HAW HAMBURG

BACHELOR THESIS
Hani Alshikh

Evaluation and use of Event-Sourcing for audit logging

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Hani Alshikh

Evaluation and use of Event-Sourcing for audit logging

Bachelor thesis submitted for examination in Bachelor´s degree
in the study course *Bachelor of Science Angewandte Informatik*
at the Department Computer Science
at the Faculty of Engineering and Computer Science
at University of Applied Science Hamburg

Supervisor: Prof. Dr. Stefan Sarstedt
Supervisor: Prof. Dr. Olaf Zukunft

Submitted on: 20. März 2023

Hani Alshikh

Title of Thesis

Evaluation and use of Event-Sourcing for audit logging

Keywords

Event Sourcing, Auditing, Audit Trail, Software Engineering, Software Architecture

Abstract TODO

Kurzfassung Falls Sie über den Titel die Aufmerksamkeit des Lesers geweckt haben, wird dieser in der Regel die Kurzfassung und das Inhaltsverzeichnis überfliegen. Im Inhaltsverzeichnis erwartet er Information über die Struktur der Arbeit, in der Kurzfassung über deren Inhalt. Die Kurzfassung sollte deshalb eine konzentrierte Verdichtung des fachlichen Inhaltes der Arbeit enthalten. Die Kurzfassung zu schreiben ist nicht ganz einfach. Sie wird immer als Allerletztes verfasst. Die Kunst besteht darin, die wesentlichen Kernaussagen der Arbeit in einer viertel bis halben Seite zusammenzufassen.

...

Contents

List of Figures

List of Tables

Abbreviations

ES Event Sourcing.

Symbols

Ω unit of electrical resistance.

1 Acknowledgments

2 Introduction

Fangen Sie nicht zu früh mit der Ausformulierung des Textes an! Benutzen Sie die Gliederungsfunktion handelsüblicher Textsysteme, um Ihre Gliederungspunkte ent- sprechend ihrer Bedeutung verschieben zu können. Füllen Sie die Textkomponenten zu Ihren Gliederungspunkten nach und nach erst mit Stichworten und Bildverweisen, später mit formulierten Sät- zen. Je besser dies gelingt, desto einfacher wird später die Ausformulierung des Textes. Damit haben Sie ein flexibles Ablagesystem für Ideen und Formulierungen.

Aufgabenstellung (Problembeschreibung inkl. Motivation, Zielsetzung, Abgrenzung),

Die Einleitung ist das erste fachliche Kapitel einer Arbeit. Hier machen Sie den Leser mit der Problem- stellung und dem Umfeld bekannt, stellen das benötigte Basis- und Umgebungswissen zur Verfügung, nehmen Bezug auf andere Arbeiten und führen so in die Problemstellung ein. Hier findet sich u.a. gewiss wieder, was Sie zu Beginn Ihrer Tätigkeit im Pflichtenheft für die Bachelorarbeit geschrieben haben, nach entsprechender Überarbeitung und Anpassung, versteht sich. Bedenken Sie, dass der normale Leser zwar allgemeines Informatik-Fachwissen besitzt, aber nicht das konkrete Umfeld ihrer Firma/Arbeit kennt bzw. die dort vorhandene Begriffswelt. Hiermit müssen Sie ihn vertraut machen sowie Ihre spezielle Zielsetzung verdeutlichen und abgrenzen. Die Einleitung sollte in mehrere Abschnitte unterteilt sein, z. B. 1.1 Problemstellung und Motivation 1.2 Zielsetzung 1.3 Abgrenzung des Themas 1.4 Überblick über den Aufbau der Arbeit (Kapitelübersicht)

Am Ende der Einleitung sollte der Leser mit dem Umfeld, der Problemstellung und der Zielsetzung in Umrissen vertraut sein.

Event sourcing is a software architecture pattern that involves storing a log of changes made to an application's data as a series of events. Instead of storing the current state of the data in a traditional database, event sourcing stores the history of changes made to the data over time. This allows developers to rebuild the current state of the data at any point in time by replaying the events in the log.

One of the primary benefits of event sourcing is that it provides a complete audit trail of all changes made to the data. Because the log of events is comprehensive and immutable, it is possible to trace the history of any given piece of data and see exactly how it has changed over time. This can be extremely useful for auditing purposes, as it allows organizations to track and monitor changes to sensitive data and ensure that they are in compliance with regulations and standards.

In addition to providing a detailed audit trail, event sourcing also offers a number of other benefits. Because the events in the log are stored in a chronological order, it is possible to use event sourcing to implement time-based queries and to reconstruct the state of the system at any point in the past. This can be useful for debugging and for performing rollbacks or reversals of changes.

Event sourcing is particularly well-suited for applications that need to handle large volumes of data and that require a high degree of data integrity. It is also a good fit for applications that need to support complex business processes and that require the ability to track and audit changes to the data over time.

Overall, event sourcing is an ideal pattern for auditing because it provides a complete and immutable record of all changes made to the data, making it possible to trace the history of any piece of data and ensure compliance with regulations and standards.

3 Audit

Wenn mehrere Teil-Kapitel zu strukturieren sind: Schreiben Sie zu jedem Teil-Kapitel eine Ein- leitung ("Hier wird die folgende Fragestellung untersucht...") und eine Ausleitung ("Hiermit ist erreicht: ... Die folgenden Probleme sind aber noch offen:...").

An audit is often defined as an independent examination, inspection, or review. While the term applies to evaluations of many different subjects, the most frequent usage is with respect to examining an organization's financial statements or accounts. In contrast to conventional dictionary definitions and sources focused on the accounting connotation of audit, definitions used by broad-scope audit standards bodies and in IT auditing contexts neither constrain nor presume the subject to which an audit applies.. [Gantz, 2014].

the International Organization for Standardization (ISO) guidelines on auditing use the term audit to mean a "systematic, independent and documented process for obtaining objective evidence and evaluating it objectively to determine the extent to which the audit criteria are fulfilled" [ISO 19011, 2018]

the Information Technology Infrastructure Library (ITIL) glossary defines audit as "formal inspection and verification to check whether a standard or set of guidelines is being followed, that records are accurate, or that efficiency and effectiveness targets are being met" [Hanna und Rance, 2011]

Such general interpretations are well suited to IT auditing, which comprises a wide range of standards, requirements, and other audit criteria corresponding to processes, systems, technologies, or entire organizations subject to IT audits.

IT auditing is a specialized discipline not only in its own right, with corresponding standards, methodologies, and professional certifications and experience requirements, but it also intersects significantly with other IT management and operational practices. The subject matter overlap between IT auditing and network monitoring, systems administration, service management, technical support, and information security makes familiarity

with IT audit policies, practices, and standards essential for IT personnel and managers of IT operations and the business areas that IT supports.

IT audit is the process of collecting and evaluating evidence of an organization's information technology (IT) systems, practices, and operations to determine whether they are adequate, efficient, and effective in meeting the organization's objectives. An IT audit typically includes a review of an organization's policies, procedures, and controls related to its IT systems, as well as an assessment of the security, reliability, and performance of its IT infrastructure. The goal of an IT audit is to identify any weaknesses or deficiencies in an organization's IT systems and recommend improvements that can help the organization achieve its goals.

An audit is a systematic, objective examination of one or more aspects of an organization that compares what the organization does to a defined set of criteria or requirements. Information technology (IT) auditing examines processes, IT assets, and controls at multiple levels within an organization to determine the extent to which the organization adheres to applicable standards or requirements.

IT auditing helps organizations understand, assess, and improve their use of controls to safeguard IT, measure and correct performance, and achieve objectives and intended outcomes. IT auditing consists of the use of formal audit methodologies to examine IT-specific processes, capabilities, and assets and their role in enabling an organization's business processes. IT auditing also addresses IT components or capabilities that support other domains subject to auditing, such as financial management and accounting, operational performance, quality assurance, and governance, risk management, and compliance (GRC).

IT audits are performed both by internal auditors working for the organization subject to audit and external auditors hired by the organization. The processes and procedures followed in internal and external auditing are often quite similar, but the roles of the audited organization and its personnel are markedly different. The audit criteria—the standards or requirements against which an organization is compared during an audit—also vary between internal and external audits and for audits of different types or conducted for different purposes.

IT auditing often occurs as a component of a wider-scope audit not limited to IT concerns alone, or a means to support other organizational processes or functions such as GRC, certification, and quality assurance. Audits performed in the context of these broader

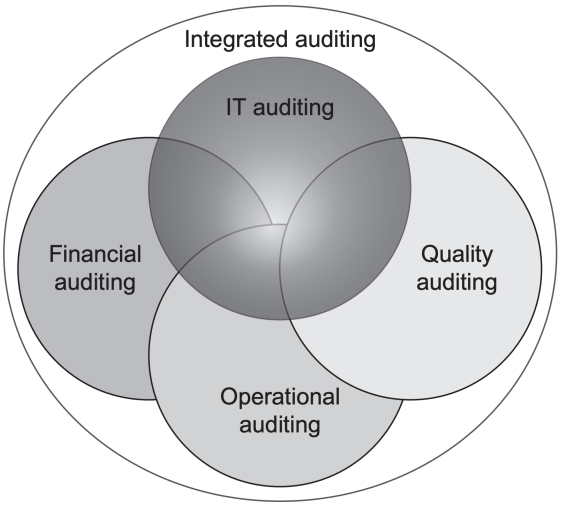

Figure 3.1: test

programs have different purposes and areas of focus than stand-alone IT-centric audits, and offer different benefits and expected outcomes to organizations.

Auditing IT differs in significant ways from auditing financial records, general operations, or business processes. Each of these auditing disciplines, however, shares a common foundation of auditing principles, standards of practice, and highlevel processes and activities.

IT auditing has much in common with other types of audit and overlaps in many respects with financial, operational, and quality audit practices.

Performing and supporting IT audits and managing an IT audit program are time-, effort-, and personnel-intensive activities, so in an age of cost-consciousness and competition for resources, it is reasonable to ask why organizations undertake IT auditing. The rationale for external audits is often clearer and easier to understand— publicly traded companies and organizations in many industries are subject to legal and regulatory requirements, compliance with which is often determined through an audit. Similarly, organizations seeking or having achieved various certifications for process or service quality, maturity, or control implementation and effectiveness typically must undergo certification audits by independent auditors. IT audits often provide information that helps organizations manage risk, confirm efficient allocation of IT-related resources, and achieve other IT and business objectives. Reasons used to justify internal IT audits may be more varied across organizations, but include:

- -

Further details on organizational motivation for conducting internal and external IT audits appear in Chapters 3 and 4, respectively. To generalize, internal IT auditing is often driven by organizational requirements for IT governance, risk management, or quality assurance, any of which may be used to determine what needs to be audited and how to prioritize IT audit activities. External IT auditing is more often driven by a need or desire to demonstrate compliance with externally imposed standards, regulations, or requirements applicable to the type of organization, industry, or operating environment.

3.1 IT Auditing

It is important to use "IT" to qualify IT audit and distinguish it from the more common financial connotation of the word audit used alone. Official definitions emphasizing the financial context appear in many standards and even in the text of the Sarbanes–Oxley Act, which defines audit to mean "the examination of financial statements of any issuer" of securities (i.e., a publicly traded company). The Act also uses both the terms evaluation and assessment when referring to required audits of companies' internal control structure and procedures.

chapter 3 on page 4 Words like assessment, evaluation, and review are often used synonymously with the term audit and while it is certainly true that an audit is a type of evaluation, some specific characteristics of auditing distinguish it from concepts implied by the use of more general terms. An audit always has a baseline or standard of reference against which the subject of the audit is compared. An audit is not intended to check on the use of best practices or (with the possible exception of operational audits) to see if opportunities exist to improve or optimize processes or operational characteristics. Instead, there is a set standard providing a basis for comparison established prior to initiating the audit. Auditors compare the subjects of the audit—processes, systems, components, software, or organizations overall—explicitly to that predefined standard to determine if the subject satisfies the criteria. Audit determinations tend to be more binary than results of other types of assessments or evaluations, in the sense that a given item either meets or fails to meet applicable requirements—auditors often articulate audit findings in terms of controls' conformity or nonconformity to criteria.

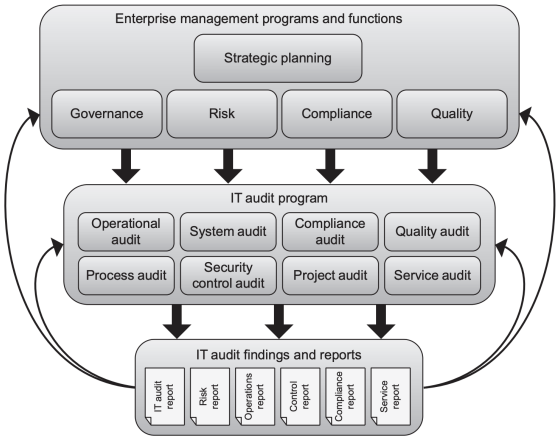

Figure 3.2: test

In contrast, a typical assessment might have a quantitative (i.e., score) or qualitative scale of ratings (e.g., poor, fair, good, excellent) and produce findings and recommendations for improvement in areas observed to be operating effectively or those considered deficient. Because auditors work from an established standard or set of criteria, IT audits using comprehensive or well thought-out requirements may be less subjective and more reliable than other types of evaluations or assessments [Gantz, 2014].

HAW Hamburg tests Tests Ω Event Sourcing (ES)

3.2 Auditing in Context

IT audit activities represent an integral part of several key enterprise management functions, which collectively contribute to the scope of the IT audit program and receive input from the output of audit processes.

- IT governance - Risk management - Legal and regulatory compliance - Quality management and quality assurance - Information security management

measured in terms of efficiency, effectiveness, and related performance metrics.

|||compare architectures in the different contexts

3.3 Internal controls

From the perspective of planning and performing IT audits, internal controls represent the substance of auditing activities, as the controls are the items that are examined, tested, analyzed, or otherwise evaluated. Organizations often implement large numbers of internal controls intended to achieve a wide variety of control objectives.

The prevalent control categorization schemes used in internal control frameworks, IT audit, and assessment guidance, and applicable legislation classify controls by purpose, by functional type, or both.

Table 3.1: Examples of Internal Controls Categorized by Type and Purpose

	Preventive	**Detective**	**Corrective**
Administrative	Acceptable use policy; Security awareness training	Audit log review procedures; IT audit program	Disaster recovery plan; Plan of action and milestones
Technical	Application firewall; Logical access control	Network monitoring; Vulnerability scanning	Incident response center; Data and system backup
Physical	Locked doors and server cabinets; Biometric access control	Video surveillance; Burglar alarm	Alternate processing facility; Sprinkler system

Just as financial, quality, and operational audits can be executed entity-wide or at different levels within an organization, IT audits can evaluate entire organizations, individual business units, mission functions and business processes, services, systems, infrastructure, or technology components. As described in detail in Chapter 5, different types of IT audits and the approaches used to conduct them may consider internal controls from multiple perspectives by focusing on the IT elements

In this work we will focus on the administrative controls for detective purposes.

administrative controls specify what an organization intends to do to safeguard the integrity of its operations, information, and other assets.

Taking a more related example where users are saved in a database and one might get the impression that user X also created

bei der Ausgestaltung der IT-Systeme und der dazugehörigen IT-Prozesse grundsätzlich auf gängige Standards abzustellen. Zu diesen zählen bspw. der IT-Grundschutz des Bundesamts für Sicherheit in der Informationstechnik, die internationalen Sicherheitsstandards ISO/IEC 270XX der International Organization for Standardization und der Payment Card Industry Data Security Standard (PCI-DSS). [BaFin, 2021].

Anforderungsdokumente können sich je nach Vorgehensmodell unterscheiden und beinhalten bspw.: - Fachkonzept (Lastenheft) - Technisches Fachkonzept (Pflichtenheft) - User-Story/Product Back-Log Nichtfunktionale Anforderungen an IT-Systeme sind bspw.: - Anforderungen an die Informationssicherheit - Zugriffsregelungen - Ergonomie - Wartbarkeit - Antwortzeiten - Resilienz.

Im Rahmen der Anwendungsentwicklung sind je nach Schutzbedarf angemessene Vorkehrungen zu treffen, dass auch nach jeder Produktivsetzung einer Anwendung die Vertraulichkeit, Integrität, Verfügbarkeit und Authentizität der zu verarbeitenden Daten nachvollziehbar sichergestellt werden [BaFin, 2021].

Geeignete Vorkehrungen sind z. B.: - Prüfung der Eingabedaten - Systemzugangskontrolle Benutzerauthentifizierung - Transaktionsautorisierung - Protokollierung der Systemaktivität - Prüfpfade (Audit Logs) - Verfolgung von sicherheitsrelevanten Ereignissen - Behandlung von Ausnahmen.

Das Institut hat nach Maßgabe des Schutzbedarfs und der SollAnforderungen Prozesse zur Protokollierung und Überwachung einzurichten, die überprüfbar machen, dass die Berechtigungen nur wie vorgesehen eingesetzt werden. Aufgrund der damit verbundenen weitreichenden Eingriffsmöglichkeiten hat das Institut insbesondere für die Aktivitäten mit privilegierten (besonders kritischen) Benutzer-/ Zutrittsrechten angemessene Prozesse zur Protokollierung und Überwachung einzurichten Die übergeordnete Verantwortung für die Prozesse zur Protokollierung und Überwachung von Berechtigungen wird einer Stelle zugeordnet, die unabhängig vom berechtigten Benutzer oder dessen Organisationseinheit ist. Zu privilegierten Zutrittsrechten zählen in der Regel die Rechte zum Zutritt zu Rechenzentren, Technikräumen sowie sonstigen sensiblen Bereichen [BaFin, 2021].

Durch begleitende technisch-organisatorische Maßnahmen ist einer Umgehung der Vorgaben der Berechtigungskonzepte vorzubeugen. Technisch-organisatorische Maßnahmen sind bspw.: die Auswahl angemessener Authentifizierungsverfahren (u. a. starke Authentifizierung im Falle von Fernzugriffen) - die Implementierung einer Richtlinie zur Wahl

sicherer Passwörter - die automatische passwortgesicherte Bildschirmsperre - die Verschlüsselung von Daten - die manipulationssichere Implementierung der Protokollierung - die Maßnahmen zur Sensibilisierung der Mitarbeiter

Planning and executing the processes associated with these programs influences the design and implementation of the IT audit program and helps organizations identify and prioritize various aspects of their operations that constitute the subject of needed IT audits. Conversely, weaknesses or deficiencies in internal controls, gaps in meeting compliance requirements, or other potential IT audit findings influence organizational decisions made at the enterprise program level about allocation of resources, risk response, corrective action, or opportunities for process or control improvement.

The effectiveness and efficiency with which organizations implement and execute IT processes is often expressed in terms of process maturity, a relative measure of how well processes are fully defined, documented, implemented, and optimized for use in an organization.

3.4 Audit log

Audit logging—Log user actions.

The purpose of audit logging is to record each user's actions. An audit log is typically used to help customer support, ensure compliance, and detect suspicious behavior. Each audit log entry records the identity of the user, the action they performed, and the business object(s). An application usually stores the audit log in a database table. [Richardson, 2018]

An audit log can take many physical forms. The most common form is a file. However a database table also makes a fine audit log. If you use a file you need a format. An ASCII form helps in making it readable to humans without special software. If it's a simple tabular structure, then tab delimited text is simple and effective. More complex structures can be handled nicely by XML.

Audit Log is easy to write but harder to read, especially as it grows large. Occasional ad hoc reads can be done by eye and simple text processing tools. More complicated or repetitive tasks can be automated with scripts. Many scripting languages are well suited

to churning though text files. If you use a database table you can save SQL scripts to get at the information.

When you use Audit Log you should always consider writing out both the actual and record dates. They are easy to produce and even though they may be the same 99% of the time, the 1% can save your bacon. As you do this remember that the record date is always the current processing date.

When to Use It

The glory of Audit Log is its simplicity. As you compare Audit Log to other patterns such as Temporal Property and Temporal Object you quickly realize that these alternatives add a lot of complexity to an object model, although these are both often better at hiding that complexity than using Effectivity everywhere.

But it's the difficulty of processing Audit Log that is it's limitation. If you are producing bills every week based on combinations of historic data, then all the code to churn through the logs will be slow and difficult to maintain. So it all depends how tightly the accessing of temporal information is integrated into your regular software process. The tighter the integration, the less useful is Audit Log.

Remember that you can use Audit Log in some parts of the model and other patterns elsewhere. You can also use Audit Log for one dimension of time and a different pattern for another dimension. So you might handle actual time history of a property with Temporal Property and use Audit Log to handle the record history.

3.5 Auditing 2.0

Auditing 2.0, also known as continuous auditing, is a modern approach to auditing that uses technology and data analytics to continuously monitor and assess an organization's financial and operational processes. Unlike traditional auditing, which is typically conducted on a periodic basis, continuous auditing is a continuous process that uses real-time data to identify and address potential risks and issues as they arise.

One of the key features of Auditing 2.0 is the use of data analytics to analyze and interpret large volumes of data. By using advanced algorithms and machine learning techniques, auditors can identify patterns and trends in the data that may indicate potential risks

or issues. This allows auditors to proactively identify and address problems before they escalate, rather than waiting for them to be discovered during a periodic audit.

Auditing 2.0 also makes use of technology such as artificial intelligence, blockchain, and cloud computing to automate and streamline the audit process. For example, AI-powered systems can be used to analyze and interpret data in real-time, while blockchain technology can be used to provide a secure and transparent record of transactions. Cloud computing can also be used to store and manage large volumes of data, making it easier for auditors to access and analyze the data they need.

Overall, Auditing 2.0 represents a significant shift in the way that audits are conducted, moving from a reactive approach to a proactive one. By using advanced technologies and data analytics, Auditing 2.0 allows organizations to continuously monitor and assess their financial and operational processes, identify potential risks and issues, and take timely and appropriate action to address them. This can help organizations to improve their risk management practices, enhance their internal controls, and ultimately reduce the risk of financial and operational errors and failures.

Auditors can use process mining techniques to evaluate all events in a business process, and do so while it is still running. van der Aalst u. a. [2010]

Auditors validate information about organizations and their business processes. Reliable information is needed to determine whether these processes are executed within certain boundaries set by managers, governments, and other stakeholders. Violations of specific rules enforced by law or company policies may indicate fraud, malpractice, risks, or inefficiencies.

Traditionally, an audit can only provide reasonable assurance that business processes are executed within the given set of boundaries. Auditors assess the operating effectiveness of process controls, and when these controls are not in place or functioning as expected, they typically check samples of factual data.

However, with detailed information about processes increasingly available in high-quality event logs, auditors no longer have to rely on a small set of samples offline. Instead, using process mining techniques, they can evaluate all events in a business process, and do so while it is still running.

The omnipresence of electronically recorded business events coupled with process mining techno logy enable a new form of auditing that will dramatically change the role of auditors: Auditing 2.0.

3.5.1 Process Mining

The goal of process mining is to discover, monitor, and improve real (not assumed) processes by extracting knowledge from event logs. Over the past decade, process mining techniques have matured and are being integrated into commercial software products (W.M.P. van der Aalst et al., "Business Process Mining: An Industrial Application," Information Systems, vol. 32, no. 5, 2007, pp. 713–732).

Business Provenance Process mining starts with the event log: a sequentially recorded collection of events, each of which refers to an activity (well-defined step) and is related to a particular case (process instance). Some mining techniques use other information such as the person or resource executing or initiating the activity, the event's time stamp, or data elements recorded with the event—for example, the size of an order.

The systematic, reliable, and trustworthy recording of events, known as business provenance, is essential to auditing. This term acknowledges the importance of traceability by ensuring that history cannot be rewritten or obscured.

3.5.2 Challenges

The application of process mining to auditing depends on the availability of relevant data, which is primarily stored in enterprise resource planning (ERP) systems. Mining such systems is challenging because they are not, despite having built-in workflow engines, process-oriented. Because data related to a particular process it usually scattered over dozens of tables, extracting it for auditing is nontrivial.

Test todo test

A second challenge of Auditing 2.0 concerns the so-called "auditing materiality" principle that guides current auditing practices. According to this principle, auditors typically consider only a small subset of data, and if they see no deviations take no further actions. By looking at all the data, auditors will inevitably find more exceptions requiring follow-up, increasing quality but also the audit's time and cost.

Finally, widespread adoption of process mining as an accepted auditing approach would require organizations such as the International Federation of Accountants to change their methodologies and issue new guidelines to companies that rely on them.

Major corporate and accounting scandals including those affecting Enron, Tyco, Adelphia, Peregrine, and WorldCom have fueled interest in more rigorous auditing practices. Legislation such as the Sarbanes-Oxley Act of 2002 and the Basel II Accord of 2004 were enacted in response to such scandals. The recent financial crisis also underscores the importance of verifying that organizations operate "within their boundaries." Process mining techniques offer a means to more rigorously check compliance and ascertain the validity and reliability of information about an organization's core processes.

Auditing 2.0—a more rigorous form of auditing that couples detailed event logs with process mining techniques—will dramatically change the auditing profession. Auditors will need better analytical and IT skills, and their role will shift as they work on the fly. Provenance data will make it possible to "replay" history reliably and accurately and to predict problems, thereby improving business processes.

4 Event Sourcing

Wenn mehrere Teil-Kapitel zu strukturieren sind: Schreiben Sie zu jedem Teil-Kapitel eine Ein- leitung ("Hier wird die folgende Fragestellung untersucht...") und eine Ausleitung ("Hiermit ist erreicht: ... Die folgenden Probleme sind aber noch offen:...").

Event Sourcing is a pattern for storing data as events in an append-only log. This simple definition misses the fact that by storing the events, you also keep the context of the events; you know an invoice was sent and for what reason from the same piece of information. In other storage patterns, the business operation context is usually lost, or sometimes stored elsewhere.

Event driven, CQRS and Evnet Sourcing are complementary patterns so it makes sense to consider them both as a set and individually. This article is the latter - a focus on event sourcing.

Martin Fowler believes that the key to Event Sourcing is that it guarantees that all changes to the domain objects are initiated by the event objects. This leads to a number of facilities that can be built on top of the event log:

Complete Rebuild: we can discard the application state completely and rebuild it by re-running the events from the event log on an empty application. Temporal Query: we can determine the application state at any point in time. Notionally we do this by starting with a blank state and rerunning the events up to a particular time or event. Event Replay: if we find a past event was incorrect, we can compute the consequences by reversing it and later events and then replaying the new event and later events.

One obvious form of return is that it's easy to serialize the events to make an Audit Log.Such an audit trail is useful for audit, no shocks there, but also has other usages. I chatted with someone who got their online accounts into an awkward state and phoned in for help. He was impressed that the helper was able to tell him exactly what he did and thus was able to figure out how to fix it. To provide such a capability means exposing

the audit trail to the support group so they can walk through a user's interaction. While Event Sourcing is a good way of doing this, you could also do this with more regular logging mechanisms, and that way not have to deal with the odd interface.

however, logging is mostly associated with debugging and has no direct relation to the system state. Logging style, verbosity....

Event Sourcing is a design approach where distributed applications keep their state as a sequence of state-changing operations. Instead of storing mutable objects, applications keep an immutable sequence of changes to such objects. Changing state and writing an event to the log is one single, therefore atomic, operation. The log becomes the authoritative source of truth, offering eventual consistency, as events propagate to different parts of the distributed application. This design entails a strongly decoupled architecture, typical of publish–subscribe systems (Clayman et al., 2010), while providing reliable auditing and logging functionality. For example, developers may bring the application back to a previous execution state, by replaying the events in the log. This feature is particularly powerful, not only for the sake of failure recovery and state management (most services can be stateless), but also for debugging and to experiment alternative what-if scenarios. Furthermore, it makes the system's data schema more flexible, as it becomes possible to recover field values or even calculate additional ones from the log.

The foundational idea of event sourcing is the domain event as described by Evans (2015). His seminal book on Domain-Driven Design (DDD), however, does not mention the pattern. Vernon (2013) describes event sourcing only briefly in his book on the implementation of various DDD patterns. Young (2017), as one of the original proposers of event sourcing, discusses the challenge of versioning ESSs. Event sourcing is also discussed in the context of CQRS (Young, 2010), a pattern strongly related to event sourcing. Recent academic literature (Erb, 2019, Zhong et al., 2019) shows an interest in applying event sourcing for research projects.

Recently, the event sourcing pattern has become a popular answer to the challenges of complex, mission-critical, scalable systems. Examples of organizations that apply event sourcing are Netflix (Avery and Reta, 2017), and Walmart's Jet.com (Gorodinski, 2017), with the goal of creating scalable and reliable critical systems. Event sourcing is informally described by Fowler (2005) as a pattern that "ensures that all changes to application state are stored as a sequence of events". Flexibility, debug-ability, and reliability are given by Avery and Reta (2017) as rationale for using event sourcing. Debski et al. (2017) and Erb and Hauck (2016) show how event sourcing can be applied

to achieve scalable, reactive systems. Kabbedijk et al. (2012) describes event sourcing as a sub pattern of Command Query Responsibility Segregation (CQRS) in his work on the improved variability and scalability of systems applying CQRS.

The events in event sourcing, as opposed to general event-driven architectures (EDAs) (Fowler, 2017), are stored as an append-only log of all state changes. Two key characteristics separate event sourcing from event-driven approaches, such as stream processing, transactional processing, and blockchain. First, events in Event Sourced Systems (ESSs) are stored as the state of the application. Other approaches use the events to communicate, while the communication aspect comes second in ESSs. The second difference is that events are closely related to events occurring in real world business processes. This allows event sourcing to be also used as a design approach. Domain-Driven Design (DDD), as described by Evans (2003), advocates events as a design tool for the process flow of a software system. Brandolini (2018) proposes event storming (analogous to brainstorming), a group design process that focuses on the events that take place in a software system. Further details on these analogous approaches are found in Section 3.

Our study regards a new research area, therefore, we apply Grounded Theory (GT). Adolph et al. (2011) describe GT as a useful approach for research in areas that have not previously been studied. A GT explains how people resolve their main concern by employing a certain process. That process is called the 'core category' of the GT. The core category of the work presented in this article is the process of designing and implementing event sourced systems, as performed by software engineers. The theoretical definition of event sourcing helps both researchers and practitioners to understand, reason about, and teach the pattern and its consequences. Section 2 explains how we applied GT to form a basis for conceptualization of ESSs from 25 interviews, and how the three essential elements are covered. From the gathered data we distill the pattern description and its consequences. This work has the following contributions:

the studie contudcted by ..... on 25 engineers of different backgrounds and roles by applying appling Grounded Theory (GT). Adolph et al. (2011) which showcase how event sourcing is ganging on popularity especially when it comes to satisfying auditory needs. As put by one of the engineers when asked about event sourcing "" The reasons for applying event sourcing can be grouped into four categories. Remarkably, all systems under study benefit from event sourcing, and no system returned to a current state model. Still, most engineers state that they would not apply event sourcing in every system. The reason given for this opinion is the added complexity of introducing event

sourcing. Engineer E2 would apply event sourcing by default, because of the benefits it gives. Overeem u. a. [2021]

Together with this description we identify four categories of rationale for the application of event sourcing, such as a decrease of complexity. In this "In Practice" submission, we also identify five engineering challenges around the pattern, with schema evolution being one of the most complex challenges. With the pattern description and its liabilities presented in this article, we enable engineers to make a considered choice.

Event sourcing is an approach to store data in an application and was originally established by Fowler (2005a). This differs from the traditional approach of storing data which, is to store the current state of an application in a database. Fowler (2002) describes this approach as active records. With the traditional approach, the data is accessed or modified by four operations, Create, Read, Update and Delete, these operations are generally referred to as CRUD operations (Betts et al. 2013). The idea behind event sourcing is to treat each change to the state as an event where each event is part of a sequence of events in the order they occurred. The sequence of events can be used to recreate the application state at any point in time (Fowler 2005a). Figure 1 illustrates how the state of two shapes can be represented with active records and with event sourcing. In Figure 1a the current X and Y values, together with the type of shape, are stored in a specific row associated with the shape identification number. In Figure 1b the same shapes are stored as a series of events. Each row consists of the type of event and the parameters for the event, together with an aggregate number which describes to which shape the event is associated. The series of events can be used to rebuild the current state which would result in the same state as in Figure 1a. One difference between the two approaches is that in Figure 1b it is possible to see that shape number 1 has moved from the original position which is information is lost when using active records.

4.1 Terminology

CQRS and event driven

4.2 Command Query Responsibility Segregation

One of the architectural patterns that in recent years emerged in the development of cloud systems is Command Query Responsibility Segregation (CQRS). The pattern was introduced by Young [5] and Dahan [6], and the goal of the pattern is to handle actions that change data (those are called commands) in different parts in the system than requests that ask for data (called queries). By separating the command-side (the part that validates and accepts changes) from the query-side (the part that answers queries), the system can optimize the two parts for their very different tasks.

Young [7] describes CQRS as a stepping stone for event sourcing. Event sourcing is a data storage model that does not store the current (or last) state, but all changes leading up to the current state. Fowler [8] explains event sourcing by comparing it to an audit trail: every data change is stored without removing or changing earlier events. The events stored in an event store are stored as schema-less data, because the different events often do not share properties. A store with an explicit schema would make it more difficult to append events in the store to a single stream. Data in schema-less stores is not without schema, but the schema is implicit: the application assumes a certain schema. This makes the problem of schema evolution and data conversion more difficult as observed by Scherzinger et al. [9]. Schema-less data is more difficult to evolve as the store is unaware of structure and thus cannot offer tools to transform the data into a new structure. Relational data stores that have explicit knowledge of the structure of the data can use the standardized data definition language (DDL) to upgrade the schema and convert the data. Another problem in the evolution of event sourced systems is the amount of data that is stored, not only the current state, but also every change leading up to that state. This huge amount of data makes the problem of performing a seamless upgrade even more important: upgrades may need more time, but they are required to be imperceptible.

The foundations of CQRS were laid by Meyer [11] in the Command-Query Separation (CQS) principle. He defined a command as “serving to modify objects” and a query is “to return information about objects”, or informally worded “asking a question should not change the answer”. Figure 1 shows the CQRS pattern: commands are accepted by the command-side and produce events which are processed by the query-side. The query-side projects these events into a form that is suitable for querying and presenting. The command-side and the query-side both have their own data store: the first store is

used to maintain data that is used in validating requested changes, and the second store is used to retrieve data for displaying or reporting.

figure to showcase CQRS

The command-side communicates with the query-side through asynchronously sending events. These events are used by the query-side to build a view of the state that can be used to query and present data. By doing this asynchronously the query-side does not influence the performance of the command-side. However, this does lead to eventual consistency. This is a weaker form of consistency that Vogels [12] defines as "when no updates are made to the object, the object will eventually have the last updated value". The system guarantees that the query-side eventually will reflect the events produced in the command-side. However, there are no guarantees on how fast this will be done. A system with a large delay is unfeasible, because in that case queries will often return data that does not reflect the latest changes send to the command-side. There are difficulties introduced by eventual consistency, such as returning items to a client that in fact are already deleted through commands send to the command-side. The patterns to overcome this difficulty and others are out of scope for the current paper.

The asynchronous sending of events between the command-side and query-side results in a weak coupling. The resulting freedom and flexibility in designing the system leads to availability, scalability, and performance among other advantages. The store used in the command-side is often an event store, because it is natural to store the events that are produced by the command-side. This proposed data storage model has a number of benefits that make it specifically useful as a store for the command-side of a CQRS system. First of all, the command-side is only used for accepting changes and never for queries, and the performance of the store is not thus not hampered by concurring reads and writes. Second, the store contains every change ever accepted into the system, making it easy to inspect when and by whom a change was done. A third benefit is the possibility to rebuild the current state (for instance the query-store) in the system by replaying the events. The replaying of events also enables easy debugging. The fourth benefit is the possibility to analyze the events for patterns in usage. This information is impossible to extract from a store that only persists the last state of the data. In the query-side a diverse range of stores can be used, such as relational, graph, or NoSql databases. The main goal of this store is to support the easy and fast retrieval of data, in whatever form the application requires.

The loosely coupled nature of CQRS combined the benefits of the event sourcing approach makes it a fitting architectural pattern for cloud systems. Event sourcing itself is not tied exclusively to CQRS, the coupling based on events is similar to that in more general event-driven architectures, as described by Michelson [13]. The events in the event store are processed by the system to build the query-side or execute complex processes. The CQRS pattern and its sub-patterns are described in more detail by Kabbedijk et al. [14]. CQRS from a practitioners viewpoint is studied by Korkmaz [15] in order to gain better understanding of the benefits and challenges. Maddodi et al. [16] studies a CQRS system in the context of continuous performance testing.

An audit log is the simplest, yet also one of the most effective forms of tracking temporal information. The idea is that any time something significant happens you write some record indicating what happened and when it happened.

4.3 Event-Sourced Architectures

Event Sourcing is the foundation for Parallel Models or Retroactive Events. If you want to use either of those patterns you will need to use Event Sourcing first. Indeed this goes to the extent that it's very hard to retrofit these patterns onto a system that wasn't built with Event Sourcing. Thus if you think there's a reasonable chance that the system will need these patterns later it's wise to build Event Sourcing now. This does seem to be one of those cases where it isn't wise to leave this decision to later refactoring.

Event Sourcing also raises some possibilities for your overall architecture, particularly if you are looking for something that is very scalable. There is a fair amount of interest in 'event-driven architecture' these days. This term covers a fair range of ideas, but most of centers around systems communicating through event messages. Such systems can operate in a very loosely coupled parallel style which provides excellent horizontal scalability and resilience to systems failure.

An example of this would be a system with lots of readers and a few writers. Using Event Sourcing this could be delivered as a cluster of systems with in-memory databases, kept up to date with each other through a stream of events. If updates are needed, they can be routed to a single master system (or a tighter cluster of servers around a single database or message queue) which applies the updates to the system of record and then broadcasts the resulting events to the wider cluster of readers. Even when the system of record is

the application state in a database this could be a very appealing structure. If the system of record is the event log, there is are plenty of options for very high performance since the event log is a purely additive structure that requires minimal locking.

Such an architecture isn't flawless, of course. The reader systems are liable to be out of sync with the master (and each other) due to differences in timing with event propagation. However this broad style of architecture is used and I've heard almost entirely favorable comment about it.

Using event streams like this also allows new applications to be added easily by tapping into the event streams and populating their own models, which don't need to be the same for all systems. It's an approach that fits in very well with a messaging approach to integration.

comparison table why event-sourced/driven [Richards, 2015]

4.3.1 Domain Event

A domain event is a class with a name formed using a past-participle verb. It has properties that meaningfully convey the event. Each property is either a primitive value or a value object. For example, an OrderCreated event class has an orderId property. A domain event typically also has metadata, such as the event ID, and a timestamp. It might also have the identity of the user who made the change, because that's useful for auditing. The metadata can be part of the event object, perhaps defined in a superclass. Alternatively, the event metadata can be in an envelope object that wraps the event object. The ID of the aggregate that emitted the event might also be part of the envelope rather than an explicit event property. The OrderCreated event is an example of a domain event. It doesn't have any fields, because the Order's ID is part of the event envelope. The following listing shows the OrderCreated event class and the DomainEventEnvelope class. [Richardson, 2018]

4.4 Challenges

However, an ESS introduces not only events and state transfers. Eventual consistency forces developers to let go of guarantees that they would have in a system using current state and synchronous processing. In a CQRS system, an update sent through a command

will not immediately be reflected in the result of a query. The system first needs to process the event into one or more projections. Engineer E12 states that "a lot of developers had to get used to information not being in place", and E2 adds that "getting people to understand eventual consistency is the biggest hurdle". Eventual consistency forces developers to rethink the basic interactions of the user with the system.

We give two examples of interactions that force developers to rethink system design. The first example is that of the expectation of users to retrieve data that they previously submitted into the system. However, in a CQRS system, the query system might not directly return the data that was submitted through a command. The user interface of the system should make it clear to the user what is going on, or even try to hide the fact that the system is eventual consistent. The second example is that of developers that more or less have the same expectation. Often developers try to use the result of the query to make decisions in an aggregate. However, the query system might not have processed all events and misses recent updates. If developers overlook this principle, the decisions lead to bugs in the system.

4.4.1 Event storage

Event-sourcing enables the reconstruction of arbitrary past application states for event-based applications. However, an entirely unbounded log size can conflict with other application requirements. We provided a first exploration of log pruning approaches for event-sourced systems and evaluated their effects as the main contributions of this work. This included an overview of the design space of pruning approaches and an assessment of the impact of log pruning mechanisms on state reconstructability. Unbounded command and event sourcing approaches enable a full reconstruction of previous states, but come with high costs in terms of storage. In practice, pure command sourcing is often not feasible due to complex re-computations. Bounded approaches restrict the log sizes, but enable reconstructions primarily for more recent states. Additional periodic checkpoints can help to maintain some older states as reference. The probabilistic approach provides a configurable bias to determine what to prune. Eventually, the choice of a log pruning approach has to reflect the application-level requirements — which entity states should be kept, how far the application needs to go back in time, and in which resolution. Application-based and user-defined pruning mechanisms as well as a detailed analysis of pruning parameters constitute future work.

some math from https://dl.acm.org/doi/10.1145/3210284.3219767 (https://dl.acm.org/doi/pdf/10.1145/321

4.4.2 Event schema evolution

Table 6.1 shows the different types of changes that can occur at each level. [Richardson, 2018]

With event sourcing, the schema of events (and snapshots!) will evolve over time. Because events are stored forever, aggregates potentially need to fold events corresponding to multiple schema versions. There's a real risk that aggregates may become bloated with code to deal with all the different versions. As mentioned in section 6.1.7, a good solution to this problem is to upgrade events to the latest version when they're loaded from the event store. This approach separates the code that upgrades events from the aggregate, which simp

4.4.3 Deleting data is tricky.

Because one of the goals of event sourcing is to preserve the history of aggregates, it intentionally stores data forever. The traditional way to delete data when using event sourcing is to do a soft delete. An application deletes an aggregate by setting a deleted flag. The aggregate will typically emit a Deleted event, which notifies any interested consumers. Any code that accesses that aggregate can check the flag and act accordingly. Using a soft delete works well for many kinds of data. One challenge, however, is complying with the General Data Protection Regulation (GDPR), a European data protection and privacy regulation that grants individuals the right to erasure (https://gdpr-info.eu/art-17-gdpr/). An application must have the ability to forget a user's personal information, such as their email address. The issue with an event sourcing-based application is that the email address might either be stored in an AccountCreated event or used as the primary key of an aggregate. The application somehow must forget about the user without deleting the events. Encryption is one mechanism you can use to solve this problem. Each user has an encryption key, which is stored in a separate database table. The application uses that encryption key to encrypt any events containing the user's personal information before storing them in an event store. When a user requests to be erased, the application deletes the encryption key record from the database table. The user's personal information is effectively deleted, because the events can no longer

be decrypted. Encrypting events solves most problems with erasing a user's personal information. But if some aspect of a user's personal information, such as email address, is used as an aggregate ID, throwing away the encryption key may not be sufficient. For example, section 6.2 describes an event store that has an entities table whose primary key is the aggregate ID. One solution to this problem is to use the technique of pseudonymization, replacing the email address with a UUID token and using that as the aggregate ID. The application stores the association between the UUID token and the email address in a database table. When a user requests to be erased, the application deletes the row for their email address from that table. This prevents the application from mapping the UUID back to the email address. [Richardson, 2018]

4.4.4 Querying the event store is challenging.

Imagine you need to find customers who have exhausted their credit limit. Because there isn't a column containing the credit, you can't write SELECT * FROM CUSTOMER WHERE CREDIT_LIMIT = 0. Instead, you must use a more complex and potentially inefficient query that has a nested SELECT to compute the credit limit by folding events that set the initial credit and adjusting it. To make matters worse, a NoSQL-based event store will typically only support primary key-based lookup. Consequently, you must implement queries using the CQRS approach described in chapter 7. [Richardson, 2018]

4.4.5 Transactions

https://www.youtube.com/watch?v=RGqr1cXjS5o

4.5 Benefits

Event sourcing can be used to implement Auditing 2.0 in a number of ways. Here are a few examples:

Storing a log of events: As mentioned earlier, event sourcing involves storing a log of events that represent changes made to the data over time. This log can be used to reconstruct the state of the data at any point in the past, which can be useful for auditing

purposes. By storing a comprehensive and immutable log of events, organizations can use event sourcing to track and monitor changes to sensitive data and ensure compliance with regulations and standards.

Analyzing event data: In addition to storing the log of events, organizations can also use data analytics and machine learning techniques to analyze the event data and identify patterns and trends that may indicate potential risks or issues. By continuously analyzing the event data, organizations can proactively identify and address potential problems as they arise, rather than waiting for them to be discovered during a periodic audit.

Automating the audit process: Event sourcing can also be used to automate the audit process by using technologies such as artificial intelligence and blockchain. For example, AI-powered systems can be used to analyze the event data and identify potential risks or issues, while blockchain technology can be used to provide a secure and transparent record of transactions. This can help to streamline the audit process and make it more efficient.

Overall, event sourcing can be a powerful tool for implementing Auditing 2.0. By storing a comprehensive log of events and using advanced technologies and data analytics to analyze the data, organizations can continuously monitor and assess their financial and operational processes, identify potential risks and issues, and take timely and appropriate action to address them.

Event sourcing is related to database systems techniques used for persistence guarantees and replication. Gray and Reuter (1992) describe how transaction logs can be used to replicate state between database systems. Every state change is recorded as a transaction, which is similar to event sourcing where every state change is recorded as an event. Kleppmann (2017) discusses event sourcing in the context of data-intensive applications, he relates the pattern to the change data capture approach, often used in Extract-Transform-Load (or ETL) processes (Vassiliadis, 2009). ETL solutions are often used for creating data warehouses. The primary difference between event sourcing and these techniques is that a transaction or a data change is a technical entity without relation to the real world, while an event in event sourcing resembles an event in the real world.

Event sourcing is a pattern that stores every state change, immutability is thus at the core of the pattern. Helland (2015) states that immutability of data is a crucial aspect for distributed systems. Although often seen as the defining characteristic of event sourcing,

immutability is not enforced in any manner, as opposed to a blockchain. In a number of the systems under study, immutability is sacrificed for a simpler schema evolution technique (see Section 7). We observed different degrees of immutability. The first degree is strict, 8 out of the 19 ESSs never change an event. The second degree of immutability is used by 3 out of 19 systems, which allow for cut-off moments. In such a cut-off moment, the event store is changed, but back-ups guarantee that no information is deleted. The goal of these back-ups is to satisfy regulations or service-level agreements, therefore, they are kept around forever. This degree of immutability still guarantees an audit trail, because the back-ups can be used to retrieve all the state changes. The last degree level of immutability is mutable, 8 out of 19 systems allow events to change. In these systems, the event store is changed on some occasions, and the back-ups are not kept forever. These systems do not satisfy the goal of a complete audit trail. However, the events can still be used to explain how the current state was reached. None of the ESSs lose information regarding the current state of a system. Events that are changed, or transformed, are in most cases changed because of technical reasons.

How can a System that Uses Event Sourcing Protect User Privacy? — Privacy regulations, such as the GDPR, are designed to protect users from being taken advantage of. Personal information should not be kept in a system for all eternity, but the system should delete it whenever someone requests that. However, such a requirement conflicts with the nature of event sourcing: retaining all the data. Engineers E20, E21, E23, E25 mention that they designed their systems to comply with these regulations. Systems HealthSys and P-PaySys use some form of anonymization and removal of information to comply. Obviously, this requires them to rewrite events. System IdentitySys takes a completely different approach. The system separates the events and the personal information in two different stores. When events are read, they are supplemented with the personal information. If that information is no longer present (because of removal requests), default values are supplied.

Easy temporal queries - Because event sourcing maintains the complete history of each business object, implementing temporal queries and reconstructing the historical state of an entity is straightforward.

5 Software Architecture and Auditing

Design patterns

Wenn mehrere Teil-Kapitel zu strukturieren sind: Schreiben Sie zu jedem Teil-Kapitel eine Ein- leitung ("Hier wird die folgende Fragestellung untersucht...") und eine Ausleitung ("Hiermit ist erreicht: ... Die folgenden Probleme sind aber noch offen:...").

Software architecture is the structure, or set of structures, which comprises software elements, the externally visible properties of those elements, and the relationships among them [Bass u. a., 2003]. This structure is an artifact from a software development process and is represented by a document composed by one or more models, which represent different perspectives about how the system will be structured, and information sets that facilitate the understanding of the proposed computational solution. It is defined based on the software requirements. Among the different types of requirements, the quality requirements are the most important for the specification of an architecture since it exerts considerable influence over it structure [Bass u. a., 2003].

5.1 Implementing audit logging

There are a few different ways to implement audit logging: Richardson [2018]

5.1.1 Audit logging code in business logic

The first and most straightforward option is to sprinkle audit logging code throughout your service's business logic. Each service method, for example, can create an audit log entry and save it in the database. The drawback with this approach is that it intertwines auditing logging code and business logic, which reduces maintainability. The other drawback is that it's potentially error prone, because it relies on the developer writing audit logging code.

5.1.2 Aspect-Oriented programming

The second option is to use AOP. You can use an AOP framework, such as Spring AOP, to define advice that automatically intercepts each service method call and persists an audit log entry. This is a much more reliable approach, because it automatically records every service method invocation. The main drawback of using AOP is that the advice only has access to the method name and its arguments, so it might be challenging to determine the business object being acted upon and generate a businessoriented audit log entry.

5.1.3 Event Sourcing

The third and final option is to implement your business logic using event sourcing. As mentioned in chapter 6, event sourcing automatically provides an audit log for create and update operations. You need to record the identity of the user in each event. One limitation with using event sourcing, though, is that it doesn't record queries. If your service must create log entries for queries, then you'll have to use one of the other options as well.

5.2 traditional persistence

The traditional approach to persistence maps classes to database tables, fields of those classes to table columns, and instances of those classes to rows in those tables. For example, figure 6.1 shows how the Order aggregate, described in chapter 5, is mapped to the ORDER table. Its OrderLineItems are mapped to the ORDER_LINE_ITEM table.

The application persists an order instance as rows in the ORDER and ORDER_LINE_-ITEM tables. It might do that using an ORM framework such as JPA or a lower-level framework such as MyBATIS. This approach clearly works well because most enterprise applications store data this way. But it has several drawbacks and limitations: - Object-Relational impedance mismatch. - Lack of aggregate history. - Implementing audit logging is tedious and error prone. - Event publishing is bolted on to the business logic. Let's look at each of these problems, starting with the Object-Relational impedance mismatch problem. [Richardson, 2018]

5.2.1 Problems

OBJECT-RELATIONAL IMPEDANCE MISMATCH

One age-old problem is the so-called Object-Relational impedance mismatch problem. There's a fundamental conceptual mismatch between the tabular relational schema and the graph structure of a rich domain model with its complex relationships. Some aspects of this problem are reflected in polarized debates over the suitability of Object/Relational mapping (ORM) frameworks. For example, Ted Neward has said that "Object-Relational mapping is the Vietnam of Computer Science" (http://blogs .tedneward.com/post/the-vietnam-of-computer-science/). To be fair, I've used Hibernate successfully to develop applications where the database schema has been derived from the object model. But the problems are deeper than the limitations of any particular ORM framework.

LACK OF AGGREGATE HISTORY

Another limitation of traditional persistence is that it only stores the current state of an aggregate. Once an aggregate has been updated, its previous state is lost. If an application must preserve the history of an aggregate, perhaps for regulatory purposes, then developers must implement this mechanism themselves. It is time consuming to implement an aggregate history mechanism and involves duplicating code that must be synchronized with the business logic.

implementing audit logging is tedious and error prone

Another issue is audit logging. Many applications must maintain an audit log that tracks which users have changed an aggregate. Some applications require auditing for security or regulatory purposes. In other applications, the history of user actions is an important feature. For example, issue trackers and task-management applications such as Asana and JIRA display the history of changes to tasks and issues. The challenge of implementing auditing is that besides being a time-consuming chore, the auditing logging code and the business logic can diverge, resulting in bugs.

event publishing is bolted on to the business logic

Another limitation of traditional persistence is that it usually doesn't support publishing domain events. Domain events, discussed in chapter 5, are events that are published by an aggregate when its state changes. They're a useful mechanism for synchronizing

data and sending notifications in microservice architecture. Some ORM frameworks, such as Hibernate, can invoke application-provided callbacks when data objects change. But there's no support for automatically publishing messages as part of the transaction that updates the data. Consequently, as with history and auditing, developers must bolt on event-generation logic, which risks not being synchronized with the business logic. Fortunately, there's a solution to these issues: event sourcing

5.3 Event Sourcing

Event sourcing is an architecture pattern that involves storing the history of events that have occurred in a system as a sequence of records. This allows the system to reconstruct past states and to track changes over time.

One way in which event sourcing can be used for auditing is by providing a complete record of all events that have occurred in the system, including information about when the events occurred and who was responsible for them. This can be useful for identifying and analyzing trends, identifying patterns of behavior, and reconstructing past states of the system.

There are several other architecture patterns that can also be used for auditing, including:

Command and Query Responsibility Segregation (CQRS): This pattern involves separating the responsibilities of reading and writing data, allowing for better scalability and security. CQRS can be used to maintain a separate audit log of all write operations, which can be used for auditing purposes.

Change Data Capture (CDC): This pattern involves capturing and storing changes to data as they occur, allowing for real-time analytics and data integration. CDC can be used to track changes to data over time, which can be useful for auditing purposes.

Two-Phase Commit (2PC): This pattern involves coordinating the execution of transactions across multiple systems, ensuring that either all or none of the changes are made. 2PC can be used to maintain a record of all transactions that have been committed, which can be useful for auditing purposes.

Ultimately, the choice of architecture pattern for auditing will depend on the specific needs of the system and the requirements of the audit process.

100% accurate audit logging - Auditing functionality is often added as an afterthought, resulting in an inherent risk of incompleteness. With event sourcing, each state change corresponds to one or more events, providing 100% accurate audit logging. [Richardson, 2018]

5.3.1 Database Driven

https://eventuate.io/whyeventsourcing.html

5.4 Blockchain

Anh et al. (2018) describes another append-only data structure: blockchain. While the data structure is similar to event sourcing, the goals of the two techniques are different. A blockchain focuses on solving problems related to distribution, consensus, and trust, while event sourcing solves problems with history, temporal complexity, and audit trails. The blockchain approach enforces the immutability of the data to solve its problems, while in event sourcing this immutability is self imposed. Event sourced systems could be build using a blockchain solution. However, the distribution and consensus features offered by blockchain do not improve the goals targeted by event sourcing.

5.5 Auditing 2.0

6 Audit Component

Wenn mehrere Teil-Kapitel zu strukturieren sind: Schreiben Sie zu jedem Teil-Kapitel eine Ein- leitung ("Hier wird die folgende Fragestellung untersucht...") und eine Ausleitung ("Hiermit ist erreicht: ... Die folgenden Probleme sind aber noch offen:...").

The event sourced system Overeem u. a. [2021]

Auditing Framework van der Aalst u. a. [2010] Event logs and process mining techniques enable new forms of auditing. Rather than sampling a small set of cases, auditors can consider the whole process and all of its instances. Moreover, they can do this continuously.

Figure 1 shows an Auditing 2.0 framework based on process mining. “Current data” events are cases that are still running, while “Historic data” events are completed cases. The figure also shows two types of process models: De jure models describe a desired or required way of working, while de facto models aim to describe reality with potential violations of the boundaries defined in de jure models (W. M. P. van der Aalst et al., “Conceptual Model for On Line Auditing,” tech. report BPM-09-19 BPMcenter.org, 2009).

6.1 Requirements

6.1.1 Use-Cases

6.1.2 Stakeholders

6.1.3 Architectural Constraints

6.1.4 Base System

6.2 System Design

6.2.1 Enabling Technologies

6.2.2 Scope and Context

6.2.3 Solution Strategy

6.2.4 Building Block View

6.2.5 Runtime View

6.2.6 Deployment View

6.3 Design Decisions

6.3.1 DD01: Registry pattern

6.4 Technical Decisions

7 Audit Browser

Wenn mehrere Teil-Kapitel zu strukturieren sind: Schreiben Sie zu jedem Teil-Kapitel eine Ein- leitung ("Hier wird die folgende Fragestellung untersucht...") und eine Ausleitung ("Hiermit ist erreicht: ... Die folgenden Probleme sind aber noch offen:...").

Process Discovery van der Aalst u. a. [2010] By analyzing frequent patterns, process mining techniques can extract from event logs models that describe the processes at hand. For example, the Alpha process mining algorithm can automatically extract a Petri net that concisely models behavior in the event log. This gives the auditor an unbiased view of what has actually happened.

Conformance Checking An auditor can use an a priori process model to check if reality, as recorded in the event log, conforms to the model and vice versa. For example, a model may indicate that purchase orders exceeding one million euros require two checks. Auditors can use conformance checking to detect deviations, locate and explain them, and measure their severity (A. Rozinat and W.M.P. van der Aalst, "Conformance Checking of Processes Based on Monitoring Real Behavior," Information Systems, vol. 33, no. 1, 2008, pp. 64–95).

Toward Operational Support Although process mining has traditionally focused on offline analysis and is seldom used for operational decision support, it can consider running-process instances and compare them with models based on historic data or business rules (W. M. P. van der Aalst, M. Pesic, and M. Song, "Beyond Process Mining: From the Past to Present and Future," tech. report BPM-09-18 BPMcenter.org, 2009). For example, an auditor can "replay" a running case on the process model in real time and check whether the observed behavior fits. The moment the case deviates, the auditor can alert an appropriate actor.

Auditors can also use a process model based on historic data to make predictions about running cases—for example, to estimate the remaining processing time and a particu-

lar outcome's probability—or provide recommendations, such as an activity that will minimize the expected costs and completion time.

7.1 Requirements

Anforderungen aufgrund von Gesprächen mit dem Anwender und aufgrund der technischen und sozialen Verantwortung des Entwicklers,Stand der Technik und ggf. der Forschung,

Keine Anforderung darf im Folgenden vergessen werden (mindestens Erwähnung in "Weiterentwicklungsmöglichkeiten"). • Jedes Feature des konzipierten bzw. realisierten Systems muss auf eine Anforderung zurück- geführt werden.

7.1.1 Use-Cases

7.1.2 Stakeholders

7.1.3 Architectural Constraints

7.2 System Design

7.2.1 Enabling Technologies

the 8 fallacies of distributed computing

protobuf vs json: https://youtu.be/CAGuhVIOT2c?t=1206

Stand des Projekts und Weiterentwicklungsmöglichkeiten. • Anhang: Entwicklerdokumentation,Benutzerdokumentation Elektronischer Anhang: Sources, Projektdateien.

7.2.2 Scope and Context

7.2.3 Solution Strategy

7.2.4 Building Block View

7.2.5 Runtime View

7.2.6 Deployment View

7.3 Design Decisions

7.3.1 DD01: gRPC proxy

Communicating using the synchronous Remote procedure invocation pattern [Richardson, 2018]

comparison: https://youtu.be/CAGuhVIOT2c?t=1829

dealing with api changes

7.3.2 DD02: Micro Frontends

7.4 Technical Decisions

Lösungsalternativen sind kritisch zu bewerten und Auswahl zu begründen.

8 Installation and Configuration

Weiterentwicklungsmöglichkeiten

8.1 Test Run

9 Conclusion

Weiterentwicklungsmöglichkeiten

Bibliography

[van der Aalst u. a. 2010] Aalst, Wil M. van der ; Hee, Kees M. van ; Werf, Jan M. van der ; Verdonk, Marc: Auditing 2.0: Using Process Mining to Support Tomorrow's Auditor. In: *Computer* 43 (2010), Nr. 3, S. 90–93

[BaFin 2021] BaFin, Bundesanstalt für F.: *Rundschreiben 11/2021 (BA) - Zahlungsdiensteaufsichtliche Anforderungen an die IT (ZAIT)*. 11 2021. – URL https://www.bafin.de/dok/16514544. – (Accessed on 12/15/2022)

[Bass u. a. 2003] Bass, Len ; Clements, Paul ; Kazman, Rick: *Software Architecture In Practice*. 01 2003. – ISBN 978-0321154958

[Gantz 2014] Gantz, Stephen D.: Chapter 1 - IT Audit Fundamentals. In: Gantz, Stephen D. (Hrsg.): *The Basics of IT Audit*. Boston : Syngress, 2014, S. 1–19. – URL https://www.sciencedirect.com/science/article/pii/B9780124171596000018. – ISBN 978-0-12-417159-6

[Hanna und Rance 2011] Hanna, Ashley ; Rance, Stuart: ITIL® glossary and abbreviations English. In: *Haettu* 14 (2011), S. 2021. – URL https://www.alaska.edu/files/oit/ITIL_2011_English_glossary_v1.0.pdf

[ISO 19011 2018] ISO 19011: *ISO - ISO 19011:2018 - Guidelines for auditing management systems*. 07 2018. – URL https://www.iso.org/obp/ui/#iso:std:iso:19011:ed-3:v1:en:term:3.1. – (Accessed on 12/13/2022)

[Overeem u. a. 2021] Overeem, Michiel ; Spoor, Marten ; Jansen, Slinger ; Brinkkemper, Sjaak: An empirical characterization of event sourced systems and their schema evolution — Lessons from industry. In: *Journal of Systems and Software* 178 (2021), S. 110970. – URL https://www.sciencedirect.com/science/article/pii/S0164121221000674. – ISSN 0164-1212

[Richards 2015] Richards, Mark: *Software architecture patterns*. Bd. 4. O'Reilly Media, Incorporated 1005 Gravenstein Highway North, Sebastopol, CA . . . , 2015

[Richardson 2018] Richardson, Chris: *Microservices patterns: with examples in Java.* Simon and Schuster, 2018

A Anhang

Glossary

HAW Hamburg Die HAW Hamburg ist die vormalige Fachhochschule am Berliner Tor.

test test with plural plural.

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original