SPECIALIZED SYSTEM DEVELOPMENT

Osama Eljabiri and Fadi P. Deek

New Jersey Institute of Technology

1. Introduction

Software development exemplifies a complex problem solving process due to various interdisciplinary variables that drive its evolution. Such variables are either problem-related or solution-based. Problem-related variables set the criteria for solution characteristics and help tailor solutions to specific problems while solution-based variables explain current options, assist in future forecasting and facilitate scaling solutions to problems. The issue of whether to find generic prescriptions to common problems [i.e. bottom-up generalization] or derive domain-dependent solutions to specific problems [i.e. top-down specialization] is debatable.

One viewpoint considers software engineering a "standardized" response to the approaches of generic methodologies and strategies as opposed to the non-systematic approaches that characterize early software development. Standardization implies generic rules, procedures, theories, and notations that mark a milestone in the development of any discipline. With standardization, software development was to witness a paradigm shift from trial and error experimentation to scientific maturity, from differing representation and implementation of concepts to unified modeling and cross platform independency, and from vague economic considerations to defined software-driven business models. The competing viewpoint of "one size fits all" has not proven to be practical in real world software development (Glass, 2000). There is no one methodology appropriate for every case, a strategy that works perfectly with every problem or off-the-shelf-prescriptions that can be applied directly without scalability, tailorability or customization. Even specific approaches that fit certain situations do not necessarily fit them all the time since change is the only constant in contemporary business. There are evolving needs that accompany innovation and emerging technologies. It can be argued that a balanced approach

between generalization and specialization can be adopted to achieve effectiveness in software development.

This chapter addresses the notion of specialized system development. The field of system specialization has been overlooked in the software engineering literature since the discipline was formally launched. Also, generic software development had only provided a "weak" strategy (Vessey and Glass, 1998) to solve problems since it only supplies guidance for solving problems and not actual solutions to problems at hand. Scalability, tailorability and specialization have become relevant issues in the software industry and software engineering research. Even general applications are not actually generic. Many of such current applications support customization features. Additionally, these systems are released in various modes that range from standard to professional to enterprise editions suiting diversity of needs and problem complexity. Such applications also evolve over the time to reflect changes in business requirements and technological capabilities. Subsequent sections of this chapter define specialized system development, discuss its drivers, present its advantages and disadvantages, and explore the types of specialized system development and its categories. Also, the need for specialized systems development and how that can be mapped to team structures is considered.

2. Principles of Specialized System Development

According to Merriam Webster dictionary, to "specialize" is to concentrate one's efforts in a special activity or field or to change in an adaptive manner. Concentration leads to more attention to details and presumably enables more efficiently problem solving. Specialization links theory to practice and makes it more meaningful. Generally speaking**,** specialized system development is about developing software systems with focus. The focus may be on the application domain, a certain phase of the development cycle, or a specific system development methodology. An example of application-domain focus is software development for pervasive computing including wireless and portable systems. An example of development phase focus is special emphasis on project management, requirements analysis or architectural design as

opposed to generic knowledge in software engineering. An example of methodology focus is systems development using structured or object-oriented strategies. However, specialization in methodology spans a wider array of approaches and tools. This includes software development process models (i.e. problem solving strategies), CASE tools and implementation techniques. Application-focused software development is the most current frequently used definition for specialized system development in the software industry.

Application-focused development can be classified into two categories: application-oriented and infrastructure-oriented. Each of these two categories can have a problem-focus or a solution focus. Problem focus can be based on the type of industry involved or the application domain. Solution domain can be based on custom-development, package-development or development aid (Glass and Vessey, 1995).

2.1 Roots For Specialized System Development

The history of specialized system development is tightly coupled with the evolution of computer hardware and technology advancement.

- **Domain-Dependent Era (pre-software development methodology):**

In the period of 1955-1965, computer hardware was application dependent. It was virtually impossible to develop business and scientific applications on the same machine. “Medical applications” and “unusual applications” were two examples of the distinct focus of application development during this era. Problem–oriented languages such as Fortran, COBOL, and ALGOL were developed to translate old software to be compatible with the requirements of new computers. **Domain-specific focus** was the major driver in building successful software systems. New disciplines emerged to support these systems such as numerical analysis and information retrieval (Vessey, 1997).

- **Domain-Independent Era (early-software development methodology):**

In the period of 1964-1980, the IBM 360 was introduced (in 1964) followed by the introduction of the lower midrange model 40 and model 67 shipped with hardware to support virtual memory shortly after. The IBM 360 combined scientific and business applications in one machine. The sociology of software development was strongly influenced by the "360" ability to end the separation between scientific and business applications. Generic applications were possible when software business become independent of hardware vendors. Competitive advantage in software development became directly proportional to the interdependency of standards, hardware or platforms. This era witnessed many attempts to institutionalize application-independent software development strategies (Vessey, 1997). This lead to the building of solid foundation for the next era of methodology-intensive software development.

- **Generic Applications Era (methodology-intensive software development):**

In the period of 1980-1995, the birth and evolution of desktop PC computing and Laptop computing was witnessed. With the availability of computers and the high degree of usability, user involvement and impact became more dominant and the availability of technology facilitated automation efforts in software implementation, which in turn enabled non-technical users to be active participants in the process (Glass, 1998). User-friendly GUI took over JCL (job control language) taking human computer interaction (HCI) to a new level. Some attempts at developing application-dependent software (such as 4^{TH} generation languages, rule-based languages, and simulation languages) were also carried out (Vessey, 1997).

- **Return to Application-Focused Development Era (software development post-methodology):**

In the period of 1995-present, the evolution of networked hardware architecture was quite dominant. Developing web-based applications marked a milestone in this era as was the emergence of web-driven tools and programming languages (i.e.: HTML, Java, Java Script, XML, VML, etc.), the evolution of friendly web interfaces through internet browsers and email agents, the emergence of web-based software engineering as a software development

methodologies, the increasing demand for software that balances speed and quality and the synchronization between business processes and software evolution.

2.2 Generic Versus Specialized Development

The shift from domain-specific computers to application-independent ones was considered an important event for software development. The subsequent advancement of application-independent computers into desktop and then notebook computing was another milestone that marked a shift toward generic infrastructure systems, applications and components with notable advantages such as:

1- **Portability:** Software can be used virtually anytime anywhere since the generic software became the dominant factor rather than the hardware.

2- **Compatibility:** One operating system can host a vast number of applications regardless of their vendors. Generic operating systems became a central repository for shared components across applications.

3- **Reusability:** One application or one module can be used across computer models, organizations, and endless number of users. It can be distributed over an organizational network or the World Wide Web. It can also be reused to develop new releases of software implementations. Furthermore, with few modifications through built-in preferences or options, the same application can be customized or tailored to more specific needs.

4- **Ease of training:** Generic applications became easier to teach and learn because of their availability and training material became cheap (or even free) due to mass production of such software products.

5- Cost–effectiveness: Since operational costs are generally lower with mass production and their revenues is usually high, they are sold or made available at competitive prices to the end user.

There are also disadvantages worth noting with generic applications. For example, such applications are based on the assumption that there are no significant differences between individuals and/or organizations that require special tailorability or scalability. This assumption applies also to generic methodologies and strategies in software development. These methodologies are rarely based on type or size of the project or technology environments and organizational settings. Such methodologies are considered one-dimensional approaches because they often do not mirror a particular organization's underlying problems (i.e. social, political and organizational development dimensions) (Avison and Fitzgerald, 2003). Generic applications also assume that businesses or individuals should be able to adapt to the infrastructure and functionalities of generic applications with limited room for changes. This can be true within the same application domain but can be extremely ineffective from a domain to another. Additionally, the assumption that business processes can be easily changed to fit a genetic software product is unrealistic and costly. Additionally, diversity of goals, market demands, stakeholder requirements, architectural specifications, non-functional requirements, organizational cultures, etc. across business domains and specializations makes generic development strategies impractical. For some organizations, adopting a specific methodology make not lead to desired result and can lead to reject methodologies altogether (Avison and Fitzgerald, 2003). Agile software development may be viewed as a reflection of this fact.

2.3 The Context of Problem Solving in Specialized System Development

Since software development essentially aims to solve problems, it is important to view specialized system development in the problem-solving context. Basically, solving problems involve two key elements: the ability to comprehend the problem and the capability to solve it. Hence, specialized system development is either problem-focused or solution-driven. Because problem types and solution strategies in software engineering vary, effective understanding of their diversity is a pre-condition to successful specialized system development. In fact, this diversity is a major driver for specialized development since differences are the catalyst for any

specialization. Relevant questions include understanding how specialization in investigating and identifying problem characteristics can help in evaluating existing solution options and selecting the most proper ones and how specialization in domain analysis and requirements engineering can help in developing effective solutions by means of proper processes or methods selection or constructing specialized processes or methods for certain applications. How can software products or solutions be adequately used, reused, customized, personalized, reengineered, or redeveloped based on application-driven or domain-specific specialization. How can specialization in problem, method, product or domain analysis assist in proper selection or successful construction of computer-based problem solving strategies that articulate suitable methods, process models, techniques and tools.

Figure 1. Generic and specialized software development in the problem-solving context

Careful examination of problem and solution diversity reveals three key drivers for specialized system development: Characteristics of the system to be developed, characteristics of system anticipated users; solution–driven capabilities, experiences and knowledge; and characteristics of system developers.

2.3.1 Characteristics of the system to be developed

This is a problem-focused category. Diversity of software systems in terms of size, complexity, time constraints, scope, underlying technology, business goals and problem environment are the most critical drivers in this category. Problems range from structured at the operational levels of organizations to semi-structured at the tactical level to ill-structured at the top management or strategic level (**vertical specialization**). Problem specialization can be between organizations in the same industry or across industries (**external horizontal specialization**) or within the same organization across its various functional departments or key business processes (**internal horizontal specialization**).

2.3.2- Characteristics of systems anticipated users

This is also a problem-focused category. Some of the drivers in this category are: age considerations, gender considerations, purpose of using the system (i.e. personal vs. business users), user background (i.e. technical vs. non-technical users), and user environment. User environment includes but is not limited to: cultures, languages, geographic locations, technical resources, financial resources, human resources, legal and ethical issues, etc. Each one of these drivers creates certain needs in systems development and therefore triggers specific specializations in responding to these requirements.

2.3.3- Solution–driven capabilities, experiences and knowledge

System specialization under this category is based on tools and resources rather than application domain. This includes capabilities and experience in project management tools, requirements analysis techniques, architectural models, user interface approaches, database management strategies, implementation languages, development tools, development methodologies and process models. These capabilities affect numerous number of specializations in the solution area.

3. Application-Based Specialized Development

The convergence of three traditional computing specializations, personal, networking and embedded, produced a new computing era referred to as “pervasive computing”. Mobile computing, wireless devices, PDA, Pocket PC, and Tablet PC are all examples of pervasive computing products. The software applications are important components in these products and the nature of these applications brings a new set of challenges in software development.

3.1 Pervasive Software Development

Pervasive applications can be distinguished by the following characteristics: Ubiquity, interconnectedness, and dynamism. These applications strive to be embedded, distributed, non-intrusive, and cost-effective (Ciarletta and Dima, 2000). This implies that software economics, system architecture and security are significant issues in pervasive software engineering. A

conceptual model is suggested to highlight the aspects of pervasive systems development in which four layers have been identified (physical, resource, abstract and intentional layers) (Ciarletta and Dima, 2000). Table 1 elaborates on the role each one of these layers play in specialized pervasive system development.

Table 1. The Roles of Pervasive Systems Development Layers

A framework of four levels can provide a sound process for developing effective m-commerce applications (Varshney and Vetter, 2001). These four levels are:

1. *M-commerce Applications:* Modifying new e-commerce applications for a mobile environment.
2. *Wireless User Infrastructure:* the modified or new mobile commerce applications should support the capabilities of user infrastructures. For example, m-commerce applications must be as effective for the mobile devices such as PDA's and cell phones.
3. *Mobile Middleware:* the new m-commerce applications must have better response time and reliability when deployed because the middleware will be used to connect e-commerce applications with different wireless networks.
4. *Wireless Network Infrastructure:* Networking requirements need to be fulfilled based on the type of m-commerce applications being deployed. Such requirements would be quality of service, network reliability, location management, roaming across multiple networks, and multicast support.

Effective m-commerce application can be deployed if network reliability and redundancy is increased. Furthermore, creating mobile commerce applications requires a unique set of knowledge and needs specific networking support factors to create effective applications (Kalakot et al, 2000) that includes wireless quality of service (QoS), efficient location management, and reliable and survivable Networks.

3.2 Real-Time Software Development

Real-time software development is not new and dates back to the 70's and continue to evolve today. The development of real-time systems requires the consideration of three basic issues (Felder, 2002) including complex timing (at the higher requirements specification levels), resource constraints (lower design levels), and scheduling constraints (lower design levels).

Gaulding and Lawson (1976) described a disciplined, engineering approach to real-time software development with a focus was on a process design methodology. The basis to this approach is a process performance requirement, a document describing the interfaces to the software, the software functional and performance requirements, the operating rules and the data processor hardware description. The goal of **process design engineering** was the development of an automated approach to the "evolutionary" design, implementation, and testing of real-time software. Gaulding and Lawson thus defined the crucial aspects of effective real time software development to include four important components: **Transformational technology** to enable traceable transformation from functional requirements to a software structure for a given computer. **Architectural Approach** which requires a top-down design, implementation, and testing techniques supported by a single process design language. **Simulation technology** which provides a capability for evaluating trial designs for real-time software processes. **Supporting tools** for automating such functions as requirements traceability, configuration management, library management, simulation control, and data collection and analysis.

An early software development life cycle for real time systems was proposed (Gomaa, 1986). This method attempts to tailor generic software development methodology to reflect the special needs of real-time software development. Table 2 describes this method, phases and applications.

Table 2. Life Cycle Phases for Real Time Software Systems

3.3 Web–based software development

Web-based software development is growing at a faster rate than any other domain. Software systems with web capabilities can maximize the business added value more effectively with their ability to reach customers, partners and enrich the business process with information (Evans and Wurster, 1999). Three criteria to assess business value in IT-based systems are: productivity, business profitability and consumer surplus (Hit and Brynjolfsson, 1996). Web applications extend traditional business goals beyond direct financial measures to encompass measures of customer satisfaction, internal processes, and the organization's innovation and improvement activities. These operational measures affect organizational financial performance (Der Zee and de Jong, 1999). Efficiency, quality, market share and penetration emerged as important measures and goals of business (Singleton et al., 1988) that can be improved by web-based systems. These influences have motivated industry to integrate internet/intranet information systems in their businesses that necessitate the adoption of new management techniques to harness the advantage of this particular technology and align it with the organizational structure.

3.3.1 E-Business Software Systems

There are more demands on quality and reliability of web-based software development that ever before. Successful configuration for web applications requires special attention to several interrelated aspects that play key roles in leveraging web engineering to the competitive advantage level. These ingredients are driven by development teams, legacy systems, value chain, and business integration and management strategies and include:

- **Skills, Structure and Management of the Development Team**

Skillful staff in web-driven software development projects can significantly boost performance. Training programs and availability of necessary resources have strong influence on the quality of e-business applications since that enables the development time to reduce the cycle time of tailoring solutions to application needs. Effective management can create the right team structure and the necessary synergy from diverse abilities.

- **Legacy Applications**

Scope and domain of legacy business problems shape the strategies needed to solve e-business software problems. The issue of negative correlation between organizational complexity and the impact of technical change is a disputable one (Keen, 1981), since the more complex are organizations the more ill structured are their business problems (Mitroff and Turoff, 1973). Even though this influences the ability to tackle such problems smoothly, information technology enables a complex organization to redesign its business processes so that it can manage complexity more effectively (Davenport and Stoddard, 1994).

- **Value Chain and Logistics Management**

Value chain is the set of activities business requires to achieve its objectives where additional values can be added as activities proceed from phase to another. E-business applications utilize internet technology to over products and services which requires integration of business processes and their relevant activities as well as the logistics of end users to original suppliers. Effective management of the entire process can add considerable value to consumers in terms of organizing, coordinating and controlling supply chain activities and logistics (Turban et al, 2000). This defines certain criteria for effective web-based development that encompasses flexibility, quality, dependability, agility and efficiency. Optimization can be assessed in terms of delivering the right product at the right time at each level of the supply chain (Vokurka et al, 2002). The value chain concept can be further utilized to build decision support systems to enhance the decision making process at the tactic and strategic management levels (Haavengen et al, 1996). Also, electronic product development (EPD) is another aspect of e-business development that relies on holistic prospective on the entire product value chain encompassing customers, designers, suppliers, manufacturers, and logistics providers toward more successful mass customization (Helander, 2000).

- **Aligning E-business Applications with Organizational Goals**

E-business solutions can effective in serving organizational goals and marketing requirements. Strategies that integrate the Internet and traditional advantages are expected to be

the kind of approach that creates potential advantages for existing corporations (Porter, 2001). E-business software systems do not rely only on the internal preparation of the company but also on the readiness of its customers and suppliers to engage in electronic interactions. By committing resources to the business problem, management can create a value driver that that boosts business readiness for e-commerce challenges (Barua et al, 2001). These e-commerce solutions link customers, suppliers, partners, and inter-organizational departments in one or more unified value chains. If these links are not well managed and efficiently aligned in synchronized frameworks, delays will occur and costs will exceed any profit earned. Clearly, this will result on financial loss and customer dissatisfaction.

Some issues may have indirect affect on the success of the E-Business applications. These are supply chain management (SCM) and enterprise resource management (ERM) which help explaining the impact of legacy business applications on the success of E-business development. Better understanding of customers and suppliers needs along with dissecting current business process and how they will affect the overall methods of supply chain and resource management will lead to flexible and manageable utilization of information technology to design reengineered business processes (Daoud, 2000).

3.3.2 Object-Oriented Development For Web Applications

Gellersen and Gaedke (1999) proposed a web composition model that defines an object-oriented approach for web development based on web implementation models. This model was developed to provide developers the capabilities of object-oriented concepts in terms of reusability, inheritance, improved modifiability and extensibility. Conallen (1999) addressed object-oriented web application architecture through a UML-based approach. This approach aims to facilitate managing complexity for web applications and enable enhanced reusability. The approach integrates three models of web application architecture: business model, navigation model and implementation model and works in conjunction with CASE tools support.

3.3.3 Customizable Web-applications

Several approaches to modeling and implementing customizable web applications have been proposed. These approaches share the characteristics for web development environments (Kappel et al, 2000) that explicitly consider user context for customization, reflecting the need for personalization for both individuals and classes of users and they consider network and device contexts together. Network context is related to network settings while device context is based on multi-delivery of different devices or classes of devices. However, they have different degrees of location and temporal contexts. Location context is related to mobile computing and portability while temporal context is based on time constraints.

3.4 Security-Driven Software Development

Software systems have evolved into global networked infrastructures, multi-dimensional databases and enterprise data warehouses that interconnect individuals, businesses, organizations, competing supply chains, numerous mobile and wireless applications and even countries. The software engineering literature typically classifies security as one of the measures for quality and reliability in software products. Moreover, the software engineering field addresses the security issue as a part of the **risk analysis** process to minimize the likelihood of intrusions, attacks, hacking or fraud in information systems. The issue of security in contemporary software applications is a critical component for business survival. There is a need for protecting organizational strategic assets such as information. In e-commerce, for example, customers, who are more aware than ever before of the ramifications of unsecured personal or private information, gain business trust with sufficient security measures, policies and standards.

The area of information systems security has evolved across paradigms and strategies (Siponen, 2002). These range from the generic, based on common sense, to the specific based on organizational culture and needs, as described in Table 3. Security-driven systems are receiving greater attention in current software development strategies including the reengineering of existing systems by adding or enhancing security features, building security-based applications to ensure security in systems such as anti-viruses, firewalls, etc., building privacy applications or

adding features that enhance privacy of individuals, and building awareness or surveillance-based applications that can help in detection and/or protection against crime and terrorism. Computer vision, image processing and multimedia-based technologies play a significant role in these applications.

As with all forms of software development, the design of such systems is not without challenges. The tradeoff between open communication channels and the potential for security threats through these same channels is one example. The remaining parts of this section present a framework for dealing with security considerations in the software development process, particularly in terms of the analysis and design of such systems.

Table 3. The Four Generations of Information Systems Security Approaches

3.4.1 Security–Driven Requirements Analysis

Since a large portion of software engineering literature was developed prior to the web era, investigating vulnerabilities was rarely addressed adequately. Web-driven applications and infrastructures have necessitated a change. For example, in terms of security, while web connectivity increased access to public information, it exposed the very same information and information systems to more risks and vulnerabilities (Deswarte, 1997). In some software engineering methodologies, security requirements are addressed in the analysis phase as non-functional requirements since software systems need to comply with internal and external security standards. Summerville (1996) classifies security requirements as external, non-functional safety and privacy requirements. This view is true from a categorization perspective, but it needs to consider that even functional requirements should be guided by security metrics or they may otherwise increase system vulnerabilities. Additional requirements or flexible requirements could expose the system to unexpected risks (Smith, 1991 and Pfleeger, 1997). Security-driven requirements analysis involves defining security objectives, setting their metrics, identifying potential risks, investigating vulnerabilities, creating what-if scenarios, reviewing current requirements, and reformulating requirements to reflect the input of the analysis phase. Analysis

output becomes the guidelines for designing security-driven solutions. Additional details are shown in Figure 2.

Figure 2. Security-Driven Requirements Analysis Process

Security objectives are usually based on organizational standards, underlying technology and magnitude of anticipated threat. Since security breaches are highly unpredictable and their nature and scope can change over the time, organizations need to be adaptive to new threats and capable to adjust their objectives to meet the demands of evolving challenges. Once objectives are determined, quantitative and qualitative measurements should be derived and extracted to establish evaluation metrics to validate and verify quality of software products in terms of security requirements. The major task in security-driven analysis is identifying potential security risks. Risk Assessment is essential because an organization may be attacked from both within and outside its network (Philips and Swiler, 2001).

Identifying potential security risks involves investigating systems vulnerabilities thoroughly. Vulnerabilities can be attributed to intentional and unintentional factors. Unintentional factors are related to human mistakes, exceptional hazards in the environment, system failures, gaps in hardware or software design, or bad requirements specifications. While external factors contribute to the existence of this category of vulnerabilities, it is the analysis, design, implementation and usability of the system that make the vast majority of security threats in most organizations. For example, a problem in data collection, data entry, data distribution, referential integrity, or authorization can result in some breaches or putting data into risky situations. The growing concern of **infrastructure vulnerabilities** where more damage with a keyboard than with a bomb (Baskerville, 1993) is an important issue for organizational management. Tracing and tracking leakages, security gaps and security-related problems across the software development process are ways to ensure security in software systems. The traceability process as shown in Figure 3 offers a strategy for a software engineering approach to system security via traceability analysis.

Figure 3. A software engineering approach to systems security via traceability analysis

Intentional factors that threaten system security include data theft, data abuse, source code theft, deliberate data manipulation, data tampering malicious damage, virus and attacks destruction, cyber crimes, terror attacks and other miscellaneous computer crimes. Computer crimes range from using the computer or computer network as a target, to using computer as a medium (i.e. misleading information), to using computers as a planning or deception tool (Turban et al., 2002). One of the current and serious challenges of information systems is how information and communication technologies can contribute to public safety (Shneiderman, 2002). Recent efforts focus on enhancing security at the technical level (i.e. network-based security) while some attention is paid to security at the analysis and architectural levels. Anti-terror system development relies not only on solution-focused capabilities but also on profound comprehension of problem domain by studying attacker's behavior has proven to be beneficial (Erland and Olovsson, 1997). **System vulnerabilities** or security gaps in any information system provide opportunities to carry out attacks or steal critical information. Identifying and securing these gaps will minimize potential risks. Holmes (2001) pointed out the need to assess system security breaching motives in order to then protect and manage the systems infrastructure, according to their vulnerabilities. Salenger (1997) relates the level of organizational internet security to their relative "functional uses" of the internet. Engineering secure systems requires a great deal of managing infrastructure vulnerability (Demuth and Rieke, 2000).

Some models suggested in designing a secure environment (Salter et al., 1998) are: adversary model, vulnerabilities model, and methodology model. The Adversary Model includes the understanding of three motives of threat potentials: what they can do, what they are willing to do, and what they want to do. The vulnerabilities model implies three steps to any successful attack: analyze the targeted system to find weaknesses, gain access quietly, and execute attack. The methodology model categorizes attacks based on their characteristics and aims to find the best protective countermeasure. While the adversary model is based on **information gathering**,

the vulnerabilities Model is driven by **risk analysis** and the methodology model is related to **response procedure** and **recovering**.

3.4.2 Security–Driven Systems Design

Designing security-focused solutions for software systems can be done at two different levels: the conceptual level and the technical level. The conceptual level provides the architectural foundation for the technical level. The key concept for security-focused architectures is defense strategies. The ability of a software system to encounter threats is tightly coupled with its capability to reduce vulnerabilities, detect threats and provide protection shields that prevent eliminate, or deal effectively with breaches and attacks. Figure 4 depicts this concept as a seven-layer conceptual model for defense strategies in security-focused system design.

Figure 4. Seven-layer conceptual model for defense strategies in security-focused system design

In this model, five key defense strategies (prevention control, detection, limitation, recovery and correction) can be used separately or combined to minimize systems vulnerabilities or system weaknesses (Turban et al., 2002). Prevention control is the most effective strategy, whether it is human error, external attack or unauthorized usage. Access control plays a significant role in this defense strategy. Figure 5 provides a basic taxonomy of various types of security controls in software systems. An intrusion detection system (IDS) is system that can identify authorized uses, misuses or abuses of computers by either authorized users or external perpetrators. Intrusions can be classifies into three categories: single intruder signal terminal (SIST), single intruder multiple terminal (SIMT) and multiple intruder multiple terminal (MIMT) (Puketza et al., 1996).

Figure 5. A basic taxonomy for security control techniques in software systems

Object–oriented and component-based architectures have proven to be maintainable structures since they allow easy replacing of defective components. Distributed object architecture and design standards provide an adequate level for generic distributed applications. But these are only the first step in building application-specific software architectures for

achieving overall system development objectives. While Commercial enterprise application integration (EAI) tools and workflow management system (WFMS) products allow to advance basic distributed standards to the commercial level, they are still far beyond mission-critical applications needs. Such needs encompass mission-critical business and information security processes. System designers should employ security solutions that reinforce each other, define relationships based on trust, and use protective countermeasures to prevent attacks.

The effectiveness of database and network design plays a crucial role in reducing system vulnerabilities. For instance, cryptographic protocol design is frequently cited in the literature as a source for distributed systems vulnerabilities. Yet, analysis and design techniques have proven useful in detecting protocol vulnerabilities (Stubblebine et al., 2002). Cybenko and Jiang (2000) discussed the vulnerabilities of the Internet and proposed a six-stage protection process to counteract malicious uses. Information-gathering techniques are the first essential step in this six-stage process for protection of infrastructures and to increase awareness of emerging threats. Information gathering techniques include: intelligence reports, unusual-incident analysis, and automated information harvesting from the Web and news services. The second essential step is a thorough risk assessment of the current system to find vulnerable areas. This risk assessment includes: modeling an attack, modeling failure of main system and modeling subsidiary failures due to main systems. The third step is **interdiction**, which includes being able to make use of current prevention methods that are already available. The forth step is **detection** of attacks through early warning systems and monitoring resources. Monitoring subsystems are able to take actions while an attack is underway, whereas a warning system can attempt to prevent an attack before it happens (Salter et al, 1998). The fifth step is implementing the proper response procedure once an attack has been acknowledged. **Response procedures** are what Cybenko and Jiang call forensic challenges. **Response procedures** can only be implemented when an attack is already underway. Once an attack is detected, the system should be able to trace the attack. The

final stage in Cybenko and Jiang's approach is **recovery** which includes learning from the attack and documenting its characteristics for future reference in a knowledge base.

4. Research Issues and Summary

The area of specialized system development can be characterized as new, huge and crucial. This field is evolving as the importance of scalability and tailorability as opposed to generic strategies and approaches in software development is realized. The theoretical foundations of specialized system development will continue to evolve to provide a roadmap for new research and development. This will, in turn, provide new challenges and opportunities to the software engineering community since specialized system development is not only new but also critical for many contemporary software applications. An important aspect in future research and development of specialized systems development also concerns the government, industry as well as academia. The government's role in decryption of information on the Internet is crucial (Fox, 2001). For example, some of the intelligence issues and policies to be further addressed have been identified (Artz, 2001; Wilson, 2000 and Zorpette, 2002). These are human role in information analysis, gaps in technical intelligence and cooperation between organizations and services that collect intelligence. While certain responsibilities are defining the role of the government, others are placing a clearer definition on organizational roles. Salenger (1997) states that the level of security implemented by organizations is directly proportional to two factors, size and income. Larger companies have the people and the resources required to properly establish and run a secure internet environment where smaller companies may not. Better protocols for defining and enforcing standards are expected to continue to emerge.

5. Defining Terms

Weak strategy: Generic approaches in problem solving not tailored to specific problem domains.

System Specialization: The concentration on unique problems and the techniques for comprehending and solving them.

Vertical specialization: Specialization in the different levels of problem complexity across inter-organizational pyramid from operational to top management.

Horizontal specialization: Specialization across various functional departments or business needs within the organization or across various domains with an industry or between industries.

Pervasive computing: The convergence of three traditional computing specializations (personal, networked and embedded), which produced a new computing era marked with wireless and portable hardware and software.

Process Design Engineering: The development of an automated engineering approach to the evolutionary design, implementation, and testing of real-time software.

Pamela: Process abstraction method for embedded large applications.

SCR: Software Cost Reduction project-Naval research laboratory.

Infrastructure vulnerabilities: Weakness points and security gaps in the physical or logical architecture of information systems that may enhance opportunities to carry out attacks or steal critical information.

Interdiction: Being able to make use of current prevention methods that are already available.

Attentive Systems: Attentive systems are those that can be used to understand user trends or log and track Internet usage across multiple sources.

Steganography: Hiding data within data.

Cognitive Fit: An approach in specialized system development where the goal is to match as closely as possible the representation to the task and user at hand. The key concept is that there should be harmony among three variables: the user's cognitive skills, the task, and the representation of the task (as presented to the user).

References

Artz, D. 2001. Digital Steganography: Hiding Data within Data. *IEEE Internet Computing* 5(3): 75-80.

Avison, D. and Fitzgerald, G. 2003. Where now for Development Methodologies? *Communications of the ACM*, 46(1): 48-72.

Barua A., Konana P., Whinston A., and Yin F. 2001. Driving E-Business Excellence. *MIT Sloan Management Review*. Cambridge. 43 (1): 36-44.

Baskerville, R. 1993. Information Systems Security Design Methods: Implications for Information Systems Development. *ACM Computing Surveys*. 25 (4): 365-414.

Baskerville, R. 1993. Information Systems Security Design Methods: Implications for Information Systems Development. *ACM Computing Surveys*. 25 (4): 365-414.

Ciarletta, L., Dima, A. 2000. A conceptual model for pervasive computing Parallel Processing. *Proceedings of the 2000 International Workshops on Parallel Processing*: 9–15

Conallen, J. 1999 .Modeling Web Application Architecture With UML. *Communications of The ACM*. 42: 63-70.

Cybenko, G. and Guofei J. 2000. Developing a Distributed System for Infrastructure Protection. *IT Professional*. 2 (4): 17 -23.

Daoud, F.2000. Electronic commerce infrastructure. *IEEE Potentials*. 19 (1): 30-33.

Davenport, T. and Stoddard D. 1994. Reengineering: Business Change of Mythic Proportions? *MIS Quarterly*. 18(2): 121-127.

Demuth, T. and Rieke, A. 2000. Bilateral Anonymity and Prevention of Abusing Logged Web Addresses. *21st Century Military Communications Conference Proceedings.* 1: 435-439.

Deswarte, Y. 1997. Internet Security Despite Untrustworthy Agents and Components. *Proceedings of the Sixth IEEE Computer Society Workshop on Future Trends of Distributed Computing Systems*. 53: 218-219.

Evans, P. and Wurster, T. 1999. Getting Real About Virtual Commerce, *Harvard Business Review*. 77(6): 85-94.

Erland, J. and Olovsson T. 1997. A Quantitative Model of the Security Intrusion Process Based on Attacker Behavior. *IEEE Transactions on Software Engineering*. 23(4): 235-245.

Felder, M. 2002. A formal design notation for real-time systems. *ACM Transactions on Software Engineering and Methodology (TOSEM).* 11(2): 149-190.

Fox, R. 2001. Privacy Tradeoff Fighting Terrorism. *Communications of the ACM*. 44(12): 9-10.

Gellersen, H. and Gaedke, M. 1999. Object Oriented Web Application Development. *IEEE Internet Computing*. 3(1): 60-68.

Glass, R. 1998. *In the Beginning, Recollections of Software Pioneers*. The IEEE Computer Society Press, Los Alamitos, CA.

Glass, R., and Vessey, I. 1995. Contemporary Application-Domain Taxonomies. *IEEE Software*. 12(4): 63-76.

Glass, Robert L. 2000. Process Diversity and a Computing Old Wives'/Husbands Tale. *IEEE Software*. 17(4): 128-129.

Gomaa, H. 1986. Software Development of Real-Time Systems. *Communications ACM*. 29(7): 657-668.

Haavengen, B., Olsen, D., Sena, J. 1996 .The Value Chain Component In A Decision Supports System: A Case Example. *IEEE Transactions on Engineering Management*. 43(4): 418–428.

Helander, M. and Jiao, J. 2000. E-Product Development (Epd) For Mass Customization. Management of Innovation and Technology, *Proceedings of the 2000 IEEE International Conference on ICMIT 2000*. 2 (2): 848–854.

Hitt, L. and Brynjolfsson, E .1996. Productivity, Business Profitability, And Consumer Surplus: Three Different Measures Of Information Technology Value. *MIS Quarterly*. 20(2): 121-142.

Holmes, N. 2001. Terrorism, Technology and the Profession. Computer. 34 (11): 134-136.

Kalakota, R., Varshney, U., and Vetter, R. 2000. Mobile Commerce: A New Frontier. *IEEE Computer Society: Special Issue on E-commerce*. 33(10): 32-38.

Kappel,G., Retschitzegger, W. and Schwinger, W. 2000. Modeling Customizable Web Applications-A Requirement's Perspective. *Proceedings of the International Conference on Digital Libraries: Research and Practice*. Kyoto.

Keen, P. 1981. Information Systems and Organizational Change. *Communications of the ACM*. 24(1): 24-33.

Kelly, J. 1987. A Comparison of Four Design Methods for Real-Time Systems. *IEEE Proceedings of the 9th International Conference on Software Engineering*: 238-252.

Kelsey, J. and Bruce S. 1999. Secure Audit Logs to Support Computer Forensics. *ACM Transactions on Information and System Security* (TISSEC). 2(2): 159-176.

Mitroff, I. and Murray T. 1973. Technological Forecasting and Assessment: Science and/or Mythology? *Journal of Technological Forecasting and Social Change*. 5(1): 113-134.

Porter, Michael E. 2001. Strategy and the Internet. *Harvard Business Review*. 79: 63-78.

Pfleeger, C.P. 1997. The fundamentals of information security. IEEE Software. 14(1): 15-16.

Puketza,N., Zhang, K., Chung, M., Mukherjee B. and Olsson, R. 1996. A Methodology for Testing Intrusion Detection Systems. *IEEE Transactions on Software Engineering*. 22(10): 719-729.

Salenger, D. 1997. Internet Environment and Outsourcing. International Journal of Network Management. 7(6): 300-304.

Salter, C., O., S., Schneier, B. and Wallner, J. 1998. Toward a Secure Engineering Methodology. *Proceedings of the 1998 workshop on new security paradigms*: 2-10.

Shneiderman, B. 2002. ACM's Computing Professionals Face New Challenges. *Communications of the ACM*: 31-34.

Singleton, J., McLean, E. and Altman, E. 1988. Measuring Information Systems Performance: Experience With the Management By Results System at Security Pacific Bank .*MIS Quarterly*. 12(2): 325-337.

Siponen, M. 2002. Designing secure information systems and software: Critical evaluation of the existing approaches and a new paradigm. Unpublished PhD Dissertation. University of Oulu.

Stubblebine, S., and Wright, R. 2002. Authentication logic with formal semantics supporting synchronization, revocation, and recency. *IEEE Transactions on Software Engineering.* 28(3): 265-285.

Smith, G.W. 1991. Modeling Security-Relevant Data Semantics. *IEEE Transactions on Software Engineering*. 17(11): 1195-1203.

Turban, E., Lee, J., King, D and Chung, H. 2000. *Electronic Commerce: A Management Perspective.* Prentice Hall, NJ.

Turban E., Rainer K., Potter R. 2002. *Introduction to Information Technology*. 2nd Edition, Wiley, New York, NY.

Varshney U., and Vetter, R. 2001. A Framework for the Emerging Mobile Commerce Applications. *Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS 34).* IEEE Computer Society.

Varshney U., and Vetter, R. 2000. Emerging Mobile and Wireless Networks. *Communications of the Association of Computing Machinery (ACM)*. 43(6): 73-81.

Vessey, I. 1997. Problems versus solutions: the role of the application domain in software, *Proceedings of the seventh workshop on Empirical studies of programmers*, Virginia: 233-240.

Vessey, I. and Glass, R. 1998. Strong vs. Weak: Approaches to Systems Development. *Communications of the ACM*. 41(4): 99-102.

Vokurka, R., Gail M., and Carl M. 2002. Improving Competitiveness Through Supply Chain Management: A Cumulative Approach. *Competitiveness Review.* 12 (1): 14-24.

Wilson, C. 2000. Holding Management Accountable: A New Policy for Protection Against Computer Crime. *National Aerospace and Electronics Conference. Proceedings of the IEEE 2000*: 272-281.

Zorpette, G. 2002. Making Intelligence Smarter. *IEEE Spectrum.* 39(1): 38-43.

Further Information

A good survey of industry frameworks is presented in the article titled **Contemporary Application-Domain Taxonomies** by Glass and Vessey published in the IEEE Software in 1995.

The authors pay particular attention to representative taxonomies, IBM industry's taxonomy, digital industry's taxonomy, digital application taxonomy and Reifer's application taxonomy.

Other good sources are:

SIMS: A Secure Information Management System for Large-Scale Dynamic Coalitions by Jiang and Dasgupta published in the IEEE Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX II), June 2001 which discusses security with large scale systems.

Attack Detection in Large Networks by Peterson and Bauman published by the IEEE Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX II), June 2001 addresses the impact of large systems characteristics on security.

Security of Distributed Object–Oriented Systems by MacDonnell et al. published by the IEEE Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX II), June 2001 addresses object-oriented security mechanisms that can provide scalable fine-grained access control in both applications and at the boundary controller using CORBA and JAVA.

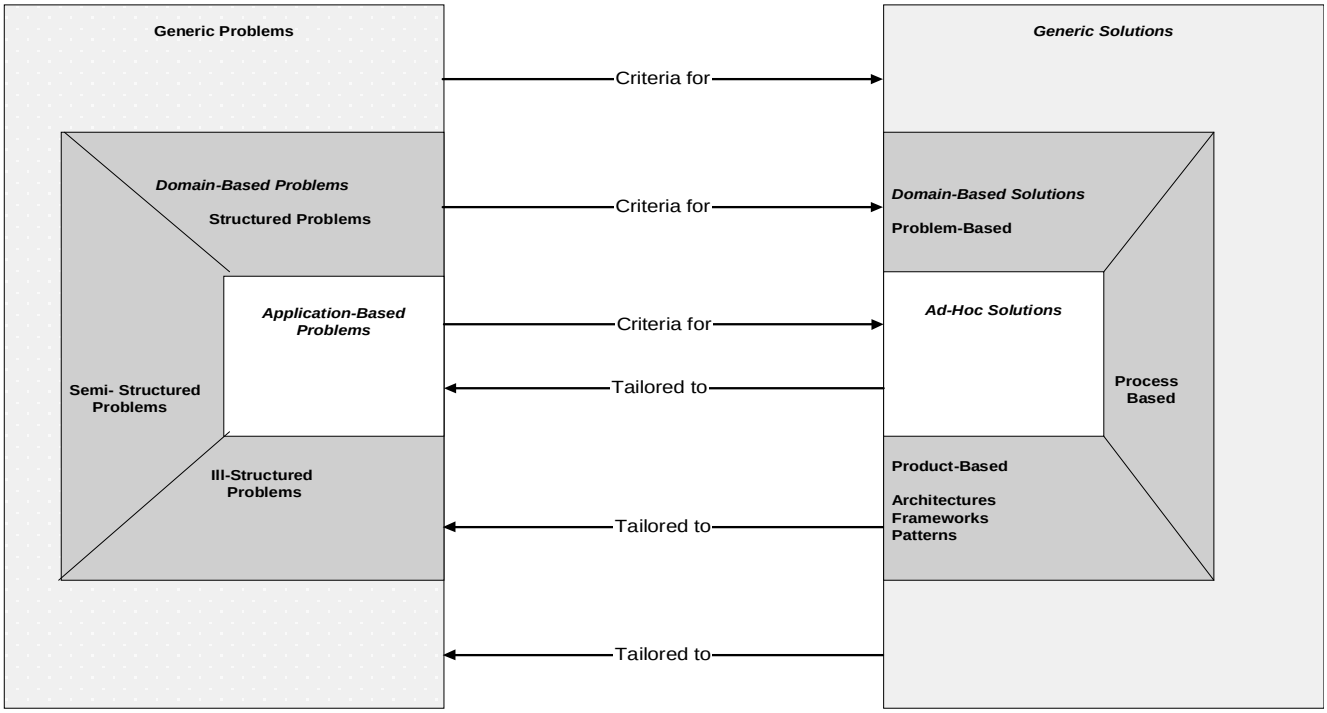

Figure 1. Generic and specialized software development in the problem-solving context

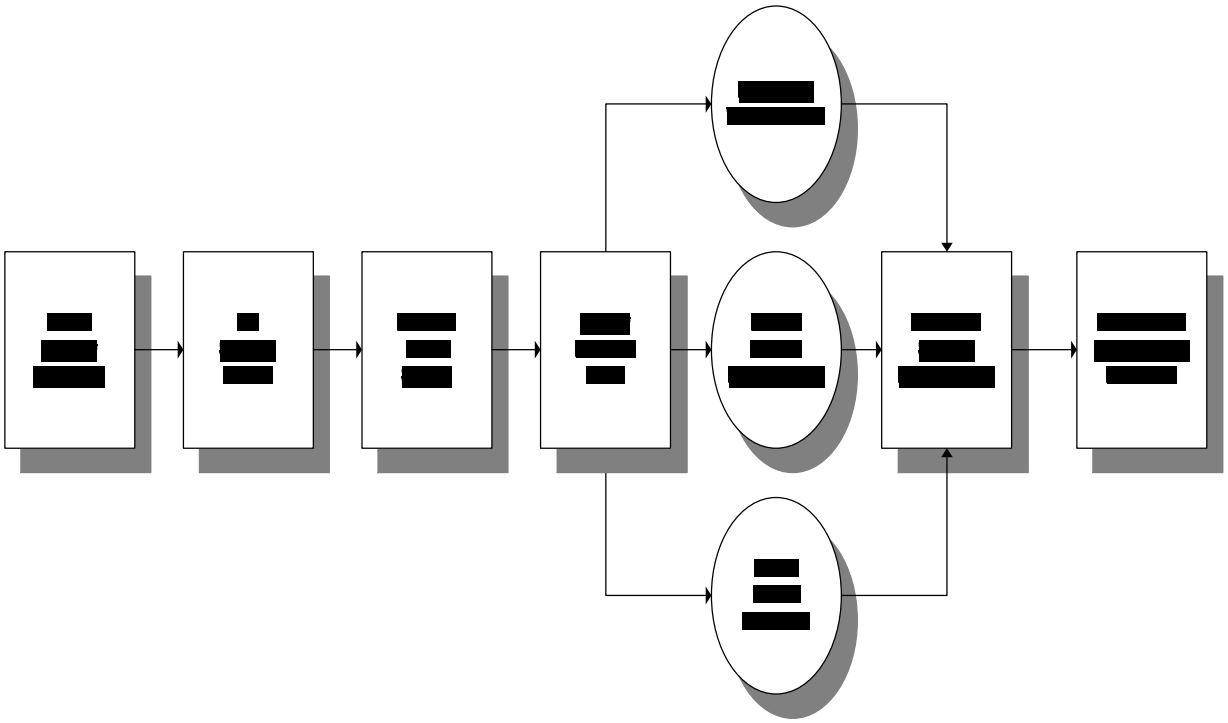

Figure 2. Security-driven requirements analysis process

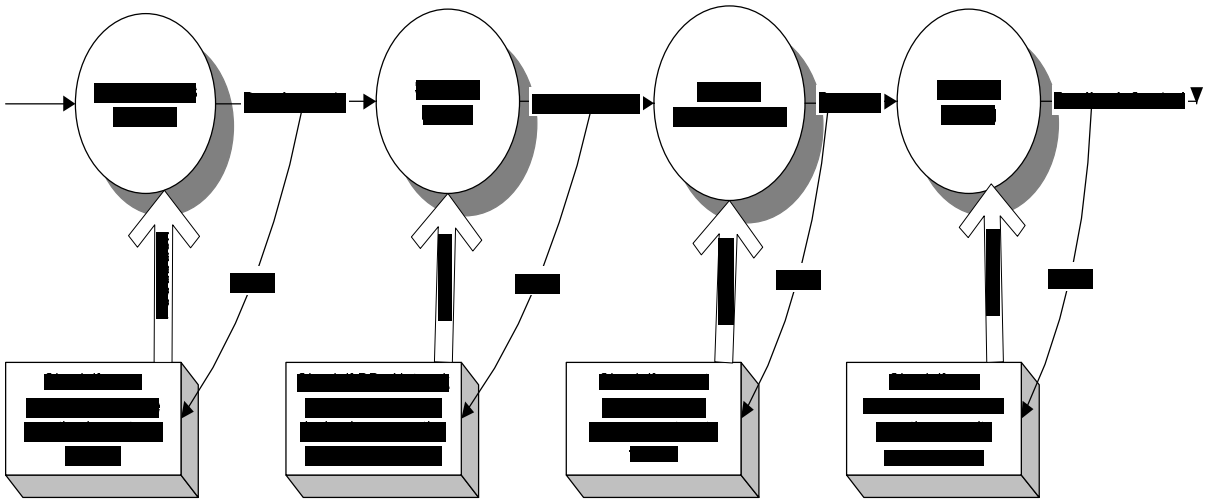

Figure 3. A software engineering approach to systems security via traceability analysis

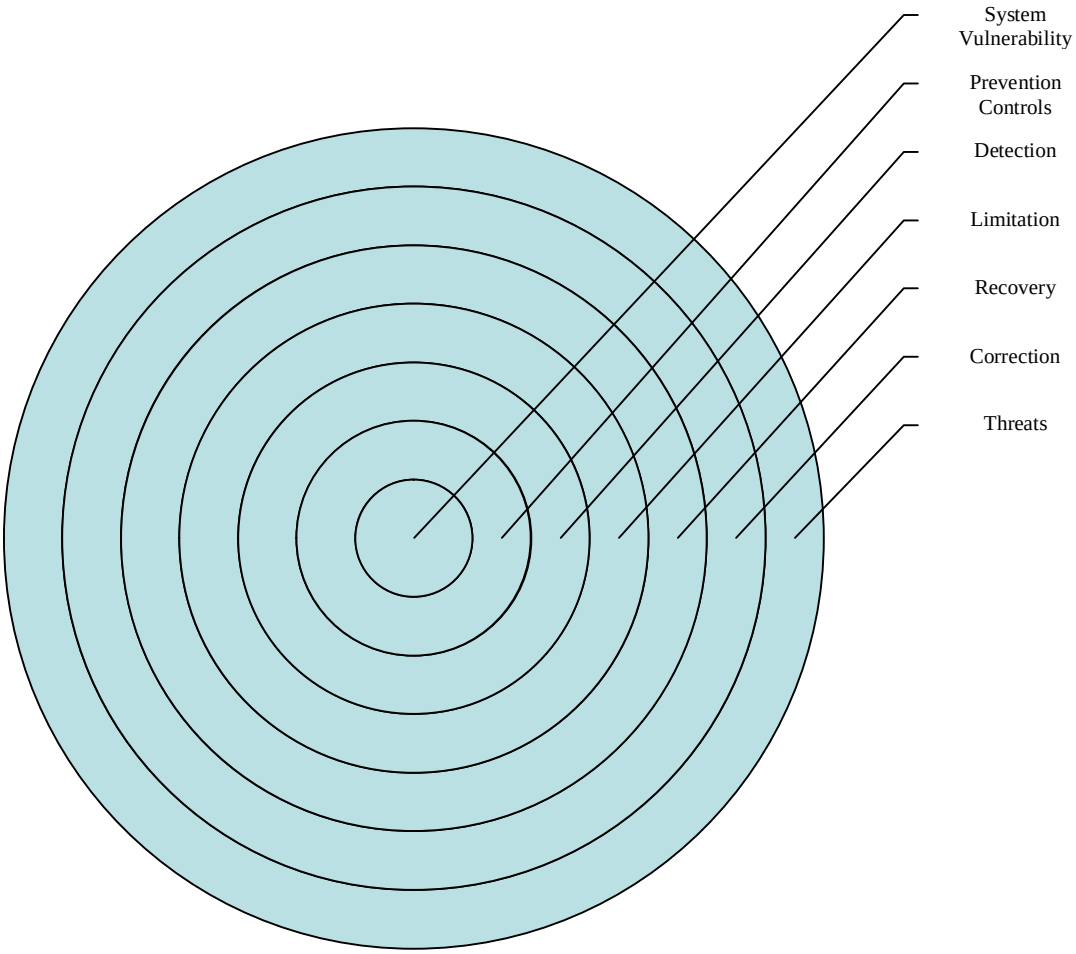

Figure 4. Seven-layer conceptual model for defense strategies in security-focused system design

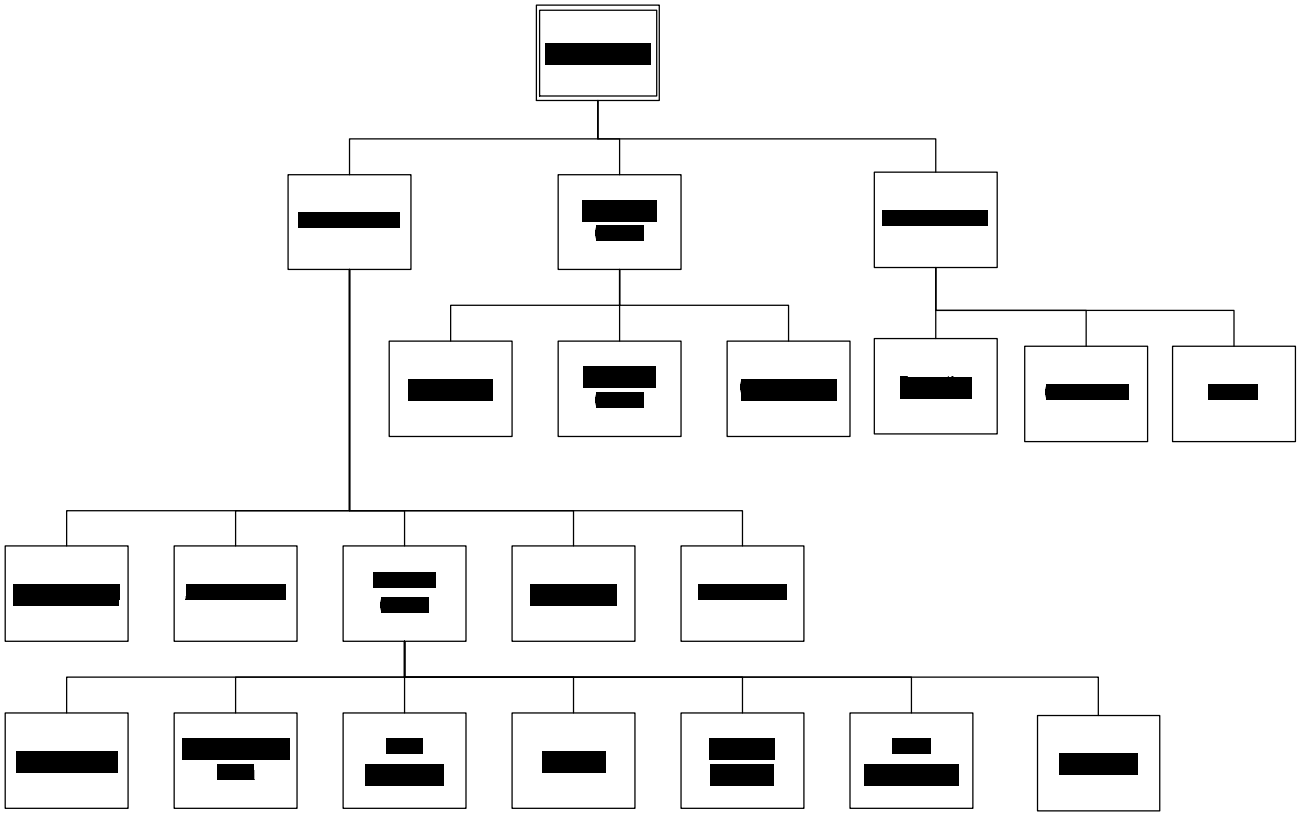

Figure 5. A basic taxonomy for security control techniques in software systems *(based on Turban et al., 2002)*

Table 1. The Roles of Pervasive Systems Development Layers

Layer	Rationale	Software Development Ramifications
Physical	-The flow of control in pervasive applications may depend on signals received from or to user's physical body. -Excellent software architecture is ineffective in pervasive devices unless well supported by hardware design that mirrors physical characteristics of humans.	Designing effective hardware architectures is crucial to software design since software effectiveness is dependent on hardware usability and hardware is irreplaceable as in desktop computing.
Resource	-Represents the infrastructure of pervasive software applications (operating systems, logical devices, system API, user interface, network protocol)	-ROM-based operating systems should be reliable with early releases since it will be very costly to make any upgrades thereafter. -System resources must be matched to user goals ands needs. -User interfaces must be intuitive and consistent. They must accommodate users' language and physical limitations. -Networking features should be automatically available, self-configuring and compatible with existing technology. -System storage must enable users to access, retrieve, and organize information the way that suit their requirements. -Execution environment and volatile memory should be responsive and provide not only speed but also sense of control with the ability of multithreading and multitasking.
Abstract	Represents the direct software application that the user will use.	-Maintaining compatibility between users' mental model "expectations" and application logic "state". -Considering short time frames available to pervasive system users for learning about the system as opposed to desktop users. -Considering the difficulty of physical conditions encountered by mobile users when designing pervasive systems. -User involvement and participation is much more critical in pervasive applications when compared with traditional applications.

Intentional	Represents user goals and purposes in using the pervasive system.	Analyzing the system to determine user goals and designing the system to fulfill these goals

Table 2. Life Cycle Phases for Real Time Software Systems

Phase	Phase Definition	Phase Application
Requirements Analysis and Specification	As in other approaches, user requirements are analyzed, and system specifications are formulated to elaborate on these requirements.	-State transition diagrams are preferred to describe the different states of the system to the user. Object-oriented-UML–based state transition diagrams is the next generation to carry out this technique more effectively. -Any operator interaction with the system should also be explicitly specified. -Throwaway repaid prototyping techniques have proven to be extremely effective in requirements analysis for real-time systems.
System Design	While the system is structured into tasks as in other software systems, real time systems are designed with focus on concurrent processes and task interfaces.	-The asynchronous nature of the functions within the system is a key characteristic that distinguishes decomposing real-time software systems into concurrent tasks. -Data flow diagrams, and event-trace diagrams are very effective techniques in mapping this phase.
Task Design	Each task is structured into modules, and module interfaces are defined.	Task-structure charts with intensive project and team management are essential to carry out task deign efficiently.
Module Construction	Detailed design, coding, and unit testing of each module is carried out.	This is similar to module construction in other system development approaches.
Task and System Integration	Modules are integrated and tested to form tasks, which, in turn, are gradually integrated and tested to form the total system.	-Incremental system development is used to achieve task and system integration.
System Testing	The whole system or major subsystems are tested to verify conformance with functional specifications. To achieve greater objectivity, system testing is best performed by independent test teams.	Automated testing is widely used for real-time systems.
Acceptance Testing	This is performed by the user.	-Extends user involvement to the validation and verification stages after system delivery.

Table 3. The Four Generations of Information Systems Security Approaches

Generation	Drivers	Strategies	Techniques	Problems
First (Early 80's)	-Generic thinking -Common sense principles.	Linking requirements "what to do" to existing capabilities: "what can be done".	Risk analysis	Gaps between generic strategies and special needs.
Second (Late 80's)	-Some focus on organization requirements.	Formal methods.	Control points and checklists.	Considering natural, functional and technical requirements while ignoring the social nature of organizations.
Third (Early 90's)	-Business processes -Focus on specific organization requirements	Information systems modeling.	-Responsibility modeling - Security semantics -Logical approach -ERM, DFD, OO and business process modeling for security.	Not enough focus on social requirements of organizations.
Forth (Late 90's up to recently)	-Socio-Technical Design -User Participation -Strong focus on specific organizational requirements	Domain-specific and application-driven design for information systems security.	-Responsibility modeling -Viable information systems	Still in its first phases.