procurement decision.

However, the systems that were demonstrated were in use in England, which has a different legal system from Scotland where the system was to be deployed. In particular, there was a more recent mental Health Act in Scotland with quite different processes for assessing patients for involuntary hospitalisation. It proved to be impossible to reconfigure the system to reflect these processes. Thus, the system did not meet one of its principal requirements – the existing manual systems had to be retained and there were no cost savings from introducing the new system.

In reality, it is not always possible to 'do it the system's way'. Existing business processes have evolved for good reasons to ensure that work is done effectively and efficiently. In complex organizations, such as a hospital, where different parts of the organization work in different ways (e.g. maternity and emergency medicine), standardized processes imposed by management who do not understand operational systems use may simply not be good enough.

The co-design of requirements, software and process is, in many respects, a good thing. Users were actively involved in the process and their comments were fed back quickly to the development team. However, the process did not properly recognise the important differences between processes in different parts of the organization and, critically, that there is no real benefit for users in spending time communicating with other users in different departments. Of course, this is not just a problem for CbC – it is also a serious problem for agile development processes.

During the system development, the major problem that this caused was the impossibility of maintaining a stable system specification – it changed daily. When this was combined with the fact that the system users and other stakeholders did not communicate effectively, it was practically impossible for anyone to understand the actually delivery schedule for the features in the system.

This co-design activity did not stop when the initial version of the system was deployed. It continued throughout the 'bedding in' period where the new processes and the new system was brought into operational use. Here, we observed a new problem. The stakeholder group with the greatest political influence drove the changes to the system, without due regard for the requirements of other stakeholders. For example, in the hospital system, this manifested itself

as changes to the user interface influenced by the hospital management and senior doctors. Nurses and junior administrators using the systems were not consulted.

Testing was observed to be a serious problem. System failures did not, by and large, manifest themselves in ways that were obvious to developers because the lack of a detailed specification meant that they did not really know what the system was supposed to do. Systems rarely failed in the sense that they crashed or produced clearly incorrect output. Rather, the failures could only be detected by users who understood their local processes and who could identify where system support was inadequate. In addition, regression testing was found to be impossible as the COTS systems were not designed to run automated test suites.

While a period of user testing was scheduled before the system was deployed, schedule slippage meant that the time for this was compressed. This meant that, rather than spending a few hours per week on testing, the developers expected the users to be available for several days before deployment. As the users were very busy people, this was practically impossible so there was very little testing carried out. The systems were deployed so that testing and usage were one and the same thing.

By and large, good change and configuration management practice has evolved in development environments where an important requirement was source code control. Existing source code control systems cannot be used alongside the COTS systems to manage the evolving configuration data, because the configuration is done using specialized support built into the system. The configuration data cannot be separated from this system itself. This made it practically impossible to revert to previous system versions when problems were discovered. In general, quality management was a serious issue as there was no shared perception of what was meant by a 'high quality system'. Good practice, such as inspections and reviews, were not carried out partly because of the problems of configuration visibility that I discuss in the next section of the paper.

4. Configuration problems

Large-scale ERP systems are so complex to configure that development is usually the responsibility of the system supplier. User-configuration is practically impossible because the learning time for the system is so long. However, for smaller-scale ERP developments (e.g. using open source ERP [4]) or for COTS-based development, the system owners are more likely to be involved in the configuration process. This was the case in the systems that we observed where the system owners set up an internal project team to develop and deploy the systems.

We discovered that there were three principal sources of difficulty faced by CbC developers:

- Understanding the configuration options
- Understanding the configuration semantics
- Understanding how a system is configured

Most configurable systems offer a range of different configuration options with, sometimes, subtle and difficult to understand interactions between these options. Sometimes, these options are obscure and poorly documented and there is rarely information available about how different options may interact.

For confidentiality reasons, we cannot discuss the specifics of the applications that we have studied but the same problems also arise in PC software. For example, my version of MS Word offers at least 8 different ways to configure the system:

- Preferences screen
- Customisation screen
- Organiser screen
- Definition of templates
- Definition of styles
- Definition of macros
- Inclusion of add-ins (e.g. Endnote)

Even with 20 years experience of using Word in many different versions, I would find it hard to explain what each of these does and how they interact.

Once a developer has discovered the different ways to configure an application, he or she is then faced with the problem of deciding which options to use. However, the semantics of the configuration options are rarely explicitly defined and developers have to rely on (often minimal) documentation and examples to try to understand them. The configuration options may reflect the underlying semantics of the system being configured and so developers have to infer these to understand the configuration possibilities.

When you are developing a software system, it is useful to be able to predict the consequences of changes to that system. Unlike a conventional programming language where the programmers understanding of the language semantics is used to decide how to implement changes, changes to configurations are typically experimental. The meaning of the configuration is defined by the underlying system, which is a black-box. Over time, gurus emerge who can make things happen with a system but cannot explain why these happen and cannot effectively transmit their knowledge to others.

The problem of understanding the current configuration of a system is a critical one for the

development and maintenance of a system. We have never yet seen any configurable system where there is a simple way of displaying the current configuration. This means that building an overall picture of a configuration is very difficult. This situation is made worse by the fact that configurations cannot be maintained in a version management system. There is therefore no single description of a configuration available.

This causes problems where changes to the system are proposed and have to be costed and implemented. Assessing the impact of a change is generally very difficult, even when system design documentation and the code of the system programs is available. It is even more difficult in configurable systems for two reasons:

1. The co-design process where system requirements are developed alongside the implementation means there is no specification of the system. This, of course, is also a problem with some agile development approaches but there, at least, the code is available to define the system. In configurable systems, there is no single description of either what should be implemented and what has actually been implemented.
2. It is not enough simply to understand the configuration to assess the impact of a change. It is also necessary to understand the underlying COTS system. During development, where consultants from the system supplier are available, this is less of a problem. However, after the system is handed over, gaining access to system information may be much more difficult.

The consequences of this are that the change costs may be much higher than expected and may take much longer than expected. For example, in the patient information systems that we observed, a small change to the user interface that was originally predicted to take 2 or 3 days ended up taking 6 weeks to implement.

4.1 Process configuration

As I have discussed, it is generally accepted that ERP and COTS system developments can only be successful if the business processes are configured to match the process model that is assumed by the software. Typically, therefore, the deployment of a new system involves process change. This is often welcomed by the organizational management who see it as an opportunity to improve and control of business processes and to ensure these are standardized across the organization. New processes may allow more effective use of new IT systems.

However, things are rarely so simple. In almost 20 years of studies with end-user organizations, we have never seen a process that has not been adapted to suit the local circumstances. The individuals involved in enacting processes always modify these processes to make them more suitable for the way in which they work. If they are forced to work with a standardized system, they will simply add on activities outside that system. For example, in a system that generated PDF reports, we observed users making use of an MS Word add-in that converted PDF to Word because they needed to add additional information to the report.

However, if they have a configurable system, enterprising users will discover how to do the configuration and will make local changes to suit their own requirements. We observed this in the patient information system that was deployed across a number of geographically dispersed clinics. The doctors in charge of these clinics had evolved their own way of working and of keeping patient records and they asked local IT staff to modify the deployed system to reflect this. Of course, this caused problems when the information from different clinics was integrated to create management reports.

It might be argued that this was a management failure and that system managers should have retained control over the changes to the system. However, there are real practical difficulties here because of the distribution of power and influence in an organization. If a senior manager (or in this case, a senior doctor) asks a relatively junior member of the IT staff to make changes to a system, it is very difficult for them to refuse to do so, whatever the organizational policy.

The possibility of relatively simple re-configuration can also lead to situations where configuration decisions are challenged and changed to reflect changing political power and influence in an organization. The patient information system that we observed was originally chosen and deployed because it had the capabilities to produce a set of reports that were required by the healthcare management board. These reported patient statistics under a number of different headings which the managers assumed were consistent with clinical record keeping.

However, the initial trial deployment of the system caused major problems because the categories under which the clinicians recorded patient information were quite different from those used by the management reports. They then insisted that the system be reconfigured so that clinical categories where used and that additional software was developed to convert these to the management categories. This proved to be impossible as the clinicians themselves did not have a standard method of recording patient information.

Some compromise was therefore sought where some management information was added to normal clinical recording. The amount of this information varied over time as different individuals were involved in the development of the system and as political power and influence changed. What should have been a 6-week process of configuring the user interface for the system, ended up taking almost a year before a stable interface was agreed.

The adoption of a thin-client approach where all interaction is with a central system makes end-user re-configuration impossible. However, it does not change the political realities where senior managers or professionals argue that (sometimes rightly) that their requirements are different from the rest of the organization and they need special support. In such situations, parts of the organization may simply play lip service to the new system but will actually develop their own parallel system to carry out their work.

We observed this in the university administrative system where the system assumed that all departments sent standard letters to students. In fact, departments which had problems in meeting their student recruitment targets (such as engineering and computer science) sent personalized letters because they believed this would attract students to them. As personalization was impossible in the university system, they simply maintained a parallel system to meet their real requirements.

5. Research challenges

Construction by configuration is a well-established development method for organizational systems. However, as I have discussed, this approach can be as problematic as other approaches to software development yet it has received very little attention from the software engineering research community. This has occurred for two reasons:

1. Many software engineering researchers are simply unaware of the scale of the change that has taken place. The changes to development practice have been driven by business rather than technical considerations and have had very little publicity in technical literature such as the Communications of the ACM, IEEE Software, etc.
2. Application systems are difficult to study in a university environment. These systems are expensive to procure and can only be used with business knowledge which is lacking in a university. However, open source middleware is widely available (e.g. Apache Axis) and this often has the same configuration problems.

This lack of involvement does, of course, cause credibility problems as it makes it much more difficult to explain the value of software engineering research. It also means that those involved in the CbC process find it impossible to relate their work to good software engineering practice. Methods and tools are reinvented and mistakes are repeated.

I believe that there are intellectually challenging software engineering problems in this mode of software engineering that the SE research community should tackle. These relate to both the COTS systems that are being configured and the configuration engineering processes.

5.1 Design for configuration

In the configurable systems that we have observed, it seems that little attention has been paid to the problem of design for configuration. What I mean by this is that the designers of COTS systems should recognize that configuring a system is time-consuming and expensive and that the generic system should be designed to simplify the configuration process and to reduce the probability of configuration errors. As researchers, we need to explore the notion of configurability and to establish design principles and guidelines for developers of configurable systems.

There are at least 3 areas where research is required:

- Design principles for configurability. What principles should be applied when designing the configuration options in a system?
- Configuration visibility. What do users require when trying to understand a configuration and how is configuration information best presented to them?
- Configuration description. How can we move away from low-level configuration to configurations that are a better reflection of business policies?

Good software design principles (such as information hiding, low coupling, etc.) have been established over many years but it is not clear how these apply to configurable systems. I believe that there is scope for research examining how these fundamental design principles can be applied in configurable systems and reflected in the configuration support tools. Examples of design principles that might apply to configurable systems are:

1. *Manageability*. In every system we have observed, it has been impossible to get an overview of the configuration then drill down through this overview to configuration details. In fact, it has usually been impossible to generate a complete view of the whole configuration. I believe that COTS systems should be structured so that the configuration is

maintained as a separate, structured entity that can be managed as a unit. As I discuss below, facilities should be provided to allow the configuration to be viewed and navigated by configuration engineers.

2. *Minimisation*. A serious problem with configurable systems is that there are often several ways to implement a configuration. It is difficult for designers to understand which of these is best and how they interact. The principle of minimization is that the number of configuration options that allow the same feature to be implemented should be minimized. Ideally, there should be only one way to implement a feature of a system.
3. *Separation*. Problems sometimes arise because configurable systems may not separate the configuration of the system for platform characteristics from the configuration of functionality? There is a need to identify distinct types of configuration and examine how these should be supported.
4. *Independence*. Following the general design principle of low coupling, the interactions between the different parts of the configuration should be minimized. Required interactions should be explicit and clearly documented.

As I have already discussed, configurable systems do not usually make it possible for developers to view the configuration 'as a whole'. Rather, they must use the different configuration options in turn to examine what has been configured. Relationships that exist between different parts of the configuration are not usually made explicit in these views.

There is a need for tools that allow engineers to see the 'configuration state' of a system and to explore dependencies across that state. Software visualization research has focused on viewing program entities and their relationships [5]. Perhaps this can be developed and extended to cover configurations and the relationships between configurations and the underlying system components?

Configuration of systems is an error-prone and ad-hoc process because the configuration is defined in terms of the underlying system entities rather than the organizational requirements and policies that must be supported by the system. We can see an example of this in security configuration where high-level security policies have to be translated into detailed commands to manage access control lists, etc.

One area of possible research, which is linked to the issue of configuration modelling, discussed in the following section, is to explore whether or not higher-level policy languages can be used as a basis for defining configurations. That is, the organization sets out what it wants to do in some policy language and an automated translator creates the configuration commands from this. Of course, this relies on configurations being separate from the application being configured

5.2 Configuration engineering

As well as research focusing on design for configuration, I believe that there is also scope for research concerned with 'configuration engineering'. This should focus on the process differences between code-based development and construction by configuration. Possible research areas include:

- Knowledge management
- Configuration modelling
- System testing
- Supporting processes

Most of the problems that arose when configuring and deploying configurable systems were a consequence of poor knowledge management. In all cases, someone in the organization knew of and understood the problem but this knowledge was simply not transmitted to managers, the development team members or other users. All to often, we heard remarks like 'I knew this would be a problem' and 'we have that problem too'. Users, in particular, were very poor at sharing their knowledge.

I believe that the most effective way to tackle this problem is to develop more effective systems for knowledge management that makes it easier to capture, classify and share knowledge about assumptions, the organization and the system itself. Such a system could also help integrate supporting processes such as change and quality management.

The modelling of software systems to remove inessential detail is accepted as good software engineering practice [6]. While system configuration makes use of business process models, we do not have methods and techniques for modelling other aspects of the configuration. Starting with a model of the generic system to be configured, is it possible to add detail to this to define the specific configuration. The extent to which this is possible with closed soruce systems is an open question but it is an area that I believe is worthy of further research.

System testing during CbC is a major problem because tests are to validate the system rather than verify it against a defined specification. This validation is difficult because of the new processes that are introduced alongside the system. Problems may be process rather than software related.

There is scope here, perhaps to explore test-first development as practiced in some agile methods [6] and to investigate whether the requirements engineering processes should be oriented towards the

definition of tests that can be applied by a development team. Other research issues in testing include the problems of providing test coverage and automated regression testing tools.

Finally, the supporting processes of configuration, change and quality management are different. We need to understand what are the quality attributes that should apply to a configuration. User-led change will remain an issue for many configurable systems and there is a need for change and configuration management support that is user-accessible and that can be integrated with more general knowledge management support.

6. Conclusions

Construction by configuration is now perhaps the most widely used development technique for large-scale organizational information systems. However, problems with this approach are well-known. Delays in delivery, unmet organizational expectations and user dissatisfaction are common. However, academic research into these problems and possible solutions is relatively limited and few researchers have explored how good software engineering practices can be adapted to support construction by configuration.

To research effectively in this area, we need to come out of the laboratory environment and interact directly with the businesses and organizations developing these systems. Approaches such as ethnographic studies [7] and action research [8] are required to develop an understanding of the problems and issues faced by CbC developers and to investigate the real utility of new tools and technologies.

Of course, there are very real difficulties in interacting with industry and studying systems as they are, rather than as researchers might like them to be. Industrial interaction takes a lot of time and confidentiality issues may limit the possibilities for research dissemination.

It is always tempting to use more manageable laboratory examples and to test new techniques on simplified systems. However, as we have seen from 30 years of software engineering research, this approach is often unsuccessful. The complex reality of systems in use makes it impossible to use techniques that may be technically superior but which, for practical reasons, are unusable in most industrial settings (e.g. formal methods).

Construction by configuration has immense economic significance. As a research area, it offers new challenges and opportunities to the software engineering research community. I believe that it is now time to embrace these challenges and to demonstrate the relevance of software engineering research to modern software development.

10. References

[1] Everdingen, Y.V., Hillegersberg, J.V., Waarts, E. 'ERP adoption by European midsize companies', *Communication of the ACM*, Vol. 43 No.4, April 2000, pp. 27-31.

[2] Law, C.C.H. and Ngai, E.W.T. 'ERP systems adoption: An exploratory study of the organizational factors and impacts of ERP success'. *Information & Management***.** Volume 44, Issue 4, June 2007, pp 418-432.

[3] Morisio M., Seaman C.B., Basili V.R., Parr, A.T., Kraft S.E., Condon S.E. 'COTS-based software development: Processes and open issues'. *J. of Systems and Software*, Volume 61, Number 3, 1 April 2002 , pp. 189-199.

[4] Serrano, N.; Sarriei, J.M. 'Open source software ERPs: a new alternative for an old need', *IEEE Software*, Volume 23, Issue 3, May-June 2006 pp 94 – 97.

[5] Gracanin, D., Matkovic, K., and Eltoweissy, M. 'Software visualization', *Innovations in Systems and Software Engineering: A NASA Journal*, Volume 1, 2005, pp 221-230.

[5] 'Model driven engineering'. *IEEE Computer, Special Issue*, Volume 39, Issue 2, February 2006.

[6] Beck, K. *Extreme Programming Explained: Embrace Change*, 2nd Edition, 2004. Addison-Wesley.

[7] Hughes, J.A. and Shapiro, D. 'From Ethnographic record to system design: some experiences in the field'. Computer Supported Collaborative Work Volume 1 Issue 3, June 1993, pp 123-147.

[8] Avison, D.F., Myers, M. D. and Peter Axel Nielson, P.A.. 'Action Research', *Communications of the ACM,* Vol. 42, No. 1, January 1999, pp. 94-97.