Requirements Eng (2002) 7:179–191
Ownership and Copyright
© 2002 Springer-Verlag London Limited

Requirements Engineering

# Authorisation Requirements on a Budget

**Darrell Raymond**
Alternative Output Inc., Waterloo, Ontario, Canada

*Engineering information system deployment is squeezed by a shrinking commitment to requirements definition and an expanding need to determine the security requirements of such systems. This paper examines the causes and effects of this squeeze. Commitment is shrinking because of past requirements experiences, misunderstood trends in system development and requirements fatigue, while needs are expanding because of recent emphasis on Internet access to data, online transactions and workflow, which greatly increase the severity of the authorisation problem. Some approaches to quantifying and addressing this problem are introduced.*

**Keywords:** Authorization; Dependency analysis; Security requirements

## 1. Introduction

Academic wisdom has it that a majority of system failures are due to poor requirements analysis. But business practice continues to favour system deployment with minimal requirements gathering. This is true even though increased emphasis on security has increased the need for requirements definition, particularly for authorisation. In this paper we study this problem in the context of systems that manage engineering and manufacturing information.

Engineering companies need to manage information about their products, which consists largely of drawings, documents, bills of materials and engineering change orders. The systems used to manage this information are electronic document management (EDM) or product data management (PDM) systems. Vendors in the EDM/PDM field include Agile, EDS, Parametric Technologies and MatrixOne. EDM/PDM systems provide the following capabilities:

- storage of large numbers of documents in an electronic 'vault';
- change management for documents, typically through the use of check-in/check-out and revision control;
- storage and change management of product data and product structure (component assembly or 'bill of material' relationships);
- storage and management of relationships between product data and documents;
- definition and execution of structured processes, known as workflow.

EDM/PDM projects are fairly typical IT projects, with a medium budget ($2–$5 million) and a medium-sized user community (1000–10,000 users). An EDM/PDM deployment usually takes from 6 to 12 months to complete, although the more ambitious efforts are developed in phases that span two or more years.

An EDM/PDM system's need for authentication, data privacy and data integrity is similar to those of other business systems, and hence requirements definition for these aspects of security may be considered typical. The same is not true of authorisation, which is an essential component of an EDM/PDM system. Managing authorisation requires that the organisation can define, control and assure:

- who can participate in certain activities;
- what kinds of operations they are allowed to perform;
- the types of data that can to be manipulated.

Authorisation is a direct, user-manipulated, functional part of an EDM/PDM system, and hence is a functional requirement of prime importance. Developing an

*Correspondence and offprint requests to:* D. Raymond, 305 Bushview Crescent, Waterloo, Ontario, Canada N2V 2A6. Email: darrell.raymond@sympatico.ca

organisation's authorisation needs in the requirements phase is an essential part of specifying the functionality of an EDM/PDM system.

## 2. The Shrinking Budget for Requirements

While it is still considered good practice to include a requirements phase in an EDM/PDM deployment, there has been a marked decrease in appetite for requirements work. Several years ago it was not uncommon for a company to spend two to four months doing requirements before a major EDM/PDM deployment, but now it is more likely for a company to spend as little as three weeks. Several factors contribute to this trend.

Many engineering companies cannot use internal resources to develop a formal statement of requirements. IT departments are chronically understaffed and may not have the skills for this kind of activity. There can also be a lack of interest on the part of IT departments to do this work, due to conflicts between IT and its users. IT is not motivated to collect new requirements when it has difficulty meeting current needs; operational departments, on the other hand, are sometimes sceptical of IT's sincerity and ability to deliver new functionality. Consequently, requirements definition is often outsourced.

Outsourcing, however, can result in a consulting boondoggle. Unless both outsourcer and client are highly motivated, it is relatively easy to generate an impressive requirements definition that is ignored. Many companies have experienced consulting engagements that produced large and costly documents but led to no deployment, or to a deployment that was not effective. These kinds of experiences lead companies to believe that requirements documents are not a very solid deliverable and so are not worth the investment.

The suspicion that requirements are a boondoggle is buttressed by emerging notions that challenge the effectiveness of a traditional requirements process. Extreme programming suggests to companies that requirements can and should be done at the same time as development [1]. The term 'lightweight requirements' carries an implicit pejorative against the traditional process, which becomes 'heavyweight' by definition.[1]

A thorough requirements process is onerous for many firms even when the work is outsourced. The overhead of learning requirements lexicon, of becoming skilled in diagramming techniques, and of purchasing and learning computer aids for developing requirements (such as UML modelling tools) is too high for many organisations. More importantly, companies find it difficult to commit much of the time of senior managers to this kind of work. Some companies suffer a malaise that might be called *requirements fatigue*. The first time a requirements definition is done it may be perceived as interesting and useful, but many companies have had to endure multiple occurrences of requirements or requirements-like interviewing for ISO certification, project management software, customer relationship management software (CRM), collaboration tools, human resources systems (HR), enterprise resource planning systems (ERP) and so on. Every new deployment starts with weeks of interviewing, self-analysis, process mapping and diagramming, often with the same managers involved in roughly the same kinds of interview. It should not be surprising if those managers quickly come to find the requirements gathering process tedious, and hence their contributions are not well focused. Where process maps and operating procedures are already well documented, up to date and conveniently to hand, then it is likely that some kind of requirements or business process re-engineering activity has recently taken place. As every requirements gatherer knows, however, these documents are often only the veneer covering the real business processes at work in the company. In a slimmed-down requirements process, it is very difficult, as well as politically unacceptable, to spend time verifying the existing process maps and procedures.

Companies suffer requirements fatigue, and they are also tired of vendor selection, a phase of project deployment in which requirements are often discovered (because one or more vendors provide features that people want) or modified (because it becomes clear that no vendor provides what was thought to be an essential feature). Selecting a vendor can take months if it is not aggressively pursued, and without careful preparation the selection process may be little better than random selection. Companies rightly suspect that the length of time devoted to this phase of the project does not lead to a measurably better choice of vendor, and so seek to shorten or eliminate this phase. In the extreme case, requirements definition becomes the output of product selection, rather than its driver. More than once we have been told, 'whatever product X does, that is how we will work' (this philosophy is commonly encountered during deployments of new ERP systems, which users assume will monolithically define the company's work processes). Another common view is that 'all the products in this area are essentially the same, so let's just pick any of them and get started'.

Another shortcut to requirements is to pick a standard. Clients will sometimes claim that 'so long as it's

[1]The technical differences between lightweight and traditional requirements are unimportant to those looking for a reason not to write requirements. The *idea* of lightweight requirements is sufficient reason to avoid a comprehensive requirements process.

compliant with StandardsBody #12345, we will accept it'. Standards bodies have made several attempts to define a set of functions that should be provided by EDM/PDM systems, including the Association for Information and Image Management's DMA and ODMA standards, ISO's STEP (10303), OMG's PDM Enablers and PDM XML. Some of these standards are codifications of existing practice, while others are attempts to define what good practice should be like, with the hope that organisations and vendors will adopt it, thereby validating the standards effort. The problem with the standards-based approach to requirements definition is that few vendor systems fully implement any of the standards.

Requirements definition is always a political process. People have significant disagreements about whether a system is needed, what it should do when and if it is deployed and which vendor is the best one. A successful deployment may be a common good, but it is typically viewed as a particular evil by at least one person or group – whose main objection to the process may simply be that they are not leading it. The longer and more public the requirements definition process, the more chance for the project to become derailed or for resistance to the project to crystallise. Thus, there are usually good political reasons to keep the requirements definition phase short.

All these factors shrink the window for requirements definition, and turn the focus of the requirements phase to determining differentiating factors between vendors: what might be called 'lightweight vendor selection'. The basic requirements question, 'What should the system do?', is often replaced with the question 'Tell me what some available systems do, and then I will tell you which one best meets my needs.'

## 3. Security Requirements

If the requirements process is going through a downturn in perceived importance, it should not be surprising that the time and effort spent on security requirements are being squeezed as well. This squeeze constricts any attempt to deal with the many special problems that exist in collecting security requirements.

### 3.1. Lack of Knowledge of Security

The first and simplest problem in collecting security requirements is that business process owners – the manufacturing and engineering personnel who actually conduct the business of the company – don't know much about computer security. Business process owners are very well informed about their data and work activities, they have an acute understanding of what improvements should be made to their processes, and they also understand that trade-offs need to be made in design; thus, requirements interviews on these topics are focused and fruitful. This is not the case for security requirements. Most engineers and managers do not consciously distinguish between authentication, authorisation, privacy and data integrity, nor do they have much knowledge of the mechanisms required to reach these goals, other than the vague notion that improved security involves digital certificates and firewalls. Few of our clients have thought it necessary to conduct a risk analysis to determine just what level of security is required by their organisation, or what cost would be borne in providing it.

There are several reasons why people don't know much about security. First, security is not a primary responsibility or interest of most people. Second, security is a very complex area that is changing rapidly, and so it is hard to keep up with security concepts and tools. Third, learning from past experience is limited by the strong tendency to minimise knowledge of security breaches [2]. Lastly, security is viewed as a system characteristic that is purchased, rather than one that derives from a mixture of mechanism and policy. Some requirements texts support this view, rating security as a 'non-functional' requirement similar to performance or reliability [3,4].

Lack of knowledge about security means that an educational process needs to be conducted before security requirements can be gathered. The small window for requirements leaves little time for such an educational process.

### 3.2. Security is a Negative Goal

It is more difficult to collect security requirements than some other requirements because security is a *negative goal* – the goal of avoiding a certain kind of result, rather than the achievement of a result [5].

Negative results are hard to measure. A secure system stops or deters security breaches, but how does one measure security breaches that were deterred? A firewall log captures information about attempts to access a system, but not every one of the disallowed accesses is a non-breach in security, nor are all the security breaches captured. Cost is not an adequate metric, since the cost of a secure system is paid whether there are attempts to breach it or not.

Negative results require an investment that few want to pay. An improvement in the security of a system is generally not enjoyed by users – their workstations are

locked down, they are required to use lengthy passwords that need frequent changing, their access to websites is restricted, their wireless network hubs are removed, and their access to and use of information is logged. These measures directly reduce the ease that people feel in doing their work, and in return they gain what appears to them to be a nebulous benefit. Contrast this with an investment in another 'non-functional' requirement, system performance: an investment in improved system performance often has an immediate and noticeable pay-off that can be measured (for example, logs may show that users have faster access to documents and hence make better and more frequent use of them).

Negative goals are poorly rewarded. A system administrator who provides 99% uptime will not be congratulated every day that the system is up, but will be under severe pressure when the system is down. Similarly, a security administrator will not receive daily kudos for doing a good job, but will be the one on the spot when systems are compromised. Paradoxically, the better that systems perform, the more people come to view this performance as typical – and one doesn't reward typical behaviour, one only rewards exceptional behaviour. Hence, the better you achieve a negative goal, the more likely it is that the only feedback you get is negative.

Negative goals are not attractive to people, and so they have less interest in being associated with them. Consequently, fewer resources will be invested in determining the requirements for negative goals.

### 3.3. The Key Differentiators Approach Doesn't Work

Earlier it was noted that the requirements process is sometimes whittled down to determining 'key differentiators' between vendors. In this approach, a full requirements document is not produced; instead, only the key differentiating requirements between various products are examined. This approach to requirements is founded on the following assumptions:

- The user's requirement for standard functionality is (thought to be) not significantly different from that provided by the vendor.
- All vendors are (thought to be) more or less equal in providing standard functionality.
- Attention should be paid to those requirements that are of most interest to the company or that will involve the most cost savings or increase in productivity.

Typical key differentiators in the EDM/PDM field include such things as:

- support for a specific ERP system, CAD package or viewing tool;
- popularity of the vendor with other companies in the same industry;
- specific kind of functionality provided by the vendor;
- vendor's product employs technology that is well understood or preferred by the company's IT staff.

When competently done, the key differentiators approach is effective in slimming down requirements definition and vendor selection. But the key differentiators approach is not a good method for identifying security requirements for EDM/PDM. Most vendors provide roughly similar authorisation capabilities, and it is hard to know whether system differences will be relevant in the eventual design. Information about authorisation methods is hard to gather; there are no comparative analyses of vendor authorisation mechanisms, there is no database of past history about vendor security and there are no commonly used benchmarks in this area. Consequently, differentiating vendors would require extensive investigation of their authorisation mechanisms, and there is no time for this in a slimmed-down requirements process.

The key differentiators approach depends on examining properties of one or more systems. A theoretical reason why this approach can't work for security is that security probably isn't a property of a system [6]. Formally speaking, properties of systems are characteristics that can be judged from a single trace of a system; many security policies involve predicates on sets of executions, and hence are not properties.

A practical limit to the usefulness of the key differentiators approach for security is that security results from a good deployment, and is not solely a function of the vendor's underlying capability. No matter what the vendor's capability, a badly designed and deployed system will be insecure.

### 3.4. Division of Responsibility for Security

A critical problem in gathering security requirements is that in most companies responsibility for security is divided:

- The IT department is responsible for acquiring and deploying security mechanisms, including firewalls, authentication systems and other technology.
- The legal department (or the executives of the company) are responsible for setting policy: for defining categories of intellectual property and rules about who can operate on data.
- The business process owners are responsible for operating the mechanism according to the policy.

This division of responsibility seems reasonable, but has important negative effects on security definition.

The first problem is that the groups are not equally well informed about security. Generally the IT department is the most knowledgeable about specific security mechanism and problems, as well as having intimate familiarity with any computer security breaches that have come to light. IT naturally concludes that the mechanisms that it can obtain define both the problem and its solution. IT's special knowledge (whether real or assumed) of this highly technical area confers authority and power, and thus security policy is heavily influenced by the IT department.

The second problem is that each group is motivated to reduce its own liability.

- The legal department is no rush to approve policy, because it understands (perhaps more clearly than the other groups) that approving certain kinds of behaviour implies liability on their part if those behaviours result in security breaches [7]. Thus the legal department tends to circle the wagons around the company's intellectual property (otherwise called 'need-to-know') and limit the company's liability by constructing an artificial notion of how the company's information is or should be distributed.[2]
- IT has deployed certain kinds of authentication and authorisation, and it wants these systems to constitute security by definition, since anything else would have to be built or purchased at considerable cost and effort. Since IT wants to minimise administration costs, it generally mandates simple rules that apply to everyone, regardless of business need (for example, 'no unauthorised software of any kind on any workstation'). IT limits its liability by constructing a computing environment based on an artificial notion of how work should be done.
- In order to limit their own liability (and to 'conduct business'), business process owners obey the letter of the security policy but sometimes violate its spirit; for example, people will resign themselves to long passwords but express their resentment at the rule by writing the password on a sticky note pasted on the bezel of the monitor. Business process owners will also get around IT department limitations by purchasing their own workstations and servers, and running 'illegitimate' software on machines they control. The consequences for actual security are fairly obvious.

Division of responsibility makes it difficult to know who to ask about security requirements. Ask a business process owner about security classifications, and you may be referred to the IT department security group. The IT department's security group's policy is often 'under development', or the executive or legal department may be currently reviewing it. If there have been no recent security breaches, then there is little urgency to complete the policy review. Paradoxically, if there *has* been a recent security breach, then policy review is considered essential but will be slowed down because no one wants to make a mistake at this point. Buck-passing can be a nested process; the local IT department may be waiting for the corporate IT department to set general IT security policies before it writes its own rules. Meanwhile, corporate IT is waiting for corporate R&D to evaluate systems for doing authentication before it sets policy. The legal department may know of or anticipate a corporate merger, sale of the business or other change of management in the near future – hence, decisions about security policy will be put off until the new administration is in place. Thus, there is often no firm policy that can be counted on to remain unchanged during system design and deployment.

The division of responsibility also makes it difficult to decide whom to blame when a security breach occurs. Was it that the legal department wasn't clear enough about the policy, and so managers couldn't enforce it? Or was it that the IT group didn't choose the right mechanism, or not recommend the right set of options on which a policy could be based? In most cases, the business process owners are the ones most directly involved in the security breach, and so they take the blame. As in other normal accidents, it is all too easy to put the mistake down to 'operator error', even when the operator is working with a system that encourages error [8].

Division of responsibility for security severely complicates requirements gathering. There is no one person or group to go to for a definitive statement about security, and in most cases no such definitive statement exists. Sometimes the requirements gatherer must actually cause the security policy to come into existence.

## 3.5. Implementation Concerns

Many EDM/PDM systems have security holes and problems of various kinds that should be understood when doing security requirements. In general, these problems spring from two characteristics. First, EDM/

[2]One company developed a simple Visual Basic application to submit an electronic request for a drawing package. They were told by the legal department that they had to remove this application since otherwise 'any contractor could just request a drawing package'. When it was pointed out to them that in the course of normal business, contractors had for years been able to simply walk down to the print shop and fill out a paper form to request copies of drawings that they needed, they were told, 'Well, you shouldn't allow that.'

PDM systems are closed-source efforts that implement their own authorisation mechanisms; hence, they have many of the problems of systems that attempt to achieve security through obscurity [9]. Second, EDM/PDM systems are more like application libraries than they are turn-key solutions. Thus, not only must we worry about security in the basic product, we must worry about security holes introduced through customisation and integration. There are a variety of places where problems can be introduced.

One typical hole is found in systems that have had a search engine 'bolted on the side' to provide Internet-like full-text search capability. When the search engine is not integrated with the authorisation model, it is sometimes possible to know that data exists and to view some of the content of data even if one is not officially authorised to access it.

A second problem area is in printing and scanning. EDM/PDM systems generally do not come with software for batch printing, application of watermarks and banners, generation of multiple rendition formats, and other device-specific software that is essential to a user's notion of what a document management system ought to do. This software is typically:

- customised or locally developed; hence not integrated with the security model;
- arcane; hence, it is difficult to evaluate its impact on security;
- produced late in the deployment cycle; since there is pressure to get the software 'out the door', bugs are likely;
- full of interactions with intelligent devices that can be points of entry for a determined hacker.

A third area for problems is in interactions with ERP systems. PDM systems maintain information such as bills of material that must be transferred to the materials resource planning component of an ERP system. The transfer of PDM bills to ERP systems usually updates data, and so the (typically) third-party software that engages in the transfer requires substantial access permissions. This data transfer is often unprotected against spoofing.

Lastly, the user interface of an EDM/PDM system tends to have security problems. Almost no company likes the out-of-the-box interfaces of EDM/PDM systems, and most undertake extensive and elaborate development of a custom interface that is typically Web-based. The security of the interface (which includes the transfer of authentication information) is subject to the quality of the company's developers or the outsourcers they hire to do the job. Systems in the recent past have been known to transfer authentication information in the clear or store transaction IDs in cookies and other vulnerable locations.

### 3.6. Intelligent Objects

Objects managed in an EDM/PDM system may have security capabilities of their own. Adobe's PDF, for example, supports password-protected authorisation controls on certain operations such as modification, printing, content selection and annotation. These authorisations are controlled through Adobe Acrobat, rather than through the EDM/PDM system's authorisation model. More recently, systems have been designed which implement online authorisation requests from distributed objects; examples of such systems include IBM's Cryptolopes, Xerox's ContentGuard and Authentica's PageRecall. There are also new systems proposed to control access to electronic entertainment media [10]. Collecting requirements about these systems and attempting to incorporate them into an overall model of security is challenging.

### 3.7. The Authorisation Problem

Earlier we stated that EDM/PDM systems have a substantial requirement for authorisation functionality. We quantify this claim through the use of the following notions:

An *authorisation specification* is a three-dimensional matrix of people, objects and operations. If the value of $(x, y, z)$ is 1, then person $x$ can apply operation $z$ to object $y$.

An *authorisation problem* is the problem of populating an authorisation matrix so that the following characteristics hold:

- We have reasonable guarantees that the matrix is correctly and completely populated.
- The matrix can be maintained in a reasonably efficient manner.

Note that the authorisation matrix is a theoretical notion introduced here to discuss authorisation without discussing actual mechanisms, such as mandatory access control or role-based access control. Authorisation in EDM/PDM systems is not done by populating such a matrix directly, but through the use of standard techniques such as roles, groups, policies and other mechanisms.

In a typical EDM/PDM deployment, the authorisation matrix is potentially very large. The number of objects $y$ can easily be in the millions. A typical manufacturing firm maintains data on $10^5$ parts and $10^6$ component-

assembly relationships. Each part may have several associated documents (specification, geometric drawing, parts list, schematic, gerber file, application drawing, etc.). Parts typically exist in multiple revisions, each of which will have a drawing and usually one or more engineering change orders. Some companies also maintain information on serialised parts – thus, $y$ is potentially as large as the number of products made by the company.

The number of operations $z$ is larger than the standard three (read–write–execute) of common file systems. A typical EDM/PDM system supports 30 or more different operations, some of which may include:

- create – generate a new object;
- read – read the attributes of an object;
- view – read the files associated with an object;
- modify – edit the attributes of an object;
- revise – generate a revision of an object;
- version – generate a version of an object;
- print – allow the files associated with the object to be printed;
- check-out (check-in) – export (import) files associated with the object;
- delete – delete an object;
- lock (unlock) – disallow (allow) object modification;
- promote (demote) – move the object to the next (previous) stage in its life cycle;
- grant – allow other users privileges similar to one's own;
- set privileges – change any of the authorisations for an object;
- link (unlink) – create (delete) a certain kind of relationship to another object;
- execute – execute one or more programs associated with an object.

Although the number of users in an EDM/PDM system is not excessively large, each of the users can belong to multiple groups and can participate in multiple roles. A 'group' is a set of persons that is identifiable for some business purpose, while a 'role' is a set of permissions that can be assumed by a user or a group for operation on some objects. Many EDM/PDM systems treat a group or a role as a substitute for a person. In other words, $x$ can be set-valued, with sets chosen from the set of persons, or as combinations of other roles or groups.

Workflow adds another dimension to the authorisation matrix, that of state: person $x$ can apply operation $z$ to object $y$ only when that object is in state $\alpha$. The set of states typically includes such things as 'draft', 'in work', 'review', 'release', 'superseded' and 'obsolete'. It is not unusual to have different sets of states for different kinds of objects; for example, parts and drawings will be subject to formal release control, but documentation may be handled more informally.

In a typical case the authorisation matrix can be of the order of $10^{12}$ entries. But recent business trends threaten to make the matrix even larger and more complex.

## 3.8. The New Economy

In the past, the authorisation problem was made tractable through the use of simple rules that dramatically reduced the size of the authorisation matrix:

- Everyone who works for us has read access to everything.
- No outsider has any access at all.
- All other operations are permitted only to a small group that does all data entry.

Under these rules, $y$ is reduced to 1 (because all objects are treated identically); $z$ is reduced to 2 ('read' and all other operations); $\alpha$ is reduced to 1 (there is only one state) and $x$ is reduced to 3 (internal, public and data entry).

This simple scenario is unworkable for modern firms. Companies are highly interested in participating in joint ventures, downsizing, e-commerce, Web portals, and other modes of work that directly impact the size and complexity of the authorisation matrix.

The simplification 'everyone who works for us has read access to everything' disappears as the notion of 'us' is becoming more complex. Manufacturing companies are increasingly less vertically integrated, and are moving towards contracting, collaboration and joint ventures as their principal mode of business.

Contractors should be shown only the data they need to do their job, but not other data. It is important to limit contractor access to information for many reasons. One is that they may also work for competitors (and hence, could transmit proprietary information); another is that access to too much information enables them to bid more effectively on new work (and hence, they may cost more to hire than they would otherwise). Contractors need more than just read access to data, because they are often actively generating new engineering and manufacturing information; however, only limited write permissions are appropriate. Workflow also becomes more important because contractors may not be physically on site, and may even be in different time zones.

Collaborative ventures are undertaken with a variety of firms, including current and future competitors. While collaborators should have access to data related to the project on which they collaborate (with exceptions, such

as cost and margin information), it is usually important to ensure that competitors do not gain access to other projects.

Joint ventures are semi-autonomous organisations that are formed by otherwise competitive firms. A joint venture may have its own IT department with its own notions about how to do authentication and authorisation; for example, it may see less need for authorisation that do its parent companies, because the joint venture's information is wholly open to either parent; meanwhile, the parent sees a need to isolate non-joint information from the joint venture.

Important customers expect online access to information about products and their as-sold (and even in some cases, as-installed and as-maintained) configurations. Customers of course should only have access to their own configurations, and not those of other customers; nor is it desirable to give customers access to all information about their configurations, because some of this information is a component of aftermarket sales or servicing, which are important revenue streams for many companies. Customers are not monolithic entities; engineering personnel at a given customer site should have different kinds of access from accounts payable personnel. As an additional complexity, the rapid pace of mergers, consolidations and divestitures means that what counts as a single customer changes over time.

These factors increase the size and complexity of $x$ and $y$ as well as increase their rate of change. It is not unusual for a merger, a joint venture, a collaboration or some other significant corporate event to occur during the deployment of an EDM/PDM system that changes the definition of 'user', introduces new IT policies and provides new data that must be managed.

Workflow will soon not be a matter of choice, but a matter of survival. Configuration management departments that once had 40 people to manage engineering change are now reduced to four. Shuffling paper change forms is no longer a viable option – particularly when so much of the change is happening 'outside' the firm. It is not sufficient for the EDM/PDM system to simply vault data – it must also mediate the life cycle of data. This means significant increases to $z$ and $\alpha$.

The demands of the new economic climate simultaneously increase the size and the difficulty of the authorisation problem.

### 3.9. Role-Based Access Control

Role-based access control is the current best approach for defining authorisation [11]. A *role* is a permission to perform any one of a defined set of operations on a defined set of objects. Roles can be assumed by a set of people (a group) to allow them to operate on a set of objects (a category). Generally objects can belong to more than one category, and people can assume more than one role and be a member of more than one group. The authorisation matrix is then expressed as $(X, Y, Z)$, where $X$ is the set of groups, $Y$ is the set of categories and $Z$ is the set of roles (here we make the simplifying assumption that role-based access is defined in terms of categories).

There are at least three benefits to role-based access control. First, it can reduce the size of the authorisation matrix if it is the case that $|X| << |x|$, $|Y| << |y|$ and $|Z| << |z|$. Second, it can reduce redundancy in the authorisation matrix: instead of specifying independently that individuals $x_1, x_2, x_3, \ldots, x_n$ can each perform operations $z_1, z_2, z_3, \ldots, z_q$ on objects $y_1, y_2, y_3, \ldots, y_m$, we define sets $X_i$, $Y_i$ and $Z_i$, and establish a single authorisation constraint between them. Third, it also simplifies updates; in the example given above, adding a single user to the set $X_i$ automatically gives the user access to all the objects $Y_i$, thus providing a multiplicative benefit under update. However, there is a corresponding downside: any error in defining the sets also results in a multiplicative defect. If person $x_k$ is mistakenly assigned to role $Z_i$, then that person gains greater access than would be the case if a single error occurred in a pairwise assignment of constraints. Thus, role-based access control can reduce the cost of populating the authorisation matrix, but can also increase the severity of mistakes.

Mistakes in assigning persons to roles and objects to categories are common. Security classifications are fundamental to security, but organisations often do a poor job of establishing workable classifications. Consider the ease with which managers might confuse such categories as 'unclassified controlled nuclear information' and 'sensitive but unclassified nuclear information' [12]. Or consider the distinction between 'classifying' data and 'categorising' it: 'PARD: Protect As Restricted Data' is not a classification level per se but 'a handling method for computer-generated numerical data or related information, which is not readily recognised as classified or unclassified because of the high volume of output and low density of potentially classified data', and ranks between unclassified and confidential, the lowest level of classification' [12]. In general, categorisation for security purposes is considered one of the most problematic areas of governmental security [13]. Defining categories is less difficult than is ensuring that managers apply them consistently and correctly. One Fortune 50 company has only four security classifications for documents, but its managers were not even able to give consistent definitions of these classifications.

Role-based access control is less compelling in situations where the role doesn't completely define authorisation across most objects. Consider for example that 'contractor' appears to be a role, and it is usually true that there is some information that no contractor should see, but on the other hand each contractor needs read and modify access on distinctly different sets of data. In such circumstances, there may be as many contractor roles as there are individual contractors, and so role-based access control does not reduce the size of the authorisation matrix, although it may still provide other benefits.

## 4. Discussion

### 4.1. Using Database Normalisation Theory

The normalising effect of role-based access control suggests a similarity to database normalisation. A good database design is a set of database schemas that controls redundancy and ensures that updates will not result in inconsistent data – much as a good role-based structure controls redundancy and makes update more efficient. Classical database design is done by data normalisation – the reduction of redundancy by splitting relations according to data dependencies. There is a considerable body of work on normalisation, including definitions of normal forms, algorithms for achieving normal forms, and complexity results [14]. A full exploration of data normalisation is beyond the scope of this paper, but the following gives some idea of how this technique can be applied to the authorisation problem.

If we define an authorisation problem in terms of attributes, schemas and dependencies, we can then apply known normalisation techniques to arrive at a normalised schema that captures the data elements in as concise a manner as possible, and minimises the update problems that can arise. We first note that the statement of the authorisation problem given in Section 3.7 does not capture the full generality of the problem. A more complete specification would take into account the following rules:

1. People can assume Roles, and the Roles determine access to a set of Objects. For example, an author may write while a reviewer may only read. Moreover, we may have an exclusion property – for example, that authors can't be their own reviewers.
2. People also belong to Groups, and Groups restrict Object access. For example, an Employee can read some Objects a Customer may not.
3. An Object may comprise multiple Files, and these Files have different access permissions. For example, a source format file may be both copied and written, while an archive format file may only be copied.
4. An Object exists in different Revisions, and different Operations can be applied to these Revisions. For example, an engineer may be able only to read the current Revision, but both read and write a new (unreleased) Revision.
5. Objects belong to one or more sets known as Categories.
6. Operations belong to one or more sets known as Capabilities.
7. The State of the Object determines the Operations that can apply to it. For example, a released Object may be read-only, but an in-process Object may be both read and written.

From the database design perspective we have identified the following attributes:

- `Person`
- `Group`
- `Role`
- `Transaction`
- `Object`
- `Revision`
- `File`
- `State`
- `Category`
- `Operation`
- `Capability`

Note that the authorisation matrix resulting from this set of attributes is considerably larger than the simple three-attribute matrix given in Section 3.7.

Database designers will recognise the authorisation matrix as an instance of the *universal relation* – that is, a single table with as many columns as there are attributes. The universal relation contains a significant amount of data duplication and some update problems. The goal of database design is to decompose this table into multiple tables that eliminate the redundancy and avoid the update problems.

The decomposition is based on dependencies that exist in the data. There are several kinds of dependency that can be eliminated, but we will focus here only on *functional* dependencies. A functional dependency

$$X \rightarrow Y$$

exists between two sets of attributes X and Y if for every value of X there exists only one value of Y. The set of dependencies that exist in an authorisation problem is case-specific, but typically the following kinds of functional dependencies occur:

**FD1** `Object, Revision → State, Transaction`

An Object of a given Revision can be in only one State and participate in only one Transaction at a time. In practice, Objects should probably be allowed to participate in multiple Transactions, but only one Transaction should permit modification of the Object. Different Revisions of an Object may be in different States and Transactions.

**FD21** `File → Object`

A given File should belong to only one Object. An Object may have more than one File.

**FD3** `Person, Transaction → Role`

A Person can only take one Role per Transaction. A Person may take different Roles in different Transactions.

**FD4** `Person, Object, Revision → Capability`

The set of Operations a given Person can apply is determined by an Object and its Revision.

We associate Capabilities with Groups and Categories as well as with Persons and Objects, leading to one or both of the following additional dependencies:

**FD5** `Category, State, Group → Capability`

**FD6** `Category, State, Role → Capability`

All Objects within a given Category in a given State permit a specific set of Operations to a Specific Group (or Role). Different Groups (or Roles) may have the same set of Operations on a given Category in a given State.

Given the above we can produce the following decomposition of the authorisation matrix. Each S*i* is a relation; the underlined attributes are the primary key of the relation:

```
S1 (Object, Category)
S2 (Person, Transaction, Role)
S3 (Person, Group)
S4 (Operation, Capability)
S5 (Category, State, Group, Capability)
S6 (Category, State, Role, Capability)
S7 (Object, Revision, State, Transaction)
S8 (Person, Object, Revision, Capability)
S9 (File, Object)
```

This schema is a normalised form of the authorisation matrix that reduces the amount of data that must be stored and reduces update anomalies.

No commercial information system allows direct population of its authorisation mechanism from a design like the one above. The value of the design is in the following characteristics:

- By separating design from mechanism, designers are able to develop authorisation logic independent of any particular choice of vendor.
- Expressing authorisation constraints as functional dependencies leads to more precise specification than found in simple use cases.
- Database design theory provides algorithms for analysing and combining dependencies, and can thus reduce the set of constraints.
- Designers can deploy the above schema using a standard database, as part of a pilot or prototype system used to qualify, demonstrate or test the authorisation system.
- After the EDM/PDM system is deployed, the schema can be used as an independent test vehicle to vet the operation of the EDM/PDM system.
- In cases where the need is great, the EDM/PDM system may be customised to take its authorisation matrix directly from the normalised schema.

The example shown here is a relatively simple translation of functional dependencies to relations, and it does not capture other dependencies that typically exist in a realistic authorisation problem. Groups, for example, are often organised in a containment hierarchy so that supergroups can be defined as the union of subgroups. As the authorisation structure becomes more complex, the use of the formal techniques become more necessary for managing authorisation.

## 4.2. A Common Notion of Things

A desirable authorisation system is one that can be applied to a wide spectrum of information resources. One barrier to a wide-spectrum authorisation system is the lack of a standard notion of *things*. All systems share a notion of persons, expecting them to be unique and stable entities, and so authentication systems at least have a common base that facilitates the concept of single sign-on (although looming on the horizon are issues such as how to separate human clones, how to safeguard bio-identification techniques against advanced surgery and genetic techniques, and so on). But authorisation deals with computer objects, and what constitutes an 'object' depends largely on which software package you use: operating systems deal with a universe of files, relational databases deal with a universe of tables and rows, ORBs deal with a universe of CORBA-compliant objects, intelligent objects deal with pages or other sub-elements,

Web servers deal with a universe of URLs, and EDM/PDM systems control many of these elements and add workflow as well. The 'impedance mismatch' between these universes makes it difficult to have an authorisation mechanism that spans them all.

One way to reduce impedance mismatch is to funnel all data through a single namespace. This is a standard technique: consider, for example, mapping objects such as sockets into a file system namespace. Today, more attention is given to the URL namespace. Netegrity and Dascom are two companies with products that provide access control to a URL namespace; if one can assign URL 'names' to all objects, then authorisation for these objects can be defined through Netegrity or Dascom. This kind of approach is useful for data access, but is less satisfactory for workflow and other operations that require update or other operations. Another namespace possibility is an object-based namespace, possibly with something like the CORBA security standard [15]. The main problem with this type of solution is that most organisations deploy many architectures, not just CORBA.

### 4.3. Architecture of an Authorisation System

From a system management point of view, it seems reasonable to have a centralised authorisation matrix, because this reduces administrative overhead. From a security point of view, however, there is an argument for *not* centralising authorisation. A centralised authorisation system suffers the problems of any centralised system: it is more susceptible to failure and error. Redundancy is a common approach to increasing robustness. In some authorisation contexts, redundancy is provided by the *two-man* rule. An ICBM launch, for example, requires the coordinated activities of a number of distinct people; this redundancy (in people) tends to make the system as a whole more robust to failure in any single component (such as error or malicious intent on the part of one of the persons). Given the possibility of authorisation failure, it may be useful to consider deploying redundant authorisation as much to overcome human error as to overcome system error.

It is also worth studying the distinction between authorisation and authentication. One proposal merges authentication and authorisation by incorporating authorisation information in a certificate used for authentication [16]. A problem with this approach is that authorisation is trending towards a time-sensitive, rule-based property, rather than an unchanging attribute of a single individual. An alternative approach that works in some contexts is to relinquish the need to authenticate a person's identity, and instead authenticate their access permissions, as is done in trust management systems. This kind of approach is unknown to most vendors and consumers of EDM/PDM products today.

### 4.4. Limits to the Requirements Process?

The experience of EDM/PDM deployments suggests that more academic thought needs to be given to justifying the value of the requirements process. Requirements specialists claim that a comprehensive requirements process reduces the possibility of system failure, yet companies continue to show little faith in the requirements method as a means of producing successful deployments. Has industry not given requirements sufficient effort? Or are there inherent limitations in the requirements process that have not yet been studied by academics, but which are empirically known to industry?

One simple question that industry continually faces is how much requirements analysis is enough. The simple answer is to do requirements until you're done – but even this is simple only if there is an efficient test for being done. A more realistic answer is that there must be some non-trivial relationship between the time spent on requirements and the return achieved from the analysis; indeed, at some point it must be the case that the marginal value of an additional day of requirements analysis is less than the benefit that would accrue (for if there were no such point, then we should continue to do requirements forever). Requirements specialists are interested in requirements and view requirements as an end in themselves; this would tend to make their threshold of marginal value low. Businesses must constantly justify expenditures in terms of outcome, and so their threshold is relatively high.

It would be interesting to know the ultimate limits to the requirements process. Even a comprehensive, well-done requirements statement has some residual error, uncertainty or lack of future knowledge that limits its correctness. It would not be surprising if in fact a more comprehensive requirements process actually introduced new error, by leading designers astray or by introducing features that themselves are error prone. This kind of effect is well documented in a field related to security: safety and accident theory. Normal accident theory teaches that safety failures cannot be foreseen or prevented in tightly coupled, highly complex systems, and that safety mechanisms sometimes reduce, rather than increase, safety. An illustrative example is how shipborne radar can result in collisions that would not have otherwise occurred: with radar, some ship captains feel able to travel at high speed in marginal weather, or indulge in anticipatory course corrections rather than

simply follow a safe base course [8]. Safety analysis is not really designed to uncover these kinds of failure mode, and requirements analysis is almost certainly similarly limited. A better understanding of the limits of requirements analysis would help businesses understand how much effort to put into requirements.

## 5. Recommendations

In the world of medium-sized software projects, the need for requirements definition is expanding, but the appetite for doing requirements is contracting. This is as true of security requirements as of any other kind. The security problems enumerated in this paper could, by themselves, easily consume the entire time that is typically available for requirements definition – but of course the time must be spent on all other aspects of the system as well. Thus we should not be surprised if security requirements are often poorly captured

In EDM/PDM deployments, we counsel business process owners to focus their limited time and resources on authorisation. Although authentication gets more attention, many security breaches are really problems in authorisation. The notorious case of Wen Ho Lee at Los Alamos is a recent example. Lee was not charged with an authentication violation (that is, pretending to be someone he was not, or attempting to gain access to information to which he did not have legitimate access), but with transferring material from a secure computer to a non-secure one – that is, with carrying out an unauthorised operation [17]. Other areas that are unlikely to be addressed by authentication are 'social engineering' techniques (used to obtain passwords by misleading individuals who know those passwords), and information thefts committed by insiders. Preventing these kinds of leaks requires more powerful authorisation, not better authentication.

Authentication, data integrity and privacy are not the direct responsibility of business process owners, but authorisation is. Business process owners must come to grips with authorisation, because they will make authorisation decisions on a day-to-day basis. Deciding who gets access to what information is a business decision, not a legal or technical one. When access to information is itself a product (as it is for many companies that make a significant profit on information about their products), the authorisation system becomes an obvious part of the production and distribution system for information.

Categorisation is not a complete answer to the problem of authorisation, whether it is used in a mandatory access control scheme or in role-based access control. Experience shows that even relatively small categorisation schemes can be misinterpreted and misapplied, while complex schemes are often subverted, thus invalidating any reasoning that might be done based on the assumption of proper use of categories. The simplest possible categorisation should be used, and steps must be taken to audit the use of the categorisation scheme.

Most EDM/PDM systems have a large variety of mechanisms for providing authorisation, but little or no functionality for assuring the result – few even allow the exporting of the matrix so that it can be studied with external tools. As a result, reasoning about the design and effectiveness of such systems is typically done informally. Application of database design techniques may help with matrix construction and provide some route to assurance. Organisations will still need to develop their own test plans and methods for auditing the matrix on a regular basis.

EDM/PDM systems are tightly coupled and highly complex systems that assuredly have failure modes that are yet to be discovered. Rather than drawing up tedious contractual-style requirements documents that attempt to legislate security from vendors, it may be better to develop a collection of 'misuse cases' that describe possible security failures, and use these to analyse systems, to consider the risks and responses associated with failures and to test vendor products in realistic scenarios. A business armed with this information will be much better prepared to make a business case for more secure systems, and will be much better prepared to deal with their inevitable failure.

## References

1. Beck K. Extreme programming explained: embrace change. Addison-Wesley, Reading, MA, 1999
2. Sagan SD. The limits of safety: organizations, accidents, and nuclear weapons. Princeton University Press, Princeton, 1993.
3. Robertson S, Robertson J. Mastering the requirements process. Addison-Wesley, Reading, MA, 1999
4. Wiegers KE. Software requirements. Microsoft Press, Redmond, WA, 1999
5. Dörner D. The logic of failure: recognizing and avoiding problems in complex situations. Perseus Press, Cambridge, MA, 1996
6. Schneider FB. Enforceable security policies. Inform Syst Security 2000;3(1):30–50
7. Anderson RJ. Liability and computer security: nine principles. In: European symposium on research in computer security. LNCS, Springer, Berlin, 1994
8. Perrow C. Normal accidents: living with high-risk technologies. Princeton University Press, Princeton, 1999
9. Garfinkel S, Spafford G. Practical Unix and Internet security. O'Reilly & Associates, Cambridge, MA, 1996
10. Content protection system architecture: a comprehensive framework for content protection. Intel/International Business Machines/Matsushita/Toshiba, 17 February 2000

11. Sandu RS, Coyne EJ, Feinstein HL, Youman CE. Role-based access control models. IEEE Computer 1996;29:38–47
12. Science at its best, security at its worst: a report on security problems at the US Department of Energy. Report of the Special Investigative Panel of the President's Foreign Intelligence Advisory Board, June 1999
13. Smith JH. Redefining security: a report to the Secretary of Defense and the Director of Central Intelligence. Joint Security Commission, 28 February 1994, US Dept of Defense
14. Dutka AF, Hanson HH. Fundamentals of data normalization. Addison-Wesley, Reading, MA, 1989
15. CORBA. Security service specification V1.2. December 1998, ch 15
16. Rubin AD, Geer R, Ranum MJ. Web security sourcebook. Wiley, New York, 1997, p 317
17. Schwartz SI. Scientist, fisherman, gardener ... spy? Bull Atomic Scientists 2000;56(6):24–30