Journal of Management Systems 4 (1) 1992, pp. 1-12.

The Developmental Duality of Information Systems Security

Richard Baskerville

School of Management
State University of New York at Binghamton
Binghamton, New York 13902-6000, USA
(607) 777-2337

Key terms: Computer Security, Information Systems Security, Information Systems Design, Information Systems Methods

ABSTRACT

The developmental duality of information systems security results because the information system and its security are separate developments. This duality may cause conflict and tension between a system and its security. This unresolved conflict leads to a shortened information systems lifespan. In particular, these coexisting developmental methods are predominantly selected from essentially different methodological generations. A commonplace tension is found in the logical engineering of the information system development method that contradicts the mechanistic engineering method used in security development. Understanding these tensions will help the designer subdue these tensions and thus extend the lifespan of the information system.

INTRODUCTION

Systems analysis and design authorities conventionally prescribe the strong consideration of security and controls during the design of information systems (*cf.* Kendall & Kendall 1988; Whitten, Bentley & Ho, 1986; Powers, Adams & Mills, 1984). Indeed, the maturing data processing auditing practices dictate that new information system designs are candidates for internal audit review *prior* to implementation stages (*cf.* Weber 1988; or Gallegos, Richardson & Borthick, 1987). Yet such ideals in the security of information systems (IS) seem to be often overlooked, or poorly detailed in systems specification and design. Evidence of such oversights can be found in the damages of computer abuse (Straub, 1990), and in the widespread experience of information systems security practitioners:

> "What is missing in many instances is the involvement of concerned users, enlightened developers, experienced EDP auditors, experienced information security specialists, or other parties who understand how to incorporate controls into systems while the systems are still in development." (Wood, 1990, p.13).

The purpose of this paper is to describe the developmental duality of information systems security, *viz.*, that the information system and its security are separate developments. This duality may lead to conflict and tension between a system and its security. This conflict helps to explain the difficulties in IS controls operation and the effects of security on the abbreviation of a system's lifespan.

Summers (1984) defines *computer security* to include "concepts, techniques and measures that are used to protect computing systems and the information they maintain against deliberate or accidental threats." *Information systems security*, however, is a broader concept:

> "Information systems security envelopes computer security, just as information

TABLE 1
Summary of Generations of Methods

Generations of Methods	Primary Attribute	Examples
First Generation: Checklist Methods	Mapping known solutions onto the information problem	Vendor's technical sales procedures
Second Generation: Mechanistic Engineering Methods	Physical identification of technical solutions to functional requirements	Black-box and "bottom-up" methods, prototyping.
Third Generation: Logical Engineering Methods	Intermediate design in an abstract model	Structured analysis, data modeling, information engineering.

systems envelopes computer systems. However, information systems security must also take into account manual systems and 'human processors'. This latter facet opens up the consideration of behavioral aspects of information security, such as motivation, cognition and the role of the system in its enveloping organization. Further, security in the flow of information across the man/machine boundaries is couched in a more effective context." (Baskerville, 1989, p. 244).

This paper proposes a framework that permits a general information systems professional to view security methods in the context of more general system development methods. This framework can also provide the information security specialist with a view of how various security methods match (or fail to match) the general approaches. The distinguishing attributes used in this study focus on abstract, native processes used in the methods. One can identify first generation methods by their limited solution space, and the usefulness of this as the foundation for security reasoning. A complete reliance on design artifacts bound to the physical world distinguishes second generation methods. Third generation methods introduce logical abstraction into the design process. These methods are summarized in Table 1.

The sections below detail each generation's distinguishing attributes, general development methods and then security development methods. Computer-supported security techniques receive separate treatment in each section. Each generation is also summarized. Three final sections analyze the development duality found in security design, offer interpretive predictions founded on this analysis, and summarize the paper.

FIRST GENERATION: CHECKLIST METHODS

Checklist methods focus on the mapping of known solutions onto the problem-at-hand. These methods arise from (and depend on) a limited set of elements that may be employed in the design project.

Checklist Security Development Methods

The checklist approach to security systems design also starts with the solutions: The known controls which could be implemented. The analyst selects the best inventory of controls according to the information problem under examination.

In both early systems analysis and early security analysis, the limited range of solutions (system elements or security controls) allowed designs to be approached from the basis of "what can be done", rather than "what needs to be done". Evidence of this is found in early texts on computer security that offer a pedagogical organization based on control typology (*cf.* FitzGerald, 1978).

Checklist methods generally do not begin with a view of the risks involved. Checklist methods begin their design with an examination of known controls. A list is provided of every conceivable control which can be implemented in a computer-based system. The analyst first checks to see if the control is already in place, determines its necessity if not found, and implements the control when required. Examples of the features of these controls include disaster recovery provisions, accuracy and integrity assurance, data access restrictions and system development policies.

This encyclopedic approach underlies much of the early work in computer security, and some very impressive checklist methods are available. Hemphill and Hemphill's (1973) *Security Procedures for Computer Systems*, the SAFE Checklist (Krauss, 1972) and the early *Computer Security Handbook* (Hoyt, 1973) are good examples of the checklist approach; as well as the still widely used *Checklist for Computer Center Self Audits* (AFIPS, 1979).

Risk Analysis in Security Design

Since checklist-style security design will suggest every conceivable control, a risk analysis technique must be used to separate appropriate and inappropriate controls in each specific situation. Risk analysis defines a risk factor *R* comprised of *P*, a probability factor related to the likelihood of a mishap occurring a given number of times per year, and *C*, a cost-loss factor related to the financial impact of such a mishap. This simple risk factor is calculated as

$$R = P \times C$$

C is derived from a exponential range table, *i.e.*, \$1000, \$10000, \$100000 and so forth. For example, a mishap resulting in an estimated loss of \$20,000 (*C* = 100,000) and likely once each three years (*P* = .33) yields an *R* of \$33,000. This figure is then used in ordering the priorities of controls suggested by the checklists. Courtney (1977) pioneered this form of risk analysis for computer security and his technique is widely adopted in security design methodology (*cf.* FitzGerald, 1978; Saltmarsh & Browne, 1983). Qualitative variations also exist, including the U.S. Department of Justice (1986) Baselines Approach and the CRAMM methodology (Farquhar, 1991).

A large portion of the computer-based computer security products are automated "decision support" tools for risk analysis. Some, such as RISKCALC and IST/RAMP, remove routine calculations from risk analysis and are too narrowly directed at discrete risk calculation to be able to model security effectiveness or select controls from a checklist. Others, such as SECURATE (Hoffman, Michelman, & Clements, 1978), the Smith and Lim (1984) Approach, and the Carroll and MacIver (1984) Knowledge-Base do provide risk analysis in a scope broad enough to assist in modelling a set of controls selected from a checklist.

First generation methods are thus marked by their foundation on a limited set of possible system elements from which a subset is selected in order to solve a problem. These methods must incorporate a cost-benefit (risk) analysis in order to isolate unnecessary members of the solution subset. Checklist methods are still in use in both IS development and IS security development. Computer-based tools, including expert systems, are available to assist security practitioners who use checklist approaches. Notably, checklist security design methods require that a complete, operational system (or its specification) precede the security design.

SECOND GENERATION: MECHANISTIC ENGINEERING METHODS

Mechanistic engineering methods aim at maximized technical functionality in an information system. They differ from checklist approaches in their focus on system functionality rather than available technology. Further, these methods differ from "logical" approaches of the third generation because the system design process remains firmly rooted in a problem space that addresses the physical system requirements at all times.

Mechanistic Engineering Security Development Methods

Like mechanistic IS engineering methods in general, mechanistic engineering security methods evolved out of first generation approaches, and retain risk analysis as a key proposal component. However, these second generation methods clearly focus on system requirements over available technologies, and lack the logical abstraction phase of third generation approaches (discussed later). Many of these methods are proprietary, but Royal Fisher (1984) details a comprehensive, requirements-oriented method to the design of data security. Fisher specifically avoids the checklists that had dominated much of the previous literature on the subject. Fisher's work is relevant to one of the examples later in this paper, so we will consider this more closely.

The Fisher Approach. Fisher first stresses the need for management to set a security/asset protection organizational policy including a security administrator and a data security plan. This plan includes a vital records plan, an access control plan, an emergency response plan, an interim processing plan, a restoration plan and a data security classification program. Once the policy is in place, Fisher defines the following five steps in his design method:

1. *Define the Data Inventory*, if not already available from data base documentation.
2. *Identification of Exposures*, using the traditional six types of exposure (accidental or intentional -- disclosure, modification and destruction) mapped onto eleven data exposure control points. These are: (1) Data Gathering, (2) Data Input Movement, (3) Data Conversion, (4) Data Communication (Input), (5) Data Receipt, (6) Data Processing, (7) Data Preparation (Output), (8) Data Output Movement, (9) Data Communication (Output), (10) Data Usage, (11) Data Disposition
3. *Assess Risk*, using Courtney's risk analysis technique (see above).
4. *Design Controls*, using a taxonomy of preventative, detective, and corrective controls classes.
5. *Analyze Cost Effectiveness*, by imposing capital investment criteria on the controls and risk profiles.

Mechanistic engineering methods like Fisher's focus on technical functionality of system requirements. In security methods, this means identification of critical exposures at physical points in the system. Checklists are only useful after the critical exposures are identified, and then only for locating effective controls for the given function. Like first generation methods, risk analysis is an important feasibility criterion. In addition, second generation methods also assume a complete operational (or specified) system will serve as the subject of a security analysis and design.

THIRD GENERATION: LOGICAL ENGINEERING METHODS

Logical engineering methods are notable for an intermediate design in a functional, abstract problem space. That is, during one or more design steps, the system is removed from its dependence on physical elements and technology, and certain design problems are resolved in the abstract. Both predecessor generations described above are entirely rooted in physical system elements during design, and thus can be differentiated from logical engineering methods.

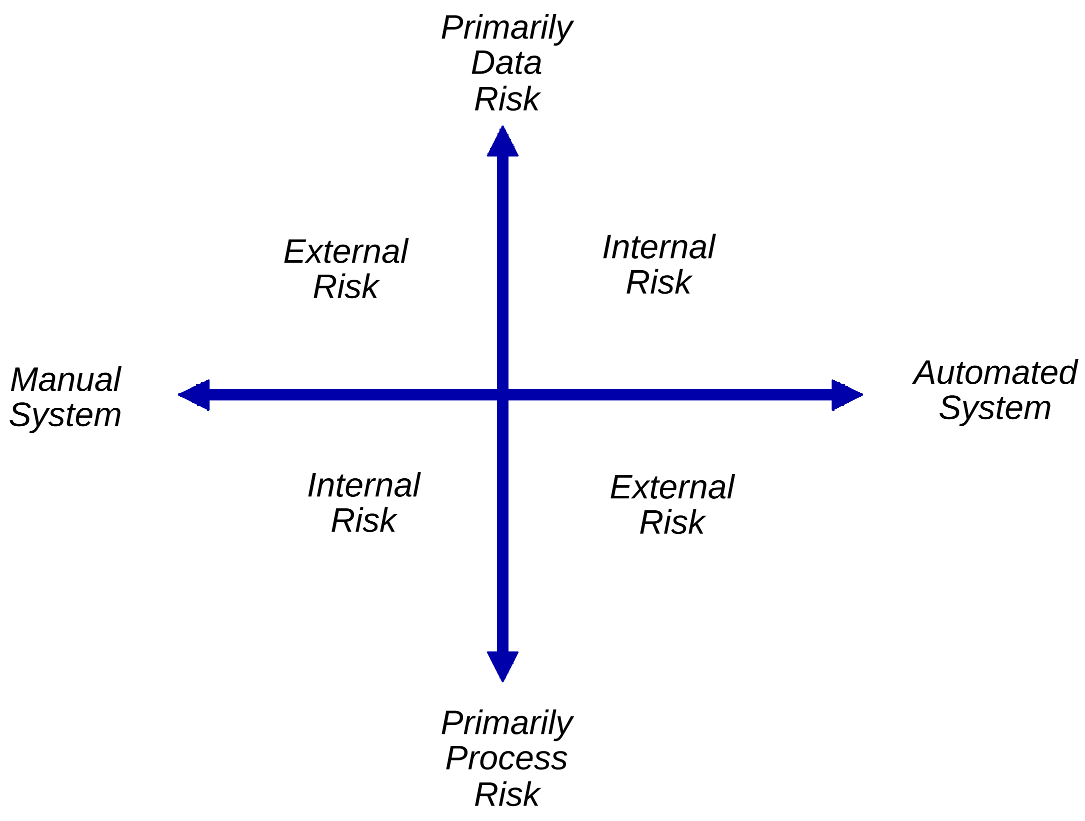
Logical Controls Design Methods

There are security approaches that shift the emphasis from physical to logical controls, for example modifying traditional structured systems analysis methods to require logical control elements (Baskerville, 1989). This approach to security design places control details in both the process model (data flow diagram) and the data model (data dictionary) of the abstract logical system design. The technique is rigorous in designing control processes and control data into the logical model, thus preventing the designer from relying completely on physical or technological security overlays. Bannon (1989) notes that such logical controls are needed. However, he finds that the approach is limited by the problems of traditional structured design in capturing the actual work patterns of human behavior.

Computer Supported Logical Engineering Methods

Most automated control design systems are components of operating system or data base protection programs (*e.g.,* RACF, TOP SECRET and ACF2). Others (such as CRAMM) support first or second generation techniques with exhaustive threats and controls databases. Some of these database systems have achieved logical model orientation by incorporating abstract logical models as the basis for a knowledge base. These would qualify as computer aids to logical security engineering. Examples include the SPAN decision support system developed at the Naval Postgraduate School (Zviran, Hoge & Micucci, 1990), and the Inversion Model Expert System (Baskerville, 1988). Since the Inversion Model is relevant to the examples later in this paper, we will examine this approach a bit more closely.

FIGURE 1
Inversion Model of Information Security



Inversion Model Expert System. This system is distinguished by its use of a dynamic model for determining the significance of controls. The model gains a certain universal strength by recognizing that, upon automation, there is an inversion of significant threat sources against certain fundamental classes of system assets. The expert system uses seventeen basic prolog rules against a security database to select appropriate controls according to this *inversion model*. This model tracks the threat severity against data and process elements in manual systems that are evolving into automated systems.

This model is illustrated in Figure 1. External risks (e.g., sabotage and espionage) are seen as primarily threats to data in manual systems, yet primarily processing threats in automated systems. Internal threats (e.g., integrity and fraud) are seen as primarily threats to data in automated systems, yet primarily processing threats in manual systems.

The knowledge base lists various resources (*e.g.*, program processes and data files), threats (*e.g.*, browsing or fraudulent changes) and controls (*e.g.*, check-digits or encryption) along with various abstract security classes of these elements. The membership of the resources, threats and controls in the abstract classes is non-exclusive. The knowledge system may identify minimum controls or maximize through several layers of controls as appropriate in the inversion model. The sample prolog program below will help clarify how this process works:

```
external(spying).
manual(paper_file).
data(paper_file).
shield(spying,paper_file,cabinet_locks).

significant(Control) :-external(Threat),
manual(Target),
data(Target),
shield(Threat,Target,Control).
```

This simplified example includes a database detailing an external threat ("spying"), a manual data element ("paper_file"), and a relevant control ("cabinet_locks") that shields this element from this threat. The rule declares that any control is significant if an external threat exists; a manual target exists; this particular manual target is a data element; and this particular control shields this particular target from this particular threat. This simple example would identify cabinet_locks as a significant control.

Third generation methods are characterized by their orientation toward abstract logical models as the primary foundation for functional security design. These approaches do not impose controls upon a physically extant, or previously designed information system, but include control consideration in all phases of system design. In a sense, first and second generation approaches embroidered or patched completed systems with security features, third generation approaches weave the controls into original fabric of the system.

THE DEVELOPMENTAL DUALITY OF INFORMATION SYSTEMS SECURITY

Third generation methods presently dominate the general IS development scene. However, the security profession has not developed or employed many logical engineering methods. This may be, in part, due to the difficulty of implementing controls separate from the physical technology. The absence of any important third generation techniques in security means that, while third generation techniques currently dominate general information systems development, first and second generation techniques currently dominate security systems development.

The conventional approach to the design of security of information systems involves the overlay of security features upon a complete and functional system. This is evident in the preponderance of security methods that presume that an extant operational subject system is to be examined, evaluated and modified for security. These are "stand-alone" information systems security analysis and design methods. Thus, a dual development process is presently conventional for secured information systems.

The developmental duality of information systems security (*viz.*, that the information system and its security are separate developments) opens a danger, rooted in the very design process, for the introduction of conflict between a system and its security. Security design, as an independent process, aims at essentially different goals than the systems design process. It fundamentally aims at strictly confining system behavior to specific, allowable uses. Unpredicted uses must be prevented.

Thus, security design can interfere with system functionality in two basic ways. First, security design may develop system security features that do not permit certain proper system functions. This is because the security designers did not recognize these important system functions. Thus, independent security features in a system may cripple certain functional features that are essential to the system's required role. Second, security features may prevent a system from adapting to changes in its environment. Since the role of security features is to confine system behavior, it necessarily follows that system flexibility in unpredictable environments is also confined. In other words, it is the nature of security to prevent adaptation of a system to unexpected changes. At the very best, maintenance costs will rise as both system security and system functionality must be modified during major system revisions.

From a developmental perspective, this conflict may be widespread. The mainstream of IS development is found today in the third, logical engineering generation (structured methods, data modelling, information engineering, *etc.*). However, the mainstream of security development is found in the second, mechanistic generation (physical systems design, prototyping). It follows that the commonplace duality in secure IS development is the conflict between the logical engineering of the system's functionality and the mechanistic engineering of the systems's security.

This conflict explains many of the difficulties in IS controls operation. This also can explain the abbreviation of some secure systems' lifespans, because IS controls overlaid on a system may interfere with that system's essential functionality. If this is dramatic, then the contradictory nature of this interference could graduate to become the principal tension of the system's existence. In such a case, the balance of the tension between functionality and security defines the lifespan of the system. Three general patterns emerge: (1) the dominant priority[1] is the security; (2) the dominant priority is the functionality; or (3) the tension is resolved, and some other aspect graduates into the dominant priority.

[1]In his sinologicalization of Marxist dialectics, Mao Tse-tung referred to this as the principal aspect of the contradiction in his pamphlet *On Contradiction*.

The dominant priority is the security. In this case the system's lifespan is defined by its ability to survive with its inadequate functionality. Security must be observed, and the system is extinguished when its limited functionality destroys its feasibility.

A brief example will illustrate this pattern: A military headquarters developed a system to support the analysis and development of its command and control information. After a logical design phase, the original designers created a local area network (LAN) of PC-based workstations. This LAN facilitated sharing of resources like peripherals and data, and most data arrived through a bridge to a wide area network. Duty officers at LAN workstations could access, process and send data gleaned from the wide area network, as well as data residing on other LAN workstations.

After the system was designed, security was overlaid using a technique similar to the Fisher approach described earlier. The military communications operating system in the wide area network was already secure, and strictly enforced data access, limiting authorized queries to specifically authorized users at specifically authorized workstations. However, the industrial LAN network operating system had only limited security features.

Focussing on the *data-communication-(output)* control point, each workstation was been carefully modified to prevent any direct data access by network users. The workstation operator had to intervene and actively release output data requested by a user at another workstation. In other words, access by a workstation to data contained in another workstation required interactive authorization by the operators of both workstations.

For example, if the intelligence duty officer (IDO) needed to access a file on the operations duty officer's (ODO) workstation, the IDO first had to send an interactive request message to the ODO. The ODO would respond with an approval or disapproval message to the IDO. If approved, the IDO could then send a query request to the ODO's workstation. The ODO must then interactively "release" the specific data to the IDO's workstation.

While this operation was a bit cumbersome, the system worked well for several years. Duty officers, working at relatively low, fairly compartmentalized decision levels, rarely required access to workstation data other than their own. Gradually, however, the senior command staff began to acquire and use LAN workstations. These senior staff officers regularly needed to access data on their own duty officer's workstation. Since these staff officers often made decisions based on broader organizational data (cross-compartmental information), they also often required accesses to data on various other duty officers' workstations. As staff officer usage increased, the duty officers began spending an inordinate amount of time with the frequent authorizations for release of sensitive data. Both staff and duty officers complained increasingly about the interference caused by this cumbersome security operation. Staff officers, not wishing to annoy their duty officers began to "desert" the system and return to earlier manual methods.

In this instance security required strong enforcement of data secrecy and could not be relaxed. According to high-level policy, information release could not be delegated without certified software, and the existing uncertified system (being a file-based industrial operating system) could not automate the access approval process. The only course was to replace the entire system with one based on a "trusted" data base that could enforce the security without direct duty officer intervention or extra aggravation to busy staff officers.

This case illustrates the two points made earlier. First, information systems security is dedicated to the confinement of system behavior to specific functions. Accessing another workstation's data was supposed to be a rare occurrence. But the system's usage changed in such a way that this cross-compartmental access (a confined behavior) became commonplace. The security requirements thus confined the system's behavior in a manner inconsistent with the system's need to adapt to changes in the organizational setting. Either the security or the system had to be changed. Where the security is paramount, as in this case, it is the entire system that had to go. The lifespan of the system was defined by the security constraints necessarily placed on its functionality.

A second illustrated point is the conflict that can rise when the basic philosophy of the two design techniques is different. In the example, a mechanistic engineering security design technique was used to overlay security on a system design that was created using a logical engineering technique. A logical engineering approach to both functionality and security could have produced security that harmonized with the functionality.

For example, an inversion model approach would have focussed on protecting the processes by which workstation data was accessed, rather than the direct data focus of the Fisher approach. This would have been dictated by the automated nature of the function, and the external nature of the risk (intentional disclosure). Had the original system security focussed on logical methods for protecting the processing elements, physical intervention by duty officers would have been minimized.

The tension could have been resolved, then, between security and functionality (or at least subdued into the background) resulting in the third pattern mentioned above. In this event, some other operational system tension would become principal, and the system lifespan would have been determined by this other contradictory tension. So perhaps the basic design of the database might have proved inflexible for the less-structured demands of the staff officers.

This third pattern is ideal for the security design, being one in which the security constraints are not interfering with the essential functionality of the system. Had the inversion security design approach been used (one consonant with the functional design approach) it would have predicted the adaptation features necessary for the system to evolve without serious security conflicts. The principle tension would focus on the database design, and the system survival depends on that issue.

But what of the second pattern, where *the dominant priority is the functionality*? This is the case where the system's lifespan is still defined by the tension between security and functionality, however, the functionality is dominant. The system survives by its endurance of the risks inherent in its unprotected operation. Functionality must be provided, and the system is extinguished when its accidental or intentional losses destroy its continued feasibility.

Another example will illustrate this pattern. In this case, we consider a small, innovative motor vehicle insurance company. This company needed an automated policy system to remain competitive. Its overhead for processing policies was far greater than its larger competitors. Such policy systems are very expensive, using large mainframes, complex tailor-made software, and very sophisticated peripherals. The expense of such a system was impossible to justify based on the relatively small company turnover. Even the modest growth that would continue after a policy system acquisition would not justify the expense for many years. A leap into such a system would be very, very risky for the company. Thus they had a limited scale problem: They could not afford to acquire and operate such a system given their present turnover, and they could not continue their growth without it.

But a creative and highly motivated data processing director contrived a scheme to construct a "patchwork" automated policy system. She managed to fabricate an interim system out of the company's existing, heterogeneous computer systems and software, linked together mostly by electronic file transfers. The company's only costs were for special file translation software (import/export) and moderate scanning and printing peripherals.

Here is a brief summary of the work flow through this patchwork system. Upon reception, the insurance application forms were keypunched into a data file, and then bit-scanned into a graphics image file. The original document was permanently archived. The application data files were physically transferred from computer to computer as they progressed from department to department. As the electronic application progressed through review, underwriting, legal and auditing offices, it accumulated additional heterogeneous files. The final package included the application data file, the underwriting data file, the policy clause script, the auditor data file, the application image, and the policy clause text. In the final stage, the actual policy was printed from this package, and the data package was archived.

The small company was thus able to achieve most of the functionality of an automated policy system at a fraction of its acquisition cost. There were several trade-offs, including operational complexity and data storage shortages. But the most recognized tradeoff was the lack of resources for providing security controls in the patchwork system. Notably, application data was physically divided across files in several magnetic tape reels in both a horizontal and vertical fashion.[2] There was considerable redundancy of important data items. Consequently, the system's basic integrity risks were high. Further, many employees had to have access to several distinct computing facilities in order to accomplish their tasks, which led to password control and separation-of-duties degeneration. Consequently system fraud and disclosure risks were also high.

In this example, the inversion model approach called for logical shielding of the data from integrity and fraud risks (primarily internal). Controls such as integrity cross-checks, encryption and record-level access security were needed. Such controls would be expensive additions to the patchwork design.

Certainly management recognized the risky nature of the patchwork system when it was

[2]That is, no data record contained a complete policy; and no file contained even partial records for every policy currently in process.

approved for implementation without the controls suggested by the inversion model. But they knew that their survival depended on breaking through to higher turnover levels. This turnover was necessary in order to afford a safer, costlier system. In their view, the risks of the unsafe operation were justified when compared with the risk of plunging into a major policy automation system too early.

Owing mostly to the care of the employees, this patchwork system operated perfectly in its early days. The company rivalled its wealthier and more carefully automated competitors. Over the years, the corporate turnover (and policy production) grew, and losses gradually grew as well. Several claims were complicated by inaccurate or incomplete policies. Policies were lost in process, and had to be reconstructed. The increased throughput began to overwhelm the system, and there were no controls in place to validate or protect the system data. Fortunately, however, as the losses became critical, turnover rose to a level that justified a proper, safe policy system. The organization replaced the patchwork system with a properly designed distributed database system (with controls).

In this case, it was the functionality that was absolutely critical, the security was necessarily optional. Proper controls were discarded since they interfered (economically and technically) with the adaptation of the system to its changing organizational environment. In effect, they "turned off" the controls in order to achieve adaptation. The lifespan of the ensuing patchwork system was defined by its capacity to survive its risks long enough to "earn" its replacement.

It is possible that this tension could have been resolved by using a different technique to specify the controls. The design approach was essentially first generation, a checklist approach. (The system was created by mapping existing hardware and software onto the problem-at-hand.) Instead of using a logical engineering approach to the security, the designers might have done better by surveying a checklist of all available controls. For example, the AFIPS (1979) checklist recommends focussing on good operational practices and adequate budget as protection against human and applications reliability. Basic controls would have focussed on user documentation and minimizing operator involvement. Using risk analysis, they could have discovered many such controls that were inexpensive enough to be feasible at the lower turnover levels.

By using such similar design philosophies for both functionality and security, *the tension between these system features is minimized* and the third pattern would be achieved. The system lifespan is thus extended and defined by some other system characteristic (perhaps the hardware lifespan). Possibly the patchwork system used in the example could have profitably survived into even higher throughput plateaus with minimum losses had such an approach been taken.

Practical Implications.
The practical implications of the duality of security design and functional design is critical. The greatest cost of information systems security may be in the shortened lifespan of the information system (Baskerville, 1988). Security designers generally overlook such costs in specifying system controls. Thus, the resolution of tension between security and functionality can mitigate this lifespan damage and reduce the overall cost of security to the organization.

For new systems, the use of harmonious development methods, such as the employment of security methods from the same generation as the development method, serves to resolve much of this tension. Thus, if a third generation systems design technique (logical engineering) is used (*e.g.*, structured design), then a third generation security design approach (*e.g.*, Inversion Modeling) should be used in concert.

For existing systems, the lifespan may be extended by preventing the the security-functionality contradiction from becoming an essential or principal tension in the system. For example, if management assures that maintenance changes do not sacrifice either the system functionality or the system security in favor of the other, then the system might well evolve with these tensions subdued and its lifespan left unmitigated.

FUTURE SECURITY METHODS

The foregoing analysis offers insight into the future of information systems security methods. It appears that there is clear room for growth of third generation methods. For security to be practical and feasible, the conflict of development duality must be mitigated. Particularly needed are security design techniques that integrate well with rising general systems design techniques such as information engineering, prototyping and object-oriented analysis.

If a fourth generation of information systems methods emerges, new security methods will be needed to harmonize with this generation. How such a generation might be distinguished is not clear, since it is always difficult to discern truly important breakthroughs until these are well past. Perhaps this generation will be spawned primarily from the sociotechnical and participative aspects of the earlier generations (*cf.* Lyytinen & Klein, 1985). These techniques, like Mumford and Weir's (1979) ETHICS are oriented toward genuine participative information systems design (*cf.* Bostrum & Heinen, 1979; Land, 1982). Despite the wide recognition of the importance of people as security factors, there are no major security design methods that involve unqualified user participation in the design process. Security methodology would have far to travel if participative techniques became the next generation of system design methods.

Alternatively perhaps, this new generation might be distinguished by features that support systems undergoing continual evolution, thus never reaching any durable structured state (Truex and Klein, 1991). These "emergent systems" ideas may have been manifested first in the organizational learning perspective of Argyris and Schon (1978) and Checkland's (1981) Soft Systems Method.

For security, the key point of this social perspective of continually evolving organizational systems would be the continuous need for security planning. Perhaps the ultimate solution to the conflict found in the duality of security design may be represented by the incorporation of security into methods for continually evolving systems. Since these methods must assume that systems never reach a stable state, and that maintenance is simply a continuation of systems development, a consistent, continuing security method would minimize the duality that might have persisted as conflict in the maintenance phase. Again, much work would be required on security methods that would harmonize with emergent systems design.

SUMMARY

The developmental duality of information systems security refers to the concept that an information system and its security are separate developments. This duality is not well recognized in either the IS development literature or the IS security literature. Despite this, it can lead to essential conflict and tension between a system and its security. This conflict surfaces as difficulties and impediments in the operation of either security features or functional features of the subject IS.

The ultimate effect of this conflict can be the early termination of an IS lifespan. Such termination depends only on the choice of functionality versus security as the most significant feature in operation of the system. Consequently, lifespan limitations will be due to functional infeasibility resulting from security significance, or due to major losses resulting from inadequate security protection as a result of the overriding importance of functionality.

An understanding of this tension can help the system developer manage the contradiction. First, by selecting compatible security and system development methods. Second, by managing system evolution to reduce the conflict between functionality and security in such a manner that it is never allowed to become the principal tension that defines the system's Lifespan.

REFERENCES

AFIPS, (1979). *Security: Checklist for computer center self-audits*. Arlington, Va: AFIPS Press.

Argyris, C. & Schon, D. (1978). *Organizational learning: A theory of action perspective*. Reading: Addison-Wesley.

Bannon, L. (1989). Discussant notes on Baskerville and Hellman. In H. Klein and K. Kumar (Eds.), *Systems development for human progress* (pp. 257-259). Amsterdam: North-Holland.

Baskerville, R. (1988). *Designing information systems security*. Chichester: J. Wiley.

Baskerville, R. (1989). Logical controls specification: An approach to information systems security. In H. Klein & K. Kumar (Eds.) *Systems development for human progress* (pp. 241-255). Amsterdam: North-Holland.

Bostrum, R. & Heinen, J. (1979, December). MIS problems and failures: a sociotechnical perspective. *MIS Quarterly, 3*, pp. 11-28.

Carroll, J. & MacIver, W. (1984). Towards an expert system for computer facility certification. In J. Finch and E. Dougall (Eds.), *Computer security: A global challenge* (pp. 293-306). Amsterdam: North-Holland.

Checkland, P. (1981). *Systems thinking, systems practice*. Chichester: J. Wiley.

Courtney, R. (1977). Security risk assessment in electronic data processing. *AFIPS National Computer Conference Proceedings, 46*, pp. 97-104.

Farquhar, B. (1991). One approach to risk assessment. *Computers & Security, 10* (1), pp.21-23.

Fisher, R. (1984). *Information systems security*. Englewood Cliffs: Prentice-Hall.

FitzGerald, J. (1978). *Internal controls for computerized systems*. San Ceandro: Underwood.

Gallegos, F., Richardson, D. & Borthick, A. (1987) *Audit and control of information systems*. Cincinnati: South-Western.

Hemphill, C. & Hemphill, J. (1973) *Security procedures for computer systems*. Homewood: Dow Jones-Irwin.

Hoffman, L., Michelman, E. & Clements, D. (1978). SECURATE - security evaluation and analysis using fuzzy metrics. *AFIPS National Computer Conference Proceedings, 47*, pp. 531-540.

Hoyt, D. (1973). Computer Security Research Group. *Computer security handbook*. New York: MacMillan.

Kendall, K. & Kendall, J. (1988). *Systems analysis and design*. Englewood Cliffs: Prentice-Hall.

Krauss, L. (1972) *SAFE*. New York: Amacom.

Land, F. (1982, May). Notes on participation. *The Computer Journal*, *25*, pp. 283-285.

Lyytinen, K. & Klein, H. (1985). The critical theory of Jurgen Habermas as a basis for a theory of information systems. In E. Mumford, R. Hirshheim, G. Fitzgerald & T. Wood-Harper (Eds). *Research methods in information systems* (pp.219-231). Amsterdam: North Holland.

Mumford, E. & Weir. M. (1979). *Computer systems in work design: The ETHICS method.* London: Associated Business Press.

Powers, M., Adams, D., & Mills, H. (1984). *Computer information systems development: analysis and design*. Cincinnati: South-Western.

Saltmarsh, T. & Browne, P. (1983). Data processing -- risk assessment. In M. Wofsey (Ed.), *Advances in computer security management* (Vol. 2, pp. 93-116). Chichester: J. Wiley.

Smith, S. & Lim, J. (1984). An automated method for assessing the effectiveness of computer security safeguards. In J. Finch & E. Dougall (Eds.), *Computer security: A global challenge* (pp. 321-328). Amsterdam: North-Holland.

Straub, D. (1990) Effective IS security: An empirical study. *Information Systems Research, 1* (3), pp. 255-276.

Summers, R. (1984) An overview of computer security. *IBM Systems Journal, 23* (4), pp. 309-325.

Truex, D. & Klein, H. (1991). A rejection of structure as a basis for information systems development. In Lee, R. & Stamper, R. (Eds.), *Proceedings of The Working Conference on Collaborative Work, Social Communications and Information Systems*. Amsterdam: North-Holland, in press.

U.S. Department of Justice (1986). *Computer crime: Computer security techniques.* Bureau of Justice Statistics Document J29.2:C86.

Weber, R. (1988). *EDP auditing: Conceptual foundations and practice* (2nd ed.). New York: McGraw-Hill.

Whitten, J., Bentley, L. and Ho, T. (1986) *Systems analysis and design methods*. St. Louis: Times Mirror/Mosby.

Wood, C. (1990). Principles of secure information systems design. *Computers & Security, 9* (1), pp.13 - 24.

Zviran, M., Hoge, J. & Micucci, V. (1990) SPAN -- a DSS for security plan analysis. *Computers & Security, 9* (2), pp. 153-160.