Compliance by Design – Bridging the Chasm between Auditors and IT Architects

Klaus Julisch [a)], *Christophe Suter* [b)], *Thomas Woitalla* [b)], *and Olaf Zimmermann* [a)]

[a)] *IBM Research GmbH, Säumerstrasse 4, 8803 Rüschlikon, Switzerland*
[b)] *PricewaterhouseCoopers AG, Birchstrasse 160, 8050 Zürich, Switzerland*

Abstract: System and process auditors assure – from an information processing perspective – the correctness and integrity of the data that is aggregated in a company's financial statements. To do so, they assess whether a company's business processes and information systems process financial data correctly. The audit process is a complex endeavor that in practice has to rely on simplifying assumptions. These simplifying assumptions mainly result from the need to restrict the audit scope and to focus it on the major risks. This article describes a generalized audit process. According to our experience with this process, there is a risk that material deficiencies remain undiscovered when said simplifying assumptions are not satisfied. To address this risk of deficiencies, the article compiles thirteen control patterns, which – according to our experience – are particularly suited to help information systems satisfy the simplifying assumptions. As such, use of these proven control patterns makes information systems easier to audit and IT architects can use them to build systems that meet audit requirements by design. Additionally, the practices and advice offered in this interdisciplinary article help bridge the gap between the architects and auditors of information systems and show either role how to benefit from an understanding of the other role's terminology, techniques, and general work approach.

Keywords: Information systems audit, CAVR, compliance, security architecture, patterns, service-oriented architecture, business processes, enterprise applications.

1 Introduction

Accounting standards such as the International Financial Reporting Standards (IFRS), United States Generally Accepted Accounting Principles (US-GAAP), and the German "Handelsgesetzbuch" provide guidelines for the transparent and comparable reporting of financial information. Moreover, publicly traded companies have to follow additional regulations that are prescribed by regulatory bodies such as the Security and Exchange Commission (SEC) in the USA. Arguably, the best-known of these additional regulations is the Sarbanes Oxley Act of 2002 (Congress, 2002).

Financial auditors verify that a company's financial statements are compliant with the applicable accounting standards and correct in all *material respects*. Financial auditors define *material correctness* as errors that are negligibly small in relation to the monetary amounts reported in the financial statements (for example, not more than 5% of earnings before interest and taxes). Financial auditors are concerned with how financial information is captured, aggregated, contextualized, and disclosed to the public. As a prerequisite, the correctness and integrity of the financial information has to be verified. This verification is the responsibility of *Systems and Process (S&P) auditors*.[1] S&P auditors examine the *business processes* that handle financial data (in the widest sense) and the *information systems* that support these business processes: They investigate whether these business processes and information systems assure the correctness and integrity of the financial data they process. We will later define what we mean by "correctness" and "integrity"; at this point, it is sufficient to intuitively understand that correctness and integrity refer to the completeness, accuracy, validity of, and restricted access to financial data.

The S&P auditor's role is often taken by Certified Information Systems Auditors (CISA) (ISACA, 2010). This certification is awarded by the Information Systems Audit and Controls Association (ISACA). The responsibilities of the S&P auditor are, however, broader than those of the CISA because the S&P auditor also has to identify (and in many cases reverse engineer) the financially relevant business processes. This requires business skills and experience.

In this interdisciplinary article, we examine information systems both from the perspectives of *auditing* and of *designing* them. More specifically, we first discuss how we review information systems in the S&P auditor role, and we show how commonly made assumptions can undermine the quality of audits if they turn out to be unfounded. We then focus on *enterprise applications* (Fowler, 2003) as the core component of many information systems and identify thirteen control patterns that, according to our experience, lead to enterprise applications that are easier to audit and less prone to negative audit findings. By doing so, this experience report shows (a) how proven control patterns can be applied to build enterprise applications that

[1] When this article mentions *financial auditors* and *S&P auditors*, we mean two different *roles* that are defined by the *activities* they perform. In practice, a single individual can play both roles on a particular audit.

satisfy S&P audit requirements by design; (b) this article further bridges the gap between the designers and auditors of information systems and gives professionals in either community practical advice on how they can benefit from a better understanding of the terminology, techniques, and work approach of the other community; (c) the article finally includes references that guide the reader to detailed material on the implementation of the control patterns.

The remainder of this article is structured as follows: Section 2 introduces our terminology and related work; Section 3 describes a generic S&P audit approach and critically appraises it using an example; Section 4 identifies the control patterns (to be) applied by information technology (IT) architects when developing applications that are subject to audits; Section 5 concludes the article and summarizes its main points.

2 Background and Related Work

2.1 Terminology in S&P Auditor and IT Architect Communities

S&P auditors and IT architects use different terminologies. Even a simple concept such as "transaction" can be the source of confusion as it is interpreted differently by either community. This section therefore introduces a common terminology that is understandable to auditors and IT architects.

Figure 1 summarizes the terminology introduced so far: *financial auditors* audit *financial statements* as directed by the applicable *Generally Accepted Auditing Standards (GAAS*, e.g., (AICPA, 2001)). The financial statements are produced by *accountants* using *financial data* which is collected, processed, and stored in *business processes*. The business processes are supported and partially automated by *information systems*, which can informally be defined as integrated hardware and software systems for collecting, storing, processing, and presenting data. *Enterprise applications* are a special type of software that forms an integral part of information systems. More specifically, enterprise applications are large, complex, physically distributed, and generally mission-critical software applications that process business data (Zimmermann, 2009). Examples of enterprise applications include software for payment processing, human resources management, customer relationship management, and business intelligence.

It is important to understand that the activities of the financial auditor focus on the financial data appearing above the dashed line in Figure 1. To assure the correctness and integrity of this data, the financial auditor commissions an auditor in the S&P auditor role. It is an S&P auditor's responsibility to audit the business processes and information systems with the objective to assure the correctness and integrity of the financial data that these processes and systems produce. As indicated in the introduction (Section 1), this article focuses on the S&P auditor's role that assesses business processes and information systems.

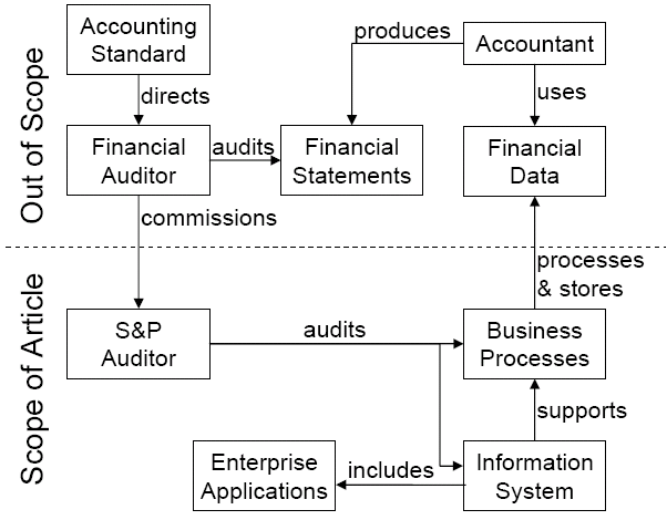

Figure 1: Roles and Relationships in Financial Audits

A business process – shown as an atomic box in Figure 1 – is a workflow that connects *activities* according to their order of execution (Leymann and Roller, 1999). Activities are units of work such as "book flight", "review travel expense", or "credit account". The flow of activities does not have to be sequential: *Gateways* fork the flow of activities into multiple parallel strands, or they merge multiple parallel strands into a single one. The flow of activities is also affected by *events.* Events are external stimuli such as customer orders arriving. Figure 2 shows how activities, gateways, and events are represented in the Business Process Modeling Notation (BPMN). Business processes frequently also contain more granular structures, which can also be expressed in BPMN. We do not discuss these BPMN structures as they are not needed to follow the article.

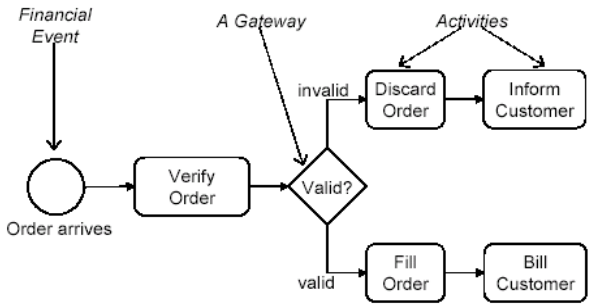

Figure 2: Description of a Business Process in BPMN

In this article, we are exclusively concerned with business processes that process financial data. An event that triggers such a business process is called a *financial event*. Financial events are known as *transactions* in the audit community, but we will avoid this term as it has a very different meaning in the field of software engineering (Fowler, 2003). A financial event always creates a data record, which is subsequently processed by business processes. We call these data records *financial Business Objects (fin-BOs)* throughout this document. In summary, *financial events* create *fin-BOs*, which are processed by the *activities* of business processes. *Gateways* fork and merge the flow of activities within a business process.

When processing fin-BOs, enterprises have the business objective of providing a fair and accurate representation of financial information. To ensure that this objective is achieved in the presence of adverse circumstances such as accidents, frauds or cyber-attacks, enterprises deploy *controls*. In general, controls are the mechanisms (including processes, structures, culture, and tasks) that an enterprise deploys in its processes and systems to ensure that business objectives are met (Bradshaw and Willis, 1998). In line with Figure 1, the Sarbanes-Oxley Act of 2002 (Congress, 2002) and the Public Company Accounting Oversight Board (PCAOB) Audit Standard No. 5 (PCAOB, 2007) focus on controls over financial reporting and distinguish between audits over controls and audits over financial statements, which are then integrated.

Examples of controls are firewalls, the four-eye principle, or the cross-checking of documents. A control is *adequate and effective* to the extent that it provides reasonable assurance (a.k.a. "evidence") that the organization will achieve its objectives reliably. *Compensating controls* complement weak or deficient controls and reduce the risk of such control weaknesses resulting in errors or omissions. Further aspects of compensating controls, such as the fact that they generally operate on a higher level than the controls they complement, are beyond the scope of this article. *Manual controls* are controls that are performed "manually", i.e., by people, while *automated controls* are implemented by machines (systems). For example, a computer program that matches invoices against orders implements an automated control, while the four-eye principle is a classic manual control. *Semi manual controls* or *IT dependant controls* combine manual activities (e.g., the verification of information) with automated procedures and information processing (e.g., report generation). Therefore the review of a report is such a control as it relies on the information in the report and does not challenge the content itself.

2.2 Related Work

In this section, we position our work relative to the ISO 27K (ISO/IEC 2005a, 2005b), COBIT (IT Governance Institute, 2005), and ITIL (OGC, 2007) standards. All three standards are used heavily in security and compliance projects. It is therefore important to understand how they are different from the practices presented in this article. We also relate our work to the classic CIA (Confidentiality, Integrity, and Availability) and AAA (Authentication, Authorization, and Auditing) security models.

ISO 27001 standardizes the *activities* in an Information Security Management System (ISMS). The activities defined in the standard describe a continuous cycle of planning, implementing, monitoring, and improving information security. In terms of ISO 27001, this article focuses on the planning and implementation activities. Moreover, the article investigates these activities towards the goal of building enterprise applications (rather than generic IT systems) that meet the specific requirements of S&P auditors (rather than generic security requirements). Section 4.4 elaborates on this point by placing our work in the context of a management lifecycle that is similar to the one defined in ISO 27001.

ISO 27002 is a catalogue primarily consisting of IT general controls whereas our article focuses on application controls and explains how to use such controls effectively to meet the requirements of S&P auditors. Moreover, it is common in S&P audits to assume that IT general controls had previously been verified (see Section 3.2). ISO 27002 with its focus on IT general controls is therefore largely complementary to our work.

COBIT (Control Objectives for Information and related Technology) is the internationally recognized framework for IT governance, i.e., for assuring that information technology is aligned with business requirements and that IT risks are mitigated by controls. COBIT covers all aspects of IT governance; security is only a part of it. Out of the 34 COBIT control objectives, only PO4, PO9, AI2, DS5, DS7-9, DS11, and DS12 are related to IT security. The remainder of COBIT focuses on other aspects of IT governance. Our paper, by contrast, is focused on compliance (not governance), enterprise applications (not entire IT systems), and architecting and auditing of such applications (not their entire lifecycle).

ITIL (IT Infrastructure Library) provides best practices for delivering IT services that are fit for purpose, stable, and reliable. ITIL Version 3 defines 25 processes that, to varying degrees, are required in the five lifecycle stages of a service. *Information security management* is one such ITIL process, which has the largest bearing on the design and operation stages of a service's lifecycle. The information security management process of ITIL Version 3 treats IT security comprehensively – including people, processes, infrastructure, and applications. While being broader in scope, ITIL is less detailed than this article in its specifics. This is particularly so as this article relates to achieving compliance with S&P audit requirements.

In summary, this article focuses on application controls for enterprise applications that process financial data and it shows how those applications can be built to meet the S&P auditors' requirements by design. By doing so, the article adds the next level of domain-specific detail to broader standards such as ISO 27K, COBIT, and ITIL. Most closely related to our work is ISACA's (2009) guide on COBIT and application controls; this article complements the ISACA guide with domain-specific technical recommendations.

Among IT security professionals, the CIA and AAA frameworks are widely used (Cross *et al.*, 2002). CIA stands for Confidentiality, Integrity, and Availability, which are key objectives of information security. AAA stands for Authentication, Authorization, and Auditing; they are three important mechanisms for achieving CIA. With respect to CIA, Section 3 will show that *integrity* is particularly important in S&P audits, while confidentiality violations pose a lesser risk; availability is

addressed by IT general controls (e.g., backups, fail-over mechanisms, etc.), which is out of the scope of S&P audits. With respect to AAA, Section 4 of this article will extend the AAA framework with additional control patterns and it will show how to apply these patterns effectively during the design phase.

3 S&P Auditor Examination of Business Processes and Information Systems

3.1 Audit Types, Scope, and Objectives

An audit is an assessment of an entity's ability to meet its objectives. There are many potential entities (e.g., products, people, organizations, projects) and objectives (e.g., reliability, policy compliance, strategic alignment, security) that can be postulated. Accordingly, there are many different types of audits, and any given audit is defined by its *scope* and *objective* (Senft and Gallegos, 2009). The *audit scope* defines the entity to be reviewed, including its geographic or functional area, time periods, documents, systems, and other specifics. The *audit objective* is a formal statement that describes the purpose of the audit. For example, a pharmaceutical company may audit its chemical plant (the scope) with the objective of assessing compliance with the applicable manufacturing quality standards such as Good Manufacturing Principles (EC, 2010).

The audit scope in this article is the business processes and information systems that are involved in the collection, storing, processing, and presentation of financial data. The audit objective is to verify that the business processes and information systems satisfy the following control objectives for handling fin-BOs:

- *Completeness (C)*: When a financial event occurs, one and only one fin-BO is entered and accepted for processing.
- *Accuracy (A)*: Fin-BOs are recorded at the correct monetary amount, in the appropriate account, and for the proper period.
- *Validity (V)*: Only authorized fin-BOs that represent financial events that actually occurred and that affect the organization's financial statements are recorded.
- *Restricted access (R)*: Fin-BOs are protected against unauthorized access and amendments.

The properties of completeness, accuracy, validity, and restricted access have jointly become known as the acronym of *CAVR* (Killmeyer, 2000). Assessing the CAVR properties defines the objective of the S&P audit.

The scope of S&P audits can be defined even more precisely, recalling that an S&P audit is generally commissioned by a financial auditor. The financial auditor seeks to assure external stakeholders of the correctness and integrity of financial accounts. Towards this goal, it is sufficient to preclude *material* accounting errors (with materiality being defined as "significant", relative to the total monetary

amounts reported). The S&P audit is therefore scoped to focus only on those business processes and information systems that have a *material impact* on the company's financial statements (Cascarino, 2007). By contrast, if an *internal audit* function commissions the S&P audit, then the scope can be defined to be any arbitrarily chosen business process. In a sense, internal audits treat *all* errors as "material", even when they result in no or only in tiny losses. In either case, the objective of the S&P audit is to assess compliance with the CAVR properties.

As an aside, the financial auditor's objectives are analogous to the S&P auditor's objectives of assessing CAVR (see Figure 1 for the relationship between financial and S&P auditors). In fact, the CAVR properties mirror the so called *financial statement assertions*, which consist of completeness, accuracy, existence or occurrence, cut-off, valuation or allocation, rights and obligations, presentation and disclosure (PCAOB, 2007). The purpose of financial statement assertions is to assert the existence of controls that assure a fair and accurate representation of financial information; the exact definition depends on the accounting standard. For example, the financial statement assertion of "completeness" states that controls have been put in place to assure the completeness of information according to the used standard (e.g., all qualified write-offs are accounted for). Additionally, the completeness assertion assures that CAVR-type controls are implemented so that, for instance, no fin-BOs representing write-offs are lost for technical reasons. A single control can, but does not have to fulfill completeness in both senses. We will not consider financial statement assertions any further in this article.

3.2 Audit Approach and Assumptions

S&P auditors use the following five-step examination method (or variants thereof) to assess business processes and information systems (Cascarino, 2007; Senft and Gallegos, 2009):

Step 1 (Scope definition): The auditor identifies the material accounts in the financial statements. The auditor then identifies the fin-BOs that have a material impact on these accounts. The business processes and information systems that create and manipulate these "material fin-BOs" constitute the scope of the S&P audit.

Step 2 (Risk assessment): Once the material fin-BOs have been identified, a *walkthrough* is performed for each of them. This walkthrough starts with the financial event that creates the fin-BO and follows it through the business processes that manipulate it, all the way to the financial statements in which it becomes a line item. During the walkthrough, the S&P auditor identifies the *risks* that threaten to destroy the CAVR properties of a fin-BO. Risks are adverse events that result in failures or losses, e.g., the modification of a charge on an invoice or the double-paying of a bill.

Step 3 (Control identification): Next, the S&P auditor identifies the controls associated with each risk. As previously explained, these controls are the mechanisms an organization has established to mitigate the risks so that the organization is able to achieve its business objectives.

Step 4 (Control testing): The S&P auditor then evaluates the adequacy and effectiveness of the controls that Step 3 identified. Two questions need to be answered here: First, are the controls (theoretically) sufficient to mitigate the identified risks, i.e., are they adequate? And second, do these controls operate correctly, i.e., are they effective? To answer these questions, S&P auditors collect *evidence*, i.e., documents, interviews, log files, test runs, physical inspections and any other relevant information that can be used to form a conclusive opinion on the adequacy and effectiveness of controls (Bitterli et al., 2009).

Step 5 (Reporting): Finally, S&P auditors document their findings and present them to management and other stakeholders, e.g., application and audit owners.

Note that S&P audits do not seek to *prove* CAVR-compliance with this method. Rather, they seek to provide *reasonable assurance* thereof. Reasonable assurance is a high, but not absolute level of evidence that provides prudent officials with a sound basis for concluding that the audit objectives are met (AICPA, 2006; SEC, 2007).

Assumptions: In conducting their work, S&P auditors make several assumptions:

1. In Steps 3 and 4, S&P auditors do not concern themselves with *IT general controls*, but rather assume that these have been assessed previously in an IT audit. In practice, the IT audit and the S&P audit can be performed by the *same* person but acting in the *different roles* of IT auditor and S&P auditor, respectively. *IT General Controls (ITGC)* (IT Governance Institute, 2005; Bayuk, 2004) are application-independent controls that protect the IT environment in which applications run. Examples of IT general controls are adequate cooling and power supply, infrastructure security, user access management, and sound management policies. IT general controls can be contrasted with *application controls* (IT Governance Institute, 2005; Bayuk, 2004), which are the unique controls that are embedded in a specific business process or information system. For example, a cryptographic signature that prevents the unauthorized modification of fin-BOs in an information system is an application control. It is noteworthy that this first assumption is not inevitable. In fact, rather than relying on IT general controls being tested, S&P auditors may also use techniques such as extensive sampling to assure that IT general controls effectively support the relevant application controls.
2. When S&P auditors test controls (step 4), they do not test the software functionality, but rather assume that the software works as specified by the software vendor. Note that by testing application controls, S&P auditors also implicitly test some aspects of software functionality; however, this is not the purpose of the S&P audit. An example of such an implicit test occurs when the testing of a debit operation also (implicitly) confirms that the software correctly subtracts numbers as specified in the software manual. Such confirmation is welcome, but it is not the objective of the S&P auditor to collect it. This second assumption may be further justified by the fact that the specified software functionality has been *certified* by a third party. Such third-party certification can be thought of as an audit of the software development

process. Successful certification means that the software development process has adequate controls to assure that the produced software implements its specified functionality correctly. In certain cases, additional audit procedures might be needed, e.g., if the customization of a software package represents a significant deviation from its off-the-shelf configuration.

3. It is assumed that processes cannot be bypassed. More specifically, during the walkthrough in Step 2 of the audit method, the S&P auditor follows individual fin-BOs in order to identify the business processes that manipulate them. In addition, the S&P auditor may utilize interviews, process maps that depict the flow of fin-BOs, or other techniques. Ultimately, the S&P auditor's time is finite and at some point it therefore has to be assumed that all processes have been identified and that each and every fin-BO follows one of the identified processes. In particular, it is assumed that potential "shadow processes" have been found and are known (i.e., visible to application owners and auditors). This assumption can be false because, for example, fraudulent insiders such as Jerome Kerviel of Société Générale (Epstein, 2008) can know loopholes that circumvent the official processes. More generally, attackers and fraudsters always target the "weakest link" in a system, while sampling or walkthroughs are inherently limited in their ability to identify all "weak links".
4. S&P auditors assume that it is not necessary to estimate the risk with mathematical accuracy. Rather, auditors assess risks based on their professional judgment which is grounded in their formal education, work experience, and contextual knowledge of past incidents and the overall IT environment.
5. It is assumed that best practices, personal and professional judgment, and experience are sufficient to select appropriate risk-mitigating controls.

The first two assumptions are born out of the need to limit the scope of S&P audits. In other words, S&P auditors have to assume that certain things (e.g., IT general controls and software functionality) just "work" because testing every control completely would require an infinite amount of time and budget. Nonetheless, it is important to be cognizant of these assumptions because they can destroy the CAVR properties when they are unjustified. For example, a failed IT general control may cause computers to be unpatched, which opens them up for hackers to create fraudulent fin-BOs – a clear violation of the CAVR principles.

The assumptions 3 to 5 affect the way in which S&P auditors perform their work, rather than the scope of their work. Each of these assumptions can have far-reaching consequences. For example, the existence of unknown (invisible) "shadow processes", i.e., processes that remain invisible during the S&P audit, can falsify assumption 3 and destroy the CAVR properties for all fin-BOs that flow through these processes.

Assumption 4 implies that the risk assessment is as good as the S&P auditor's judgment. However, human judgment is fallible and this may be most apparent when risks are correlated. For example, assume the S&P auditor notices that an audited enterprise has weak access restrictions ("R") for its order fin-BOs. Will the

S&P auditor classify this risk as "low" because the impact of disclosing order data is small? Or will the S&P auditor ask what a malicious employee could do if (s)he had information about past orders? Would the S&P auditor notice that, by having knowledge of past orders, a malicious employee could issue second payments to fictitious suppliers whose names resemble the names on the order forms and thereby pass the control ensuring that payments match orders? Would the S&P auditor then check whether there are controls that prevent such double-payments of the same order? And how would the S&P auditor finally rank the *real* risk of disclosing order fin-BOs?

Assumption 5 is problematic on several levels. First, the argument can be made that we do not have enough actuarial-like data in information security to know what (truly) best practices are. Second, the actual individuals performing S&P audits are typically entry-level associates, which may lack the experience or knowledge to choose appropriate controls. Third, *regulatory capture*, i.e., the situation where regulators serve the interests of the institutions they regulate (Kidwell *et al.*, 2008), threatens to undermine best practices. Lastly, best practices evolve at the speed of consensus-driven committees (i.e., very slowly), while threats evolve at Internet speed (i.e., very fast). For example, while best practices still approve of virus scanners as a means to control malware, the hacking community has long known ways to evade virus scanners and to render them less effective than desired (Baker et al., 2009; Symantec, 2009). Similarly, to prevent fraudulent or erroneous payments, best practice has long recommended controls to check that each supplier invoice matches a purchase order and no order is paid twice. These controls have been bypassed by suppliers that did not ship what had been ordered and invoiced, i.e., they sent either fewer or inferior items. This fraud (and frauds in general) work until they become so common and prevalent that best practices are updated to include further controls that prevent them (and, in fact, today's three-way-matching control (Schaeffer, 2006) prevents the particular fraud in this example). The problem is not that one cannot provide controls mitigating all risks, but rather that the use of best practices systematically introduces control gaps and weaknesses, hence falsifying Assumption 5.

Unfortunately, we have no silver bullet that makes the above assumptions justified in all instances. However, in Section 4 we will outline proven control patterns and methods that, according to our experience, reduce the risks of these assumptions being incorrect and, consequently, becoming the source of control failures.

3.3 Example S&P Audit of a Sales Process

We now describe the S&P audit of a sales process. The sales process and its underlying information system are depicted in Figure 3. To simplify the description, we do not consider any cost and price calculation for goods sold; we also exclude aspects such as sales order confirmations. The key elements are as follows:

1. A customer triggers an order (the financial event at the left-hand side of the figure).
2. The producer receives the order and checks if it can fulfill this order out of its inventory.

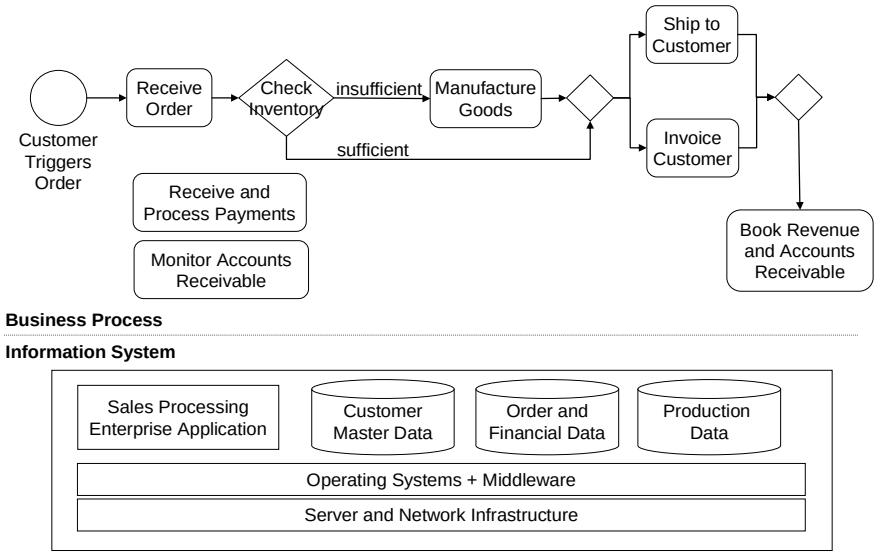

Figure 3: Business Process and Information System for Sales Processing

3. If the inventory does not contain the products ordered, then they are manufactured.
4. Once the ordered goods are available, they are shipped to the customer and the customer is invoiced.
5. The revenues are booked on the income statement, and the money that the customer owes to the producer is recorded under accounts receivable on the balance sheet of the producer.
6. Two additional activities that the producer engages in are the receiving and processing of payments, as well as the monitoring of accounts receivable. The first of these activities handles the book keeping for payments that customers make to settle their invoices. The second activity checks for customers whose payments are overdue and sends them reminders.
7. Below the dotted line, Figure 3 shows the information system that supports the sales process. The information system includes one enterprise application and three databases containing customer master data, order and financial data, and production data.

With the audit scope being the process and system of Figure 3 and the audit objective being to assess if the business process guarantees CAVR, the S&P auditor identifies the customer order as the material fin-BO and follows it from the start to the point where it gets recorded in the financial statements. Along the way, the S&P auditor identifies various risks that threaten to destroy CAVR. The auditor further assesses if there are adequate and effective controls to mitigate these risks, and (s)he makes various assumptions about the information system. By way of illustration, we now consider three examples.

Example 1: Erroneous customer order

- *Risk:* The S&P auditor identifies the risk that the customer's order may be erroneous, e.g., quoting the wrong product price, containing incorrect billing or shipment information, or being entirely fictitious to meet some sales incentive (step 2 of the audit method from Section 3.2).
- *Controls:* The S&P auditor checks which controls exist to mitigate these risks (step 3 of audit method). Based on professional experience and the evidence the auditor collects, (s)he further judges whether these controls are adequate and effective (step 4). For example, the S&P auditor may be satisfied to find a control that cross checks the information on the order with the customer master data that the producer maintains about its customers. However, the auditor may notice that orders are submitted using unencrypted email. The risk here is that anyone could fake such orders. This destroys the validity property of CAVR and constitutes an audit finding.
- *Assumptions:* The S&P auditor learns that customer orders are received and processed by an integrated Enterprise Resource Planning (ERP) package, which is protected by a firewall to fend off cyber attacks. Firewalls are IT general controls, and following Assumption 1, the S&P auditor therefore assumes that the firewall is adequate and effective at preventing cyber attacks.

Example 2: Erroneous invoices

- *Risk:* The S&P auditor identifies the risk of erroneous invoices, such as the wrong invoice amount, incorrect Value Added Tax (VAT) amounts in cross-border shipments, failure to invoice certain goods shipped, or the existence of invoices for goods that were not shipped.
- *Controls:* The S&P auditor identifies a control that verifies invoices against orders to assure correct quantities, delivery addresses, and prices. Furthermore, an evaluation of log files provides evidence that this control operates correctly. Sampling the control with test data further corroborates this judgment. The S&P auditor therefore assesses that the risk of erroneous invoices has been mitigated.
- *Assumptions:* The Value Added Tax (VAT) is calculated by an ERP system and – given the ERP system's certification – it is assumed that the VAT is always correct.

Example 3: Transfer of fin-BOs between integrated information systems

- *Risk:* The S&P auditor identifies a manually operated file transfer interface between the sales processing application and another information system not shown in Figure 3, the production system. More specifically, the S&P auditor observes that when inventory is insufficient to fulfill an order then an operator manually transfers a file containing the order information from the sales processing application to the production system.

- *Controls:* The S&P auditor identifies a control that restricts access to the file and thereby enforces the R-property (restricted access). The file contains also a cryptographic signature which leads to an error the file or its signature is manipulated during the file transfer. This signature secures validity according to CAVR.
- *Assumptions:* The copying of files between the sales processing system and the production system is standard software functionality and therefore assumed to work correctly.

4 Architecting for CAVR Compliance

As explained in Section 3, Completeness, Accuracy, Validity, and Restricted access (CAVR) are the four properties that S&P auditors assess when auditing business processes and enterprise applications. The challenge for IT architects therefore is to design and implement enterprise applications that are CAVR-compliant; they can do so by properly configuring the native controls of commercial software packages or by designing additional controls into their custom developed enterprise application architectures. The latter is frequently done by complementing enterprise applications with special-purpose, out-of-the-box Governance Risk and Compliance (GRC) software, which (among other things) offers libraries of automated controls (Trent, 2008). In this section, we present a systematic approach for achieving CAVR compliance during these activities.

Our approach to building CAVR-compliant applications has three parts: First, we compile a *control pattern catalog* from the literature that highlights proven mechanisms that, according to our experience, are most suitable to control risks and prevent violations of the CAVR properties (full pattern descriptions and implementation advice can then be found in the referenced literature). Second, we present a *design method* for applying controls from the catalog to one's architecture. Third, we outline how to realize selected controls in *layered enterprise application and integration architectures*. We also discuss the importance of application lifecycle management for CAVR compliance and describe several risky designs that architects should be aware of.

4.1 Control Pattern Catalog for CAVR Compliance

Controls are the mechanisms that an enterprise deploys in its processes and systems to assure that business objectives are met. Various control catalogs have been suggested, e.g., by the IT Governance Institute (2005), ISO/IEC (2005b), and Ross et al. (2007). Here, we focus on application controls helping enterprise applications to satisfy the CAVR properties. Other controls such as IT general controls relating to physical, network and server security are outside the scope of this article and can be found in the previously mentioned standards. The software engineering community recommends architectural *patterns* (Blakley and Heath, 2004; Schumacher et al.,

2006; Yoder and Barcalow, 1997) as a state-of-the-art way to sketch and share reusable designs. Hence, Table 1 collects thirteen proven *control patterns* that IT architects can apply to assure compliance with each of the four CAVR properties.

We selected these thirteen patterns from the vast body of knowledge that can be found in the literature. The rationale justifying this particular compilation is:

- All thirteen patterns address design problems imposed by one or more of the CAVR properties.
- These patterns have been successfully applied in practice in multiple occasions (according to our own industry project experience and the literature).
- Combining them into an end-to-end information system and enterprise application architecture is technically feasible and yields a complete response to our goal (i.e., solution to our design problem), compliance by design.

The remainder of this section and Section 4.3, sketch the control patterns in more detail; for space reasons, full pattern descriptions and implementation advice could not be included in this article but can be found in the referenced literature.

STANDALONE OBJECTS. To ensure that all fin-BOs are entered and accepted for processing once and only once (namely when a financial event occurs), they should be represented explicitly in the architecture so that they are easy to locate for architects, developers, application maintainers, and reviewers. This can be accomplished a) by tagging them as fin-BOs in an analysis- and design-level *domain model* (Evans, 2003; Fowler, 2003) and b) by adding fin-BO components to the functional architecture. With respect to a), a *domain model* specifies the key concepts in a particular application domain (e.g., pharmaceutical industry or telecommunications) in a standardized, often machine-readable way; it covers data structures and behavior as well as the relations between the different concepts. Class diagrams from the Unified Modeling Language (UML) are often used to document domain models during requirements engineering and architecture design. During development, the concepts from the domain model become a source of components in the architecture, e.g., HTML forms displayed in a Web browser, server-side business objects and service components coded in Java, PHP, or other programming languages, and tables in a relational database. The usage of such components should be controlled similarly to the access to the entire system (e.g., using access management middleware providing authentication and authorization services).

SINGLE ACCESS POINT. An application is difficult to test and audit when it has multiple “front doors”, “back doors”, and “side doors” for entering the application. It is therefore recommended to set up only one way to access an application (Yoder and Barcalow, 1997). In our context, this means setting up only one interface through which fin-BOs can be created and modified. That way, it becomes easier to assure completeness because a single interface has to be tested for duplications and omissions and correct treatment of relations to and from other fin-BOs.

Table 1: Application Controls Enforcing CAVR Compliance

CAVR Property	Eligible Control Patterns
Completeness (C): Fin-BOs are entered and processed once and only once; each financial event is represented by exactly one fin-BO	• STANDALONE OBJECTS: Model fin-BOs explicitly and represent them as dedicated components in the functional architecture; • SINGLE ACCESS POINT: Provide a single interface through which fin-BOs can be entered into the application; • RELIABLE MESSAGING: Use messaging as integration style and message channels that guarantee delivery to prevent losing or duplicating fin-BOs that are passed among distributed components; • LOGGING: Maintain an audit log to document the operations performed on fin-BOs as well as the subject that triggered these operations; • COMPLETE MEDIATION: Every time a fin-BO is accessed, intercept and check the access attempt and deny it if it is not authorized;
Accuracy (A): The information in fin-BOs is correct and precise	• CONTROLLED INTERFACES: Give fin-BO processing components narrow Application Programming Interfaces (APIs) and/or communication interfaces that restrict the operations a user (or other component or other system) can perform on them; • LEAST PRIVILEGE: Grant subjects only the privileges they absolutely need to accomplish their tasks; • INPUT AND OUTPUT VALIDATION: Define pre- and post-conditions for each fin-BO and check all input and output against these conditions; in particular, cross-check the data in fin-BOs against master data repositories; • IDENTITY GRANULARITY: Assign each subject (user or activity) an identity of its own and force it to operate under this identity; avoid using group identities because such group identifies destroy accountability of subjects; • CRYPTOGRAPHIC SIGNATURES: Have the creator and modifiers of a fin-BO cryptographically sign it to vouch for its correctness; • LOGGING: As above; • COMPLETE MEDIATION: As above.
Validity (V): Fin-BOs correspond to real-word financial events	• SEPARATION OF DUTY: No single user has the power both to create fin-BOs and to review/approve these fin-BOs; • CROSS-CHECKING: Cross-check related fin-BOs (e.g., orders, invoices, and shipment confirmations) to detect inconsistencies; • INPUT AND OUTPUT VALIDATION: As above; may help detect inconsistencies such as payments to parties that are not in the supplier database; • CRYPTOGRAPHIC SIGNATURES: Require creators and modifiers of fin-BOs to certify their validity by cryptographically signing them; • IDENTITY GRANULARITY: As above; • LOGGING: As above.
Restricted Access (R): Fin-BOs are protected against unauthorized access and modification	• ENCRYPTION: Encrypt fin-BOs to prevent unauthorized access; • SEPARATION OF DUTY: As above; • COMPLETE MEDIATION: As above; • LEAST PRIVILEGE: As above; • IDENTITY GRANULARITY: As above; • LOGGING: As above.

RELIABLE MESSAGING. The "once and only once" aspect of the completeness property suggests *messaging* to be the preferred integration style to connect physically distributed software components within a single enterprise application, but also different enterprise applications (Hohpe and Woolf, 2004). *Message channels* then transfer messages with exactly-once *guaranteed delivery* semantics (Hohpe and Woolf, 2004), even if the message receiver becomes unavailable temporarily. In such a setting, the fin-BOs are treated as textual *document messages* that are transferred from application (component) to application (component) over the message channels (Hohpe and Woolf, 2004).

The accuracy property of CAVR requires that fin-BOs are transported from one processing component to another without modification. Hence, the message channels are subject to audit control; all systems management patterns from (Hohpe and Woolf, 2004) can be leveraged to facilitate such audits. For instance, a *wire tap* component intercepts a message flow without affecting it, thus allowing tools and humans to inspect the message content. A *message history* can be created for logging purposes.

LOGGING. Logging allows documenting the lifecycle of fin-BOs starting with the creation of fin-BOs and ending with their inclusion as line items in financial statements. All major operations performed on fin-BOs should be logged along with the user who triggered them. Major operations that should be logged are sometimes referred to as CRUD operations, which stands for the operations *create*, *read*, *update*, and *delete*. The *message store* pattern (Hohpe and Woolf, 2004) can be used to centralize these logging capabilities in loosely coupled integration architectures.

CONTROLLED INTERFACES. Fin-BOs should implement well-defined and restrictive interfaces which allow subjects to do exactly what they have to do, but not more. For example, consider a fin-BO that stores the number of items ordered. In a good design, the `add_item()` method is invoked when an additional item is ordered. A poor design, by contrast, would offer a method `set_items(int no_of_items)`, through which a subject can set the number of orders to *any arbitrary number*. Clearly, the `set_items(int no_of_items)` method offers more flexibility, and more opportunity to introduce inaccuracies. Tulach (2008) elaborates on this control pattern within the larger context of API design.

COMPLETE MEDIATION. Every access to every fin-BO must be checked to ensure it is allowed (Saltzer and Schroeder, 1975). This principle requires a sound method for identifying the subjects that trigger operations on fin-BOs. Complete mediation further restricts the use of caching: Rather than remembering the result of a prior authority check, the system should check each time if a subject (e.g., end user or activity in a business process as introduced in Section 2) is authorized to perform a given operation on a fin-BO. While this design has a negative impact on performance, it leads to a simpler implementation that can be tested and audited more easily.

LEAST PRIVILEGE. Each subject should be given only those privileges that are needed to complete its tasks. Again, subjects are users or activities. Howard and LeBlanc (2003) discuss the least privilege pattern in detail and offer many practical examples.

INPUT AND OUTPUT VALIDATION. All fin-BO processing activities that involve algorithms (e.g., customer segmentation, calculation of prices and discounts, loan processing, etc.) should follow the *design-by-contract* principle and check at run-time that inputs and outputs satisfy pre- and post-conditions (Meyer, 2000) that collectively ensure accuracy and validity. Input validation should pay particular attention to verifying border values (lower and upper boundaries), invalid input, canonization issues, and exceptional situations. Master Data Management (MDM) repositories should be used where available to check input and output data against the authoritative records. That way, inaccurate addresses or account data can be detected. Howard and LeBlanc (2003) offer more detailed advice and examples on performing input and output validation in practice.

IDENTITY GRANULARITY. Each subject (user or activity) should be assigned a dedicated identity of its own; subjects should then be forced to operate under these different identities (NIST, 1983). Having a unique identity allocated to each subject (be it an individual or an activity in a business process) is a prerequisite for having fine-grained access-control and accountability in a system. As a consequence, group identities such as "sales department" and default identities such as "guest" or "administrator" should be avoided. Hansen *et al.* (2008) discuss the privacy issues related to the use fine-grained identities.

CRYPTOGRAPHIC SIGNATURES. Require subjects to cryptographically sign fin-BOs that they create or modify. By virtue of this signature, subjects vouch for the validity and accuracy of the fin-BOs they create and manipulate. This principle enforces accountability, as signatures are tied to subjects in a non-repudiable manner. Key management and storing secret information such as cryptographic keys are generally considered the weakest link of cryptographic applications. Howard and LeBlanc (2003) as well as Ferguson and Schneier (2003) offer practical solutions to these problems.

SEPARATION OF DUTY (SOD). SoD (Benantar, 2006) is the principle of requiring more than one person to complete an activity. That way, SoD helps prevent fraud and accidental errors that jeopardize the validity or accuracy of fin-BOs.

CROSS-CHECKING. Invalid fin-BOs can be detected and prevented by comparing them to other fin-BOs that should contain matching information (Bragg, 2010). For example, a payment fin-BO without an order fin-BO is invalid in most circumstances. Similarly, a "capital expenditure" fin-BO that is not matched by a "new asset" fin-BO should raise warning flags. Bragg (2010) offers further examples of how the cross-checking control pattern can be applied in practice. When designed into an enterprise application, this control pattern can report such processing errors to users and log files.

ENCRYPTION. Encryption "scrambles" data so that only authorized parties – who have the key to descramble it – can read the data. This restricts the access to the data. Practical implementation specifics can be found in (Howard and LeBlanc 2003) and (Ferguson and Schneier, 2003).

For space reasons, we could only outline the patterns in this section. Having selected one or more of them, many subsequent design decisions have to be made – by definition, patterns are "soft around the edges" and only sketch a solution to a recurring design problem (Fowler 2003, Hohpe and Woolf, 2004). Section 4.3 adds another level of detail by describing how the control patterns should be applied in a Service-Oriented Architecture. Further details, e.g., on implementation activities, can be found in the referenced literature.

4.2 Design Method for Achieving CAVR Compliance

We now present a design method for selecting the specific controls that are best suited for achieving CAVR compliance. When applying the proposed method, we recommend that the following design principles are also applied:

- *Defense in depth*, i.e., never rely on one control alone;
- *Automation*, i.e., seek to automate controls as much as possible;
- *Fail-safe*, i.e., controls that break should default to a state that protects assets even though this may reduce usability or performance.

In other words, solely relying on a single control, on manual controls, or on controls that expose assets in the event of their failure is not advisable. With this in mind, the IT architect can use the following five-step method to build CAVR-compliance into new enterprise applications:

Step 1 (Define Control Objectives): This step determines which of the four CAVR properties have to be assured for which fin-BOs. In most cases, the control objective is to enforce all CAVR properties for all fin-BOs. We have, however, also encountered situations where not all four CAVR properties were required, and the control objectives could be relaxed. According to the state of the art in the related field of security engineering, control objectives are derived from the need to defend against attack threats, such as the threats given by the STRIDE model (Howard and LeBlanc, 2003). CAVR can therefore be seen as a catalogue of threats that are specific to S&P audits and largely complementary to STRIDE or other conventional threat modeling frameworks.

On our own development and integration projects, we found it important to perform Step 1 of our method during the requirements analysis phase. To do so, we successfully extended object-oriented analysis and design (Rumbaugh et al., 1999) with techniques to solicit and prioritize non-functional requirements (including the CAVR properties). One such technique is Attribute-Driven Design (ADD) (Bass et al., 2003). More recently, we added agile practices such as user story telling (Cohn, 2004) to our requirements engineering portfolio.

Step 2 (Create Architecture Overview): The architect identifies the business processes that process fin-BOs. Specifically, it is important to identify the subjects (users or activities) that interact with fin-BOs and the operations that these subjects invoke on the fin-BOs.

If object-oriented analysis and design techniques are used, the actor information found in UML use cases is an important source for the identification of subjects and operations. In UML, actors represent the external parties (i.e., human users or other systems) that interact with the system under construction; these interactions are CAVR audit relevant if fin-BOs are involved. When agile practices are applied, the personas in the user stories provide similar input. For example, a user story that describes an end user interaction such as "as a call center agent, I want to create orders that are processed and invoiced at a later stage…" reveals important information about subjects and how they manipulate fin-BOs.

In our industry projects, we customized both object-oriented analysis and user agile development methods so that audit-relevant information on subjects, fin-BOs, and their interactions is captured explicitly. Additionally, we found value in modeling *misuse cases* that capture problematic situations such as those outlined in the examples from Section 3.

Step 3 (Assess Risk): For each fin-BO, subject, and operation, the IT architect assesses if and how the fin-BOs' pertinent CAVR properties (according to Step 1) may be destroyed. In practice, such risk assessments suffer from a lack of hard data on the probabilities and impact of most threats. We found the DREAD approach to be helpful (Howard and LeBlanc, 2003), which weights the Damage potential, Reproducibility, Exploitability, Affected users, and Discoverability of threats. When assessing the damage potential it is important to not only work with the financial amounts represented by the fin-BOs, but to also consider the reputational and legal risks of CAVR violations. The purpose of Step 3 is to rank the threats to CAVR by decreasing risk, so as to prioritize the work in Step 4.

Step 4 (Select Mitigating Controls): The architect selects suitable control patterns, e.g., those introduced in Section 4.1, to mitigate the (major) identified risks. According to our overarching design principles, automated controls are preferable, and at least two controls should be implemented to mitigate each risk (defense-in-depth). Controls should be configured in such a way that they do not leave any assets exposed in the event of their failure.

This step represents a non-trivial design task for which typically no optimal solution can be found under real-world constraints such as budget limitations and other forces. For instance, non-functional requirements such as performance, usability and security conflict with each other, and project sponsors are not always willing to pay for all security measures that would be useful in theory. One would think that multiple control patterns from Section 4.2 should always be applied to ensure defense in depth; however, this is not always feasible for technical and/or economic reasons. To give an example, if an existing legacy system participates in a business process whose source code can not be modified (e.g., because the required skills and experience are not available at a reasonable cost), it can be rather expensive to add a proxy that performs INPUT AND OUTPUT VALIDATION.

Dealing with such conflicts and making appropriate tradeoffs is an important part of the expertise of application and security architects; it can be supported by meth-

ods and techniques such as ADD and the Architecture Tradeoff Analysis Method (ATAM) (Bass et al., 2003). Concepts, methods, and tools for these activities are developed in the architectural patterns community (Schumacher et al., 2006) and in the architectural knowledge management community (Zimmermann, 2009).

Step 5 (Realize Controls): Finally, the architect enforces the selected controls during the realization of the enterprise application architecture.

According to our experience, this step is best supported by design-time methods such as developer coaching and providing architectural templates (i.e., working sample solutions), code generation in model-driven development, and explicit architectural decision identification, making, and enforcement (Zimmermann, 2009).

As the following section shows, controls become easier to select and realize in architectures that follow the service-oriented architecture style.

4.3 Realizing Control Patterns in Service-Oriented Architectures

In Step 5, "Realize Controls", architects must implement the selected risk-mitigating controls in the enterprise application. Not surprisingly, the ease of implementation depends on the architecture of the enterprise application, and it is easier to add controls when applications are well-engineered. In this section, we focus on layered enterprise applications that follow the *Service-Oriented Architecture (SOA)* style, and we show how this style facilitates the implementation of controls. For space reasons, we cannot consider the "reverse" problem of auditing controls in SOAs.

An *architectural style* is a set of architectural principles, constraints, and patterns that share a common design intent and are aligned with each other to make architectures recognizable and their construction repeatable (Zimmermann, 2009). SOA is an architectural style that builds enterprise applications according to the following principles and patterns (Josuttis, 2007; Zimmermann, 2009):

- *Layering*. Applications should be organized into logical layers to separate concerns. The access to components in a given layer is restricted to components in the same and in higher layers. Layered application architectures with well-defined layer boundaries benefit the CAVR properties.
- *Service*, *service contract*, *service provider* and *service consumer*. Applications are structured into mutually invoking services. *Services* are software components that provide distinctive business functionalities via network-accessible interfaces. These interfaces are specified and exposed by *service contracts*: A *service provider* implements an interface, and a *service consumer* invokes it. Services can either be atomic or be composed from others via *service composition*. Business processes are often realized as composed services; as outlined in Section 2, the Business Process Modeling Notation (BPMN) allows specifying such composed process services.
- Integration via *Enterprise Service Bus (ESB)*. All consumer-provider interactions are brokered by an ESB that decouples service consumers from service providers to promote loose coupling principles such as protocol, for-

mat, and location transparency. Messaging middleware is often used to implement the ESB pattern.

- *Service registry*. A service registry provides a facility to discover service providers so that service consumers are able to acquire all information that is required to invoke these providers.

Extending the layered architectures described by Fowler (2003) and Zimmermann (2008), Figure 4 presents a canonical reference model for CAVR-compliant enterprise applications that follow the SOA style. The figure shows that enterprise applications serve multiple *end users* (humans or other IT systems) over one or more *channels* (vertical lines); financial events may occur and fin-BOs be created when end users interact with the enterprise application. The software and hardware components that the end user works with are known as the *frontend tier* of the enterprise application. The *middle tier* serves these client components; it is logically layered into the presentation, business logic, and data access & application integration layers. The middle tier uses the services provided by databases and other systems residing in the *backend tier*. The tiers may be physically distributed (e.g., different servers, different network zones, and/or different geographical locations).

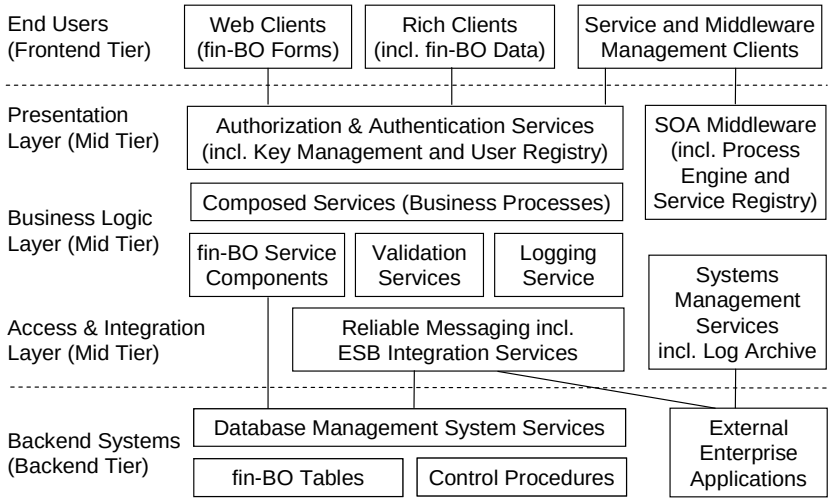

Figure 4: Building CAVR-Compliant SOA with Control Patterns

Being service providers and/or service consumers, each functional component in the application (represented as a box in Figure 4) exposes certain compliance risks, but also gives an opportunity to implement controls. The same holds true for the integration channels (represented as vertical lines). The two dotted horizontal lines represent boundaries of physical tiers; therefore, they expose network access points which have to be secured properly (e.g., access control).

The concrete architecture of an enterprise application is an *instance* of this canonical reference model, i.e., it follows the same principles of componentization, layering, and ESB integration, but applies them to a specific architecture. For instance, the information system supporting the sales process in Figure 3 in Section 3 contains such an enterprise application; in the architecture of this enterprise application, the sales process may be realized as a composed service which is supported by fin-BO service components and database tables for order and financial data. Additional service components and fin-BO tables may represent the customer master data and production data from Figure 3 in the architecture.

Logical layering and the SOA patterns facilitate achieving CAVR-compliance. This is because the control patterns from Section 4.1 can be implemented with particular ease in layered architecture that follows the SOA style. Specifically, the control patterns can be realized as follows:

STANDALONE OBJECTS. Fin-BOs should be stored as data objects in the database management system in the backend tier (Figure 4). To facilitate the implementation of other controls, it is further important that the database management system stores one and only one copy of each fin-BO (i.e., there are no duplicates). This copy is then referenced and accessed throughout the application, e.g., from composed services and service components that process fin-BOs in the mid tier.

SINGLE ACCESS POINT. The database management system becomes the single access point for the backend tier through which fin-BOs are created, updated, queried, and otherwise manipulated. It is important to suppress any access to the fin-BOs that does not "pass through" the database management system. It is further important to tightly control the number of components (i.e., business processes and fin-BO service components) that can access and manipulate fin-BOs in the database. Such tight control is important because otherwise these components may become "loopholes" that enable indirect access to fin-BOs while circumventing the access control of the database management system. Authentication and authorization services therefore constitute the SINGLE ACCESS POINT of the mid tier in our canonical reference model (Figure 4).

RELIABLE MESSAGING. Reliable messaging is represented as an architectural component in the mid tier of Figure 4. *Message channels* and *messaging endpoints* (Hohpe and Woolf, 2004) can be provided "out-of the box" by messaging middleware, which may also realize the ESB patterns. The remaining design tasks for the architect are *message construction*, *message routing and transformation*, *message consumption*, and *systems management* (Hohpe and Woolf, 2004). Detailed architecture and design patterns for these tasks have to be selected and implemented for each message endpoint and channel in the system. In a SOA, service consumers and service providers are message endpoints; service invocations translate into message exchanges.

LOGGING. The logging control pattern can be realized by a logging service component in the mid tier in Figure 4 which feeds a log archive (appearing in the systems management services box). The remainder of the application should then use this logging service for all their logging needs. The log archive applies the *message store*

pattern (Hohpe and Woolf, 2004) to create an audit trail of all services and other components that processed fin-BOs. Such an audit trail is useful for answering questions about data provenance such as "who created a fin-BO?", "where and when was it created?", "who modified it?", and so on.

CONTROLLED INTERFACES, COMPLETE MEDIATION, INPUT & OUTPUT VALIDATION, CROSS-CHECKING. These controls extend the SINGLE ACCESS POINT pattern. On the backend tier (Figure 4), these controls should be enforced by implementing a well-defined and restrictive interface to the fin-BOs, by enforcing that only authorized subjects have access to fin-BOs, and by validating (and cross-checking) inputs so that invalid or potentially malicious data can be handled safely. Using foreign key relationships and stored control procedures the database management system can help doing this (Davidson *et al.*, 2006).

It is recommended that *all* mid-tier service components are protected with the CONTROLLED INTERFACES, COMPLETE MEDIATION, and INPUT & OUTPUT VALIDATION controls. This is particularly important for service components that manipulate fin-BOs. CROSS-CHECKING is required for all service components working with multiple related fin-BOs; the relationships between these fin-BOs should be specified in a *domain model* as introduced in Section 4.1 in the context of the STANDALONE OBJECTS pattern. The SOA style facilitates such comprehensive protection because it composes software out of services (i.e., self-contained components). Each of these service components can (and should) be protected by the above four controls. In this context, it is important that each service implements its own controls and does not rely on other services to enforce the necessary controls. In particular, the lower architectural layers of Figure 4 should implement their own input validation services rather than "trusting" that higher layers check the input data. While such trust assumptions are not uncommon in practice, they bear risks and should be taken with caution.

For COMPLETE MEDIATION (and, to a lesser degree, the other control patterns) one has to decide *where* in Figure 4 access control is implemented. One option is that each service component implements its own access control. A second option is that middleware enforces all access control according to a user-defined security policy (Buecker, 2007). Within the middleware, access control can be provided either by the container (e.g., Java Enterprise Edition) running the services or by standalone authentication and authorization services acting as gateways (Ebbers et al., 2008). Today's best practice advocates the middleware option because it makes access rights explicit and centralizes them in a single place. Access control rights become easier to verify, audit, and revise this way.

LEAST PRIVILEGE and SEPARATION OF DUTY (SOD). The controls of minimizing user privileges and of not giving one subject "both keys to the bomb" are related to the COMPLETE MEDIATION control pattern. More precisely, they are enforced by configuring the COMPLETE MEDIATION control in a manner that prohibits any access to a service if such access violates the LEAST PRIVILEGE or SEPARATION OF DUTY patterns. This can be achieved by configuring the authentication and authorization services adequately. Such configurations of the COMPLETE MEDIATION controls can be derived from the end user information in the requirements specification, e.g.,

from the actors in UML use case models. If the requirements specification turns out to be incomplete or in conflict with the audit control requirements solicited in steps 1 to 4 of our design method (see Section 4.2), it might be necessary to revise the requirements specification at this point.

IDENTITY GRANULARITY. Intuitively, the identity of a user is the unique "name" under which an information system knows the user after (s)he logged in. Managing and controlling identities is a particularly difficult challenge because identities tend to proliferate as most applications introduce their own user registries and identities. The complex design decisions involved in managing this proliferation of identities go beyond the scope of this article; they have been described in the literature (Bertocci, Serack and Baker, 2008). However, we will highlight certain practices that are advisable from a controls point of view: First, we recommend adopting a *Single Sign On (SSO)* solution (Bertocci, Serack and Baker, 2008), in which a single authentication service (a.k.a. *identity provider*), maintains all user accounts. Users then "log into" the authentication service, and the authentication service subsequently vouches for the users' identities to other services. Among other benefits, the authentication service can centrally enforce authentication policies (such as password aging) and it can reduce the risk of users adopting multiple identities.

A second recommendation concerns the common practice that a user logs into a service A, which then accesses further services B and C *under its own identity* rather than the identity of the user. A preferable solution to this is to use *delegation* where the user authorizes service A to interact with B and C on its behalf (Cantor, 2005; OAuth, 2009). All activities can be tied back to the user who initiated them. As delegation is not widely implemented, we recommend *propagating* the identity of request originators and to diligently log when a service component uses its own identity to act on behalf of somebody else. These implementation options and the associated trade-offs are further discussed by Rosen *et al.* (2008) as well as Meier *et al.* (2003).

CRYPTOGRAPHIC SIGNATURES and ENCRYPTION. Cryptographic operations are typically provided in operating system libraries. As such, the SOA style offers no advantages here. However, SOA can help with the management of the secret keys that much of today's cryptography is based on. This so-called *key management* comprises the generation, recording, distribution, installation, storage, change, disposition, and control of cryptographic keys (Murray , 2007). In a SOA, the functionality of the key management system should be embedded in a service (see Figure 4).

4.4 Impact of Application Lifecycle Management on CAVR Compliance

An architecture based on strong controls is an important prerequisite for attaining CAVR compliance, but it is not sufficient. The other stages in the lifecycle of an application play an equally important role towards achieving CAVR compliance. Consistent with COBIT (IT Governance Institute, 2005) and ISO 27001 (ISO/IEC,

2005a), we distinguish four application lifecycle phases which we call *plan*, *build*, *deploy and operate*, and *monitor:*

Phase 1 (Plan): The plan step identifies the control objectives, risks, and mitigating controls as per Steps 1 to 4 of our method from Section 4.2. The plan step also defines the roles and responsibilities for the tasks that are to be performed in the subsequent lifecycle phases 2 to 4.

Phase 2 (Build): In the build step, the application architecture is designed as described in Step 5 of our method (Section 4.3). Additionally, application and security architects derive the security policies that configure the controls. Defining appropriate security polices is essential because controls like COMPLETE MITIGATION and INPUT AND OUTPUT VALIDATION are of limited value unless they are configured correctly and consistently. In the build step, the enterprise application is also implemented and tested using representative sample data. Potentially, formal correctness proofs for key algorithms may be used to achieve a desired level of quality and confidence or to pass software certification for custom code. As discussed in Section 3.2, S&P auditors may assume such software certification to exist.

Phase 3 (Deploy and operate): System administrators install the application and work with security experts to configure the application and its associated controls. Further deployment activities are to change default passwords, to deactivate all infrastructure resources that are not required (e.g., network ports) and to take precautions to protect test data and program source code from accidental disclosure (these activities are often summarized under the term application and infrastructure *hardening*). Once deployment has completed, the application becomes operational in a production environment.

Phase 4 (Monitor): System administrators continuously monitor the application to detect compliance violations, control failures, or security threats. They further respond to such problems by refining security policies or by adding further controls. It is important that all changes to enterprise applications are controlled, including impact assessment, formal approval, testing, and documentation.

4.5 Anti-Patterns and Risks of New Technologies

As the last part of our experience report, this section briefly highlights some "risky designs" that we encountered in practice. As these designs may be forced upon architects by circumstances (e.g., legacy system limitations or other environmental constraints) they are not "bad" per se. They are, however, more risky in terms of CAVR compliance, and should be used cautiously:

- If the *file transfer* pattern (Hohpe and Woolf, 2004) is chosen as the integration style (e.g., as described in Example 3 in Section 3.3), architects and auditors may lose control over who accesses and processes fin-BOs that are stored in and exchanged as files. Shared hard drives in particular pose significant audit risks; although operating systems allow system administra-

tors to restrict and log access to files, the probability that shared drives are not configured properly is, in practice, rather high.

- *Introspection* (i.e., self reflective, adaptive, dynamic programs that make use of runtime meta-information about the program code) is a popular programming paradigm in certain developer communities. This paradigm allows the creation of generic and therefore flexible programs that can work with many different types of input data; however, in such a setting it is rather difficult to identify and tag fin-BOs, both during design and audit. As a consequence, our five-step design method (cf. Section 4.2) can no longer be followed; it is also unclear how to apply the control patterns (Section 4.1) in such a setting.
- *Scheduled batch jobs* that are invoked automatically rather than by a triggering user activity run the risk of bypassing the control patterns without notice. An example is a home grown Perl script that is configured to run every day and directly accesses the fin-BO database (e.g., as a crontab job on UNIX systems). Such a script might violate the SINGLE ACCESS POINT control if the access management procedures are not configured properly.
- Similar risks arise if *homegrown ad-hoc workflows* exist (e.g., realized in spreadsheets, structured emails with textual attachments, or groupware applications). If such ad hoc workflows are part of audited business processes then special care has to be taken to implement suitable controls. Otherwise, the uncontrolled nature of ad-hoc workflows may expose fin-BOs to threats that destroy their CAVR properties. For example, an ad-hoc workflow may stipulate the use of the operating system clipboard to cut-and-paste fin-BOs between activities. Clearly, such an approach exposes fin-BO data and threatens to destroy CAVR compliance. As a consequence, the auditor's assumptions about the existence of general IT controls and correct functioning of certified software (these assumptions were discussed in Section 3.2) may no longer be valid.
- Via openly accessible *data marts* and *data warehouses*, fin-BOs can be exported as reports and then modified in personal productivity tools such as spreadsheet editors. This may violate the restricted access property, in particular if the exported reports are further distributed by email or other uncontrolled means. If the modified data is re-imported into the operational fin-BO database, the other CAVR properties may also be violated.

It is worth noting that the risks introduced by these bad designs cannot be mitigated by adding GRC systems as such systems typically protect properly engineered, possibly certified software packages such as ERP systems and not the proprietary (custom) use of the technologies listed above.

Risks of new technologies. While the SOA style facilitates the achievement of CAVR-compliance in many ways, it is also afflicted with certain problems. These problems arise when applications are built by mixing and matching services in different combinations as required by the business logic. As a consequence, any given service may be shared among multiple enterprise applications. The crux with this setup is that each enterprise application comes with its own compliance require-

ments, which in turn translate into compliance requirements for constituent services. Consequently, each service is subject to multiple compliance requirements from multiple enterprise applications, and it is not always clear how services can accommodate all of them. The situation is further complicated when services are composed dynamically at runtime (i.e., during phases 3 and 4 of the application lifecycle from Section 4.4). Such problems did not exist previously when monolithic enterprise applications "owned" all their parts and could consequently control these parts entirely.

Virtualization is another new technology that poses new challenges to auditors. Virtualization is the practice of simulating multiple computers (so-called *virtual machines*) on a single physical machine. Virtual machines and the programs they execute can be moved dynamically between physical machines with the objective to balance load, to become more tolerant to failures, to speed up deployment, and to cut cost. The drawback of such flexibility is that it complicates the control of enterprise applications. For example, fin-BOs may dynamically be relocated to different physical machines; services or entire business processes may no longer execute within fixed system boundaries. Hence, these modern paradigms pose important challenges from an S&P audit's point of view, which are hotly debated in research and development communities at present.

5 Summary and Conclusions

In this article, we disclosed our experience with a risk-based systems and process audit method. This method is based on a *walkthrough*, where the auditor identifies major classes of financial business objects and follows them through the business processes that manipulate them. In doing so, the auditor tests each financial business object for compliance with four properties – Completeness, Accuracy, Validity, and Restricted access (CAVR). We illustrated the audit method with an example and identified five major assumptions that the method is based on:

1. The IT infrastructure of an information system that supports enterprise applications and business processes is assumed to be controlled and audited separately (e.g., by auditors specializing in IT general controls).
2. All software, particularly ERP packages and custom developed enterprise applications, works as specified by their creators (e.g., a software vendor).
3. Due to the automated nature of IT systems, it is assumed that a small sample of walkthroughs is sufficient to identify all business processes and test all controls.
4. The auditor's professional judgment is considered adequate to assess risks; mathematical models are not employed.
5. It is assumed that best practices and professional judgment are sufficient to select appropriate risk-mitigating controls.

Having described the audit method and its underlying assumptions, we discussed how IT architects can build enterprise applications that are easier to audit and that

more justifiably fulfill the auditor's assumptions. This led us to the identification of a set of thirteen proven control patterns that IT architects apply when designing enterprise applications that are subject to systems and process audits. We further described a five-step architecture design method for compliance, and we showed how control patterns and design method fit into the larger lifecycle of enterprise applications. We also presented how the control patterns and design method can be applied to layered, service-oriented enterprise application architectures and highlighted several risky designs that he have encountered in practice.

While not directly applicable to off-the-shelf software packages that come with predefined architectures, our control patterns and design method are still useful during package customization, e.g., when integrating packages with other systems or when modifying the meta-model and database schemas in a package. Moreover, package architects should consider our patterns when designing additional package capabilities or outlining the architectures for new packages.

Our patterns and methods are applicable in most sectors, including pharmaceuticals, finance, and government. Eventually, our work could reduce audit time in these sectors, e.g., by reducing the need to test controls: It might be possible to thoroughly audit a reference implementation of the control patterns and to build anchor points into enterprise application architectures where these reference implementations can be "plugged in". The audit process could then be reduced to testing whether the reference implementations have been integrated correctly into the anchor points. Applying this approach consequently strengthens the simplifying assumption that one sample is sufficient. Detailed sector-specific design guidance exceeds the scope of this article.

The interdisciplinary work leading to this article was in many ways insightful and instructive. In particular, we highlight three key points:

- We realized that auditors and IT architects use different vocabularies, assumptions, and methods when discussing and analyzing systems. For example, S&P auditors and IT architects use the term "transaction" in very different ways, as explained in Section 2.1. Moreover, IT architects do not always fully understand the differences between IT General Controls audits, S&P audits (this paper's focus), and financial audits; furthermore, they cannot be expected to be aware of the entire set of unique requirements imposed by each audit type. Auditors, on the other hand, are not always familiar with Business Process Modeling Notation (BPMN), domain models, architectural patterns, Uniform Modeling Language (UML), user stories and other design artifacts that can support audits. As such, we consider it an important contribution of this article that it presents a joint business process-oriented view on enterprise applications, including a shared notion of fin-BOs, control objectives that are associated with fin-BOs, a consistent risk-based approach to design and audit, and a catalogue of proven control patterns. A common language as established by our control patterns catalog and our canonical SOA reference model increases the efficiency in joint meetings and avoids misunderstandings when reviewing each other's work (e.g., design specifications or audit reports).

- Understanding architectural methods and artifacts enables auditors to ask for more specific documentation to inform the audit process. Moreover, we found that understanding how CAVR can be engineered systematically into enterprise applications makes it easier for auditors to identify ad-hoc or improvised controls, which are more likely to be deficient in practice.
- For IT architects, it is important to appreciate that compliance is not just another non-functional requirement, but has the potential to delay software development projects significantly or even prevent the developed solutions from being deployed. As such, it is important to understand S&P audits and how software can be built that satisfies them. The work leading to this paper also reinforced the importance of secure software engineering principles (Howard and LeBlanc, 2003). What was interesting, though, was the realization that *completeness* and *validity* (the C and V in CAVR) are particularly difficult to assure using the known software security methods. This difficulty originates from the fact that completeness and validity require (electronic) fin-BOs to correctly represent real-world financial events. This is a difficult task because IT controls are confined to the *electronic world* and can only partially control what, in effect, happened in the *real world*.

Both enterprise application design and systems and process audits remain demanding tasks, even in the light of the methods and patterns presented in this article. Dealing with the intricacies of large and complex systems is never easy. This is particularly true when requirements change as they frequently do in practice. For example, it is not uncommon for business executives to request new application features that unwittingly break compliance; laws and regulations constantly change, which generally imposes new requirements on enterprise applications; IT architects and their development teams may have been unaware of audit requirements so that the corresponding functionality has to be added in hindsight. The methods, principles, and patterns we reported on in this article clearly help in dealing with these and other realities; however, they do not render them mechanical or easy.

Acknowledgements

The authors would like to thank the anonymous reviewers for their thoughtful comments on this work.

In conducting the research leading to these results, Klaus Julisch has received funding from the European Community's Seventh Framework Program (FP7/2007-2013) under grant agreement No. FP7-216917.

References

AICPA. American Institute of Certified Public Accountants. Generally Accepted Auditing Standards. SAS 95, 2001. URL: http://www.aicpa.org/download/members/div/auditstd/AU-00150.PDF

AICPA, American Institute of Certified Public Accountants. Statements on Auditing Standards (SAS) No. 104. Amendment to Statement on Auditing Standards No. 1 Codification of Auditing Standards and Procedures ("Due Professional Care in the Performance of Work"). 2006.

Baker WH, Hutton A, Hylender CD, Novak C, Porter C, Sartin B, Tippett P. 2009 Data Breach Investigations Report, Verizon Business, 2009.

Bass L, Clements P, Kazman R. Software Architecture in Practice, Second Edition. Addison Wesley, 2003.

Bayuk JL. Stepping Through the IS Audit: What to Expect, How to Prepare. Second Edition. ISACA, 2004.

Benantar M. Access Control Systems: Security, Identity Management and Trust Models. Springer; 2006.

Bertocci V, Serack G, Baker C. Understanding Windows CardSpace. Addison-Wesley, 2008.

Bitterli PR, Brun J, Bucher T, Christ B, Hamberger B, Huissoud M, Küng D, Toggwhyler A, Wyniger D. Guide to the Audit of IT Applications. ISACA; 2009.

Blakley B, Heath C. Security Design Patterns. Technical Report G031, ISBN 1931624275, Open Group, 2004.

Bradshaw W, Willis A. Learning about Risk: Choices, Connections and Competencies. Toronto: Canadian Institute for Chartered Accountants, 1998.

Bragg SM. The Ultimate Accountants' Reference: Including GAAP, IRS and SEC Regulations, Leases, and More. 3rd Edition. John Wiley & Sons; 2010.

Buecker A. Understanding SOA Security Design and Implementation. IBM Redbooks, 2007.

Cantor S. SAML 2.0 Single Sign-On with Constrained Delegation. Working Draft, Oct. 2005. URL: http://shibboleth.internet2.edu/docs/draft-cantor-saml-sso-delegation-01.pdf .

Cascarino RE. Auditor's Guide to Information Systems Auditing. John Wiley & Sons; 2007.

Cohn D. User Stories Applied. Addison Wesley, 2004.

Congress of the United States of America. Sarbanes-Oxley Act of 2002, H.R. 3763.

Cross M, Norris LJ, Piltzecker T. Security+ Study Guide. Syngress Publishing, 2002.

Davidson L, Kline K, and Windisch K. Pro SQL Server 2005 Database Design and Optimization. Apress; 2006.

Ebbers M, Barrus B, Bonazebi S, Daly P, Lee C. DataPower Architectural Design Patterns. IBM Redbook, 2008.

EC, European Commission. EudraLex - Volume 4 Good Manufacturing Practice (GMP) Guidelines. 2010. URL: http://ec.europa.eu/enterprise/sectors/pharmaceuticals/documents/eudralex/vol-4/index_en.htm .

Epstein J. Security Lessons Learned from Société Générale. IEEE Security & Privacy, 6(3), 2008.

Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison Wesley; 2003.

Ferguson N, Schneier B. Practical Cryptography. 1st Edition. Wiley; 2003.

Fowler M. Patterns of Enterprise Application Architecture. Addison Wesley; 2003.

Hansen M, Schwartz A, and Cooper A. Privacy and Identity Management. IEEE Security & Privacy; March/April 2008.

Hohpe G, Woolf B. Enterprise Integration Patterns. Addison Wesley, 2004.

Howard M and LeBlanc D. Writing Secure Code. 2nd Edition. Microsoft Press; 2003.

ISACA. COBIT and Application Controls: A Management Guide. ISBN 9781933284859; 2009.

ISACA. Certified Information Systems Auditor, CISA Certification Overview. 2010. URL: http://www.isaca.org/Template.cfm?Section=CISA_Certification&Template=/TaggedPage/TaggedPageDisplay.cfm&TPLID=16&ContentID=43558 .

ISO/IEC. Information Technology – Security Techniques – Information Security Management Systems – Requirements. ISO/IEC 27001:2005.

ISO/IEC. Information Technology – Security Techniques – Information Security Management Systems – Guidelines. ISO/IEC 27002:2005.

IT Governance Institute. Control Objectives for Information and related Technology 4.0 (COBIT). ISBN 1-933284-37-4; 2005.

Josuttis NM. SOA in Practice. O'Reilly; 2007.

Kidwell DS, Blackwell DW, Whidbee DA, and Peterson RL. Financial Institutions, Markets, and Money. Tenth Edition. John Wiley & Sons; 2008.

Killmeyer J. Information Security Architecture. Auerbach Publications; 2000.

Leymann F, Roller D. Production Workflow: Concepts and Techniques. Prentice Hall; 1999.

Meier JD, Mackman A, Dunner M, and Vasireddy S. Building Secure Microsoft ASP.NET Applications: Authentication, Authorization, and Secure Communication. Microsoft Press; 2003.

Meyer B. Object-Oriented Software Construction. 2nd Edition. Prentice Hall; 2000.

Murray WH. Principles and Applications of Cryptographic Key Management. In: Tipton HF, Krause M, editors. Information Security Management Handbook, 6th Edition. Auerbach; 2007.

NIST, National Security Institute. Trusted Computer System Evaluation Criteria. Department of Defense; 1983.

OGC, Office of Government Commerce. The Official Introduction to the ITIL Service Lifecycle; 2007.

OAuth Core Workgoup. OAuth Core 1.0 Revision A. June 2009. URL: http://oauth.net/core/1.0a .

PCAOB, Public Company Accounting Oversight Board. Auditing Standard No. 5: An Audit of Internal Control Over Financial Reporting That Is Integrated with An Audit of Financial Statements. PCAOB Release No. 2007-005A, Nov. 2007. URL: http://pcaobus.org/Standards/Auditing/Pages/Auditing_Standard_5.aspx .

Rosen M., Lublinsky B., Smith KY, and Balcer MJ. Applied SOA: Service-Oriented Architecture and Design Strategies. John Wiley & Sons; 2008.

Ross R, Katzke S, Johnson A, Swanson M, Stoneburner G, and Rogers G. Recommended Security Controls for Federal Information Systems. National Institute of Standards and Technology. Special Publication 800-53, Rev. 2. Dec. 2007.

Rumbaugh, J., Jacobson, I., Booch, G., The Unified Modeling Language Reference Manual. Addison-Wesley, 1999.

Saltzer JH, Schroeder MD. The Protection of Information in Computer Systems. pp. 1278-1308. Proceedings of the IEEE, 63(9), 1975.

Schaeffer, MS. Accounts Payable and Sarbanes-Oxley: Strengthening Your Internal Controls. John Wiley & Sons, 2006.

Schumacher M, Fernandez EB, Hybertson D, Buschmann F, Sommerlad P. Security Patterns: Integrating Security and Systems Engineering. Wiley; 2006.

SEC, Securities and Exchange Commission. Commission Guidance Regarding Management's Report on Internal Control Over Financial Reporting Under Section 13(a) or 15(d) of the Securities Exchange Act of 1934. RELEASE NOS. 33-8810; 34-55929; FR-77; File No. S7-24-06. 2007.

Senft S, Gallegos F. Information Technology Control and Audit. Auerbach Publications; 2009.

Symantec. *Web Based Attacks*. Symantec Corp., Technical Report. Feb 2009.

Trent H. Products for Managing Governance, Risk, and Compliance: Market Fluff or Relevant Stuff? Burton Group. Mar.18, 2008.

Tulach J. Practical API Design: Confessions of a Java Framework Architect. Apress; 2008.

Yoder J, Barcalow J. Architectural Patterns for Enabling Application Security. In Proceedings of the Fourth Conference on Pattern Languages and Programs; 1997.

Zimmermann O. An Architectural Decision Modeling Framework for Service-Oriented Architecture Design. PhD Thesis. University of Stuttgart, 2009.