Enhanced Version Control for Unconventional Applications

by

Ahmed Saleh Shatnawi

A Dissertation Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

in Computer Science

at

The University of Wisconsin–Milwaukee

December 2017

Abstract

Enhanced Version Control for Unconventional Applications

by

Ahmed Saleh Shatnawi

The University of Wisconsin–Milwaukee, 2017
Under the Supervision of Professor Ethan V. Munson

The Extensible Markup Language (XML) is widely used to store, retrieve, and share digital documents. Recently, a form of Version Control System has been applied to the language, resulting in Version-Aware XML allowing for enhanced portability and scalability. While Version Control Systems are able to keep track of changes made to documents, we think that there is untapped potential in the technology. In this dissertation, we present novel ways of using Version Control System to enhance the security and performance of existing applications. We present a framework to maintain integrity in offline XML documents and provide non-repudiation security features that are independent of central certificate repositories. In addition, we use Version Control information to enhance the performance of Automated Policy Enforcement eXchange framework (APEX), an existing document security framework developed by Hewlett-Packard (HP) Labs.

Finally, we present an interactive and scalable visualization framework to represent Version-Aware-related data that helps users visualize and understand version control data, delete specific revisions of a document, and access a comprehensive overview of the entire versioning history.

I dedicate this dissertation to my mother, Sarah Bsoul, and my father, Saleh Shatnawi. Without their support, it would not have been possible for me to have started, much less finish.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to sincerely thank my advisor Professor Ethan Munson for the time he spent assisting me in the ideation and research processes, for his patience, for the encouragement I found from him, and, above all, his constant willingness to offer his support. Being Professor Ethan's graduate student has by itself been a gift, besides his knowledge in the field and stature as a renowned figure in software engineering and intelligence, he is the kind of person you will learn from about all matter of things on every encounter.

Special thanks to Prof. Xu Guangwu and Prof. Tian Zhao for the many years of support and help. Thanks to Prof. Michael Zimmer and Prof. Yi Hu for their insights and comments about my research that have always been helpful. Thanks to my brother and my sister for their support. Thanks to my friend Osama Al-Khaleel for his continuous help and support. I would like to thank my friend Mohammad Hajjat for his insights and providing valuable resources.

Finally, I would like to thank my wife, Rasha and my sons Ayham and Aws for standing beside me throughout my graduate journey and being in my life. I couldn't be the man I am without them, and their support means the world to me.

Chapter 1

Introduction

A document lifecycle is defined as a set of stages, where a document goes through some or all these stages. The stages of a document lifecycle might include: creation, storage, retrieval, searching, sharing and archiving. Not all documents will pass through all stages.

Recent events, such as the strong hurricanes that hit the U.S. in 2017, show that even in the developed world, online services may only function intermittently. In order to deal with such cases, computer scientists should think seriously about alternative approaches that support information technology services during offline periods while maintaining security.

Today's use of documents has changed substantially from 20 years ago. Documents are being accessed through portable devices, the rate at which documents are generated has increased, and there has also been an increase in the number of Web services and social media applications, many of which are document-based. Considering these issues, Simske and Balinsky [1] concluded that the typical document life-cycle does not provide sufficient privacy or security.

The Extensible Markup Language (XML) is a widely used language that defines rules to represent digital documents in a readable way for both humans and machines. Many modern applications rely on XML to store, retrieve, and process their data. For example, MS Office and LibreOffice "files" are archives containing multiple XML files. Modern Electronic Health Record (EHR) systems make extensive use of XML [2, 3, 4]. Thus, XML document security must be maintained through the whole document life-cycle as they might contain private or classified information. Specifically, unauthorized changes and the complete sequence of changes made

to the document should be identifiable. In this dissertation, we will focus on XML documents and their applications.

However, since we are talking about documents, we need to discuss the importance of collaborative authoring where people work together on shared documents to accomplish certain tasks. Collaborative authoring could happen online through a centralized service or could happen offline via secondary storage devices such as flash drives, compact disc (CD) and digital versatile discs (DVD).

Society has adopted collaboration on documents because the document is a shared responsibility, individual authors seek help from others to improve productivity and innovation, and a document is modified over time by people whose roles change or evolve. However, collaboration on documents presents security challenges that could affect the correctness of documents, expose confidential information to unauthorized users, and compromise the availability of the data. Besides, many collaborative frameworks nowadays rely on access control and security policies to maintain security, where both access control and security policies are determined at the time the document is created and are assumed to be unchanging [5]; or they can be modified afterward by the owners/administrators in a manual fashion [6].

The norm that a document will be initially assigned a set of static security parameters without taking into consideration the possibility that the content of the documents could change over time. This may lead to the exposure of sensitive information to unauthorized people (information leakage).

In an effort to address these issues, HP Labs developed a framework called Automated Policy Enforcement eXchange [7] (APEX) in order to ensure the enforcement of security policies while the document's content changes dynamically. We will discuss APEX in detail in Section 2.2.

The subject of document security is a rather broad topic; therefore, a brief description of what is in the scope for our work is in order. Historically, the *Confidentiality*, *Integrity*, and *Availability* (CIA) principles have been considered to be the backbone

of information security [8]. A system that maintains *Confidentiality* prevents data from being revealed to unauthorized users. A system that maintains *Integrity* guarantees the accuracy of data during the whole document life-cycle. A system that maintains *Availability* ensures continuous data availability when it is needed.

The second order security principles are *Assurance*, *Authenticity*, and *Anonymity* (AAA). These principles were designed to support the CIA principles as security researchers tried to cope with modern computer security challenges. A system that maintains *Assurance* will provide a trusted system by making sure to enforce policies and permissions. A system that maintains *Authenticity* should identify users and determine whether a particular user is authorized to do a certain task. A system that maintains *Anonymity* requires separating any digital transactions from identifying information that can be used to identify authorized users.

The *authenticity* principle leads to the *non-repudiation* security principle. A system that supports *non-repudiation* is able to provide a non-repudiable proof of actions the authorized users have made. Non-repudiation can be accomplished by using digital signatures.

Relying only on both CIA and AAA to achieve information security without paying attention to implementation, usage, and design details can lead to security flaws in system design, and hence, to a non-secure system. For instance, to make a document's content confidential, an encryption algorithm should be used to transform the document plaintext to ciphertext. If a weak encryption algorithm or a weak key is used, security problems will follow.

A Version control system (VCS) is a tool that is used to track, store, and manage changes made to documents. VCSs were developed initially to help software developers collaborate on the production of correct software documents. Using a VCS allows management of revisions, switching between revisions, and merging documents. In addition, version control data provides a rich source of information to enhance users' collaboration process and help to produce a higher quality of work. More details

about VCSs and their basic operations can be found in Section 2.1.

Recently, a novel version control technology called Version-Aware XML documents was presented by Thao and Munson [9]. Version-Aware XML documents store the entire revision history of a document inside the document itself without relying on a central repository. Adopting Version-Aware XML document technology frees users from any obligations to use a repository or any network connections to collaborate. Hence, it provides offline documents the ability to have many of the same sophisticated version control features as traditional, online software version control systems.

Our primary goal of this Ph.D. dissertation is to provide an enhancement to Version-Aware XML document technology and then use the technology in innovative ways to improve the security and performance of other applications.

The research will address four main topics:

- Enhancing Version-Aware XML documents.
- Enhancing the Automated Policy Enforcement eXchange(APEX).
- Maintaining integrity and non-repudiation in secure offline documents.
- Providing interactive graphical user interfaces for version control systems.

Chapter 2 presents the background information and previous research. Subsequently, Chapter 3 discuss determining UID length, our results and evaluation. Chapter 4 discuss enhancing the Automated Policy Enforcement eXchange (APEX) the motivation and our proposed solutions. Chapter 5 discuss maintaining integrity and non-repudiation in secure offline documents, our results and evaluation. Chapter 6 discuss interactive graphical user interfaces for Version-Aware XML document our results, and evaluation. Chapter 7 summarizes the contributions of our research and presents possible future enhancements.

Chapter 2

Background

This chapter presents background information and previous research important for understanding the ideas presented in this dissertation. Section 2.1 will discuss version control systems; in particular, Version-Aware XML documents and related work. Section 2.2 will discuss related work in the secure document literature, and focus on the APEX framework, noting issues that arise from it. Section 2.3 will discuss research related to maintaining integrity and non-repudiation in secure offline documents. We will also discuss the need for generating unique XML tag IDs and choosing an optimal length for unique IDs. Finally, Section 2.4 will discuss other work on version control visualization systems.

2.1 Version Control

Version control systems are used to track, store, and manage all the changes made to documents. Version control systems generally support a common set of operations:

- Create: The create operation issued to create a new empty repository to store files in the version control system.
- Checkout: The checkout operation issued to get a snapshot (copy) of a version from the repository in order to work on. We call this copy the *working copy*.
- Commit: The commit operation issued to write changes made on the working copy to the repository. The operation crests a new version of the file.
- Diff: The diff operation finds the differences between the working copy and another version or between any other two versions.

- Branch: The branch operation is used to create an independent version path (or *branch*). An example would be when two users collaborating on a repository for a program, and it has several bugs. The first collaborator plans to fix these bugs and publish the corrected code, while at the same time, the second collaborator wants to create a new version of the software that contains extra features. Thus, the branch will be used to go in both directions, fixing the bugs and creating a new version of the software.

- Merge: The merge operation issued to merge changes made in two branches. Using the previous example, after the first collaborator finishes fixing the bugs in one branch and the second collaborator finishes adding features in another, they merge both branches to get a new version without bugs and with support for the new features added.

Current version control systems use two main techniques to store versioning data. They either store a complete copy of each new revision, or they store deltas, which represent only the difference between two revisions. Examples of version control systems that store only deltas are Visual SourceSafe and Source Code Control System [10] (SCCS), whereas a modern VCS like Git stores snapshots of revisions instead of string deltas [11].

As version control systems have evolved, two main types emerged: centralized and distributed. Many papers in the literature discuss both types of version control and illustrate the advantages and disadvantages of using each type [12, 13].

A centralized version control system has a central repository, where a client-server approach is adopted to store the revisions in the central repository. In contrast to the distributed version control systems, centralized version control systems cannot be used offline, the users must be connected to the system at all the times. Also, a single point of failure is a possible scenario when one server is being used [14, 15, 16, 17, 18]. An advantage of using a centralized VCS over distributed VCS is that when the project has a large version history, or the project has many large files, the distributed

version control systems would not be efficient at handling the load. Sink [19] presents a more detailed contrast. A summary of the advantages and disadvantages of both distributed and centralized version control systems are shown in Table 2.1. In this dissertation, portability and speed will be the most prioritized features that will discuss.

Table 2.1: Centralized vs. Distributed Version Control Systems

	Centralized	**Distributed**
Operations Speed	Relatively, slower that distributed.	Fast for most of the operations(i.e., clone, push, etc.)
Portability	Users must be connected to the central repository to be able to do any operation.	Disconnected operation is possible without being connected to the network
Implicit Backup	If the central repository data is corrupted, data might not be recovered.	Version control data can be fully or partially recovered due to information replication by distributed users.
Hardware Requirements	Since all the users access and do their operations on the central server, very high hardware specifications must be chosen for the server.	Since all the extensive operations happen on the client side, decent hardware specifications are needed for the client.
Locks	Lock operation issued to prevent other users to from committing any changes to the locked file. Locks are very useful while a group of people collaborating online.	There is not much support for a lock since it is meant to be a feature for online usage.
Obliterate	It's an easy task compared to distributed VCS.	It's difficult and expensive to obliterate all copies of versioning data data.

As stated above, a distributed version control system, known as Version-Aware XML documents, was recently developed by Thao and Munson [9]. Version-Aware documents utilize reverse deltas stored inside the document file itself, without relying on a central repository. In this case, reverse deltas represent the changes between each version of a changed document, rather than storing the versions of a document as a whole document every time a change has been made on top of the document. When reverse deltas are applied to a current document, a previous version of the document is retrieved. The Version-Aware technology was initially evaluated using Inkscape graphics editor.

The Version-Aware XML document technology essentially has three XML extensions on top of the XML document:

- A Revision-history element wraps a list of all revisions in the form of reverse deltas. Each revision history node has a unique ID attribute to support both branching and merging and has an attribute called "parent" that references the previous revision elements.

- A unique ID is added to each XML document element to identify each node. The use of unique IDs helps assure that changes are identified correctly and efficiently. The unique ID-based technique shows a performance enhancement to distinguishing changes made in each revision, and improved conflict detection compared to the hash-based model [20].

- An XML signature for the version data, which is used to verify its integrity.

In order to see if the Version-Aware approach is practical for modern office software, Pandey and Munson [21] modified LibreOffice Writer application to see if the unique document element IDs used by the Version-Aware approach could be supported. They showed that this was possible with very modest changes to the Writer code base.

Microsoft Office documents are actually a set of compressed XML files that store

data, style, settings, etc. Coakley *et al.* [22] created a custom Microsoft Word add-on to support Version-Aware XML document technology. Revisions of the document content were stored as a separate copy (snapshot) in a sub-directory inside the document. Success in applying the Version-Aware XML document technology to Microsoft Word was limited, they were not able to stamp each XML node in the underlying document representation with unique IDs due to Microsoft security restrictions. Recently, Alexandre *et al.* [23] were able to adapt the unique IDs technique in MS Word documents using the Microsoft Word's Revision Save ID (RSID), which will allow the use of efficient merging and differencing algorithms discussed in [20].

2.2 Automated Policy Enforcement eXchange framework(APEX)

Maintaining document security during the whole document life-cycle has become an important requirement nowadays and is considered a significant challenge in the document engineering field [1].

Security challenges arise when users collaborate on documents. The changes made can affect the correctness of documents, expose confidential information to unauthorized users, and compromise the availability of the data. Confidential data can be exposed to unauthorized users when emailing, printing or sharing XML files that contain sensitive data. Additionally, conventional document security approaches apply a static set of security parameters to a document or its component elements [24, 25].

Throughout this section and in Chapter 4, we will use *exposure operations* to denote operations that could lead to information leakage. These operations include printing, emailing, moving documents, sharing, and more; all of which can make it possible for unauthorized users get exposure to sensitive data inside documents.

Relying on static security policies and parameters without taking into consider-

ation the fact that the content of the documents could change over time can lead to information leakage. For example, suppose a document is created and does not contain any classified or confidential data at the time of creation. If the security parameters are set to allow usage of *exposure operations*, and, later on, confidential data is added to the document, the fixed security parameters will allow exposure operations, leading to data leakage.

Trying to address the above security issues, HP Labs developed a framework called Automated Policy Enforcement eXchange (APEX) to ensure the enforcement of security policies as the content of document components changes over time. The APEX [7] framework aims to:

1. Prevent unintentional actions that would violate security policies such as printing sensitive data on a shared printer or emailing confidential information outside the company, etc.

2. Handle insider threats considered one of the biggest security concerns to any organization, APEX would alleviate this threat by logging all actions performed by insiders on confidential documents.

3. Decouple security policy enforcement actions from the software interface, ensuring that all the documents in a local environment such as an organization, company, and government agency will be monitored and protected during the entire document life-cycle.

As shown in Figure 2.1, the APEX framework architecture is a client-server model.

- The server side has a central database and a policy editor.
  - The central database holds policies to be enforced on documents inside an organization or agency.
  - The policy editor lets administration staff edit or create security policies without manually interacting with the central database.

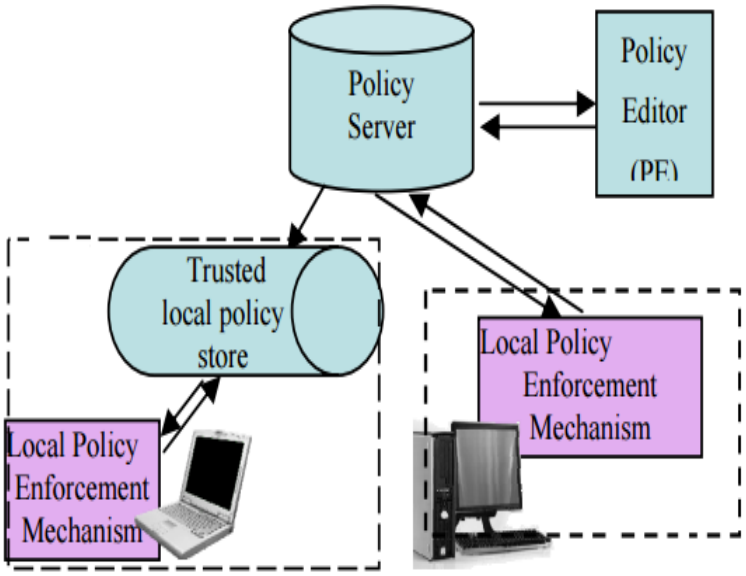


Figure 2.1: APEX Framework Architecture. [7]

- The client side's main component is a daemon or process that intercepts any user or system process action that can leak confidential content or metadata. The daemon scans the entire content of the document components and their metadata. When a user or a system process tries to perform certain actions on a document or its associated metadata, a rule will be evaluated to determine which security policy should be applied. The evaluation is based on the current content of the document, its current metadata and the current security privilege of the user or the system process that initiated the action.

Simske and Balinsky evaluated APEX framework in [26]. A limitation of their framework will be when users worked with large documents; APEX added a noticeable delay while analyzing the document against policies.

Although there is considerable research in the literature related to preventing data leakage, there is little mention of dynamic policy enforcement as a mechanism to secure resources during collaboration. In contrast to static policy enforcement, dynamic policy enforcement takes the current content of the document into account when determining which policy to apply. Yixiang *et al* [27] presented a technique to prevent data leakage of sensitive information in multiple XML documents. The

authors discussed a data publishing applications as a use case to show how data leakage of sensitive information is possible through common knowledge. They required the publisher (a human user) of XML files to specify manually which data should be protected. This can be a laborious task when dealing with large XML files, and, they did discuss fine-grained access control to prevent unauthorized users from executing a particular action on top of a document.

Bertino *et al.* [28] developed a Java-based program called Author-X that provides an access control system to let users browse and author XML documents. Author-X will grant or deny access based on credential-based security policies. These security policies control access at different granularity levels for XML documents based on users' credentials. Author-X supported browsing and authoring policies. Browsing policies manage users' privileges to read or navigate through the XML document contents, whereas authoring policies manage users' privileges to modify the XML document content such as append, write, delete, or insert operations. Author-X does not prevent people with sufficient privileges to access to the XML documents from using *exposure operations* that could lead to data leakage.

2.3 Maintaining Integrity and Non-Repudiation in Secure Offline Documents

An important contribution of this dissertation research is to develop an efficient approach for maintaining the integrity and non-repudiation in offline XML documents. In this section, we will focus on medical information, to show the importance of integrity and non-repudiation properties; and on MS Office which is widely used with over 1.2 billion Office users [29]. Both medical information and MS office have rigorous security requirements.

Verifying the integrity of medical data is essential. If a person can make changes to medical documents offline and push these changes through authorized users, then

they have effectively circumvented access controls. Back in 2013, two nurses were charged with falsifying stroke patients' records in the United Kingdom [30], as a consequence, 82 women died after having her records falsified by the nurses. Therefore, knowing what document changes are made, and who made the changes, when sharing Electronic Health Records (EHR) offline is a vital requirement.

EHR is an electronic-based patient health records contains medical history, diagnoses, medications, and other sensitive patient data. Although there is no standard format for exchanging EHR in the U.S., many XML based standards have been adopted by health care providers [31].

Similarly, the need to preserve XML document integrity and provide non-repudiation can also be found in legislative and governance domains, where multiple parties collaborate to construct laws, policies, and regulations. Since sharing XML-based documents can happen offline via flash drives and emails, the integrity of the files must be checked before finalizing such documents. Before a document is shared for collaboration, it is necessary to provide a mechanism to track changes that will be made to the document and verify the integrity of the document.

Both integrity and non-repudiation security features have been discussed in the literature for medical record systems [32]. The authors conducted a survey paper to identify frequently adopted security and privacy features for EHR systems. Related security features of interest to us are integration and sharing of EHR and these security policies.

However, rural areas and other regions with limited or unreliable internet connections make heavy use of offline sharing, where protecting document integrity is not a straightforward task. Unreliable internet connections, viruses, and lost hardware can compromise the integrity of documents as all of these factors could lead to illegally altering of data. Lacking the support of a central online version control repository to maintain document integrity across users, we need to have secure offline document control systems.

Multiple researchers have addressed the preservation of data integrity while collaboratively editing XML documents online. For instance, in [33] the authors used a security region-object parallel flow (S-RPF) graph protocol that supports simultaneous updates of different parts of an XML document, while preserving both confidentiality and integrity. In [34], Miklau and Suciu suggested a framework that provided both accuracy and authenticity for web published data. The authors used datasets annotations with unique data such as census data and medical databases that provided authorship evidence of the data. Liu *et al.* [35] discussed how to preserve XML document integrity through concatenated hash functions, but they did not discuss the mechanisms used to validate the users' certificates.

While using EHR was efficient for working, storing and extending paper-based medical records, researchers conducted their studies to analyze the impact on information integrity using EHR in health systems.

Bowman *et al.* [36] discussed the consequences of using EHR while neglecting security-oriented concerns and about document integrity. They discussed multiple potential risks that could impact the information integrity in EHR systems including user interface usability; lack of or poor documentation capture; copy and paste practices; use of preexisting templates; and other errors related to software workflow. However, they did not consider the risks from information tampering, other malicious activity, or accidental integrity problems resulting from modifications during offline access.

Mohammed *et al.* [37] discussed the consequences of noncompliance with United States privacy legislation (e. g. HIPAA and HITECH) in situations where integrity and reliability of health care systems can be compromised. The authors suggested conducting security assessments on a regular basis, and the institution of regular system audits and information audits.

Considerable research has been published on providing secure access control and updates in a distributed, cooperative environment for online tree-structured docu-

ments like XML [38, 39]. One important issue is knowing whether shared files have been tampered with by a non-authorized third party. An intruder or unauthorized user might provide false medical information or inject harmful scripts in files that will infect applications that parse the documents. While the negative effects of false information are fairly obvious, the risks from XML injection could be quite severe and can include data destruction or denial of service via malicious queries. An example of XML injection is the billion laughs attack [40] that affects Microsoft XML Core Services (MSXML) in Mac, this type of attack depletes the memory and crashes MS Word.

MS Word documents are designed to be a portable, non-centralized way to collaborate and share data. MS Word itself provides a limited form of version control and also allows users to sign their documents. However, Word does not provide a complete version history, allowing the recipient of a signed document to know only who last signed the document. Our approach provides an extensive document history with author signatures at each step of the version history. We will discuss in more detail about MS Word and version control in Chapter 5.

EPedigree [41] is an electronic document that provides support for chain tracking dangerous drugs from the manufacturer to the consumer by storing data on the history inventory using the pedigree XML schema. The data stored at each stage in the chain from the manufacturer of the product to the seller is digitally signed to provide both integrity and non-repudiation over the information stored. We will discuss and contrast our approach verses ePedigree in Chapter 5.

EPedigree has been successfully deployed to secure the supply chains of drugs in the pharmaceutical industry in the US [42]. This technology has been adopted to prevent anti-counterfeiting by providing a full history of a particular batch of a drug from the drug manufacturer to the consumer. It provides both integrity and non-repudiation for each step in the drug distribution chain using digital certificates.

However, ePedigree does not provide a complete offline certificate validation process. Also, ePedigree has a rapid document growth while propagating through the chain.

2.4 Interactive Graphical User Interface for Version-Aware Documents

Dealing with raw XML documents is not a straightforward task. Nontechnical users cannot read or interpret XML files. Even for technical users, reading large XML files, understanding the structure, and analyzing the data can be problematic [43].

Relatively few research projects have been performed regarding visualization tools to convey a version control system's versioning data. A large portion of this research has been devoted to version control as applied to software development. We think that concepts of VCS visualization apply equally across the software development and XML domains, and have included this software-development focused research in our literature review.

Ball and Eick [44] presented novel visualization techniques for production-scale software systems. Until now, they are some of the most useful visualization techniques due to their scalability and direct mapping of the visual components to the source code and vice-versa. They show different aspects of software code through four types of representation, along with color-coding to show some statistics of interest. The types of representations are:

- Line Representation: maps each text line to a single horizontal line while preserving the line length and indentation of the original text.
- Pixel Representation: This kind of representation maps each line of code to a small number of pixels. An advantage of using this technique over line representation is its compactness for visualizing more information within a single screen. Pixels are color-coded to show a statistic of interest.

- File Summary Representation: maps each file by a rectangle. Each rectangle's height would fall into one of four predefined quartiles of a set size, measured by the number of lines in each file.

- Hierarchical Representation: maps structured data such as file systems (sub-systems and sub-directories) to a tree-map to show statistics of interest.

All of these visualization techniques are used to present software code information in a compact way. They also provide a global overview of the software code that fits on a single screen, along with the ability to navigate from visualization components to the software code easily.

Wu *et al.* [45, 46] suggested a set of requirements for better version control visualization support. They constructed a set of questions using the "5W+2H" pattern, where the 5W and 2H are respectively:

- **W**hat has happened to the program I am collaborating on since the last time I made changes?

- **W**ho was the user that made the change?

- **W**here were the changes made?

- **W**hen did each change in the project take place?

- **W**hy were these changes made?

- **H**ow did each file changed in details?

- What is the **H**istory of a given file?

They implemented a graphical user interface, called Xia, that would satisfy the requirements they identified. Moreover, they evaluated Xia with a small user study and small software projects. We view the small size of their study is a limitation.

In other research, Xie *et al.* [47] relied on existing visualization tools (e.g., cv3D) to answer a set of questions based on data extracted and mined from CVS repositories.

They claimed that their different views would help users answer questions about the version control systems' versioning data. The paper lacks any user studies to demonstrate that users found their tool provided an enhanced user experience.

Ogawa *et al.* [48] presented StarGate, a system for visualizing software repository. Their system solution focuses on the authors of the software rather than the source code. The software project directory structure is represented using a layered space-filling radial hierarchy called the Gate component, where developers are represented as colored circles inside the Gate radial hierarchy, called stars. The size of each star reflects the number of modifications the developer made to the software. Their solution does not have any usability studies to evaluate the user experience using their system. Moreover, the set of survey questions they proposed were limited and developer-centric.

Collberg *et al.* [49] presented Gevol, a system that visualizes version control systems to help to understand the evolution of software. They proposed a new graph-based visualization technique. Gevol answers specific questions regarding the evolution of software programs over time. They have three main graphs, inheritance graphs, call graphs, and control-flow graphs to show relationships and patterns in software projects. We could not find any usability studies, and, they provided a limited scope of questions to be answered about version control systems.

Shrestha *et al.* [50] used a two-dimensional graphic spatio-temporal technique, Storygraph, to present the revision commit time, along with the geographic location of developers, on free and open-source software (FOSS) projects. Their visualization tool composed three axes: two parallel vertical axes representing latitude and longitude for commit locations, and a horizontal axis representing the time of the commit. Their system does provide an interactive interface allowing a user to filter contributions based on selected locations. However, their solution does not show any fine details about collaborators, such as collaborator name and commit city. The lack of a usability study led us to question the qualitative user experience when dealing

with their model. Heller *et al.* [51] represented user profiles and repository metadata from the GitHub VCS to convey information about contributions made to projects, successive commits, and relationships among followers. The authors used existing graphical techniques, such as geo-scatter maps, multiple small displays, and matrix diagrams to aid researchers and developers in inferring coding relationships and social relationships. They theorized that these relationships would allow a user to detect patterns relating geographic distance to social connectivity, developer relationships, and influence among cities. The authors did not conduct any usability studies to support their assertion the techniques they employed made the data represented more accessible to a wider audience of researchers and developers.

On review of this research, we think there is an ill-considered gap in the existing studies. We think that graphical version control tools should have a user-centric design in which user experience is the primary motivator.

Chapter 3

Determining UID length

In Section 2.1, we discussed in detail the background information for version control systems. In this chapter, we will discuss the list of limitations in the current state of the Version-Aware XML documents that motivated us to enhance the Version-Aware XML documents, followed by our solution, and it's evaluation.

3.1 Problem Statement

A unique ID must identify each revision XML element in the Version-Aware XML document where the ID must be collision-free for branching and merging operations. Relying on very long auto-generated IDs to achieve collision resistance without determining the optimal length affects human readability, storage requirements, and the performance of processing these documents.

We performed a mathematical analysis to determine the optimal unique ID length. We will discuss this in detail in the next section.

3.2 Solution and Evaluation

Although labeling XML tags with unique IDs is not a common practice in XML editing tools, Thao and Munson [52] used IDs for XML tags to enhance the performance of XML document merging. In Version-Aware technology, the same technique was used to identify each revision tag. Version-Aware tags used IDs represented by hex-character strings of length 32 to identify and classify revisions as parent-child relationships.

In our integrity and non-Repudiation framework, we adopt the same labeling

technique but have performed additional analysis to determine an optimal ID length. For our framework to be useful to end users, both certificate tag IDs and digest tag IDs must be essentially unique. Since a long-lived document might have many such IDs, we wanted to show that the space required for the IDs would not be burdensome. As a starting point, we have sought to ensure a low probability of collisions among one million (10^6) IDs. We think one million, is reasonable and represents a very low probability of collision while dealing with number of revisions in XML documents.

We analyzed the minimum length of an ID comprised of base64 encoded characters to maintain a 10^{-12} probability of collision with 10^6 IDS. In the equations below, k denotes the maximum number of relevant XML elements in a document, P denotes the maximum acceptable probability of collision, C denotes the number of characters in the encoded text, l_{min} indicates the minimum number of characters in an ID to handle k, and, finally, l_{safe} denotes the minimum length of an ID to achieve P with k IDs.

So we have $l_{min} = \lceil \log_C k \rceil$, when $C = 64$ and $k = 10^6$, we get

$l_{min} = \lceil \log_{64} 10^6 \rceil = \lceil 3.321 \rceil = 4$.

To calculate l_{safe}, let N be the total number of possible IDs, i.e., $l_{safe} = \lceil \log_C N \rceil$. First consider P_k, the probability of selecting k IDs (with replacement, from the N available ones) without collision. Then

$$P_k = \prod_{i=0}^{k-1} \left(1 - \frac{i}{N}\right). \tag{3.1}$$

Using standard approximation for cryptographic hash collisions [53], we get

$$P_k \approx e^{-\frac{k(k-1)}{2N}}.$$

We want the probability of having a collision to be at most P, namely,

$$
\begin{aligned}
& & 1 - P_k &\leq P \\
\text{or} & & 1 - e^{-\frac{k(k-1)}{2N}} &\leq P \\
\text{or} & & e^{-\frac{k(k-1)}{2N}} &\geq 1 - P \\
\text{or} & & -\frac{k(k-1)}{2N} &\geq \ln(1-P) \\
\text{or} & & N &\geq -\frac{k(k-1)}{2\ln(1-P)}.
\end{aligned}
$$

Then

$$l_{safe} = \lceil \log_C N \rceil \geq \left\lceil \log_C \left(-\frac{k(k-1)}{2\ln(1-P)} \right) \right\rceil.$$

Setting $k = 10^6$, $P = 10^{-12}$, $C = 64$, we get

$$l_{safe} \geq \lceil 13.121045472 \rceil = 14.$$

From Equation 3.1, we found that the minimum number of characters for unique IDs is fourteen characters in length (using a 64-value alphabet) with probability 10^{-12} of collision should be sufficient for this purpose.

Chapter 4

Enhancing the Automated Policy Enforcement eXchange (APEX)

In Section 2.2, we discussed in detail the background information for version control systems and the APEX framework. In this chapter, we will discuss the list of limitations that motivated us to enhance APEX, followed by our solution, and it's evaluation.

4.1 Problem Statement

Electronic documents such as manuals for a car or an airplane, as well as legislative documents and judicial archives, are not frequently changed over time. Deep scanning such documents each time a user needs to conduct *exposure operations* will be time- and resource-consuming. *Exposure operations* are operations that can lead to information leakage. The APEX framework requires a deep scan of the entire document to evaluate which security policy should be enforced while users or processes are executing *exposure operations* on the document or its metadata. The time spent to deep scan the document's content and its metadata will be proportional to the size of the document and its metadata. For example, suppose that, in an environment that deployed the APEX framework, a user is trying to use many *exposure operations* on a document that has half million characters or more. Each time the user initiates an *exposure operation*, APEX has to scan the entire document to determine which security policy should be used. In contrast, if an improved APEX only had to scan a few small deltas, the cost of actions may be substantially reduced.

To avoid conducting a deep scan every time a user makes any *exposure operation*

on a document, we developed a solution that only requires deep scans after changes in security policies. Our solution will mainly take advantage of the Version-Aware technology and a memory cache concept for this purpose.

Throughout the rest of this dissertation, we will refer to HP's APEX as APEX, and our solution as eAPEX. We will discuss eAPEX in detail in the next Section.

4.2 Solution and Evaluation

Our solution (eAPEX) relies mainly on three elements:

1. Integration of Version-Aware XML document technology.
2. The assignment of UIDs to each document element
3. A secondary database mapping documents IDs to security policy data. This database functions as a *cache* for security policy data. The database will be added to the trusted local policy store. The logical definition of the table is as follows:
    - Table primary key: The document ID.
    - One column that stores the policy update(PU) ID used during the latest session.
    - Multiple columns to store the most recent rules evaluated at the end of each session.

Throughout this subsection, we will use the term *session* to denote the period where users open a document, work on it, and then close it.

The enhanced processing steps will be:

- Every time a user asks for a new session on a document whose ID is X, the daemon will look up X in the secondary database. If a record for X exists, the daemon will fetch X's policy data and the process proceeds to the next step. If a record for X does not exist, then we have a *cache miss* and the daemon will add the ID element to the document, deep scan the whole document, evaluate the policy rules, and then store the rules in the database.

- If a database entry for document X was found at the beginning of a session, the daemon will evaluate whether the last PU ID from the secondary database is equal to the last PU ID from the local client's policy store. If they are equal, then we have a *cache hit* and the daemon will use the latest evaluated rules to enforce the document policy. Alternatively, if the two PU IDs are not equal, then we have a *cache miss* and the daemon will deep scan the whole document, and evaluate the rules corresponding to the newest PU and then store them in the database.

- When a user is editing the document and invokes an *exposure operation* during a session, we have two different approaches that the daemon will use to evaluate the security policies for the changes made to the document. The first approach is an eager approach where the daemon will scan each changed paragraph as soon as the user finishes editing it. Once a user initialize an *exposure operation*, the daemon will rely on the evaluated security policies to determine the actions in response to the *exposure operations*. The other approach is lazy. In this approach, every time the user finishes editing a paragraph, we mark the paragraph as having changed. When the user initiates an *exposure operation*, the daemon will scan the marked paragraphs, and evaluate the security policies for them. After completing the scan, the daemon will clear the marks.

  Both approaches utilize paragraph IDs and the deltas that represents the edit-

ing changes to evaluate the rules and update the corresponding values in the database at the end of each session.

- The added document ID element has a read-only privilege. Only the daemon has the sufficient privilege to modify or delete this element to prevent any alteration of this element intentionally or unintentionally. Moreover, if the document is permitted to be disclosed, the daemon will obliterate the document ID element in the disclosed copy of the document. The reason for this obliteration is to make sure that the disclosed document will be treated as a new document(if it is related to the new system) due to the possibility that sensitive data may have been added to it outside the APEX framework.

We analyzed the expected performance of APEX and eAPEX. In the equations below, T denotes the total time that will be spent through different sessions to evaluate and enforce the security policy, C_{ds} denotes the cost in time to perform deep scanning the document; C_{δ} denotes the cost in time of scanning the intrasession modified paragraphs of the document, C_{Para} denotes the time to scan one modified paragraph in the document, $N_{session}$ indicates the number of sessions a user perform on the document, $Cache_{miss}$ indicates a cache miss, $Cache_{hit}$ indicates a cache hit, N_{para} indicates the number of paragraphs changed per session, and finally, $EOPS$ denotes the number of *exposure operations* conducted per session.

$$T_{APEX} = \sum_{j=1}^{N_{session}} \sum_{i=1}^{EOPS_j} C_{ds_i}$$

$$T_{eAPEX_{Lazy}} = \sum_{j=1}^{N_{session}} \sum_{i=1}^{EOPS_j} (Cache_{miss_i} \cdot C_{ds_i} + Cache_{hit_i} \cdot C_{\delta_i})$$

$$T_{eAPEX_{Eager}} = \sum_{j=1}^{N_{session}} \sum_{i=1}^{N_{para_j}} (Cache_{miss_i} \cdot C_{ds_i} + Cache_{hit_i} \cdot C_{Para_i})$$

An example that shows the impact of eAPEX solution would be when a user is working on a new large document (500,000 characters), the user opened the document ten times a day and made 5 paragraphs changes to the document without adding or deleting any sensitive data, then tried to print and share the manual with other colleagues. During the 10 sessions, no policy changed happens. The total time will be spend using the APEX, and the eAPEX approaches will be as follows:

$$T_{APEX} = C_{ds} \cdot 2 \cdot 10 = 20 \cdot C_{ds}$$

$$T_{eAPEX_{Lazy}} = 19 \cdot C_{\delta}$$

$$T_{eAPEX_{Eager}} = 49 \cdot C_{Para}$$

We assume $C_{Para} \ll C_{\delta} \ll C_{ds}$. So $T_{APEX} \gg T_{eAPEX_{Lazy}}$ and also $\gg T_{eAPEX_{Eager}}$. Thus, for this constructed example, since the number of paragraphs in the document is large, then eAPEX reduced the time needed to determine the security policies by almost 20 times.

To see the effect of our enhanced solution in terms of time per `exposure operation`, we analyzed the expected performance of APEX and eAPEX. In the equations below,$T_{perExpo}$ denotes the time that will be spent per each `exposure operation`, C_{Para} denotes the cost in time performing deep scanning for one modified paragraph in the document, C_{δ} denotes the cost in time performing deep scanning the intrasession modified paragraphs of the document, and C_{ds} denotes the cost in time performing deep scanning the document. The time will be spend per each exposure operation in APEX, and the eAPEX approaches will be as follows:

$$T_{perExpo_{APEX}} = C_{ds}$$

$$T_{perExpo_{eAPEX_{Lazy}}} = Cache_{hit} \cdot C_{\delta} + Cache_{miss} \cdot C_{ds}$$

$$T_{perExpo_{eAPEX_{Eager}}} \leq Cache_{hit} \cdot C_{Para} + Cache_{miss} \cdot C_{ds}$$

We assume $C_{Para} \ll C_{\delta} \ll C_{ds}$. So $T_{perExpo_{APEX}} \gg T_{perExpo_{eAPEX_{Lazy}}}$ and also $C_{ds} \gg T_{perExpo_{eAPEX_{Eager}}}$. When the security policy changes ($Cache_{miss}$), eAPEX and APEX should have similar performance at the next exposure operation, because a deep scan is required.

4.2.0.1 Evaluation:

To evaluate the performance of eAPEX, we reached out to HP to gain access to APEX, but were unsuccessful. Instead, we implemented an application that contains the minimum essential components of APEX [7] that we needed to make a testbed of our solution and measure its performance.

To compare our enhanced solution to APEX, we needed to take advantage of APEX's limitations as established by our analysis. Therefore, we sought a large text file and found one in "Jargon File" that was obtained from [54]. The "Jargon File" is a glossary specialized in the slang of the programmers. The first "Jargon File" was a compilation of terms used by MIT AI Lab, the Stanford Artificial Intelligence Laboratory (SAIL), and others. From the "Jargon File" we extracted 404,801 characters, satisfying our understanding of large. In addition, the "Jargon File" undergoes many revisions that change a small percentage of the elements, making it an ideal candidate for the improvements provided by our proposed solution.

The "JARGON FILE" was obtained as text. The text document was converted to an XML format with each paragraph wrapped in a < P > element containing a unique ID. Evaluations were conducted on a Dell OPTIPLEX running Windows 10 with 3.3 GHz Core i5 processor and 8 GB of RAM. Our implementation was written in C# using Windows Forms, and used the database engine MariaDB. The StopWatch class available in the .NET diagnostics package allowed us to measure elapsed time for each *exposure operation* and evaluate it.

For searching we chose the Aho-Corasick text search algorithm [55]. We made this decision because the Aho-Corasick algorithm can be run in linear time $\mathcal{O}(n+m+z)$,

where n is the length of the text, m is the sum of characters in the keyword, and z is the number of occurrences of the keyword.

The Aho-Corasick algorithm is implemented in two parts. First, constructing a Trie with keywords from the policies, and the failure function for each node in the Trie. A Trie is an ordered tree Abstract Data Type (ADT) that is used to store keywords. The second part is using the given text to traverse the Trie and identify any matching keywords.

When testing both the lazy and the eager approaches, we set up test cases to measure the amount of time taken to apply an exposure operation based on a percentage of paragraphs changed. Test cases applied changes to 1% of the paragraphs (10 paragraphs), 5% of the paragraphs (50 paragraphs), 10% of the paragraphs (100 paragraphs), 20% of the paragraphs (200 paragraphs), 40% percent of the paragraphs (400 paragraphs), 60% of the paragraphs (600 paragraphs), 80% of the paragraphs (800 paragraphs), and 100% of the paragraphs (1000 paragraphs). The paragraphs selected to be changed were chosen randomly using a uniform distribution among other paragraphs. Each test case was run 10 times and a mean was calculated.

Figure 4.1 plots the average elapsed time per exposure operation using the lazy approach against the percentage of paragraphs changed, while Figure 4.2 plots the same data set achieved with the eager approach.

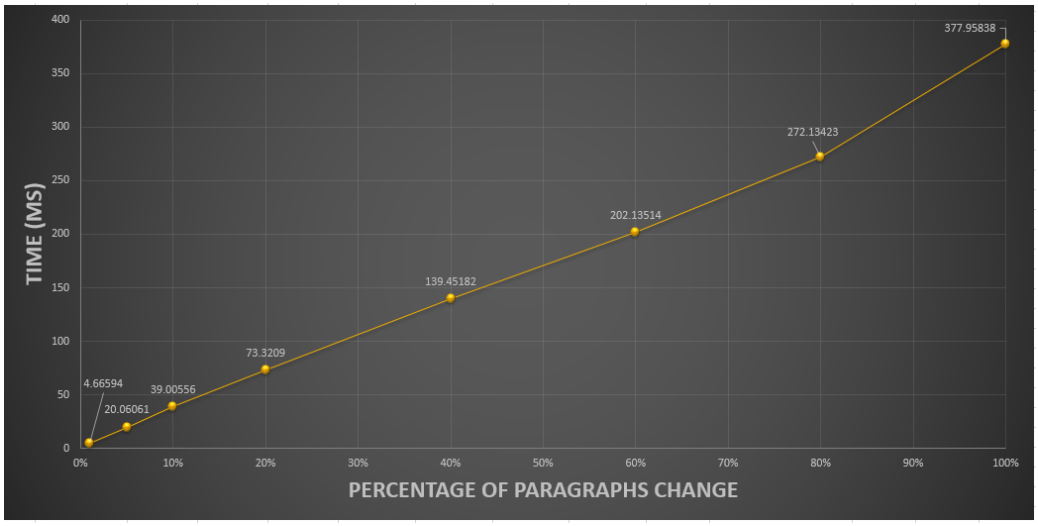


Figure 4.1: Lazy Approach: Average Elapsed Time Per Exposure Operation

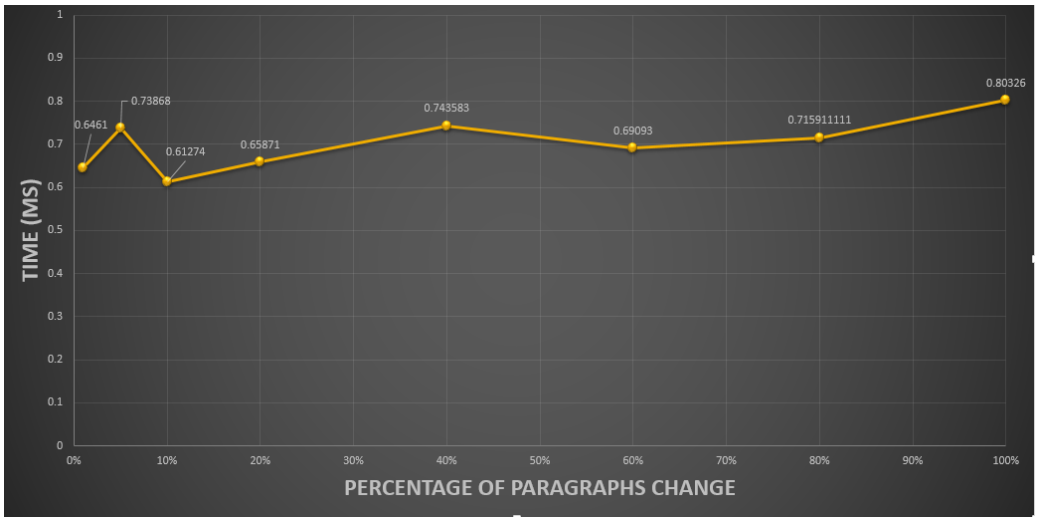


Figure 4.2: Eager Approach: Average Elapsed Time Per Exposure Operation

As one might expect, the time our lazy approach solution spent evaluating the policies is linearly correlated to the number of paragraphs changes associated with an exposure operation. The nature of the document, however, plays an essential factor in this approach: If the users are working on a stable document and are making minimal changes between each exposure operation, this approach would work better than APEX's initial approach.

On the other hand, our eager approach shows that whenever a user executes an *exposure operation* without having any policy changes, the elapsed time between execution and exposure will be no worse than scanning one paragraph. The eager approach has inherent inefficiencies due to the human nature involved in the editing process. A user may rewrite a sentence in a paragraph 10 times before being satisfied with it. Assuming the user left the paragraph and returned back to it 10 times. For this one sentence, the eager approach will perform 10 commits, hence, 10 times scanning the paragraph. On the other hand, the lazy approach will only perform 1 after a user execute an *exposure operation*.

Figure 4.3 demonstrates a series of user-initiated document events plotting against the cumulative time spent by the system to execute all events.

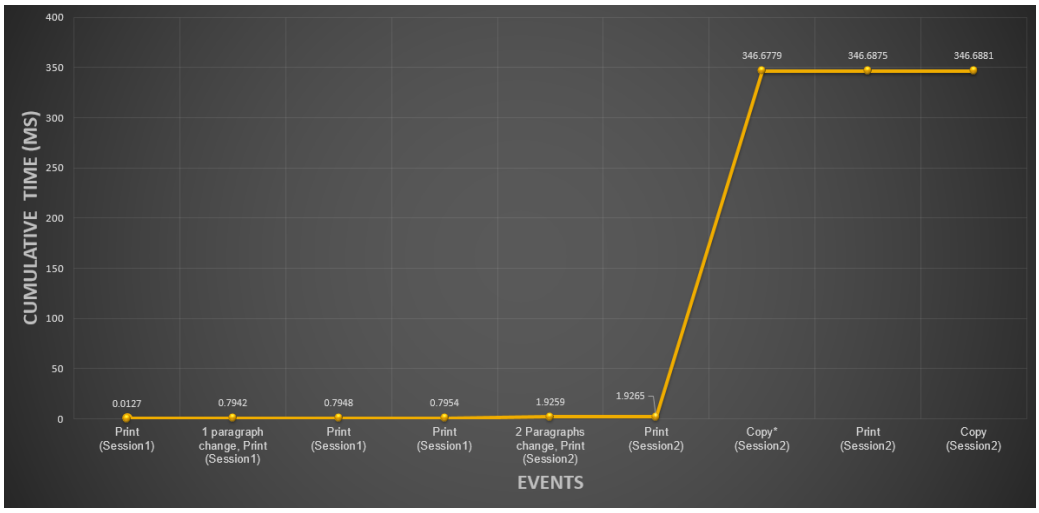


Figure 4.3: eAPEX Cumulative Elapsed Time for two sessions with Multiple Edits and Exposure Operations

In a new session, the first, the third, and the fourth events shows the user request to print a document that has been scanned before without any change in the security policies. Our enhancement rapidly determines whether to allow or deny the print request according to the security policy. Within the same session, the second event as the user changing one paragraph and requesting to print the document. Our enhancement will spend time to reevaluate the policy, but that time is low relative to the total amount of time the system will spend using the original APEX implementation since we are scanning a small portion of the document paragraphs.

For the fifth event, the user has exited the previous session (session 1) and opened a new session(session 2) to the document, followed by a request to print the document. The decision will be made instantly because no policies changed since the last session.

The seventh event(the event with asterisk) had a security policy change followed by a user event to copy the document and thus requires a deep scan. For this exposure operation, our solution has the same performance and behavior as original APEX.

We have created a simulated APEX line as shown in Figure 4.4 in order to make a comparison of our approaches behavior to that of the original APEX. Note the increase in time regardless of any change of document content.

Figure 4.4: APEX Cumulative Elapsed Time for two sessions with Multiple Edits and Exposure Operations

The test results allow us to conclude that both of our approaches improve on APEX's running time (Figure 4.4) when a user executes an inter- and intra-session exposure operation. As long as the security policy remains static, our solution is expected to maintain its efficiency (Figure 4.3). In situations where the security policy has changed, our solution is expected to have the same performance and behavior as APEX, due to the necessity of running a deep scan on the document's contents and metadata.

Chapter 5

Maintaining integrity and non-repudiation in secure offline documents

In Sections 2.1 and 2.3, we discussed in detail the background information for version control systems and for maintaining integrity and non-repudiation in documents. In this chapter, we will discuss the list of limitations in literature that motivated us to build a framework for maintaining integrity and non-repudiation in secure offline documents, followed by our solution, and it's evaluation.

5.1 Problem Statement

Since XML-based Electronic Health Records (EHR) being an important application of document engineering and the integrity of medical records being so important throughout their life-cycle, we chose EHR as one of our case studies. EHRs are used by many health care providers [56].

EHR systems store sensitive patient data such as medical history, medication, and immunization records. These records can be shared with many different health institutions where medical staff read and update them.

EHR documents may be shared offline via flash drives or emails. The integrity of medical records is important throughout their life-cycle. However, offline sharing of EHRs can cause many problems, especially when an unknown person modifies an offline version in an unknown way. EHRs are a digital version of patients' medical history used by health professionals, to improve efficiency by avoiding paperwork and providing easier sharing of patients' health records between health care providers, and to reduce the costs to manage this data.

SmartCare [57, 58] is a distributed XML-based electronic health records system developed for adoption in countries where both internet and electricity are limited and unreliable, such as Zambia and Nigeria. Failing to prove the integrity of these medical records could lead to life-threatening consequences if unauthorized persons tamper with the data. Failing to identify who collaborated on these documents prevents intentional and unintentional false medical information from being connected to the authorized health professionals who altered it.

For an example of non-repudiation failure using SmartCare, conceder the following scenario. A doctor was supposed to prescribe a particular medication to a patient, but the medical records show a different medication. Later on, the patient showed signs of potentially fatal complications after taking the drug prescribed in the medical records, but the doctor denied ordering the drug in the record for the patient. Because the patient medical records were shared in a distributed fashion (offline), we do not have answers to the following questions:

1. Were the modifications on the patient's medical records made by the doctor?

2. What changes did the doctor make to the patient health record? Moreover, can we prove that these changes were made by the doctor himself?

3. What was the sequence of changes made by the doctor to the patient health record? in addition, do we have non-repudiable evidence of what we know?

   Question 1 addresses integrity, Questions 2 and 3 addresses non-repudiation.

Another example that shows the importance of our proposed solution would be MS Word. The MS Word application is the most popular editor that people rely on to collaborate authoring documents that might be passed among a series of collaborating authors performing different changes in parallel, in an environment where it is important to be sure that only known users are making changes. Relying on MS Word's signature feature would limit each known user to verify the signature of the previous version only. MS Word neither supports sophisticated version control

functinalities nor provides users with the ability to keep signing a document multiple times without removing the previous signature.

In a broader scope, if documents are shared in a distributed fashion (offline), can we answer the below questions:

1. If the documents have been changed, were these modifications made by authorized users?

2. What changes did authorized users make to the documents? Moreover, can we prove it?

3. What was the sequence of changes made to the documents? in addition, do we have non-repudiable evidence of what we know?

Our proposed solution answers these questions by introducing a new security framework for XML-based documents that preserve XML data integrity and non-repudiation for the entire document reversion history. The framework will support the offline sharing of documents among collaborating authors by using Version-Aware XML document technology [17, 6] to store reversion information inside the XML based document itself. We will use digital certificates in order to verify the integrity of the XML document and to ensure the non-repudiation of changes and authorship. Specifically, our framework will enable users to answer the following questions:

1. Has the XML document been modified?

2. Were the modifications made by known users?

3. What changes the known users made to the document? Moreover, can we prove it?

4. For each change made to the document, which known user made the change, what computer was used to make the change, and at what time the change made with non-repudiable certainty?

We will discuss our solution in detail in the next Section.

5.2 Solution and Evaluation

The current solution set for offline XML-based documents lack fundamental protection for maintaining integrity and non-repudiation. Our solution is to build a secure framework for XML documents that will preserve data integrity and non-repudiation when electronic documents are shared offline between both authorized and unauthorized users. In theory, our framework applies to any kind of XML-based application that allows adding versioning data inside files. For example, our framework could be used in educational environments where a teacher asks students to edit a document collaboratively, or in legislative and governance domains where multiple people collaborate to write or revise legislative documents.

Our framework uses both Version-Aware XML technology and Digital Signature [59].

Our proposed solution is designed to work on any XML-based document. Having identified two affected areas as medical records and MS Word documents, we evaluated our solution on simulated medical records and collaborative MS Word documents. Due to the lack of access to the SmartCare medical health system, we built an XML editor that would mimic the work of other XML-based medical software. As shown in Figure 5.1, users can download our XML editor, which means that they had to go online at least once to get both the XML editor and their certificates.

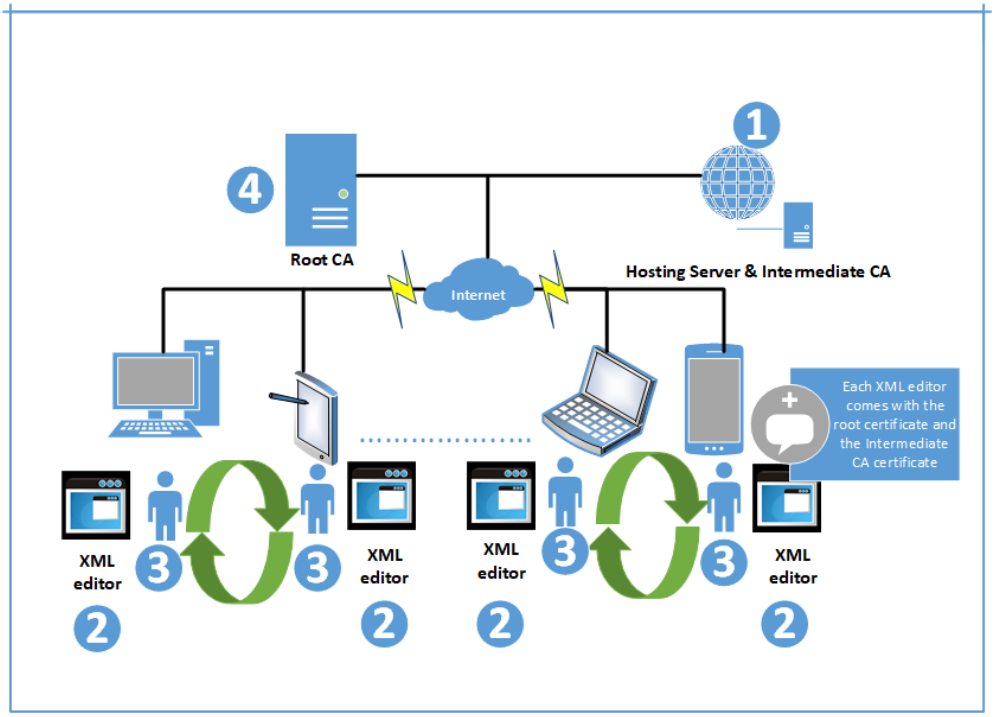


Figure 5.1: Initial setup to download the XML editor along with obtaining the end user's certificate signed by the intermediate Certificate Authority (CA) through secure channel. While the curved arrows show how end users could share the files offline, where the thunder sign represents a network connection for the initial setup and not necessary reliable afterwards.

Figure 5.1 illustrates our framework where the key entities are marked with numbers. The first time that an end user (denoted by 3) wants to start collaborating on a document using our framework, that user must get access to our XML editor. We currently use an *application host* (1) to provide starter documents that have the plug-in embedded. The application host also acts as an intermediate certificate authority (CA) that is verified by a very well known root CA (e.g, Verisign). The user can provide and confirm some identifying information (i.e. name, email address, cellphone number, etc.) in order to become a known user and receive a personal certificate that will be used to sign documents. In addition to the plug-in, the documents also contain the certificates of the root CA (4) and the intermediate CA (1), which were used in the chain validating the end user's certificate. The end users (3) are shown sharing the documents (2) that they are collaborating on, which they do using our XML editor. The XML editor helps the authors by automatically signing

the document each time it is committed. The authors can then share the document with their collaborators and the person who is in charge to generate the final version of the document; knowing that everyone can verify the document's integrity.

When any user creates/changes and then signs a document, the following integrity XML tags and data will be added in the XML file:

- The current editor's certificate will be embedded in the XML file inside the certificate's XML tag.
- Version-Aware data will be embedded in the XML file.
- A signed digest for the entire XML file will be embedded in the XML file along with the certificate ID that has been used for signing.

Our framework then checks for both integrity and non-repudiation by using the information stored inside the SigAware element as shown in Figure 5.2.

Figure 5.2: A sample Health Level 7 (HL7) Clinical Document Architecture (CDA) document signed by two different users. Box (1) represents the entire integrity and Version-Aware document with seguro and molhado namespaces defined in Boxes (2) and (3). Box (4) contains the integrity data. Both user's certificate along with latest document signature by user's private key. Box (5) represents version data. Box (6) contains the HL7 CDA document content (latest version). Users' certificates are only shown partially in the figure due to lack of space.

Our XML editor was built as a Windows application using C# Windows Forms. That editor edits, validates, and verifies XML documents, using our framework to preserve integrity and non-repudiation. We tested multiple XML documents to check the integrity of documents shared by other users; we also validated and verified the certificates while importing both users' certificates and inline stored certificates.

Our XML editor has the capability to import and verify the user's digital certificate and to verify the integrity of the document when loaded. When the user is done editing the document, the user is able to either sign the document through the menu or just close the XML editor, in which case the user will be notified to sign the

document before exit. Both integrity data and the user's certificate will be added to the document after it has been signed. If a user opens an XML document that has been changed without a legitimate signature being found, the user will be notified of possible tampering. Also, if a user tries to open a file that does not contain any integrity-aware data (signatures and certificate) the user will be informed by a message that he/she will be responsible and will take responsibility for the content of the document if he/she signs it.

5.2.1 Evaluation Results:

5.2.1.1 Evaluation 1: XML Simulation of EHR

We tested our framework with the following scenarios:

- Integrity Scenario: We used a Clinical Document Architecture (CDA) document XML file from Farfan*et al.* [60]. This scenario involved a user, User A, editing a fresh copy of the file. After all edits were in place, A signed the document with his certificate. We obtained the certificate for A from the intermediate CA. A then shared the document with another user, User B. Upon receiving the document, B used our system to verify the integrity of the document using User A's certificate.

  User B altered the XML document and decided to use a new certificate that was not issued by our intermediate CA. After signing the document, B sent the document to a third user, User C. Using our framework, C checked the file and found out that the certificate used to sign the document was not legitimate. Our framework reports all such incidents to the legitimate user C who requested validation of the file.

- Non-repudiation Scenario: Using the same CDA document XML file, we considered the case where the most recent document change was made and signed by a User D, but at some later time, D denies making the change. The validation

framework rejects this attempt to repudiate changes, because the certificate associated with the change was authenticated by the CA.

Other scenarios we tested involved an attempt by a malicious anonymous user to tamper with User A's file contents. Our framework detected such tampering attempts and notified the user who received the tampered file.

5.2.1.2 Evaluation 2: Version-Aware Word Plug-in

Our second evaluation was done by extending Coakley *et al.*'s Version Aware Word Document plug-in [6] to add our security features. The plug-in, implemented by Coakley *et al.*, maintains a version history by saving document snapshots in a new sub-folder of the MS Word file (actually, a zipped folder) called "history" without securing the integrity of these snapshots and the document content. This sub-folder contains an index file called "revision-history.xml" that points to further sub-folders named with unique IDs. The plug-in is implemented in C#. We found, as did Coakley *et al.*, that Word's plug-in API was challenging to work with.

Our extension modified Coakley *et al.*'s Version Aware Word Document plug-in to support both document integrity and non-repudiation, along with storing more end-user information to be used for forensics (i.e., IP address, MAC address, etc.).

Figure 5.3 illustrates our framework used on top of MS word, in which key entities are marked with numbers similar to Figure 5.1. The first time that an end user (denoted by 3) wants to start working with our system, that user must get access to our plug-in (denoted by 4). We currently use an *application host* (denoted by 1) to provide starter MS documents (denoted by 2) that have the plug-in embedded and the actual document contents (denoted by 5). The application host also acts as an intermediate certificate authority (CA) that is verified by a very well known root CA (e.g Verisign), to which the user can provide and confirm some identifying information (i.e. name, email address, cellphone number, etc.) in order to become a known user and receive a personal certificate that will be used to sign documents. In

addition to the plug-in, the documents also contain the certificates of the root CA (denoted by 6) and the intermediate CA (1), which were used in the chain validating the end user's certificate. The end users (3) are shown sharing the documents (2) that they are collaborating on, which they do using the standard Word application. The plug-in helps the authors by automatically signing the document each time it is committed. The authors can then share the document with the newly committed changes with their collaborators and the person who is in charge to generate the final version of the document; knowing that everyone can verify the document's integrity.

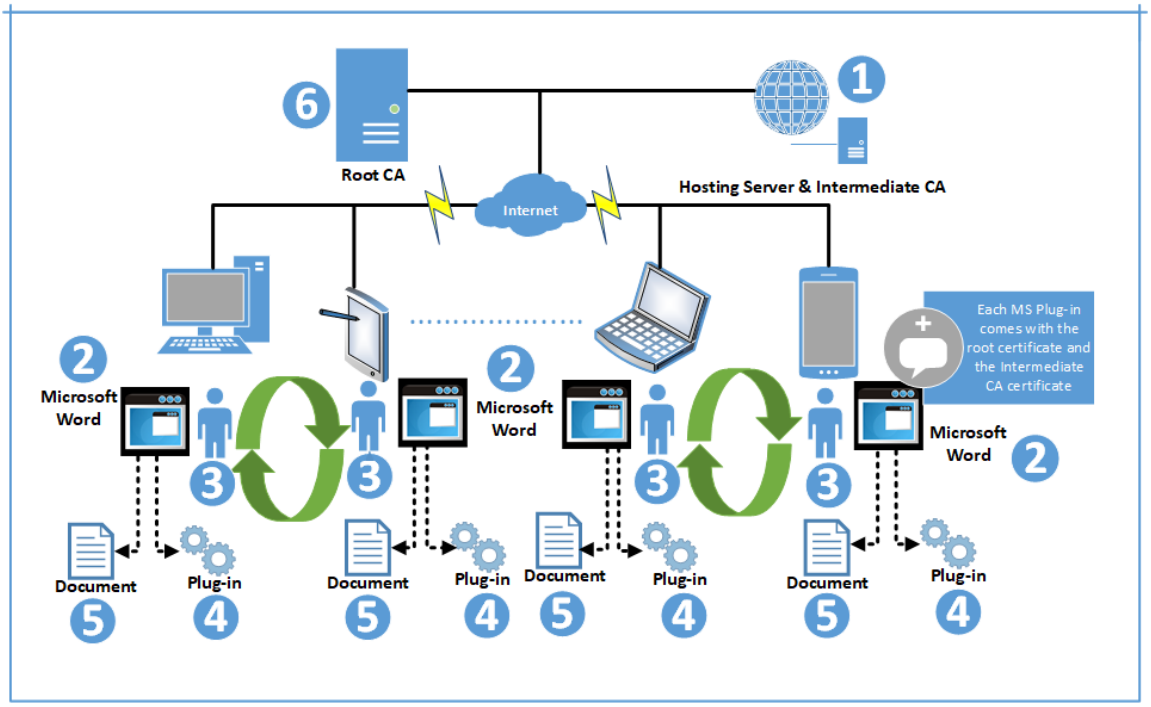


Figure 5.3: Initial setup to download the Word secure Version-Aware document along with obtaining the end user's certificate signed by the intermediate Certificate Authority (CA) over a secure channel. The curved arrows show that end users could share the files offline, and the lightning symbol represents a network connection present for the initial setup but not necessary reliable afterwards.

In this framework, the only times that end users must go online are when they first need to get a document containing the plug-in and whenever they need to obtain a new certificate. The former case could be a one-time only event, but the latter case might have to occur periodically since certificates can expire or be revoked.

After downloading the Word document, end users may modify and share the

Word document offline or via online services. When a user commits changes to the document, the following integrity XML tags and data are added in the revision-history.xml file. These features will be embedded in the revision-history.xml as shown in Figure 5.4:

(a) Version-Aware data.

(b) The current author's certificate.

(c) A signed digest for the whole document along with the certificate ID that has been used for signing.

(d) Other forensic information will be attached to the current revision to identify the user who made the revision.

Our framework then checks for both integrity and non-repudiation by using the information stored in the revision-history XML file inside the Word document as shown in Figure 5.4.

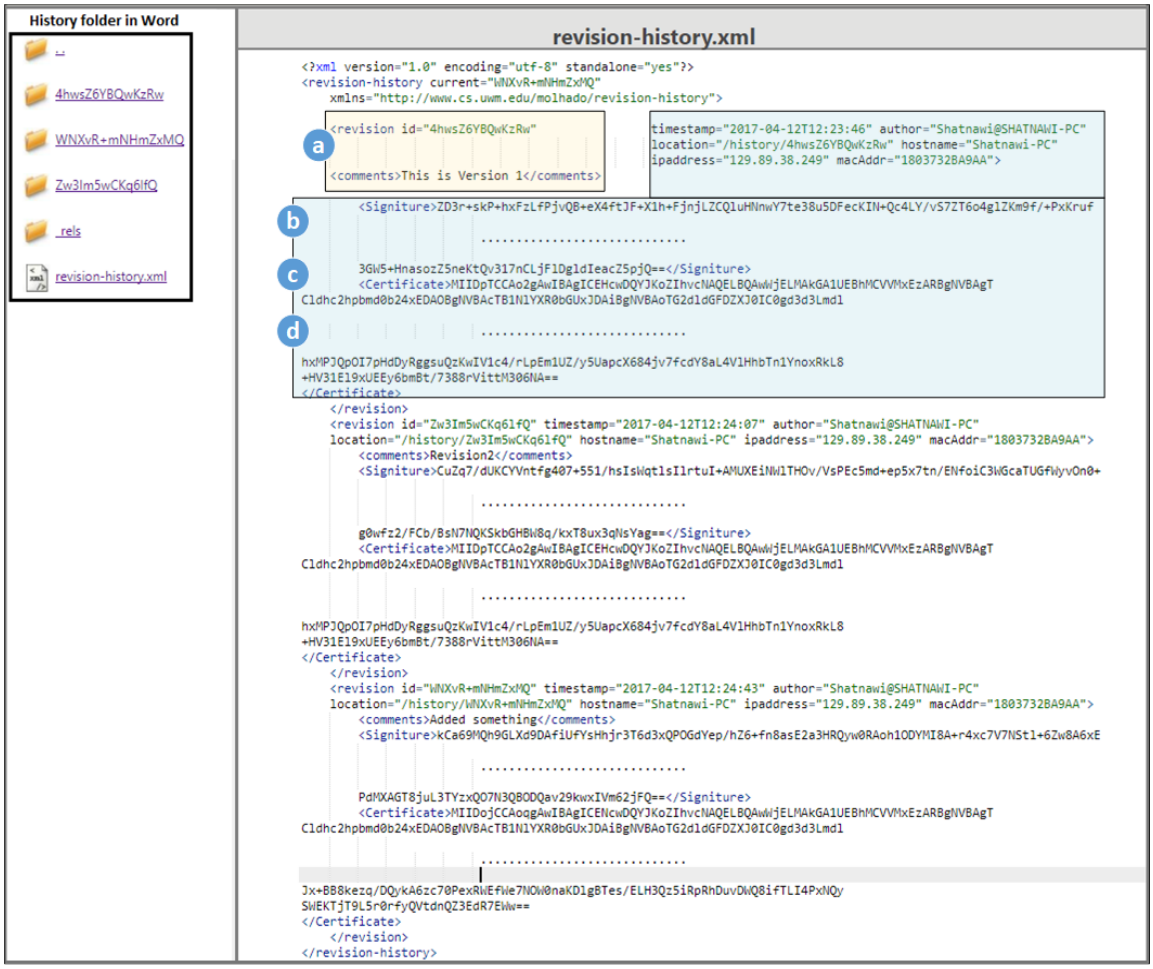


Figure 5.4: A sample Background Check Report word document signed by three different users. The lines of asterisks represents elided parts of the certificate and forensic data.

Although our framework does not prevent illegal tampering by altering or injecting data in the Word document, it will detect any modifications made by unknown users in the Word file, even including style changes.

Our extended plug-in adds the capability to import the user's digital certificate, verify the user's certificate and verify the integrity of the document when loaded. When the user is done editing the document, the user has the ability to sign the document. Signatures are persistent and the document holds the signatures for every revision made. This is different from Word's own document signing feature, which removes all previous signatures when a new collaborator signs the document.

We performed a range of tests using a document that records a background check

written with MS Word. We chose this particular document because it is a natural example of a document that might be passed among a series of collaborating authors performing different parts of an employee background check, in an environment where it is important to be sure that only known users are making changes. Our use case scenarios fall into two categories:

1. Integrity based: Testing whether our framework preserves the integrity of documents and their revisions while shared among multiple users.

2. Non-repudiation based: Testing whether our framework provides a non-repudiable evidence of which users made which changes.

We tested many scenarios, focusing on the questions we posed earlier and we verified that our proposed system is able to:

1. Detect whether or not a shared file been modified.

2. Determine whether each modification was made by a known user.

3. What exact changes the known users made on top of a document along with a solid proof which user made which change.

4. Determine the author, the computer making the change, and the time for each change made.

One interesting integrity-based user case scenario we conducted shown in Figure 5.5, involved a collaboration among four users: A, B, C, and D. If the four users used their valid certificates using our plug-in, any other known users were able to see the exact changes made by the users, verify the integrity of their revisions, and see proof of what they changed in the document. However, let us suppose that when User D loads the document, the system says that the changes apparently made by C are not signed by a valid certificate. In this case, D knows that those changes are not trustworthy. At this point, D can either reject the document or D can try to

validate the changes by a secondary pathway (e.g. directly contacting C for a new, valid document), or D can reject the changes and work from the valid revision signed by B.

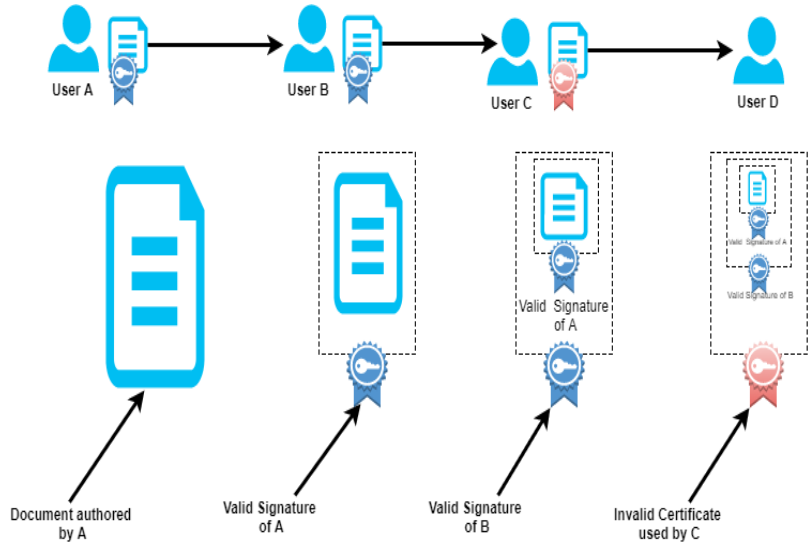


Figure 5.5: Integrity based scenario 1

Another integrity-based use case scenario we conducted, as shown in Figure 5.6 involved a malicious attempt by an anonymous user who tampered with User C's file contents. Our framework detected such tampering attempts and notified Use D who received the tampered file that the document contents have been tapered with in an illegal way. Our framework also detects when a user has not relied on our plug-in for editing.

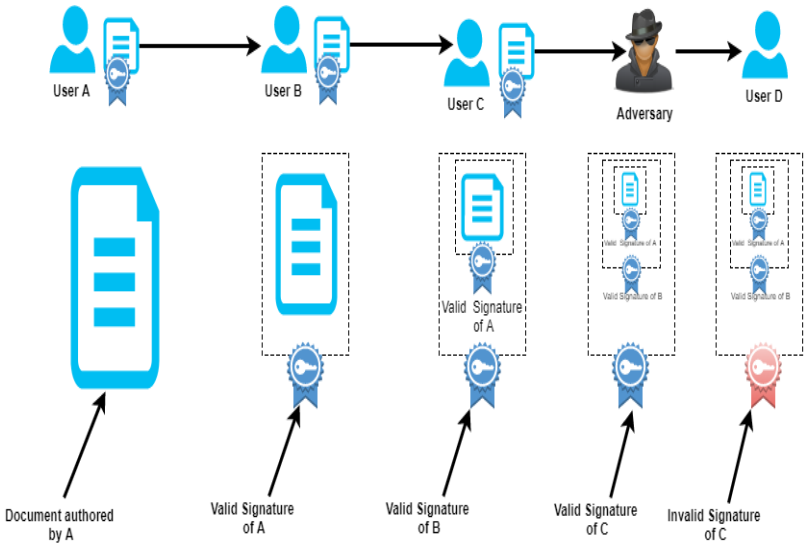


Figure 5.6: Integrity based scenario 2

Figure 5.7 shows a non-repudiation based user case scenario involved a collaboration among four users: A, B, C, and D. If all the four users have used their valid certificates using our plug-in, any other known users are able to have a non-repudiable proof of the actions A, B, C, and D have made. However, let us suppose that when User D got the document, User B denied his role editing the document and claimed someone else made the changes. In this case, because User B's certificate was valid, User D has a non-repudiable evidence that User B made certain changes to the document.

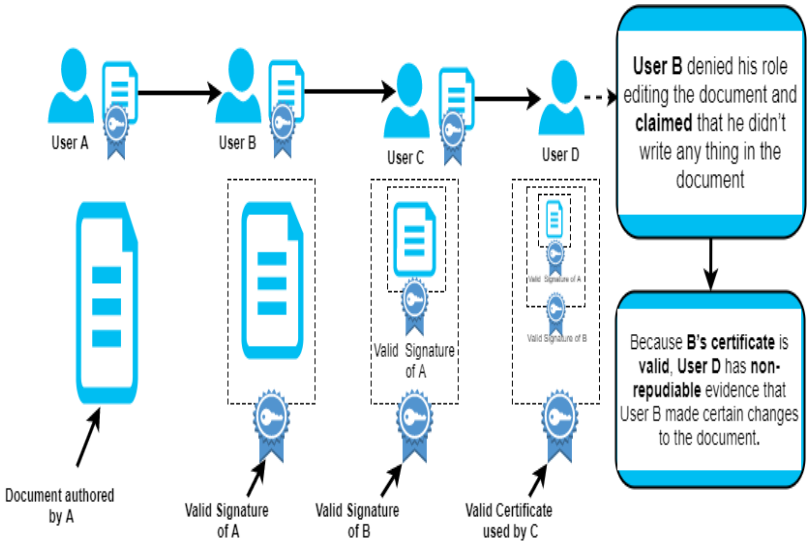


Figure 5.7: Non-repudiation based scenario

Reverse engineering, such as re-saving a document without a signature and determining the signature by subtraction has been considered in our proposed framework. Since each user will share his committed document with all the collaborators, any user can check whether any information has been subtracted by comparing it with other signed documents If a signature does not appear in a document, but appears in other documents, then the signature must have been removed from that document. We did not implement the feature to check for this type of problem automatically, but it could easily done in the future.

As we have discussed earlier in Chapter 2, ePedigree is an infrastructure that preserves both integrity and non repudiation over drug/food chains in the distribution flow. A significant disadvantage of adopting ePedigree is the rapid document growth while propagating from one part to another through the chain [61]. In our solution, we apply the same level of security without keeping any redundant information in the document. Our framework stores the reverse deltas each time the document has been changed. Whenever a user wants to validate the integrity of all the revisions of the document, our tool will apply the reverse deltas per each revision and verifies the integrity of the information.

Another difference between our proposed solution and ePedigree is that we provide a complete offline certificate validation process by storing all the chain certificates used to sign end users' certificates in our solution application, while ePedigree you have to rely on online services to verify certificate validity.

One more point of difference is that our framework targets more domains in practice for collaboration on a document, while collaborators want to work in a parallel fashion rather than sequentially, as proposed by ePedigree.

A final point of difference is that the three way Merging technique discussed in [20] with Version-Aware XML documents provided us with this efficient mechanism to distribute our XML-based document and let the collaborators work in parallel on the document. In the end, we can merge their documents in a single document to be

the final production. Areal case scenario that is not applicable by ePedigree is when any intermediate chain wants to merge two shipment drugs and send them as a one package, this scenario is definitely applicable using our solution.

Chapter 6

Interactive graphical user interfaces for Version-Aware XML document

In Section 2.4, we discussed in detail the background information for different graphical user interfaces for version control systems. In this chapter, we will discuss the list of gaps and limitations in the literature that motivated us to build an interactive graphical user interface for Version-Aware XML document, followed by our solution, and it's evaluation.

6.1 Problem Statement

Reading, modifying and interpreting raw XML data is not a trivial task, especially for a non-technical user. Version control data contains valuable information that helps users enhance their collaboration process on documents. Because of this, researchers try to present version control data using graphical user interfaces that allow collaborators to better understand the data. We will show that no previous work in the literature provides a sufficient set of visualization services for version control data to meet reasonable requirements. In addition, these systems have no graphical support for necessary operations related to revision deletion.

We established a core set of requirements for a Version-Aware visualization infrastructure. Based on these requirements, we implemented a graphical user interface prototype that allows end users to better understand version control data and enables perform key operations on the revision history.

We will discuss our solution in detail in the next Section.

6.2 Solution and Evaluation

Visualizing all versioning data in a single view is a challenging task as a lot of information has to be seen. This chapter takes on that task via a graphical user interface prototype designed to improve the understanding of version control system data, which in turn could increase the efficiency of collaboration and code quality through these systems.

The interaction design process can be divided into four steps: establishing requirements, designing alternatives, prototyping, and user-centric prototype evaluation. This chapter will focus on the requirements we established, the prototype functionality implemented to address these requirements, methodology chosen for prototype evaluation, and analyze the results of the prototype evaluation to identify areas for future improvement.

At the onset of the chapter, We established a base set of questions that we hypothesized a user should be able answer about versions of XML files via our prototype. Through research and interviews, this set was enhanced and updated until we felt it was satisfactorily complete. The final set is as follows:

1. Who worked on a particular XML file (list of all users who collaborated to edit XML file)?

2. What is the timeline of changes made to a specific Version-Aware XML file?

3. How many users collaborated on an XML document?

4. How many changes were made by each user on a particular Version-Aware XML file?

5. For each revision made by a user, is the integrity verified on the resulting revision?

6. Can we know the revision that has been changed the most?

7. Can we filter the visualized data using graphical controls e.g. slider, check boxes, etc?

8. Can we visualize geographical locations (e.g. map visualization) of the users who made changes on a particular Version-Aware XML file?

9. Can a user delete a revision branch or even clear the whole revision history?

10. What is the structure of XML document tree?

11. Can we identify unusual or special XML elements, such as elements that has been changed the most or least?

Some of the above questions were answered partially through different research papers using different visualization techniques in the literature [45, 47, 48, 49]. Our prototype integrates functionality that addresses all the questions listed above.

Analyzing this question set, we extracted a list of 10 requirements for our prototype, and along with each requirement, the type of requirement (Version Control or Security Related Attributes). Table 4.1 lists each requirement along with its classification.

Table 6.1: Prototype System Requirements

Req. ID	**Requirement**	**Classification**
FR1.SRS.1	The system should be able to determine the contributions users made on a specific document.	Version Control
FR1.SRS.2	The system should be able to display the editable XML document elements in a way that helps users visualize them.	Version Control
FR1.SRS.3	The system should be able to show the XML elements and their lengths along with how many changes have been made to each element	Version Control
FR1.SRS.4	The system should be able to show the revisions made by UserX.	Version Control
FR1.SRS.5	The system should be able to determine the number of changes made by UserX.	Version Control
FR1.SRS.6	The system should be able to validate the integrity of a revision/version of a document.	Security
FR1.SRS.7	The system should be able to display the changes that have been made to a document in a way that helps users to visualize them.	Version Control
FR1.SRS.8	The system should be able to the geographical locations for collaborators map.	Version Control and Security
FR1.SRS.9	The system should be able to display the timeline of revisons for a specific document including number of changes.	Version Control and Security
FR1.SRS.10	The system should be able to to allow users to delete a revision record or to clear the whole revision history.	Version Control

In the design phase, we maintained a goal of keeping the system components as modular as possible. This viewpoint led to 3-tier architecture. Figure 6.1 shows our 3-tier architecture that is consists of a Data Access Tier, a Business Tier, and a Presentation Tier. This architecture allows each tier to be decoupled from the others in the event of a need to adopt the prototype to other VCSs in the future.

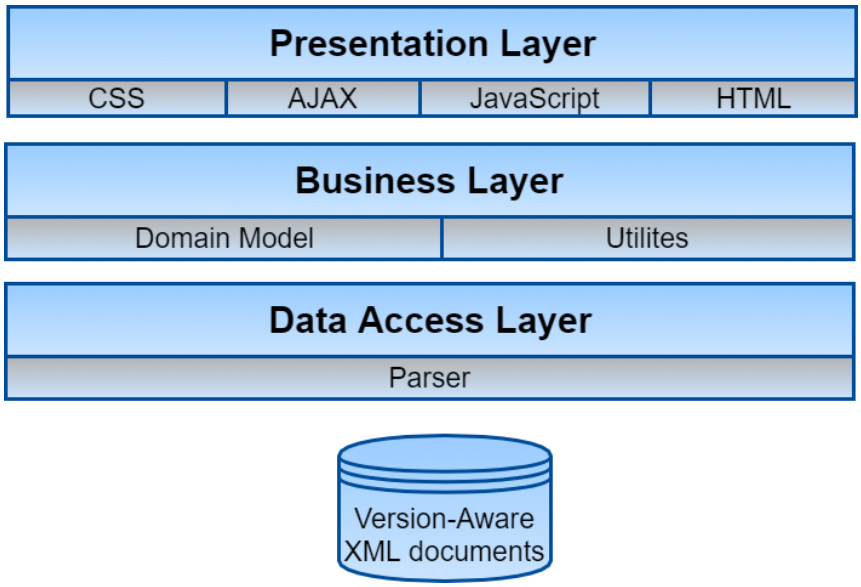


Figure 6.1: Prototype Software Architecture

The Data Access Layer handles the input and parsing of the Version-Aware XML document. This layer is the most loosely coupled layer, allowing extension of our framework to utilize with other types of VCSs. The Business Layer encompasses the logic necessary to create a domain model from the output of the Data Access Layer. This domain model consists of data inherent to XML documents as well as derived data processed or calculated by this layer. At this stage, a variety of utility classes are implemented to analyze, calculate, and cluster the required derived data. For example, given the IP address that submitted a revision, the Business Layer queries an online service to map a longitude and latitude associated with a revision. This data is helpful when the need to visually represent geographical data arises. Finally, the Presentation Layer encompasses user interface components for a web browser front-end. These components include cascading style sheets (CSS), AJAX

calls, JavaScript, and HTML. The Presentation Layer provides a bridge between end-users and the Business Layer, formatting the information for the interface view being accessed.

Contrasting our requirements with the structure of the Version-Aware XML data led to an early realization that the existing data model for the Version-Aware XML revision information needed to be extended. Our extended data model shown in Figure 6.2 added IP address information, MAC address information, the collaborators certificate, and the revision digest.

Figure 6.2: A sample enhanced Version-Aware XMl document with added information needed to meet the requirements.The lines of dots represents elided parts of the certificate and forensic data.

In designing the user interface, we wanted to adhere to the best practices user experience (UX) design. Applying these principles meant that each page must have a similar visual style. In other words, a set color theme, typeface and icon set. This maintains a sense of consistency and cohesion across the system. In addition, icons and buttons should mimic general controls as much as possible. This should increase the intuitiveness of use and greatly decrease the learning curve of the system [62].

There was no part of the prototype where the meaning of an element is dependent solely on its color. So, there will always be at least one other way to identify the

function to avoid problems with people who are color-blind. For instance, on the geographical map, the participants can know how many editors on each geographic place are relying on the code colors or just hovering the mouse over each location for the list of editors.

Error messages across pages abide by Shneiderman's rules [63] in the following ways:

- Error messages do not include negative terms such as FATAL. Instead, informative feedback provided when a user attempts an unsupported action on our prototype.
- Error messages avoided using ALL CAPS
- Error messages are designed to be intuitive to users and refrain from using unnecessary engineering jargon on the main interface. Messages displayed on the views and tooltips are concise and easy to read. In cases where it is necessary for a longer message to be inserted in a tooltip box, the text will be shown in a scroll format to avoid overly large text boxes.

Our prototype provides 8 views: Collaborator Actions, Document Tree, Components, Revision History, Geography, Timeline, Revision Management, and Upload. The views are implemented using Web APIs. On the server side, we used an ASP.NET web application framework backing by C# code-behind. On the client side, we used controls from D3.js [64] and ASP.net for creating interactive interfaces as well as cascading style sheets (CSS) for the look and feel.

Our initial high fidelity prototype design alternative was based on the requirements; we began by selecting the controls we thought were scalable, efficient at conveying information, and easy to use. For example, we chose the d3.js tree diagram consisting of a collapsible tree with auto-sizing to represent an XML document structure that was constructed from interactive parent and children nodes. On a click event, the nodes would collapse it's descendent nodes in the tree and could be

re-clicked to expand the descendant nodes. We discovered that this graphical representation technique was not scalable when applied to displaying large XML files. To solve this, we developed an idea to combine a magnifying glass tool with a flexible force-directed graph layout, allowing us to achieve both scalability, efficiency, and ease of use.

The Collaborator Action View shown in Figure 6.3, addresses requirments 1 and 5:

FR1.SRS.1	The system should be able to determine the contributions users made on a specific document.
FR1.SRS.5	The system should be able to determine the number of changes made by UserX.

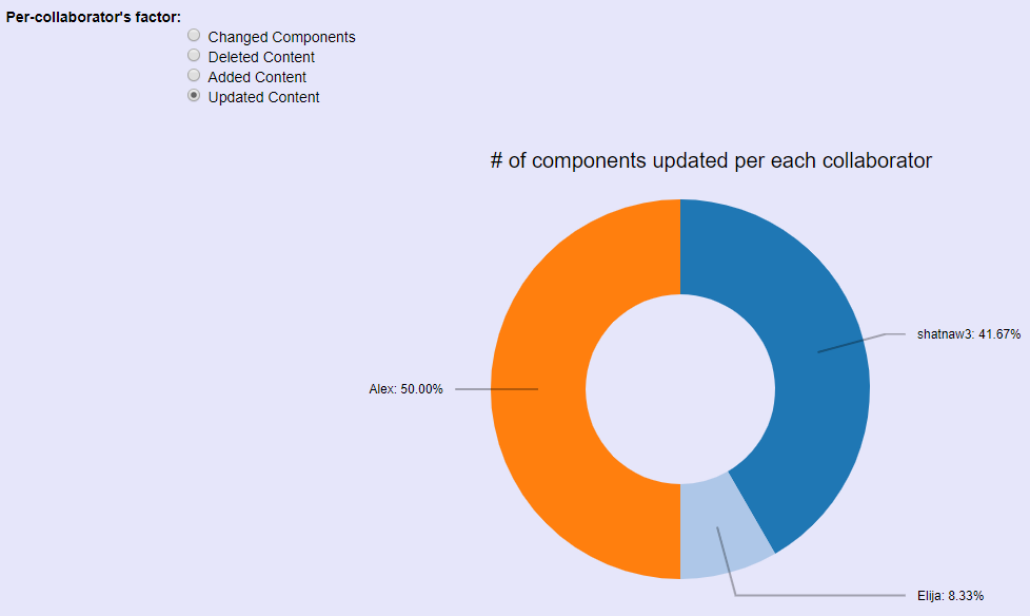


Figure 6.3: Collaborator Action View

The focal point of the Collaborator Actions View is a donut chart denoting contribution numbers associated with each collaborator. The donut chart provides the user with a familiar graphical representation of statistics, allowing for a quick grasp of the presented information. It should be noted that, for the sake of this example, a screenshot of a three-user collaboration was used; however, the system is fully capable of scaling up to a larger number of collaborators.

The Collaborator Actions View uses two features for displaying visual information

related to collaborator contributions. The first is a filtering mechanism in the upper left corner of the interface. This feature allows users to narrow the scope of the information presented by opting to see all contributions, only updates, only deletions, or only additions.

The second feature allows a user to obtain more in-depth information associated with a collaborator's contributions. When hovering the mouse on a collaborator's donut slice, a tool tip (text box overlay) appears containing more detailed information such as the number of components added/deleted/updated per each user. We think that usability and accessibility are both increased by providing detailed information on-demand, as opposed to a flood of information that a user must parse his or her self.

We think the Collaborator Actions View increases the precision and accessibility of the information, and, such, improves on prior models.

The Document Tree View shown in Figure 6.4 addresses requirement 2:

FR1.SRS.2	The system should be able to display the editable XML document elements in a way that helps users visualize them.

The view provides an interactive graphical tree displaying all editable elements in the document.

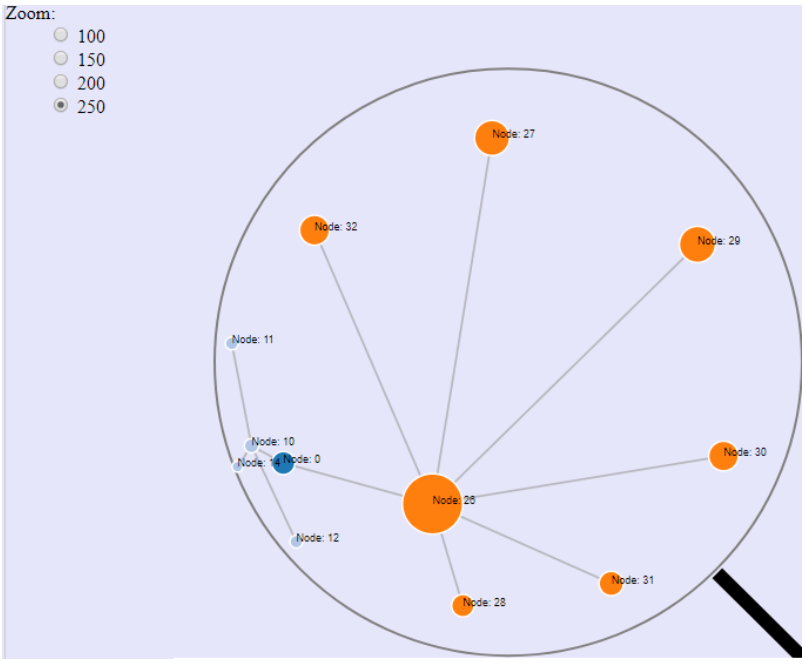


Figure 6.4: Document Tree View

A tree graph is a standard visual metaphor associated with the XML hierarchy of a document, and its implementation here will convey the structure of the XML. To enhance the speed of comprehension, leaf nodes that have shared ancestry also share a common color. The Document Tree View has two interactive features that enhance the tree image. The first feature is a control box in the upper left corner that allows the user to zoom in on the entire tree. The second feature is a magnifying glass attached to the mouse cursor. It is common to have a Version-Aware XML file with hundreds of XML elements, and thus hundreds of tree nodes. Our magnifying glass feature allows a user to focus on a dense portion of a tree and clearly identify connections. When the magnifying glass hovers over a specific node it displays the content of the node. This view improves prior tools because it shows many nodes in a tree while still being able to show details about specific nodes.

The Components View shown in Figure 6.5 addresses requirement 3:

FR1.SRS.3	The system should be able to show the XML elements and their lengths along with how many changes have been made to each element

The view provides a graphical representation of a document's components via

blocks have been scaled to the component's size (in terms of characters).

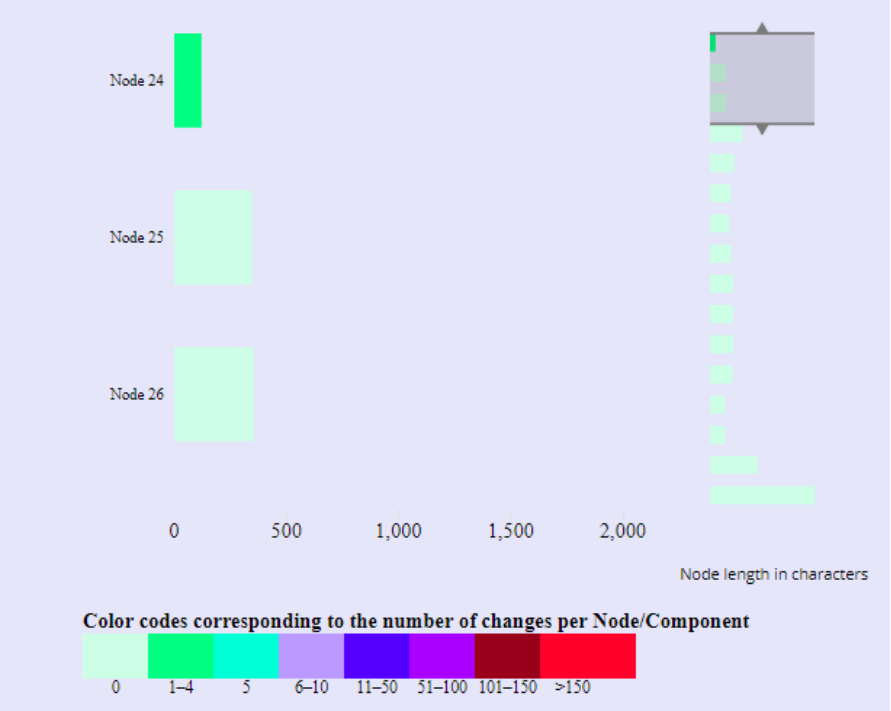


Figure 6.5: Components View

The Components View shown in Figure 6.5, consists of two main parts: a central viewing window and an overview panel. The overview panel on the right side contains the entire list of editable components and an adjustable box that paints an area of interest to the central viewing window. The size of the brush box may be adjusted as well as the portion of the overview that it paints.

In the central viewing window the user can hover the mouse over a component to obtain its exact size in characters. In addition, color codes corresponding to a clearly defined legend are used signify the number of changes made on each component.

The central viewing window updates in real-time as users adjust the area of interest. This is akin to how the Bell Labs' SeeSoft [65] visualization tool represents lines of software code. We tried to follow the same model using the brush visualization, which we think it gives the user more flexibility of control in both details and big picture.

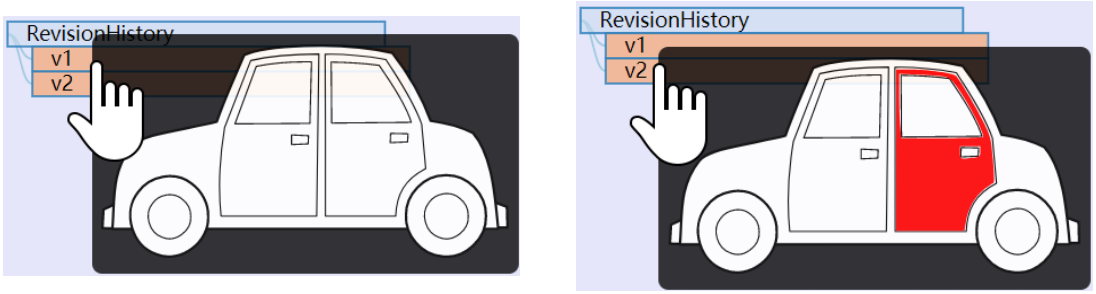
The Revision History view addresses requirements 6 and 7:

FR1.SRS.6	The system should be able to validate the integrity of a revision/version of a document.
FR1.SRS.7	The system should be able to display the changes that have been made to a document in a way that helps users to visualize them.

The view provides a graphical representation of document's prior reversions and enables a user's ability to verify revision integrity using color codes.

We initially thought it would be challenging to instantaneously display prior revisions of a document; however, we designed a novel solution that allowed us to immediately retrieve and render the revision history by applying the reverse deltas to the current document. The design implements a control that allows the user to visualize the evolution of a Version-Aware XML file. The document-tree-style control is composed of squares and can be expanded and collapsed when needed. As shown in Figure 6.6 a and b, there are two revisions have been made to an SVG image, as the user hovers the cursor over each revision, a tooltip shows the rendered SVG image corresponding to that reversion. This feature gives the user the ability to switch back and forth between revisions to compare the changes that have been made across multiple revisions. This feature has the potential to be applied to many XML-based documents and we think there are numerous possibilities for expansion.

Any revision node that has illegal alteration or data corruption has been color coded. This feature is intended to draw the user's attention to a violation in the security parameters.



(a) Revision1

(b) Revision 2

Figure 6.6: Revision History View

The Geography View shown in Figure 6.7 addresses requirement 8:

FR1.SRS.8	The system should be able to the geographical locations for collaborators map.

The view shows collaborators on a graphical map of the world.

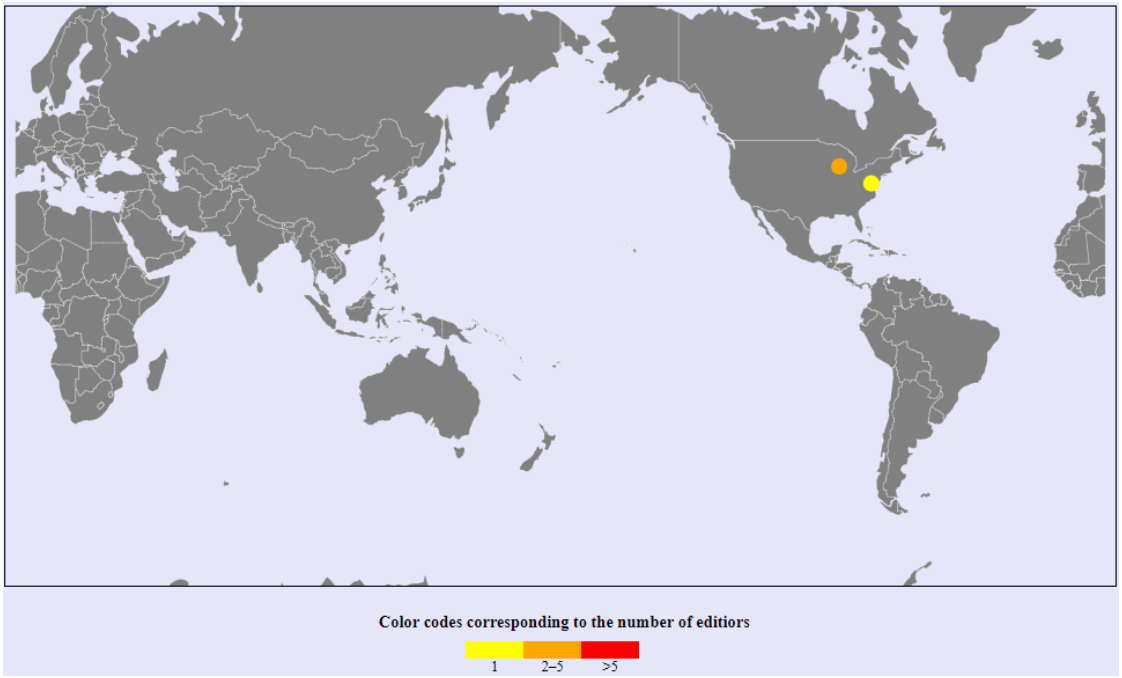


Figure 6.7: Geography View

Knowing the geographical locations collaborators may be helpful to discover patterns in open source projects, such as the effect of developers' locations on productivity and malicious behavior. We present a geographical map that shows the collaborators' locations as color coded pins. The color codes give insight into how many editors are in each location. When a user hovers the cursor over a pin, a tooltip is provided containing details about the collaborators and the exact locations associated with the location. We parsed the IP addresses stored in the Version-Aware repository and queried a web service for the location where each revision been committed. The mapping groups IP addresses by city, and color coded pins are used to indicate the number of collaborators in that city. In the future, it would be interesting to achieve a finer grain mapping for each IP address. Such mapping may help organizations and companies to understand their employees' behaviors and contributions, i.e how much of the work has been on-site vs how much has been done remotely.

The Timeline View shown in Figure 6.8 addresses requirements 4 and 9:

FR1.SRS.4	The system should be able to show the revisions made by UserX.
FR1.SRS.9	The system should be able to display the timeline of revisions for a specific document including number of changes.

The view plots revisions on a visual timeline.

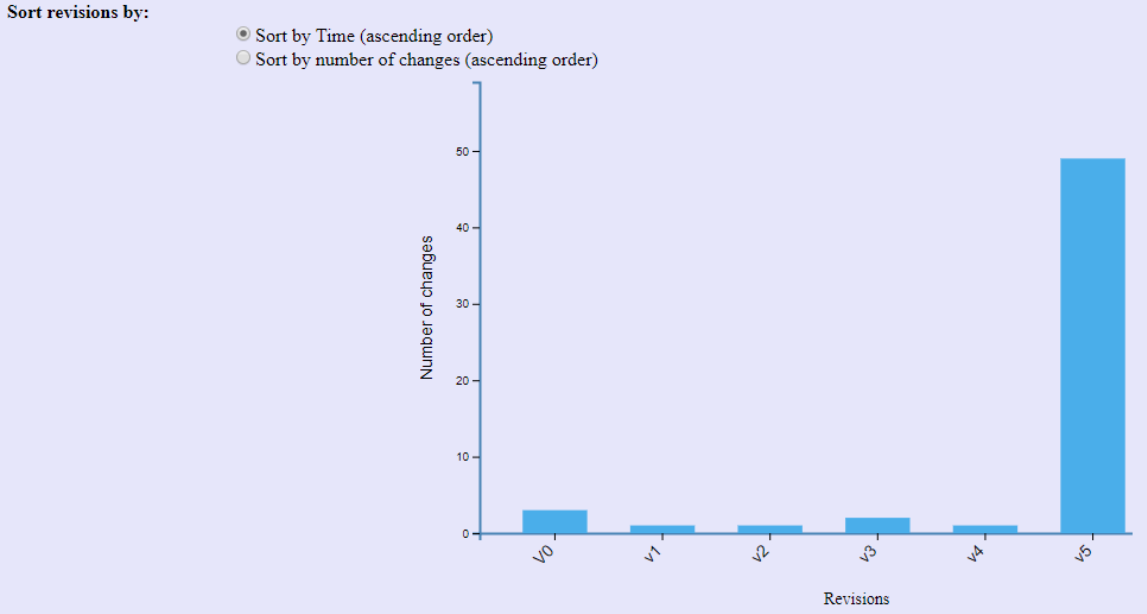


Figure 6.8: Timeline View

The Timeline View creates a bar graph of document which can be sorted by time of revision or by number of modifications associated with each revision. The x-axis of the graph is revisions, while the y-axis is number of modifications. The user can hover the mouse over a bar to obtain more detailed information about the author of this revision, the date/time of the revision, and the number of modifications in the revision.

The Revision Management View shown in Figure 6.9 addresses requirement 10:

FR1.SRS.10	The system should be able to allow users to delete a revision record or to clear the whole revision history.

Th view provides a graphical interface for deleting version history.

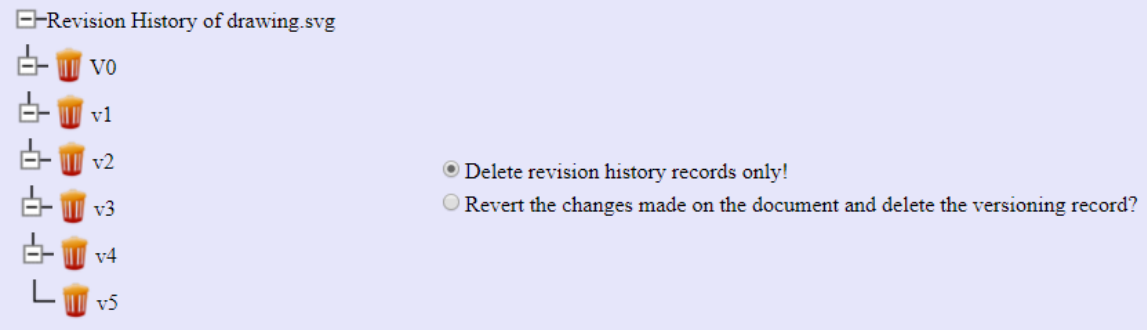


Figure 6.9: Revision Management View

From our point of view, a beneficial feature that is absent in the literature is the support for visually managing the revision history data by deleting a branch or the entire revision record. We think it it should be possible for users who are not familiar with editing code or using command line features to be able to rollback a version-controlled document. Managing the revision records is another novel part of our prototype. Our prototype provides the ability to rollback to a certain revision and delete the ensuing revision records. We built this control using ASP.net to update the Version-Aware document directly from our user interface.

The Upload View shown in Figure 6.10 is a simple interface provided as a user's first contact with the system. The upload interface is a simple control with two buttons: browse for a file and upload the file. The user is notified if he or she tries to hit the upload button before choosing an appropriate document to be uploaded.

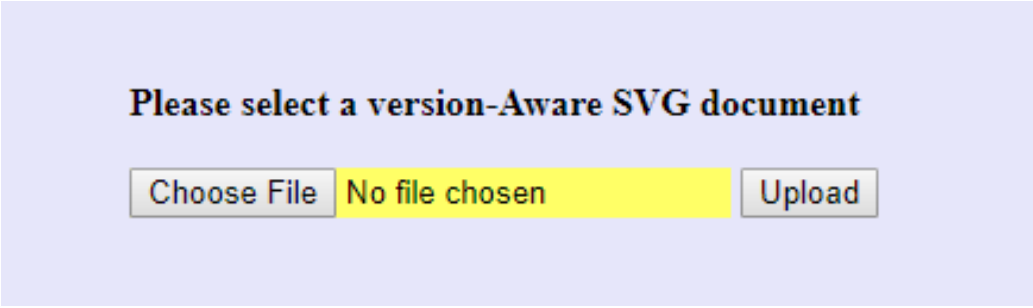


Figure 6.10: Upload View

6.2.1 Evaluation:

To evaluate the graphical user interface prototype we conducted usability tests using volunteers with a general technological background. As with all Human Computer Interaction (HCI) research, recruiting subjects was a challenge, but we were eventually able to recruit fifteen subjects to participate in the study, two females and thirteen males aged 19 to 53 years old. Our method of recruitment was via a department-wide email to graduate and undergraduate students in Computer Science at the University of Wisconsin-Milwaukee (U.W.M).

The fifteen recruited subjects were 13 graduate and 2 undergraduate students at U.W.M. The selection process recruited subjects who had no experience, little experience, and experts using version control systems, and thus the tests will attempt to reflect a true sample of the overall user base. All participants could be considered to be potential primary users of version control system.

The usability test was designed using a semi-structured format in which participants interacted with our prototype and were asked to perform a series of tasks relating to different use cases. The usability tasks are described in Appendix A.

In developing the scenarios and tasks for the usability test, our main goal was to understand the ease of use of our user interface prototype and whether the system helps the primary system users easily perform the most frequent tasks and access the versioning data.

During the usability tests, we did not describe the purpose of each visual component; rather, we observed the participant's ability to infer the purposes based on their interaction with the interface. If a participant asked a question during the test that would warrant a functional explanation, the question was tabled until the end of the test. Once the participant completed each of the required tasks (or declined to finish a task), we provided answers for functional questions we had declined to answer during the test. After the participant's questions had been answered, the participant was asked to complete a short questionnaire regarding the different interfaces they

interacted with.

At the start of the usability test, we greeted the participant, led the participant to a predefined location, and briefly introduced the facilitator (the author) and the purpose of the test. Before starting each test, each participant was given an IRB informed consent form outlining the purpose, benefits, and risks of the study. The IRB informed consent described in Appendix A.

The Usability study was designed to test the prototype's functionality, and ease of use as the performed seven (7) scenarios composed of between 2 and 6 tasks. A narrative description of the scenarios and their tasks can be seen in Appendix A.

The seven core scenarios in addition to an ad-hoc scenario deemed a useful addition after initial test design, encompass all of the prototype views. In each Scenario, the first task addresses how the subjects initially interact with the prototype through the Upload View.

- The first scenario contains eight tasks that pertain to both general collaborator contribution on a Version-Aware SVG image (updating, adding, and deleting) information and specific XML element contributions associated with each collaborator. This scenario applies to interaction with Collaborators View.

- The second scenario contains two tasks that pertain to the order of revisions based on the time that each revision committed. This scenario applies to interaction with Timeline View.

- The third scenario contains two tasks that pertain to the order of revisions based on the number of modifications made in each revision. This scenario applies to interaction with Timeline View.

- The fourth scenario contains three tasks that pertain to collaborators' locations and the distribution of the collaborators. This scenario applies to interaction with Geography View.

- The fifth scenario contains three tasks that pertain to revisions management by deleting a revision or clearing the entire revision history. This scenario applies to interaction with Revision Management View.

- The sixth scenario contains two tasks that pertain to the XML structure of the SVG image. This scenario applies to interaction with Document Tree View.

- The seventh scenario contains three tasks that pertain to determining the lengths and number of changes that have been made to each XML element in an SVG. This scenario applies to interaction with Components View.

The additional ad-hoc scenario that was added contains one task that pertains to how the subjects interacted with the system to visualize the changes that have been made from one revision to another. This scenario applies to interaction with Revision History View.

6.2.2 Evaluation Results:

The key findings from the usability tests, with an emphasis on areas of improvement we can make to our user interface prototype were:

- Overall, the participants had a positive reaction to the prototype.

- Participants managed to complete 90 percent of the seven scenarios' tasks.

- Almost all participants understood our prototype and its purpose right away.

- The wording in the Managing Revisions View was frustrating for most of the participants.

- Participants found the ability to see the changes that had been made from one revision to another in a visual way as a good feature.

- For the Document Tree View, users were prone to not realize the availability of the tooltip. There was also a loading glitch that would require a user to move the cursor to the visualization area before the tree was displayed.

- Six participants suggested that we should have some kind of help button to provide information about what should they do and to learn how to interact with different visualization controls.

- The views theme (font styles, color scheme, and visual components positioning) received mixed reviews. Some participants had positive comments on the theme, while others were not fond of it.

- Having a navigation bar to switch between views on each page was well-received.

- Participants liked that they could continue navigating between different views without going back to the main menu page.

- There is a need to review the Collaborator Action View. Four participants suggested that they like to see collaborators information in a tabular fashion in addition to donut charts.

- Five users found the Components View difficult to understand upon first seeing it.

- After the users spent two to three minutes exploring the Components View, they finished the tasks successfully. This may be due to the new visualization technique of the brush, but this certainly should be addressed.

- Color codes used were very helpful for most of the participants, two participants found difficulties to distinguish between adjacent colors in the legends.

At the conclusion of each subject's usability test, the subject was provided with a paper-based questionnaire. The questionnaire questions are described in Appendix B.

The goal of the questionnaire was to obtain quantitative feedback from the subjects, as the results from the usability test itself was primarily qualitative in nature. The majority of the survey questions were written according to the following five-point Likert-type scale:

(5) Strongly Agree (4) Agree (3) Neutral (2) Disagree (1) Strongly Disagree

The subjects were allowed to select "N/A" to allow for the possibility that the subjects did not experience what we were inquiring about or they found the question not clear enough to answer.

The questionnaire was comprised of fourteen (14) questions in three main question categories. The upload page category (5 questions) and interface design category (7 questions) were designed to find out how participants felt about specific aspects of our prototype interface. The feedback category concluded the survey with two open-ended questions to obtain additional qualitative feedback from the subjects.

After the evaluations with the participants have been completed, we analyzed and interpreted the results from the post questionnaire. For each question, we assigned values on a 1-5 scale to each response in order to report a single average response for each question. Converting the results to a single number helped us do comparisons, contrasts and reasons about questions in the questionnaire. Since some of the questions were not related to every subject/participant, the total number of respondents for each question is reported in both Figure 6.11 and Figure 6.12.

Figure 6.11 shows that subjects responded very positively about our prototype. The subjects reported less satisfaction on the placement of the error messages on the page (Question 3), loading animation while uploading the document (Question 4), and feedback about redirecting to the next page (Question 5) compared to the other Upload page related questions with averaging score 4.50, 4.45, and 4.30 respectively on a 5 point scale assessing. While participants supports that placement of

the upload control in the page and the error feedback messages received have a very good design (questions 1 and 2 got an averaged 4.73 and 5.0 respectively). This data suggests that the participants were less satisfied with the placement of the error messages on the page, loading animation while uploading the document, and feedback about redirecting to the next page than other measured features in the upload page interface.

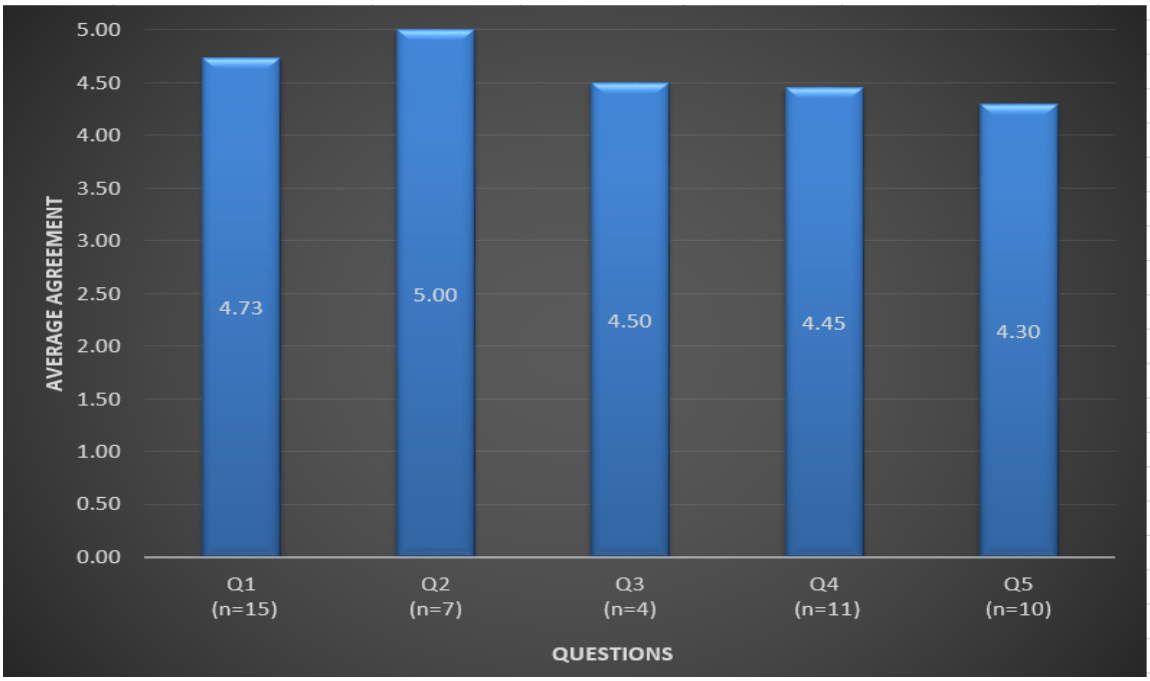


Figure 6.11: Average agreement with questions for Upload page Interface (1 = strongly disagree; 5 = strongly agree)

Figure 6.12 shows that subjects responded very positively about our prototype. The subjects reported lower satisfaction of the look and feel of the theme used across pages (Question 1), placement of the buttons on the main page interface (Question 2), and using/looking at the donut chart to convey required information (Question 4) compared to the other interface design related questions with averaging score 4.13, 4.40, and 4.27 respectively on a 5 point scale assessing. While participants supports that the navigation menu feature was helpful (Question 3), the the tooltip feature was useful to covey more information (Question 5), the layout on each page was a

good choice (Question 6), and overall, they did find our prototype user interface easy to use (Question 7). Questions 3, 5, 6, and 7 got an averaged 4.60, 4.53, 4.46, and 5.0 respectively. This data suggests that the participants has lower satisfaction for backgrounds colors and the placement of the main menu buttons. It also suggests they does not prefer using donut charts to read the visualized information, instead the users made comments that they prefer a tabular format instead to show the data.

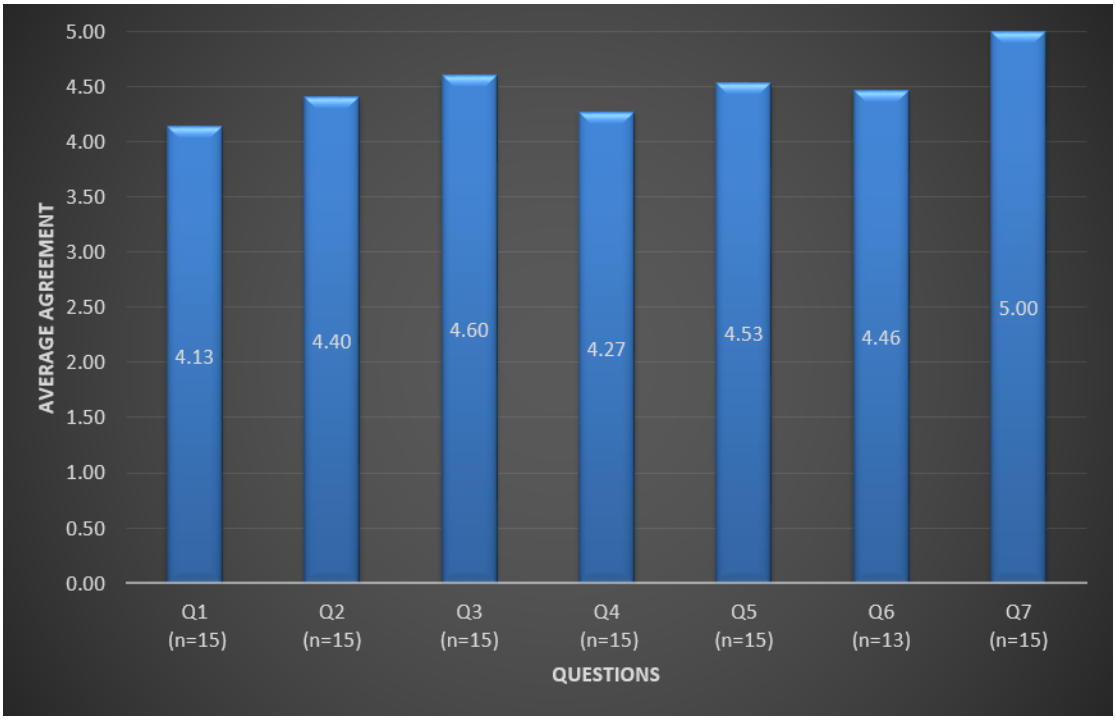


Figure 6.12: Average agreement with questions for interface design (1 = strongly disagree; 5 = strongly agree)

Based on the results of the usability tests and questionnaire, we plan to incorporate the following enhancements, grouped by scope, to our prototype in the future.

- All views
  - Change the look and feel of the main menu view, so it is larger and clearer to the user.
  - Change the wording of multiple buttons to better communicate their purpose.

  - Adding more system messages to better inform the user of acceptable behavior. For example, "You should hover your cursor on the sectors of the donut chart to get more information".
  - Consider color codes for people who have color blindness.
  - Create help documentation and tutorials. Determine how such documentation should be accessed, viewed, and searched.

- Document Tree View
  - Change the Document Tree View to draw the tree immediately when the user activates this page, so that the user does not need to move the cursor into the visualization area to make the tree visible.

- Collaborator Action View
  - Add a tabular visualization of collaborators information in addition to the donut charts so users can find the best way to read the information.

- Geography View
  - Add visible controls for zooming the map to zoom in and zoom out to be more obvious for the user. In the prototype, zooming has supported using the scroll mouse and many participants didn't realize this possibility.

- Upload View
  - Ensure that the upload button is inactive until the user chooses a valid document to be visualized.

- Revision Management View
  - After deleting, provide clear feedback on whether the operation was successful or not.
  - Find a more intuitive icon for version deletion.

We think that our usability test has room for improvement, but, we also thought the results from our participants were positive. Evaluating the results confirmed one of our main goals in the design and building our prototype: to enhance the user experience of accessing revision information from a VCS. In contrast, [44, 65, 50, 51, 66] did not conduct any usability evaluation for the effectiveness of their visualization techniques, leaving a question as to the qualitative value a user would place on interfacing with their systems.

The ability to manage version control systems revision via a graphical user interface which provides an easily accessible way for collaborators and administrators to manage the revisions regardless of technical expertise was considered a key part in our prototype. Accomplishing this task sets our prototype in contrast with the solutions presented in the literature, of which none had this feature present.

With the rapid release of emerging technologies at present, along with the ever present security threat of the modern world, security awareness and protecting one's sensitive data has become an essential part of an individual's life habits. In our effort to raise security awareness, our Revision History View clearly and visually indicates whether a revision's integrity is valid or not. This visual feature is lacking in the current state-of-the-art.

It is sometimes tricky to unite ease of use and complexity of data, but our user study participants reported that they found our graphical user interface prototype intuitive to use due to our reliance on conventional representations of the data. Using the design principle "Do not let me think" [62] laid a good ground rule to the design of a graphical interface, and we think it led to better evaluation results. We understand that maintaining the ease of use of the user interface is not always possible, as it depends upon the system complexity domains. However, we think that our user studies provided evidence on how real users interacted with our prototype.

A desire for portability and maintenance led us to build our visualization prototype as a Web application. Both Xia and SeeSoft tools [45, 65] are Windows

applications, therefore porting their applications to different platforms cannot be easily done compared to our web-based prototype.

Our prototype allowed for the filtering and sorting of the data, thus providing a granularity of detail. With contrast to our work, both [50, 66] have not provided filtering and sorting feature.

Using relatively small Version-Aware XML documents to test our prototype was a limitation of our testing. In our future designs, we intend to test our prototype with real-world large VCS repositories and re-evaluate our prototype with the enhancements we determined necessary in Section 6.2.2.

Chapter 7

Contributions and Conclusions

We have shown a set of new ways to use version control techniques to enhance the security and performance of other applications.

We have already implemented our proposed integrity framework and the XML Unique ID generator, integrating both together in an XML editor that mimics the characteristics of EHR applications. We have also implemented our framework on MS Word documents.

We have defined a sufficient set of requirements that should exist in any modern version control visualization tool. We implemented an interactive graphical user interface that meets these requirements. Our prototype provides an easy way for users to understand and manage version control data.

We used Version-Aware technology to enhance the performance of the APEX framework. Our approach reduced the time needed to evaluate security policies when exposure operations are requested, especially for large documents.

The main contributions of this dissertation's research are:

- Our eAPEX solution improved the performance of the secure dynamic policy enforcement infrastructure (APEX), specifically improving its performance when dynamic policies are applied to large documents.

- Our secure framework for XML documents improves security for offline (Word) documents and their provenance. It provides persistent integrity and tampering detection as well as provides tools for doing forensics. Our approach provides an extensive document history with author signatures at each step.

- Our visualization prototype takes steps to improve users' understanding of

version control data. It may also improve the efficiency of collaboration on documents while promoting security awareness.

- Our mathematical analysis used to determine the optimal unique ID length for Version-Aware documents that enhances human readability and reduces storage requirements.

We would like to investigate and consider the applicability of our proposed framework to strengthen the quality of health care and avoid any life threatening consequences in developing countries and remote geographical areas. We would also like to explore our framework's capacity to assist in times of crises, such as natural disasters, when online services might be expensive or intermittent. For example, more than 7 million people lost internet services due to Hurricane Irma. In situations like this, information technology applications in critical locations such as hospitals and police stations should stay function . Failing to maintain data integrity of these applications while operating offline could lead to severe consequences.

In addition, we intend to pursue an application of our framework to the military domain where internet may not be available on the front lines and maintaining the integrity of information collected from, and relayed to, the field can mean the difference between life and death.

As of now, our secure framework for XML documents does not protect against the removal of the most recent revision. We are planning to develop a protocol that addresses this issue by adopting a communication procedure that requires authors to have some Internet/Cloud access at one point to push their revisions to a trusted site or person.

We plan to support eAPEX as a plugin that can be integrated into applications without modifying the underlying application in any way. An interesting example of such an integration would be MS Word. Implementing eAPEX as a plug-in for the Microsoft Word application would provide portability because the plug-in would be attached to the document itself and it is going to be scalable because there are no

fixed limits on the numbers of users who can collaborate in producing the document.

Finally, even though we felt the usability tests we conducted for our visualization prototype were successful, our requirements and questionnaire were not perfect. In response, we plan to revise the requirements and the survey questions, conduct additional usability tests, and implement the enhancements listed in Section 6.2.2. We also intend to conduct further evaluation on our visualization prototype with more realistic documents such as Git and Concurrent Versions System (CVS) repositories.

Bibliography

[1] Helen Balinsky and Steven J Simske. Secure document engineering. In *Proceedings of the 11th ACM symposium on Document engineering*, pages 269–272. ACM, 2011.

[2] G. Dimitrios Katehakis, Stelios Sfakianakis, Manolis Tsiknakis, and C. Stelios Orphanoudakis. An infrastructure for integrated electronic health record services: the role of xml. *Technology and Health Care*, 8(3/4):189–190, 2000.

[3] Hongyan Li, Shiwei Tang, Dongqing Yang, and Yi Yu. An xml based electronic medical record integration system. In *Advances in Web-Age Information Management*, pages 160–167. Springer, 2001.

[4] Robert Steele, William Gardner, Darius Chandra, and Tharam S Dillon. Framework and prototype for a secure xml-based electronic health records system. *International journal of electronic healthcare*, 3(2):151–174, 2007.

[5] Tim Sheahan. Hp exstream. *PrintWeek*, 2012.

[6] Using Business Activity Monitoring ES Dashboard. Adobe livecycle es version 8.0 adobe, 2007.

[7] Steven J Simske and Helen Balinsky. Apex: automated policy enforcement exchange. In *Proceedings of the 10th ACM symposium on Document engineering*, pages 139–142. ACM, 2010.

[8] Michael E Whitman and Herbert J Mattord. *Principles of information security*. Cengage Learning, 2011.

[9] Cheng Thao and Ethan V Munson. Version-aware xml documents. In *Proceedings of the 11th ACM symposium on Document engineering*, pages 97–100. ACM, 2011.

[10] Nayan B Ruparelia. The history of version control. *ACM SIGSOFT Software Engineering Notes*, 35(1):5–9, 2010.

[11] Diomidis Spinellis. Git. *IEEE software*, 29(3):100–101, 2012.

[12] Brian De Alwis and Jonathan Sillito. Why are software projects moving from centralized to decentralized version control systems? In *Proceedings of the 2009 ICSE Workshop on cooperative and human aspects on software engineering*, pages 36–39. IEEE Computer Society, 2009.

[13] Caius Brindescu, Mihai Codoban, Sergii Shmarkatiuk, and Danny Dig. How do centralized and distributed version control systems impact software changes? In *Proceedings of the 36th International Conference on Software Engineering*, pages 322–333. ACM, 2014.

[14] Viktor Green. Version control systems in corporations: Centralized and distributed-an explorative case study into the corporate use of version control systems. 2015.

[15] Brian De Alwis and Jonathan Sillito. Why are software projects moving from centralized to decentralized version control systems? In *Cooperative and Human Aspects on Software Engineering, 2009. CHASE'09. ICSE Workshop on*, pages 36–39. IEEE, 2009.

[16] Christian Rodríguez-Bustos and Jairo Aponte. How distributed version control systems impact open source software projects. In *Proceedings of the 9th IEEE Working Conference on Mining Software Repositories*, pages 36–39. IEEE Press, 2012.

[17] Nic Bertino. Modern version control: creating an efficient development ecosystem. In *Proceedings of the 40th annual ACM SIGUCCS conference on User services*, pages 219–222. ACM, 2012.

[18] Vladimir Jotov. An investigation on the approaches for version control systems. In *Proceedings of the 9th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, page 73. ACM, 2008.

[19] Eric Sink. *Version control by example*. Pyrenean Gold Press, 2011.

[20] Cheng Thao and Ethan V Munson. Using versioned tree data structure, change detection and node identity for three-way xml merging. In *Proceedings of the 10th ACM symposium on Document engineering*, pages 77–86. ACM, 2010.

[21] Meenu Pandey and Ethan V Munson. Version aware libreoffice documents. In *Proceedings of the 2013 ACM symposium on Document engineering*, pages 57–60. ACM, 2013.

[22] Stephen M Coakley, Jacob Mischka, and Cheng Thao. Version-aware word documents. In *Proceedings of the 2nd International Workshop on (Document) Changes: modeling, detection, storage and visualization*, page 2. ACM, 2014.

[23] Ethan V Munson, Cheng Thao, et al. Improving version-aware word documents. In *Proceedings of the 2017 ACM Symposium on Document Engineering*, pages 129–132. ACM, 2017.

[24] Manoj Jayabalan and Thomas ODaniel. Access control and privilege management in electronic health record: a systematic literature review. *Journal of medical systems*, 40(12):261, 2016.

[25] AA Abd El-Aziz and A Kannan. Literature review on xml security and access control to xml documents.

[26] Helen Balinsky, David Subiros Perez, and Steven J Simske. System call interception framework for data leak prevention. In *Enterprise Distributed Object Computing Conference (EDOC), 2011 15th IEEE International*, pages 139–148. IEEE, 2011.

[27] Ding Yixiang, Peng Tao, and Jiang Minghua. Secure multiple xml documents publishing without information leakage. In *Convergence Information Technology, 2007. International Conference on*, pages 2114–2119. IEEE, 2007.

[28] Elisa Bertino, Silvana Castano, and Elena Ferrari. Securing xml documents with author-x. *IEEE Internet Computing*, 5(3):21–31, 2001.

[29] KALUKA WANJALA. Microsoft office user base crosses the 1.2 billion mark, 2016.

[30] Steven Morris. Nurses jailed for falsifying stroke patients' records, 2016.

[31] Dimitrios G Katehakis, Stelios Sfakianakis, Manolis Tsiknakis, and Stelios C Orphanoudakis. An infrastructure for integrated electronic health record services: the role of xml (extensible markup language). *Journal of Medical Internet Research*, 3(1), 2001.

[32] Fatemeh Rezaeibagha, Khin Than Win, and Willy Susilo. A systematic literature review on security and privacy of electronic health record systems: technical perspectives. *Health Information Management Journal*, 44(3):23–38, 2015.

[33] Yunhua Koglin, Giovanni Mella, Elisa Bertino, and Elena Ferrari. An update protocol for xml documents in distributed and cooperative systems. In *Distributed Computing Systems, 2005. ICDCS 2005. Proceedings. 25th IEEE International Conference on*, pages 314–323. IEEE, 2005.

[34] Gerome Miklau and Dan Suciu. Managing integrity for data exchanged on the web. In *WebDB*, pages 13–18, 2005.

[35] Baolong Liu, Joan Lu, and Jim Yip. XML data integrity based on concatenated hash function. *CoRR*, abs/0906.3772, 2009.

[36] MJ Sue Bowman, CCS RHIA, et al. Impact of electronic health record systems on information integrity: quality and safety implications. *Perspectives in Health Information Management*, page 1, 2013.

[37] Derek Mohammed, Ronda Mariani, and Shereeza Mohammed. Cybersecurity challenges and compliance issues within the us healthcare sector. *International Journal of Business and Social Research*, 5(2):55–66, 2015.

[38] Andreas Ekelhart, Stefan Fenz, Gernot Goluch, Markus Steinkellner, and Edgar Weippl. Xml security - a comparative literature review. *J. Syst. Softw.*, 81(10):1715–1724, October 2008.

[39] José Luis Fernández-Alemán, Inmaculada Carrión Señor, Pedro Ángel Oliver Lozoya, and Ambrosio Toval. Security and privacy in electronic health records: A systematic literature review. *Journal of biomedical informatics*, 46(3):541–562, 2013.

[40] ER Harold. Tip: Configure sax parsers for secure processing, 2005.

[41] Pedigree Ratified Standard EPCglobal. 1.0., epcglobal (2007).

[42] California e-pedigree law. `https://web.archive.org/web/20080515195728/http://www.pharmacy.ca.gov/about/e_pedigree_laws.shtml`, 2008. [Online; accessed 19-September-2017].

[43] Holger M Kienle. Exchange format bibliography. *ACM SIGSOFT Software Engineering Notes*, 26(1):56–60, 2001.

[44] Thomas Ball and Stephen G Eick. Software visualization in the large. *Computer*, 29(4):33–43, 1996.

[45] Xiaomin Wu, Margaret-Anne Storey, Adam Murray, and Rob Lintern. Visualization to support version control software: Suggested requirements. *VisSoft. Amsterdam, Netherlands*, pages 80–86, 2003.

[46] Xiaomin Wu, Margaret-Anne Storey, Adam Murray, and Rob Lintern. Visualization to support version control software: Suggested requirements. *VisSoft. Amsterdam, Netherlands*, pages 80–86, 2003.

[47] Xinrong Xie, Denys Poshyvanyk, and Andrian Marcus. Visualization of cvs repository information. In *2006 13th Working Conference on Reverse Engineering*, pages 231–242. IEEE, 2006.

[48] Michael Ogawa and Kwan-Liu Ma. Stargate: A unified, interactive visualization of software projects. In *2008 IEEE Pacific Visualization Symposium*, pages 191–198. IEEE, 2008.

[49] Christian Collberg, Stephen Kobourov, Jasvir Nagra, Jacob Pitts, and Kevin Wampler. A system for graph-based visualization of the evolution of software. In *Proceedings of the 2003 ACM symposium on Software visualization*, pages 77–ff. ACM, 2003.

[50] Ayush Shrestha, Ying Zhu, and Ben Miller. Visualizing time and geography of open source software with storygraph. In *Software Visualization (VISSOFT), 2013 First IEEE Working Conference on*, pages 1–4. IEEE, 2013.

[51] Brandon Heller, Eli Marschner, Evan Rosenfeld, and Jeffrey Heer. Visualizing collaboration and influence in the open-source software community. In *Proceedings of the 8th working conference on mining software repositories*, pages 223–226. ACM, 2011.

[52] Cheng Thao and Ethan V Munson. Using versioned tree data structure, change detection and node identity for three-way xml merging. In *Proceedings of the 10th ACM symposium on Document engineering*, pages 77–86. ACM, 2010.

[53] Jeff Preshing. Hash collision probabilities. `http://preshing.com/20110504/hash-collision-probabilities/`, 2011. [Online; accessed 19-May-2015].

[54] Jargon file, version 2.6.2, 1991.

[55] Alfred V Aho and Margaret J Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[56] Ashish K Jha, Catherine M DesRoches, Eric G Campbell, Karen Donelan, Sowmya R Rao, Timothy G Ferris, Alexandra Shields, Sara Rosenbaum, and David Blumenthal. Use of electronic health records in us hospitals. *New England Journal of Medicine*, 360(16):1628–1638, 2009.

[57] H Fusco, T Hubschman, V Mweeta, Benjamin H Chi, Jens Levy, Moses Sinkala, et al. Electronic patient tracking supports rapid expansion of hiv care and treatment in resource-constrained settings. In *3rd IAS Conference on HIV Pathogenesis and Treatment*, 2005.

[58] Keith Mweebo. Security of electronic health records in a resource limited setting: the case of smart-care electronic health record in zambia. 2014.

[59] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17(2):281–308, April 1988.

[60] Fernando Farfán, Vagelis Hristidis, Anand Ranganathan, and Redmond P Burke. Ontology-aware search on xml-based electronic medical records. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 1525–1527. IEEE, 2008.

[61] Mark Harrison and Andy Shaw. Electronic pedigree and authentication issues for aerospace part tracking, 2006.

[62] Steve Krug. *Don't make me think!: a common sense approach to Web usability.* Pearson Education India, 2000.

[63] Ben Shneiderman. Shneiderman's eight golden rules of interface design. *Retrieved July*, 25:2009, 2005.

[64] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

[65] SC Eick, Joseph L Steffen, and Eric E Sumner. Seesoft-a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering*, 18(11):957–968, 1992.

[66] Yuri Takhteyev and Andrew Hilts. Investigating the geography of open source software through github. *Manuscript submitted for publication*, 2010.

Appendices

Appendix A: IRB Informed Consent Form, Scenarios, And Tasks

Dear participant,

You are invited to participate in a research study, entitled "Evaluation of a version-aware XML visualization prototype". The study is being conducted by Ahmed Shatnawi and Prof. Ethan Munson at the University of Wisconsin - Milwaukee.

The purpose of this research is to study the user experience while interacting with our user interface for a version-aware control system. Approximately, 25 subjects will participate in this study. If you agree to participate, you will be asked to complete certain tasks as example scenario is presented to you. Using a scenario, I will ask you to complete tasks using the prototype interface. I will be writing down your strategies for completing those tasks. During the test, you may have questions for me. I generally will be unable to answer your questions during the test. When we have completed all of the scenarios and tasks, I will address any questions that you raised during the test. After all the questions have been answered, I will ask you to complete a short questionnaire about the prototype interface. This will take approximately 35 minutes of your time.

Risks that you may experience from participating are considered minimal. There are no costs for participating. There are no benefits to you other than to further research.

No identifying information will be collected. Your responses will be treated as confidential and all reasonable efforts will be made so that no individual participant will be identified with his/her answers. The research team will not make any identifiable information anywhere and all study results will be reported without identifying information so that no one viewing the results will ever be able to match you with your responses. Data from this study will be saved on a networked, password-protected computer in a locked room (EMS 1010) until May 2018. Only Prof. Ethan Munson and Ahmed Shatnawi will have access to your information. However, the Institutional Review Board at UW-Milwaukee or appropriate federal agencies like the Office for Human Research Protections may review this study's records.

Your participation in this study is voluntary. You may choose not to take part in this study, or if you decide to take part, you can change your mind later and withdraw from the study. You are free to not answer any questions or withdraw at any time. Your decision will not change any present or future relationships with the University of Wisconsin Milwaukee. There are no known alternatives available to participating in this research study other than not taking part.

If you have questions about the study or study procedures, you are free to contact the investigator at the address and phone number below. If you have questions about your rights as a study participant or complaints about your treatment as a research subject, contact the Institutional Review Board at 414-229-3173 or irbinfo@uwm.edu.

To voluntarily agree to take part in this study, you must be 18 years of age or older. By signing the consent form, you are giving your consent to voluntarily participate in this research project.

Thank you!

Ahmed Shatnawi
Milwaukee, WI 53211
571-502-3179
shatnaw3@uwm.edu

The below are the scenarios provided to the participants. Each scenario has several tasks that we asked the participants to complete using our prototype interface.

Scenario 1:

"You are collaborating offline on an SVG image (will be provided for the participant) with a group of people, and you need to know who worked previously on the SVG image, who are the collaborator/s who added, deleted, and updated content in the document(SVG image). Please rely on our prototype to answer these questions"

Task 1.

Upload the SVG image "drawing.svg" file from the desktop to be processed and visualize the revision history.

Task 2.

"You are automatically redirected to the Visualization Views page. Which view will you choose to answer the question stated in the main scenario?"

Task 3."

"[If Collaborators Actions view page selected, start with this.] Who are the collaborators who changed content in the document? Can you list them? How many unique components each collaborator changed?"

Task 4.

""[If Collaborators Actions view page selected, start with this.] Who are the collaborators who deleted content in the document? Can you list them? How many components each collaborator deleted?"

Task 5.

"[If Collaborators Actions view page selected, start with this.] Who are the collaborators who updated content in the document? Can you list them? How many components each collaborator updated?"

Task 6.

“[If Collaborators Actions view page selected, start with this.] Who are the collaborators who added content in the document? Can you list them? How many components each collaborator added?”

Scenario 2:

“You are collaborating on the same SVG image provided in the previous scenario, and you need to know what is the timeline of revisions made by all collaborators on top of the SVG image.”

Task 1

“Which view will you choose to answer the question stated in the main scenario?”

Task 2.

“[If sorted revisions view page selected, start with this.] What is the latest revision made on the document? What is the ascending order of revisions based on the time each revision made? Can you list them in ascending order?”

Scenario 3:

“You are collaborating on the same SVG image provided in the previous scenarios, and you need to know what is the order of revisions made on the document based on the number of modifications made per revision.”

Task 1.

“Which view you will choose to answer the question stated in the main scenario?”

Task 2.

“”[If sorted revisions view page selected, start with this.] What is the revision that has the highest number of modifications? What is the ascending order of revisions based on the number of modifications made in each revision?”

Scenario 4:

"You are collaborating on the same SVG image provided in the previous scenarios, and you need to know the collaborators' geographical locations."

Task 1.

"Which view will you choose to answer the question stated in the main scenario?"

Task 2.

"[If collaborators locations map view page selected, start with this.] Where are the locations the document edited in?"

Task 3.

""[If collaborators locations map view page selected, start with this.] Can you tell whether multiple editors made changes from the same geographical location?"

Scenario 5:

"You are collaborating on the same SVG image provided in the previous Scenarios, and you need to manage revisions by deleting a revision or even clear the whole revision history."

Task 1.

"Which view will you choose to answer the question stated in the main scenario?"

Task 2.

""[If Manage revisions view page selected, start with this.] Can you just delete all the revisions' records?"

Task 3.

""[If Manage revisions view page selected, start with this.] Can you revert the changes made on the document and delete the versioning record?"

Scenario 6:

"You are collaborating on the same SVG image provided in the previous scenarios, and you need to see the SVG XML tree hierarchy of the image."

Task 1.

"Which view will you choose to answer the question stated in the main scenario?"

Task 2.

""[If document tree view page selected, start with this.] What is the current parent element in the SVG document?"

Scenario 7:

"You are collaborating on the same SVG image provided in the previous scenarios, and you need to know the length of all of the component/s in the SVG image."

Task 1.

"Which view will you choose to answer the question stated in the main scenario?"

Task 2.

""[If SVG components view page selected, start with this.] Which component/node has the longest length in characters?"

Task 3.

"[If SVG components view page selected, start with this.] Which component/node has been changed the most?"

Appendix B: Post-Study Questionnaire

The following are the official questions that participants were asked via the post-test questionnaire.

The Upload page Interface:

Q1- I like the placement of the upload control in the page.

Strongly Agree Agree Neutral Disagree Strongly Disagree N/A

Q2- I like the error feedback received when I press upload button without selecting a file.

Strongly Agree Agree Neutral Disagree Strongly Disagree N/A

Q3- I like the placement of the error messages on the page.

Strongly Agree Agree Neutral Disagree Strongly Disagree N/A

Q4- I like that loading animation while uploading the document.

Strongly Agree Agree Neutral Disagree Strongly Disagree N/A

Q5- I like that the feedback about redirecting to the next page.

Strongly Agree Agree Neutral Disagree Strongly Disagree N/A

Interface Design

Q1- I like the look and feel of the theme across pages?

Strongly Agree Agree Neutral Disagree Strongly Disagree N/A

Q2- I like the placement of the buttons on the main page interface.

Strongly Agree Agree Neutral Disagree Strongly Disagree N/A

Q3- I like the navigation menu feature.

Strongly Agree Agree Neutral Disagree Strongly Disagree N/A

Q4- I like using/looking at the donut chart to convey the required information clearly

Strongly Agree Agree Neutral Disagree Strongly Disagree N/A

Q5- I like the tooltip feature and I found it useful.

Strongly Agree Agree Neutral Disagree Strongly Disagree N/A

Q6- I like the layout on each page.

Strongly Agree Agree Neutral Disagree Strongly Disagree N/A

Q7- Overall, did you find our prototype user interface easy to use?

yes No

Q8- Why?

Other Suggestions

If you have any other suggestions for this prototype interface, please write them here.

Appendix C: Publications

[**1**] Ahmed Shatnawi, Ethan V. Munson, and Cheng Thao. "Maintaining Integrity and Non-Repudiation in Secure Offline Documents." In Proceedings of the 2017 ACM Symposium on Document Engineering, pp. 59-62. ACM, 2017.

Curriculum Vitae

AHMED S. SHATNAWI

+1(571) 502-3179 **shatnaw3@uwm.edu**

Advisor: Professor, Ethan V. Munson

EDUCATION

M.S.	2012	Software Engineering	George Mason University, Fairfax, VA
B.A.	2007	Computer Engineering	Jordan University Of Science And Technology, Irbid, Jordan

PUBLICATIONS

Shatnawi, Ahmed, Ethan V. Munson, and Cheng Thao. "Maintaining Integrity and Non-Repudiation in Secure Offline Documents." In Proceedings of the 2017 ACM Symposium on Document Engineering, pp. 59-62. ACM, 2017.

Abdel-Hafeez, Saleh, Ann Gordon-Ross, Asem Albosul, Ahmad Shatnawi, and Shadi Harb. "A shadow dynamic finite state machine for branch prediction: an alternative for the 2-bit saturating counter." Informatica 35, no. 2 (2011).

WORK EXPERIENCE

University of Wisconsin–Milwaukee ***WI, USA*** **8/2014-12/2017**

Computer Science Department

Graduate Teaching Assistant

- Primary instructor: prepared assignments, quizzes, exams, lectures, and lab activities for intermediate programming using Java (Spring 2017).
- Ran laboratories and graded for five additional courses including:
  - Introductory computer programming
  - Data structures and algorithms
  - Introduction to computer security
  - Text retrieval and its applications in biomedicine
  - Introduction to database systems.

University of Wisconsin–Milwaukee ***WI, USA*** **1/2014-8/2014**

Information security office

Security Coordinator-Risk Assessment Support

- Worked with System administrators to fix security issues on servers and workstations found using a vulnerability scanner.
- Managed vulnerability scanner.
- Worked on various technical and non-technical projects for the office as assigned. (Worked on a web vulnerability scanner project. Worked on and completed the project to change the authentication for the scanner to the Active Directory).
- Developed two automated tools for the office using C#.

Blue Sky Broad band, ***MN, USA*** **8/2012-1/2013**

Senior Engineer

- Designed LAN/WAN and configuring routers and switches according to the network design.
- Installed, maintained and supported Linux Servers (i.e., AAA, SIP and web server).
- Installed, configured and monitored the operation of wireless devices (Alvarion, Motorola, Radwin, and Ubiquity).
- Web applications and services development using C#, ASP.NET and JavaScript.

George Mason University, VA, USA **6/2011-8/2011**

Center of Intelligent Spatial Computing for Water/Energy Science

Web application programmer

- Developed an open-source standalone client to utilize the cloud computing and WPS services to support Earth and geography science communities. Windows Azure Platform was used.

JORDAN UNIVERSITY OF SCIENCE AND TECHNOLOGY, Irbid, Jordan 2/2007-8/2010

Department of Computer Engineering

Teaching Assistant

- Trained undergraduate students on new technologies, including using the Verilog Hardware Description Language; on lab tools; and on relating theories to practice.
- Taught students how to write synthesizable code to implement digital circuits using Altera Quartus II Software.
- Supervised the systems programming course projects which included the design and implementation of emulators, assemblers, linkers and loaders.
- Taught the web technologies lab which primarily focused on web applications design and development. Several technologies were used in the lab including ASP.NET and Ajax.
- Helped in designing the lab component of several courses including Hardware Description Languages, Visual Programming, Computer Networks, Systems Programming

HORIZONS MEDIA AND INFORMATION SERVICES, Amman, Jordan 9/2006-1/2007

Software Engineer

- Part of the team that developed the intranet of the Ministry Of Economy and Commerce in Qatar. My role was focused on SharePoint Technology.
- Web applications and services development using C#, ASP.NET and JavaScript.
- Development of internal libraries using C++ and C#.

Awards And Honors

- Chancellor's Graduate Student Award - UWM, 2015, 2016, 2017.
- ACM SIGSOFT, CAPS Travel Award, 2016
- PHI KAPPA PHI Honor Society, 2016
- Distinguished Academic Achievement Award - Department of Computer Science, GMU, 2012.
- Outstanding Academic Achievement Award - Office of International Programs and Services, GMU, 2012.

Computer Skills

Programming/Scripting: C++, .NET Framework (C++.NET, ASP.NET and C#), JAVA, XHTML, JavaScript, PHP, XML, AJAX, CSS, Servlet, JSP, JSON, MVC.

Assembly Languages: MIPS R2000, Motorola 68000

Hardware Description: Verilog

Database Systems: MYSQL, Oracle

Operating Systems: Windows (Client and Server), Linux, UNIX