Security and Compliance Testing Strategies for Cloud Computing

Dr Dipankar Dasgupta, Durdana Naseem
Center for Information Assurance,
Department of Computer Science
University of Memphis, TN
ddasgupt@memphis.edu, dnaseem@memphis.edu

ABSTRACT
Due to rapidly changing Information Technologies, it is becoming more expensive for companies/organizations to regularly update hardware and software, and also maintain a big IT department with highly technical staff. So many organizations are adopting cloud services to reduce the cost and increase the flexibility of their IT infrastructures. While different sectors are adopting the cloud for their IT need, they are also very concern about data security (both in rest and in motion) and various compliance requirements such as PCI DSS, HIPAA, GLBA, SOX, etc. There are several cloud service models, where one model sits on top of another with lowest one as Infrastructure-as-a-Service (IaaS), and above that is Platform-as-a-Service (PaaS) and Software-as-a-Service (SaaS). While security requirements are essential for all service models, they vary in the degree of defensive measures. Moreover compliance for each sector requires specific protection for online data such as Accountability, Transparency, Accuracy, Security and Access. Essentially all compliances have the Network and Data Security as the important area of responsibility.

The main focus is to find new methods to manage risk factors and compliance violations in the cloud computing security, which the business or companies can utilize to manage processes and decision rights. Hence this provides new testing methods for the companies in reducing risk and compliance efforts. In this paper, we propose a framework that uses a compressive research on different compliance model requirements against each service model, which can be used as a framework by the companies to test risk and compliances against cloud service providers. The developed tool will generate a SLA document for the organization to help in finding correct required services from the vendor in order to certify the compliance in their industry structure.

Keywords

1. INTRODUCTION
Traditionally software requires that you purchase licenses from vendors, install the software locally on the local server or on employee local machines, which needed onsite IT expert resource to maintain the infrastructure cost, facilities cost, labor costs plus maintenance costs and also add back up data, maintain security and servers, upgrades and patches, make sure redundancy plus uptime, etc. Consider when your business starts growing there are additional purchases like licenses, infrastructure, which increases the costs.

Cloud computing have evolved the concept of how we deploy, maintain, and access software, platforms, and infrastructure theory utilizing the concept of Internet as a service. Analysts forecasting a long-running trend where all types of business services will be virtualized, enabling massive interoperability, which will potentially lead to huge cost savings.

Moving application and data to the cloud has many advantages but when it comes to sensitive data it is very risky. The important concept for the data is „Know where your data is and know where your data is going... [1]. This concept plays a major role in achieving and maintaining security compliance with all areas of security compliances. In satisfying the compliance requirements and achieving any of the compliances depends upon ability of the vendor to implement network access controls, and periodically test their effectiveness. When moving to a cloud platform, where the underlying infrastructure is outsourced the problem rises. Hence to address this the compliances for different sectors are introduced whose approach is to understand the architecture, access control and network security [3-5].

1.1 Cloud Computing Models:
Some of the key compliance includes PCI DSS, HIPAA, GLBA, and SOX etc. Although the term Cloud Computing and different sector compliances are widely used, it is important to note that all Cloud models are not the same. As such, it is critical that organizations don't apply broad-brush one-size fits all approach to security across all models [2]. Cloud Models can be segmented into Software as a Service (SaaS), Platform as a service (PaaS) and Integration as a Service (IaaS). When an organization is considering Cloud security it should consider both the differences and similarities between these three segments of Cloud Models.

1.2 Benefits of Compliance:
Cloud Providers need to be carefully assessed by each customer to verify if the key requirements of data security and auditing are met. This can be a very tedious and risky task for the enterprise or customers, and potential hazard to breach. Being Compliance ensures the following:
- Best and improved protection of Companies critical Data and Information.
- Reduce the liability due to security breach.
- Timely Audition to ensure full compliancy and Reporting.
- Cost efficient.

Along with the above-mentioned benefits there is a trust built in compliant cloud providers to provide guaranteed, highly secured, fully compliant datacenter services, which also increases their business [6-7].

1.3 SLA (Service Level Agreement):
SLA is the Service level Agreement provided by the Cloud Service Provider to the Customer so that the Customer request the
requirements provided by the SLA form the Vendor [8]. The purpose of this agreement is to make sure that the proper elements are in place to provide the cloud service support to the customer by the service provider. The Objectives of this agreement is to provide clear description of service provision to the customer, provide a clear reference, Match provision of expected service provision with actual service support.

1.4 Problem Definition:
The problem addressed in this paper is, with increase in security compliances and cloud computing services demand, every company states to best provide all checklists required for different compliance. Organizations demand and vendors provide assurances by passing third-party vulnerability assessments, encrypting their data, onsite visits, or taking on contractual liability in the event of a breach or service degradation, etc. This requires vendors to implement what the organizations may not be able to do for them without incurring significant expense. Consider a Payment industry where the 12 requirements of PCI DSS must be satisfied by the Vendor Organizations to provide security certification, but it increases the cost. Hence, when an organization wants to use only Platform as a Service, then they may not need the vendor to fulfill all the requirements compared to the organization that request for Software as a Service from the cloud.

All the current existing Tools provide a framework that fulfill all the requirements for the particular sector compliance but do not filter the requirements that are necessary based on individual service model. This will be addressed in this project by developing the relational model for compliance with the service model. Once the organization mapping is done the tool generates a SLA document in the PDF format for the customers, which they can use to find the right vendor and use as a service agreement between them. This framework will also provide the companies a tool to test risk and compliances against cloud service providers.

2. BACKGROUND
Cloud systems are not to be misunderstood as just another form of resource provisioning infrastructure. Though the concept of,.clouds,. is not new, it is undisputable that they have proven a major commercial success over recent years. In this section we will describe cloud computing, different service models and compliances.

2.1 Cloud Computing
Cloud computing involves delivering hosted services over the Internet. A cloud service has three different characteristics based on the service models.

2.2 Service Models
The Service models are divided into three categories: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS)

2.2.1 SaaS
SaaS involves the vendor using their cloud infrastructure and cloud platforms to provide customers with software applications such as email. Users can use web browser to access the end user applications eliminating the need for the user to install or maintain additional software. The vendor controls and maintains the Network, Servers, Operating Systems, physical computer hardware, and software applications. Where as the customer only controls and maintains limited application configuration settings specific to users such as creating email address distribution lists.

This particular model is focused on managing access to applications. Some of the SaaS vendor services include Salesforce.com Customer Relationship Management (CRM), Google Docs and Google Gmail, Microsoft Office 365 [9].

2.2.2 PaaS
PaaS involves the vendor-providing Infrastructure as a Service plus operating systems and server applications such as web servers. PaaS enables customers to use the vendors cloud infrastructure to deploy web applications and other software developed by the customer using programming languages supported by the vendor. The vendor controls and maintains the Network, physical computer hardware, operating systems and server applications. Where as the customer only controls and maintains the software applications developed by the customer. The primary focus of this model is on protecting data and storage as a service. An important element to consider within PaaS is the ability to plan against the possibility of an outage from a Cloud provider. Some of the PaaS vendor services include Google App Engine, Force.com, Amazon Web Services Elastic Beanstalk, and the Microsoft Windows Azure platform [9].

2.2.3 IaaS
IaaS involves the vendor providing physical computer hardware including CPU processing, memory, and data storage and network connectivity. The vendor may share their hardware among multiple customers using virtualization software. IaaS enables customers to run operating systems and software applications of their choice. The vendor controls and maintains the physical computer hardware where as the customer controls and maintains the operating systems, software applications. Some of the IaaS vendor services include Amazon Elastic Compute Cloud (EC2), Go Grid and Rackspace Cloud [9].

<table>
<thead>
<tr>
<th>Table 1: Different Service Models with Who Controls What?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network</td>
</tr>
<tr>
<td>Servers</td>
</tr>
<tr>
<td>Operating Systems</td>
</tr>
<tr>
<td>Storage</td>
</tr>
<tr>
<td>Application</td>
</tr>
<tr>
<td>Configurations</td>
</tr>
</tbody>
</table>

2.3 Security Compliances
The term compliance describes the ability to act according to an order, a set of rules or a request. Some of the key compliance includes PCI DSS, HIPAA, GLBA, SOX, and FISMA.

2.3.1 PCI DSS
PCI-DSS was setup by the major card companies in order to secure the information for the financial transaction by the credit or the debit cards. When the business organization wants to use one of the compliance then he will be directed to the following security offered by the compliances. PCI DSS compliance was set up to improve the Information Security of financial transactions related to credit and debit cards [10-13].
2.3.2 **HIPPA**

HIPAA (Health Insurance Portability and Accountability Act) requires insurance portability, administrative simplification and fraud enforcement like privacy and security. Migrating to the cloud in the health care industry can have extensive benefits such as extensively reducing the size of an on-site data center, but the move obviously needs to be done in a way that ensures the safety of patient information as well as the organization's HIPAA compliance status [14-16].

2.3.3 **GLBA**

GLBA (Gramm-Leach-Bliley Act) requires financial institutions to develop, implement, and maintain a comprehensive written information security program that protects the privacy and integrity of customer records. This compliance requires analyzing the risks before moving the customer information into emerging technology models. It mandates emphasize the need for each bank, thrift, and credit union agency to adopt a proactive information security and technology risk management capability [17].

2.3.4 **SOX**

SOX (The Sarbanes-Oxley Act of 2002) is legislation that came into existence in response to the high-profile Enron and WorldCom financial scandals to protect shareholders and the general public from accounting errors and fraudulent practices in the enterprise. SOX is not a set of business practices and does not specify how a business should store records; rather, it defines which records are to be stored and for how long [18].

2.3.5 **FISMA**

FISMA (The Federal Information Act of 2002) was established to address the importance of information security related to both the economic and national security interests of the United States. It maintains minimum-security requirements and controls to be abided by all federal agencies [19].

3. RELATED WORK

Below are few of the Tools that provide framework to support compliances and security products for the organizations to certify the compliances:

3.1 **WatchGuard, Compliance Tool**

3.1.1 **Tool Description:**

WatchGuard [20] is an easy-to-manage network security tool that provides small and enterprise offices compliance solutions. They also provide solutions that deliver security, performance and rock-solid reliability for different industries like Hospitality, Education, Energy, Healthcare and Finance. WatchGuard uses XTM multifunctional firewalls, Next generation Firewall etc. This tool provides all the required software and act as a vendor for the organizations that are looking for cloud computing compliances services. It is a very comprehensive tool that is useful to all kind of service sectors with high-end technology

3.1.2 **Functionality:**

- It also provide Cyber Bullying for Schools and Workplace
- It provides the customers with different Service Models with complete compliance fulfillment
- It provides not only Fireware, which intelligently block evasive application such as Skype communications but also, provides gateway antivirus.
- It provides the SSL certificate
- All the agreements are standard across all services provided.

3.2 **Trust Wave, Compliance Tool**

3.2.1 **Tool Description:**

Trust Wave [21], is the leading provider of on-demand data security and payment card industry compliance management solutions to businesses and organizations throughout the world. They also provide solutions that deliver security, performance and rock-solid reliability for different industries like Hospitality, Education, Energy, Healthcare, Government agencies, POS Providers, Retailers, Utilities, Hosting Providers, Higher Education and Finance Services. It provides services from small business to large companies with different cloud components. It was awarded as 2011 finalist for best security company.

3.2.2 **Functionality:**

- It manages logs to identify security events and potential incidents
- It provides encryption, Web application firewall, tokenization, Network access control, data loss prevention, file integrity monitoring, application security and authentication.
- It also provide SSL Certificates
- Detect, scan, inventory and analyze the Web site contents.

3.3 **FedRAMP Process:**

3.3.1 **Description:**

Federal Risk and Authorization Management Program (FedRAMP) [22] is a government-wide program that provides a standardized approach to security assessment, authorization and continuous monitoring for cloud-based services. FedRAMP uses a “do once, use many times,” framework that intends to saves costs, time, and staff required to conduct redundant agency security assessments and process monitoring reports.

FedRAMP uses a security risk model that can be utilized by different agencies based on a consistent security baseline. The FedRAMP PMO can create a framework for agencies to utilize the security authorization. This framework requires Federal agencies to use FedRAMP when assessing, authorizing, and continuously monitoring cloud services. Four distinct areas of FedRAMP security authorization process are: Security Assessment, Leverage the Authority to Operate (ATO), Continuous Monitoring and 3PAO Accreditation.
3.3.2 **Purpose:**
- Ensure that cloud based services have adequate information security;
- Eliminate duplication of effort and reduce risk management costs; and
- Enable rapid and cost-effective procurement of information systems/services for Federal agencies.

3.4 **MEGHNAD:**

3.4.1 **Description:**
MEGHNAD [23,24] is a Java based framework to estimate security coverage for different type of service offerings. This framework tested for various cloud service security requirements, it has 100 security tools from six defense levels. It is primarily useful for cloud insurance providers in offering a security package meeting customer security expectations while satisfying all service requirements.

The optimization method used in the MEGHNAD is scalable to accommodate tool specific configuration settings, and customer-required standards and compliances to make a fine-grained coverage estimate.

3.4.2 **Functionality:**
- Builds repositories of security tools available in the market and evaluate their features, strength, and weakness in order to calculate security index.
- Applies an intelligent search and optimization technique, a multi-objective genetic algorithm which works as a specialized Cloud Doctor in prescribing security toolsets based on the cloud services and the level of security assurance required.
- Provides a visualization interface to analyze results and detailed reports on recommended security tools, their costs and estimated coverage.

3.4.3 **MEGHNAD Architecture:**

The MEGHNAD Tool uses a pool of feasible solutions as a result of evolutionary search, with ranking based on overall weighted fitness score and individual objective. The Tool has Expert Input Panel, Operator Input Panel and a Results Analyzer. The products/tools are arranged from left to right, sorted according to the coverage index. The lines passing through tools at various levels represent the selected solution. MEGHNAD, coverage estimation approach not only uses local experts input but also considers other reviews and ratings.

3.5 **PROBLEM CLARIFICATION:**
The above-mentioned tool and other similar tools provide compliance and security for different cloud models. All the current existing Tools provide a framework that fulfill all the requirements for the particular sector compliance but do not filter the requirements that are necessary based on individual service model. The main problem is when relying on cloud computing partners many organizations tend to overlook the importance of finding the right compliance vendor and the responsibilities they bear for ensuring ongoing compliance with appropriate sector.

Solutions to these problems are to make the right vendor selection with complete knowledge of what requirements to be requested and select the appropriate compliance vendor.

4. **PROPOSED APPROACH**
A Java based framework will be developed based on different service models, which describes the controls between vendor and customer. An algorithm will be developed to map the security model and compliance requirement into different categories based on the „who controls what matrix?“. In this framework the user need to select the security compliance with the cloud model to generate the mapped checklist of compliance requirements. Finally the framework will generate a Service level agreement document for the customer to use as a guide to select the appropriate vendor and also use as an agreement between the vendors to satisfy the compliance. This framework will also provide the companies a tool to test risk and compliances against cloud service providers.

Following features will be available in the framework:
1. Authentication for different customers
2. Service and Compliance level requirement details
3. Company Details with SLA criteria
4. Security compliance requirement details based on compliance and model selected
5. SLA Document in PDF to download.

4.1 Authentication module
An authentication module will be added to accommodate the different organization login. The organization user can login using id and password

4.2 Input Details
The information related to the Organization structure, area of sector, services, SLA criteria and compliance details act as an input to the system.

4.3 Mapping algorithm
An algorithm will be developed to map the security model and compliance requirement into different categories based on the „who controls what matrix?“, described in section 2.2 Table 1.

4.4 Result Details
4.4.1 Generate Security Compliance Details
This will generate the mapped checklist of compliance requirements based on the cloud security model and Compliance.

4.4.2 SLA Document
This will generate a Service level agreement document for the customer to use as a guide to select the appropriate vendor and also use as an agreement between the vendors to satisfy the compliance in PDF format.

4.5 Integration with MEGHNAD
An extension to this framework is a phased approach to integrate the Compliance tool in MEGHNAD. This framework once integrated will increase the Security service recommendation that will be provided to the Customers. Based on the companies requirements it will provide a security control baseline and Compliance requirements recommendations associated with the service model and service sector. This will be incorporating the security insurance as part of SLA. The cloud providers can use this tool to inspect or test their services offered to different compliance sectors and test the security baseline they offer can.

This Tool will be able to provide following enhanced features:
- Based on Companies needs provide different areas of compliance requirements recommendations associated with the service model and service sector.
- Leveraging MEGHNAD, it can determine the security level of coverage.
- It can provide prescription for different security tools to establish appropriate level of security baseline.

Compliance Requirements Vs. Service Mapping:
The work involved understanding the concepts of Cloud Computing, security Models and Compliances. This involved extensive research on the subject. As a first step gathered requirement for PCI DSS and HIPAA, secondly mapped those requirement in a matrix based on „Who controls what matrix?„. This was achieved manually, which have produced the steps to develop an algorithm to provide as base to test the framework.

- **PCI DSS Requirement Mapping matrix:**

<table>
<thead>
<tr>
<th>Requirement</th>
<th>SAAS</th>
<th>PAAS</th>
<th>IAAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security Management Process</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Assigned Security Responsibility</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Workforce Security</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Information Access Management</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Access Control</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Audit Control</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**USABILITY**
This tool will generate a SLA document to test the required requirements for selected sector and the service model. It will benefit the Vendor to prepare the service agreement and convince the customer or organization. This can also be used by organizations to self asses what requirements or services they will need before approaching the Cloud vendor.
SUMMARY
Cloud Computing provides services as per usage and service models, the more customer delegate the IT infrastructure to cloud it becomes increasingly important to continuously monitor Quality of Service (QoS) as the data security and compliance is a continues ongoing responsibility. Therefore finding the right service provider with required terms in SLA Document is the backbone of any service. The importance of this work is that the customer gets a clear idea of what services requirements need to be requested in order to be compliant in their business sector. Also the vendors can use this document to persuade the customer with right service they need in a cost effective way.

This project tool will categorize the compliance requirements that need to be requested from the vendor based on the service model for different industry standards such as PCI DSS, HIPAA, SOX, GLBA, and FISMA etc. This framework will also help the companies to test risk and compliances against cloud service providers and also generates a SLA document that provides security compliance requirements terms to be requested from the vendor.

REFERENCES
Dynamic Systems Security Testing using Function Extraction

Alan R. Hevner
College of Business
University of South Florida
Tampa, FL 33620
ahhevner@usf.edu

Richard C. Linger
Cyberspace Sciences and Information Intelligence Research Group
Oak Ridge National Laboratory
Knoxville, TN 37831
lingerr@ornl.gov

Abstract— We describe an approach for applying Function Extraction (FX) technology to the dynamic testing of security in large-scale, operational software systems. FX is used proactively as an intrusion detection and prevention system (IDPS) within a security infrastructure surrounding the operation of a critical software system. An innovative aspect of the FX approach is the concept of computational security attributes (CSA). The CSA approach to software security analysis provides theory-based foundations for precisely defining and computing security attribute values. The translation of a static security property expressed as an abstraction of external data to the dynamic behavior of a program expressed in terms of its data and functions is a key to the CSA approach for verification of behaviors that meet specific security properties. The paper concludes with a discussion of future research and development directions for applying FX to dynamic security testing.

Keywords-Systems security testing, Intrusion detection, Intrusion prevention, Function extraction, Computational security attributes

I. INTRODUCTION

Recent statements by the FBI’s top cyber cop, Shawn Henry, have highlighted the immense challenges of security in critical computer systems and data networks (Barrett 2012). Current public and private security approaches are unsustainable. Henry states, “I don’t see how we ever come out of this without changes in technology or changes in behavior, because with the status quo, it’s an unsustainable model.” (emphasis added)

Our research on the emerging technologies of Function Extraction (FX) for software system understanding and analysis provides a paradigm change that can result in new ways of thinking about the security testing of software systems. The objective of FX is to compute the behaviors of software systems to the maximum extent possible with mathematical precision. FX presents an opportunity to move from the current range of slow and costly security testing processes to fast and cheap computation of system behaviors, including behaviors related to security, at machine speeds. Because a principal objective of testing is to validate system behaviors and qualities, automated computation can be expected to streamline testing processes and permit increased focus on system-level issues such as security and sustainability (Pleszkoch et al. 2008).

In this paper, we propose an approach for applying FX technology to the dynamic testing of security in large-scale, operational software systems. FX is used proactively as an intrusion detection and prevention system (IDPS) within a security infrastructure surrounding the operation of a critical software system. An innovative aspect of the FX approach for IDPS is the concept of computational security attributes (CSA). The CSA approach to software security analysis provides theory-based foundations for precisely defining and computing security attribute values (Walton et al. 2009). The translation of a static security property expressed as an abstraction of external data to the dynamic behavior of a program expressed in terms of its data and functions is a key to the CSA approach to verification of behaviors that meet specific security properties.

The paper concludes with a discussion of future research and development directions for applying FX to dynamic security testing.

II. FUNCTION EXTRACTION

Software behavior computation begins with the observation that sequential programs can be regarded as rules for mathematical functions or relations. That is, programs can be treated as mappings from domains to ranges (Linger et al. 1979).

For illustration, consider the following simple program fragment operating on small integers. It takes in values for a, b, and c (used but never set) and produces values for r, s, t, w, x1, and x2:

\[
\begin{align*}
  r &:= b \cdot b \\
  s &:= a \cdot c \\
  t &:= 2 \cdot a \\
  w &:= \sqrt{r - 4 \cdot s} \\
  x1 &:= (-b + w) / t \\
  x2 &:= (-b - w) / t
\end{align*}
\]

Final values for r, s, and t are self-evident, and final values for w, x1, and x2 can be easily composed through successive algebraic substitution:
The objective of FX is to compute the behavior of software with mathematical precision to the maximum extent possible. As noted, sizable programs can contain a massive number of execution paths; however, they are constructed of a finite number of nested and sequenced control structures, each of which makes a knowable contribution to overall behavior. These structures correspond to mathematical functions that can be computed in a stepwise process that traverses the finite control structure hierarchy. At each step, procedural details are abstracted out, while net effects are preserved and propagated in extracted behavior.

FX technology is initially being implemented for programs written in or compiled into Intel assembly language, with a current emphasis on malware analysis (Pleszkoch and Linger 2004) and use of computed behaviors to augment or replace certain forms of testing (Pleszkoch et al. 2008). The focus of malware analysis is on the use of computed behavior to unravel complex program logic and remove control flow and no-op block (code with no functional effect) obfuscation inserted by intruders, followed by computation of the behavior of the remaining functional code.

The overall architecture of an FX system is depicted in Figure 1. The starting point is a definition of the functional semantics of the programming language. The Function Extraction process generally proceeds from transformation of an input program into functional form, followed by structuring and behavior computation. There is a lot more to the process than this, but the figure provides a notional view of the major steps involved.

**Figure 1: The Function Extraction process**

Function Extraction has potential for widespread application across the software engineering life cycle, as discussed in (Collins et al. 2011). At this point, FX is an emerging technology that can be built out for many wide-ranging evaluation and operational applications.
III. FX FOR INTRUSION DETECTION AND PREVENTION

With the use of FX technology, an opportunity exists for systems testing and customer acceptance testing to shift from defect detection to certification of fitness for use. In particular, we highlight the application of FX for the purposes of intrusion detection and prevention in operational software systems. Intrusion detection is the process of monitoring dynamic events in a system and analyzing them for indications of violations or threats of violations of security policies, acceptable use policies, or standard operating procedures for security practices (Scarfone and Mell 2007). An intrusion prevention system works with the detection mechanism to proactively react and attempt to stop security violations. Here we outline FX as an element of an intrusion detection and prevention system (IDPS). A key aspect is use of FX for the analyses of Computational Security Attributes as described in (Walton et al. 2006).

IV. COMPUTATIONAL SECURITY ATTRIBUTES (CSA)

Fast and reliable analysis of security attributes is vital for every sector of our software-dependent society. For example, access to enterprise applications and data must be restricted to those who can provide appropriate proofs of identity. Applications and data must be protected so that attempts to corrupt them are detected and prevented. Healthcare systems must protect personal data while allowing controlled access by authorized personnel. Enterprises must be able to demonstrate that every accounting change is auditable. The flow of data through enterprise applications and the flow of transactions that drive the data must be logged and reported as proof of what actually happened.

In the current state of practice, security properties of software systems are typically assessed through labor-intensive evaluations by security experts who accumulate system knowledge in bits and pieces from architectures, specifications, designs, code, and test results. Ongoing program maintenance and evolution limit the relevance of even this hard-won but static and quickly outdated knowledge. When systems operate in threat environments, security attribute values can change very quickly. To further complicate matters, security strategies must be sufficiently dynamic to keep pace with organizational and technical change.

A fundamentally different approach recognizes and leverages the fact that the problem of determining the security properties of programs comes down in large measure to the question of how the software behaves when invoked with stimuli intended to cause harmful outcomes. Because security properties have functional characteristics amenable to computational approaches, it is appropriate to focus on the question “What can be computed with respect to security attributes?” The computational security attribute approach provides a step toward a computational security engineering discipline. The ultimate goal is to develop and describe mathematical foundations and their engineering automation to permit:

- rigorous specification, evaluation, and improvement of the security attributes of software and systems during development,
- specification and evaluation of the security attributes of acquired software,
- verification of the as-built security attributes of software systems, and
- real-time evaluation of security attributes during system operation.

While analysts have often characterized many security attributes as “non-functional” properties of programs, it turns out that they are in fact fully functional and thereby subject to FX-style automated analysis. Complete definitions of the required behavior of security attributes of interest can be created based solely on data and transformations of data. These definitions can then be used to analyze the security properties of programs. Thus, as illustrated in Figure 2, computational security attribute (CSA) analysis consists of three steps (Walton et al. 2009):

1. **Define required security behavior.** Specify security attributes in terms of required behavior during execution expressed in terms of data and transformations on data.
2. **Calculate program behavior.** Apply function extraction to create a behavior catalog that specifies the complete “as built” functional behavior of the code.
3. **Compare program behavior to required security behavior.** Compare the computed behavior catalog with required security attribute behavior to verify whether it is correct or not.

Requirements for security attribute behavior must explicitly define expected behavior of code in all
circumstances of interest. Thus, the requirements for security attribute behavior must include a minimal definition of required behavior for all inputs of interest to the security attributes, including desired inputs (for example, an authenticated user id) and undesired inputs (for example, an unknown user id). Usage environment conditions related to security attributes are specified in the same manner as inputs to the system. For example, availability of the network might be specified by a Boolean value that indicates whether or not the network is currently available. Security successes and failures are also specified in terms of data. For example, system control data can be used to indicate whether the current user has been authenticated using a trusted authentication mechanism.

Verification that a security property is satisfied requires verification of both the data at rest (i.e., the control data values) and the data in motion (i.e., the mechanisms used to perform the data transformations). Some common tasks to verify data at rest include checking to make sure that a specific task (for example, an audit task) will always be carried out to validate the contents of a specific control data structure. Advantages of this approach to security attribute verification include the use of constraints and boundary conditions that can make any assumptions explicit. People and process issues can be handled by the CSA approach by using assumptions and constraints as part of the behavior catalogs. Behaviors can embody requirements for a given security architecture. The attribute verification process will expose security vulnerabilities, making it easier to address evolution of code, environment, use, and so forth.

The CSA verification process can provide important opportunities for improved acquisition and third-party verification. A “user” of a system might be a person, a device, or a software component. The user may be the intended user or may be an unexpected and/or hostile user. An issue that must be considered with commercial off-the-shelf (COTS) products and reuse is that the definition of “user” embodied in the security behavior requirements may not be the same definition that was employed in the COTS or reused component. The same issue occurs when unknown components are employed as “black boxes” in systems of systems. If, in the composition of components or systems, it doesn’t matter what a specific “black box” component does with respect to security attribute requirements, then that component can be used. However, if the behavior of a component does matter, it cannot be used until its security attributes have been verified. In this case, a behavior catalog can be calculated for the component using its executable, even if documentation and source code are not available. Only externally observable behaviors are of interest to security attribute analysis. Thus, while the behavior catalog will have to be produced for the entire system in order to extract the externally observable behaviors, there is no need to expose the algorithm or source code, and there’s no need to understand the entire state space.

V. CSA Exemplars

Security properties are fully functional and are dependent on the execution behavior of software. We briefly describe seven security attributes to illustrate the range of CSA analyses that can be performed via the use of FX. Three of these attributes (confidentiality, integrity, and availability) are important to information. The other four attributes (authentication, authorization, non-repudiation, and privacy) relate to the people who use that information. The behavioral requirements for each of these attributes can be completely described in terms of data items and constraints on their processing. The processing can be expressed, for example, as logical or quantified expressions or even conditional concurrent assignments, which can be mechanically checked against the calculated behavior of the software of interest for conformance or non-conformance with CSA requirements. A fuller discussion of these CSA analyses can be found in Walton et al. (2006, 2009).

- **Authentication**: Authentication requires that a trusted user has been bound to the behavior. That is, the system will only allow the program to be executed if the user has previously been determined to be a trusted user. To verify authentication, one must examine the net effects on the control data related to authentication: verify the data that provides evidence that the binding took place, and verify that this evidence data was not changed before completion of any operation that required authentication.

- **Authorization**: Authorization requires that a user has the right to perform the requested process. To verify that an authorized operation took place, one must examine the net effects on the control data to verify that it provides evidence that authorization occurred before the operation, and that the evidence data for the authorization was not changed before that operation completed.

- **Non-Repudiation**: Non-repudiation of data transmission requires that neither the sender nor the recipient of the data can later refute his or her participation in the transaction. Non-repudiation of changes to a dataset requires that the means for authentication of changes cannot later be refuted. For the purposes of this discussion we treat data change as a special case of data transmission, where receipt of the data transmission includes making and logging the requested change to the dataset. To verify non-repudiation one must examine the net effects on the control data related to non-repudiation.

- **Confidentiality**: Confidential data access or confidential data transmission requires that unauthorized disclosure of one or more specific data items will not occur. Confidentiality is often described in terms of a security policy that specifies the required strength of the mechanisms that ensure that the data cannot be accessed outside the system. For example, the security policy may require
verification that approved encryption mechanisms are used for the output. To verify confidentiality, one must examine the net effects on the control data related to confidentiality.

- **Privacy**: Privacy requires that an individual has defined control over how his/her information will be disclosed. To verify privacy, one must examine the net effects on the control data related to privacy.

- **Integrity**: Integrity requires that authorized changes are allowed, changes must be detected and tracked, and changes must be limited to a specific scope. Integrity is defined as a property of an object, not of a mission. To verify integrity, one must examine the net effects on the control data related to integrity. That is, one must be able to: isolate the object, isolate all the behaviors that can modify the object, detect any modifications to the data, and ensure that all transformations of the data across the object are within the pre-defined allowable subset.

- **Availability**: Availability requires that a resource is usable during a given time period, despite attacks or failures. To verify availability, one must examine the net effects on the control data related to availability. To avoid having to consider temporal properties, one can specify non-availability rather than availability (i.e., specify under what conditions the program’s behavior catalog do not apply).

VI. RESEARCH STATUS AND FUTURE DIRECTIONS

Computational security attribute (CSA) analysis is a step toward a computational security engineering discipline. It can potentially transform systems security engineering by rigorously defining security attributes of software systems and replacing or augmenting labor-intensive, subjective, human security evaluation. Advantages of the CSA approach include the following:

- A rigorous method is used to specify security attributes in terms of the actual behavior of code and to verify that the code is correct with respect to security attributes.
- The specified security behaviors can provide requirements for security architectures.
- Traceability capabilities can be defined and verified outside of the automated processes.
- Vulnerabilities can be well understood, making it easier to address evolution of code, environment, use, and users.
- The use of constraints provides a mechanism for explicitly defining all assumptions.

CSA technology addresses the specification of security attributes of systems before they are built, specification and evaluation of security attributes of acquired software, verification of the as-built security attributes of systems, and real-time evaluation of security attributes during system operation.

Our future directions include the development of prototype automation to support application of CSA technology. This automation will be based on a vision of human-computer interaction that would complement and amplify human capabilities for reasoning about software security attributes during systems development and for real-time evaluation of a system’s security attributes during operation. These tools will be constructed in accumulating increments to maximize earned value and minimize risk.

CSA supports a usage-centric evaluation of security attributes that can explicitly consider the objectives and constraints of specific execution environments. Such an approach will support modeling, analysis, and evaluation of the security attribute values of software, as constrained by the policies of specific execution environments. In order for this approach to be widely used, tools are needed to support user input and query of security requirements, including automatic mapping of the model of user-specified acceptable function calls and safe behavior to the code’s behavior catalog. The ORNL FX project is developing tools that will be used to compare behavior catalogs. These FX tools, combined with the CSA approach and proposed CSA tools, will support security analysts in the comparison of security attribute requirements and constraints with behavior catalogs, thus providing a mechanism for automated security attribute analysis.

ACKNOWLEDGMENT

We gratefully acknowledge our collaborators at the Oak Ridge National Laboratory in this research.

REFERENCES


Mark L. Gillenson, Ph.D., University of Memphis
Jasbir Dhaliwal, Ph.D., University of Memphis
Son Ngoc Bui, University of Memphis

Abstract
Since data is a central element of every information systems environment, it follows that data and its management is crucial in software testing efforts. In fact, there are a variety of kinds of data that either enter into the software testing process or are created during software testing. These include databases that the software under test (SUT) will access in its eventual production environment, the test cases created to test the software, and the data generated by the defect management system during testing. Taking a broader view of “data”, it includes requirements, systems analysis diagrams, and systems designs diagrams and specifications. If the SUT is an upgrade of an existing application, the data includes the historical production data that has been saved and the regression test case suite that has been saved for future “smoke tests.” This paper presents a conceptual framework for developing data design and management techniques in the software testing environment. It then proposes a new indexing system for tracking and cross-referencing the various kinds of data in the test environment.

1. Introduction

As information systems pervade every aspect of commerce and daily life, the expectations of the software behind them performing according to plan become ever more important. This means that the software must accomplish the functional goals set for it, must execute as quickly as expected, must be secure, user-friendly, and, in many cases, make sense in a wide range of languages and countries. Furthermore, in today’s complicated and intertwined systems, the software must often be interoperable with a variety of other software systems and applications. Developing software is a complex process and in order to produce software that works properly, the software must be tested, corrected, and tested again. The point of all of this for this paper is that the increasing demands placed on software require increasing sophistication in the software testing effort. This is an even more acute issue due to industry's emphasis on financial performance which continues to limit the resources available to the software testing effort.

There are many aspects to software testing, including test planning, developing test cases, executing the test cases and tracking defects that are discovered, monitoring the correction of the defects, developing testing process metrics and associated reports, and managing testing personnel, among others. An underlying feature of virtually all of these software testing aspects
is data and there are actually a wide variety of types of data involved. Some types of data involved in testing, such as lists of requirements, come directly from the software development effort and become part of the testing milieu when they have to be tested. Other types of data involved in testing, such as test cases, are developed directly as part of the testing effort. This paper will take a broad view of data in software testing. It will list and describe all of the different kinds of data in the testing environment, discuss their development, their design, their management, and their importance in the software testing effort.

2. Data in Testing

In order to list and discuss the types of data in testing, we must first establish the bounds of what we mean by testing in the software development environment. For a particular testing effort, these bounds are set forth in the test plan (D1)\(^1\) (Craig & Jaskiel, 2002; "IEEE Standard for Software and System Test Documentation," 2008) and so the test plan, itself, becomes the first piece of data in the testing effort. Traditionally, testing has meant the testing of the code produced, i.e. the testing of the end product of the development effort. We take a broader view of testing that begins with testing the requirements (D2) set forth for the application. Following this approach, the list of application requirements, the systems analysis diagrams and documentation (D3) derived from the requirements, and the systems design specifications (D4) derived from the systems analysis work all become testable data.

Moving on, the program code (D5) produced in the software development project is itself a form of data. In fact, it is critical to track the different versions of the software as it goes through testing and refinement. Most software applications are designed to read from and write to databases of various kinds. By definition, the data in these databases (D6) is a kind of data required in the software development process. Furthermore, testing the code generates several kinds of additional data. To begin with, these include the test cases (D7) created to test, respectively, the functionality of the code, its performance, its security or vulnerability, its usability, and such additional issues as its appropriateness for different countries in the international arena. Key functionality test cases are grouped together and stored as a regression test suite (D8). These can then be used in the future to provide confidence, in so-called "smoke tests" that the software still works properly as it is upgraded and modified in the future (Steve, 1996). As the code is tested and defects are found, an array of data is produced as part of the defect management process (D9).\(^2\) It then follows that the metrics (D10) produced from the defect management process are another level of data, as are the reports to management (D11) incorporating the metrics that summarize the progress of the development and testing efforts.

---

\(^1\) For easy reference in this paper we will label each type of data as Dx.

\(^2\) Note: If defects found in the requirements, systems analysis artifacts, and systems design artifacts are not handled in the defect management process the same way that defects in code are handled, then defect reports created for these artifacts become yet another kind of data in testing. For convenience in this paper we will assume that all such defects are handled by the defect management process.
3. Data Design in Testing
While it is not the purpose of this paper to go into detail about how to design each of the types of data listed in Section 2 above, it will be instructive to state and briefly describe the appropriate design methodologies, particularly for the data, such as test cases, that are created as part of the testing process. First, the design of the test plan should follow the well-established IEEE Std. 829, versions of which were published in 1998 ("IEEE Standard for Software Test Documentation," 1998) and 2008 ("IEEE Standard for Software and System Test Documentation," 2008). At the time of writing this paper, the 2008 version is being further updated (Reid, 2012). Requirements (D2) are often written by the user department requesting the application, but in large enough organizations may be written or refined by dedicated requirements writing personnel. Requirements should specify the main purpose, the “critical path,” of an application, as well as subordinate purposes and significant exception conditions (Abhijit Chakraborty, Mrinal Kanti Baowaly, Ashrafu Arefin, & Bahar, 2012; Craig & Jaskiel, 2002). Systems analysis artifacts (D3), created by professional systems analysts, normally include such standard diagramming techniques as data flow diagrams (Craig & Jaskiel, 2002; Sharma & Vishwakarma, 2012) and supporting documentation. Systems design artifacts (D4) for large-scale new applications may be in the form of a hierarchy of subsystems and, ultimately, of programmable modules (Basu, Bensalem, Bozga, Bourgos, & Sifakis, 2012; Craig & Jaskiel, 2002; Hinchey & Coyle, 2012). These are created by chief or lead programmers, in preparation for coding. Additional or alternate documentation is needed if the programming effort is directed towards updating or modifying an existing application.

The program code (D5) has, in effect, already been designed in the systems design step. Beyond that, organizations may have programming standards that they expect their programmers to follow when writing code. The standard procedure for database design (D6) is to develop entity-relationship diagrams, derive logical database structures from them, and then modify the logical design for performance and security purposes to obtain the final physical design (Hansen, 2003). Furthermore, the logical design can be tested with standard data normalization techniques to assure the avoidance of data integrity issues. A further issue regarding the data in these test databases is privacy. Since it is generally assumed that test environments are less secure than production environments, standards and regulations have been developed to protect the privacy of data in databases used in testing. An example of this is the Payment Card Industry Data Security Standard (PCI DSS) ("Payment Card Industry Data Security Standard Requirements and Security Assessment Procedures," 2010). Other relevant standards and regulations include the U.S. Gramm-Leach-Bliley Act, the European Union Personal Data Protection Directive, and the Japanese Data Protection Directive, among others (Gramm–Leach–Bliley Act, 1999; Protection of Personal Data, 1995; Protection of Personal Information, 2003)
A critical element in systems testing is the design of test cases (D7). As we said earlier, different test cases must be designed for several different purposes. First and foremost are test cases designed for functionality testing which test whether the software performs the application functions specified in the requirements (Craig & Jaskiel, 2002; Khel, 2012). Such test cases, which can be designed to test code ranging anywhere from individual code modules all the way through to the complete application, consist of a set of input variable values and the expected output when those values are executed in the appropriate program code. There are three major ways that functionality test cases can be designed. One is directly from requirements (or associated artifacts known variously as use cases, scenarios, or user stories.) In this instance, a test case is created to directly assess whether the program code satisfies the stated requirement. The second method of functionality test case design is through several heuristic methods that comprise developing test cases for “black box” testing (Craig & Jaskiel, 2002; Leonardo, 2012; Vos et al., 2012). This method begins with recognizing that the output from a piece of software can fall into one of several “equivalence classes” based either on specific outcomes comprising equivalence classes or based on specific calculations in the software comprising equivalence classes. The idea then is to have a representative test case for each equivalence class. This leads to the next step, known as “boundary value analysis”, which recognizes that software defects often occur at the boundary between adjoining equivalence classes, necessitating test cases at and near such boundaries. Other tools and techniques for creating test cases for black box testing include “domain analysis”, the use of decision trees, “pairwise analysis”, and the use of state transition diagrams (Kashyap, Roberts, Sarkani, & Mazzuchi, 2012; Qureshi, 2012; Tai & Lie, 2002). The third method of functionality test case design is “white box” testing (Craig & Jaskiel, 2002; Khel, 2012). This entails developing test cases by tracing paths through the program code itself with techniques such as “control flow analysis” (Craig & Jaskiel, 2002; Khel, 2012). Performance testing requires the generation of very large numbers of test cases in which the outputs are generally of no concern (Craig & Jaskiel, 2002). These test cases can be generated by a program written for this purpose based either on random number generation or on what commonly expected input values will be for the software once it is in production. They can also be taken from production history if the software being tested is an upgrade of existing software. They can even be massively reproduced test cases that were created for black box testing. Finally, test cases for security testing, usability testing, and internationalization testing are created to test those specific issues. For example, an expert in software vulnerability should be engaged to create test cases that will probe the software for typical security flaws. Finally, regression test suites (D8) are designed, in the sense of deciding which functional test cases to include in them, based on the criticality of the individual test cases. For example, test cases that are designed to test the critical path requirements of the software should be included in the regression test suite. Alternatively, or in addition, the full set of test cases generated by pairwise analysis may be included. Note that as the software in question is upgraded and modified in the future, the regression test suite may well have to be modified in tandem.
The data in the defect management process (D9) consists of details about the defect, including its description, the test case that caused it, the person who discovered it, and on to data relating to its being fixed. The design of this data is spelled out in a variety of sources including the IEEE Std. 829 ("IEEE Standard for Software and System Test Documentation," 2008). Furthermore, there is a wide variety of defect management software products available for purchase, such as HP Quality Center ("Serena HP Quality Center Integration," 2010), and open source software such as Bugzilla ("Bugzilla," 2012), each of which has its own variation on defect management data design. Accumulated defect management data can be organized or "designed" into several very useful metrics (D10) (Craig & Jaskiel, 2002). These begin with the number of defects found during testing, or better yet during each stage of development and test, and the number of defects found in production. The Defect Removal Efficiency, also known as the Defect Detection Percentage, is the number of defects found during testing divided by the sum of the total number of defects ever found (i.e. in test and production.) The Defect Spoilage is a weighted measure that tracks when defects were found relative to when they were created. Defect Density is the number of defects divided by the size of the software as measured by the number of lines of code or the number of function points. The Defect Coverage is the number of test cases produced for functionality testing relative to the number of requirements. The design of the storage of all of these defects can be as simple as a table or a spreadsheet. Finally, reports to management (D11) can be free-form in design, containing an overall description of the testing effort, including the generated metrics.

4. Data Management in Testing

We assert that there are certain commonalities in the management of all eleven of the types of testing data listed and described. If an artifact is important enough to be part of the testing process, it is important enough to be assigned a unique identifier, a name, and a textual description. Furthermore, it should be easy to find by anyone who has a legitimate need to find it. With this as an initial statement, we next go on to look at the management of each of the eleven kinds of data we have identified.

Traditionally, the management of the eleven different kinds of data has been handled independently and in a variety of different ways, if, indeed, they have been carefully managed at all. Test plans (D1) are typically just handled as ordinary documents, although in more sophisticated environments they are given unique identification numbers (Bauer & Finger, 1979; Craig & Jaskiel, 2002). Requirements (D2) management can range from the very informal to the formal using software designed for this purpose (Bourque, Dupuis, Abra, Moore, & Tripp, 1999; Craig & Jaskiel, 2002). Systems analysis (D3) and systems design (D4) artifacts are most often held informally but may be stored using specially designed software for this purpose (Craig & Jaskiel, 2002; Sharma & Vishwakarma, 2012). Program code (D5) should be managed in a reasonably sophisticated system that tracks versions of the code (Craig & Jaskiel, 2002; Hangal
Database test data (D6) is, of course, managed by a database management system. The crucial point is to name the databases appropriately in order to keep track of them as test databases and not confuse them with production databases.

All too often, test cases (D7) are handled informally or in groups. We believe that as key elements of the testing process, test cases should be uniquely identified and very carefully managed, both individually and when incorporated in regression test suites (D8) which should also be uniquely identified. Defect management data (D9) is managed in specially designed defect management software of which there are numerous examples available for sale (e.g. HP Quality Center) ("Serena HP Quality Center Integration," 2010) and as open source software (e.g. Bugzilla) ("Bugzilla," 2012). Metrics (D10), when they are kept, are most often stored in an ad hoc fashion, although some defect management software products are capable of storing and managing metrics they produce (Craig & Jaskiel, 2002). Finally, reports to management (D11), which can vary greatly in content and style, are text documents that are managed in an ad hoc manner.

Unfortunately, the independent, disconnected, and often informal ways in which data in testing are maintained are not conducive to a well-managed test environment. We believe that managing test data effectively is a key to having an effective test effort. It is essential to be able to store all eleven kinds of test data in well-organized manner, to be able to retrieve all of the components efficiently as needed, and to be able to relate the various elements of the test data to each other.

We propose a new system for managing the data in testing to be known as the Global Index for Test and Evaluation Data (GIFTED). The acronym is appropriate as this will truly be a "gifted" system that will greatly improve the management of the test environment and lead to more successful test and evaluation efforts. GIFTED's basic concept is simple. Suggesting that all eleven kinds of test data be integrated into a single storage schema would clearly be unrealistic. Instead, we propose a federated schema in which the eleven kinds of test data continue to be stored as they have been, but with a new, GIFTED index built over them as a unifying device. Each instance of each of the eleven kinds of data would continue to have a unique identifier (or would have to have one now, if it didn't before.) But, in addition, there would now be the GIFTED index that would be capable of referencing and seeking all of the test data. The manner of creating the index items would be to simply add the appropriate Dx designation before the test data's unique identifier. Thus, test case number 47294 would be reference in the GIFTED index as D7-47294. The index would then point to the physical location of the test data associated with its index reference.

A further benefit of GIFTED is the ability to cross-reference the various types of data in testing. It would be a simple matter to create a cross-reference table to indicate which test cases
are in which regression test suites, which test cases are derived from which requirements, which metrics apply to which program code version, and so forth. Clearly, this would be a huge advantage in managing the test and evaluation environment.

5. Conclusion

Software testing is a challenging endeavor for a variety of reasons. Resources are generally limited, expectations are broad and high, and testers are often not engaged early enough in the software development process. Another challenge is recognizing all of the different types of data involved in testing, much less being able to effectively manage them. GIFTED is a concept that takes an engineering approach to cataloging, managing, finding, and cross-referencing all of the various types of data in the software testing environment. We believe that the realization of this concept will be a major step forward in effective software testing.

References


Serena HP Quality Center Integration. (2010).


Abstract

This paper examines an important role of a specific leadership style (i.e., transformational leadership) and its attributes and behavioral characteristics that might be positively related to software testing team performance in virtual settings. In this study, we suggest a preliminary research model that could be further developed into a more specific model including mediation and/or moderation once data collection is in process. The theoretical implications call for testing managers’ attention to the importance of transformational leadership development in software testing teams in virtual settings.

I. INTRODUCTION

In the face of both heightened levels of competition and an uncertain economic environment, many organizations are adopting team-based structures to better compete and survive (Sundstrom 1999). Extending the basic form of a team-based structure, work situations in business settings have evolved such that leaders and followers increasingly collaborate with physically dispersed team members. Owing to the introduction of new communication technologies in workplaces that alleviate the barriers of time and space, these “virtual teams” have become even more attractive to organizations. Virtual teams have emerged as increasingly popular options because of the high levels of efficiency and task-oriented effectiveness that they achieve while simultaneously enhancing the flexibility of time, space, labor forces, and other resources compared to traditional format of face-to-face teams.

This trend is not unique to software testing teams. Recently, software testing teams have been faced with the demands associated with virtual teams, in addition to the core competency of information technology (IT) use required to complete their missions. The members in virtual software testing teams work independently and interdependently in a situation where they are geographically dispersed and cross-functional. Counterbalancing the strengths that virtual teams offer, leadership of virtual teams presents unique challenges to leaders due to physical distance, an innate characteristic of virtual teams (Malhotra et al. 2007). Because virtual team communication is likely to be mainly done through electronic means with minimal face-to-face interaction, effective leadership, at least as identified within a traditional setting, needs further adaption to better deal with the distance-related concerns embedded in virtual software testing teams. As yet, however, little is known about effective leadership styles in virtual teams in general and in virtual software testing teams in particular, nor have large-scale field studies been conducted in this area.

Likewise, the role of effective leadership in software testing teams has gone largely unexplored, partly because leadership is frequently seen as interchangeable with the managerial role. Research in the IT field, including software testing team situations, has often focused on very specific managerial roles, such as allocating resources, monitoring outcomes, and controlling and coordinating people and work environments. While these managerial behaviors are undoubtedly important, they encompass only a small portion of a leader’s role and focus primarily on achieving efficient operations. Of course, the role of a leader also includes motivating employees and adapting to changing conditions (Yukl 2008). Given this fact, it seems necessary to investigate the effect of a specific leadership style in software testing teams.

Thus the primary purpose of our study was to explore the influence of transformational leadership, which focuses on inspiring followers by appealing to their values and ideals, and ultimately motivating them to perform beyond expectations (Bass and Riggio 2006). Transformational leadership has been documented as one of the most influential leadership styles in leadership history. Despite such effectiveness and the high demand for transformational leadership in the field of management, to date little has been discovered about the effect of transformational leadership on software testing team-related outcomes. Therefore, identifying the effects of transformational leadership in software testing teams will add theoretical and practical implications to both the literature and practice. In examining this issue, we will consider a virtual team setting as our contextual factor.
II. THEORETICAL FOUNDATION

A. Leadership versus Management

Previous research has demonstrated that the effective use of leadership behaviors is predictive of both individual (e.g., employee performance and satisfaction) and organizational outcomes (e.g., increased productivity, increased sales) (See Bass and Avolio 1994; Dvir et al. 2002). Despite these findings, insufficient attention has been paid to the relationship between leadership and software testing groups. Research in software testing teams often focuses on very specific managerial roles, such as allocating resources, monitoring outcomes, and controlling and coordinating people and work environments, while investigating the relationship between leadership and IT outcomes (e.g., Karahanna and Watson 2006; Tan et al. 1999).

As Mintzberg indicated (2009), management comprises a series of roles that managers fulfill. These roles allow the manager to gain access to information, and ultimately allow him or her to make decisions. Software testing managers routinely have to work within complicated organizations that possess many formal and informal lines of communication. They deal with complex processes and methodologies, staffs with varying degrees of skill, a wide array of tools, budgets, and estimates. While these behaviors are undoubtedly important, they encompass only a small portion of a leader’s role. More importantly, many of the aforementioned managerial behaviors are focused primarily on obtaining efficiency by control and pushing members in the right direction. For example, the test manager, who may be recognized as simply the figurehead for the testing group, tends to focus on stability and day-to-day operation for the attainment of organizational goals in an effective and efficient manner through planning, organizing, staffing, directing, and controlling organizational resources (Kotter 2001). Whereas management focuses primarily on stability, order, and efficiency (e.g., the issue of how things get done), leadership focuses more on change, flexibility, and innovation (e.g., the issue of what things mean to people).

In addition to these managerial roles, often corporate or IT governance is regarded as an alternative role of leadership. Corporate governance is aligned with the managerial function; it is defined as managing teams by the set of processes, customs, policies, laws, and management practices that affect a way an entity is controlled and managed (Berndroter 2008). Typically, corporate governance is more focused on corporate-level effectiveness to achieve organizational goals than on team effectiveness, whereas leadership relies on the proactive and voluntary actions of followers resulting from their intrinsic motivation and willingness to achieve a common goal that is accepted as their own goal (Kotter 2001). Leaders seek to have followers not only see the importance of the goal, but also agree on the most important things to be done.

According to House et al. (1999), leadership may be defined as “an individual’s ability to influence, motivate, and enable others to contribute toward the effectiveness and success of the organization” (p. 184). The core of leadership is the ability to capitalize on followers’ motivation and willingness to achieve, which plays a critical role in achieving individual- and collective-level performance. Leaders are frequently regarded as change agents—that is, as initiating and adapting innovative changes as positive stimuli in a team context (Yukl 2006). These efforts have been seen as one of the major driving forces to elicit followers’ satisfaction with their job, commitment to their team, and ultimately better performance than had originally been expected (Bass 1985). As such, it seems prudent to investigate the role of leadership in software testing teams, where technologies are ever changing and motivating employees to effectively utilize them can be a challenge.

Based on this rationale, in the next section, we discuss transformational leadership and its effect on work outcomes in teams and organizations in detail.

B. Transformational Leadership

Investigating the effect of transformational leadership in software testing teams is important because such leadership has been validated as one of the most effective leadership styles for motivating employees to perform better and for improving group and organizational effectiveness (Judge and Piccolo 2004). Since Bass’s pioneering work (1985) on transformational leadership, numerous studies have made a great effort to examine the behavioral characteristics of transformational leaders and their positive relationships on work outcomes in organizations [see Yammarino, Dionne, Chun, and Dansereau’s (2005) review on transformational leadership].

Transformational leadership focuses on inspiring followers via their values and ideals, and ultimately motivating them to perform beyond expectations (Bass et al. 1994; Bass 1985). As a result of such leadership, followers feel trust, loyalty, and reverence toward the leader and often transcend self-interest for the sake of the group (Avolio et al. 2004). According to Bass and his colleagues, transformational leadership consists of four behavioral components: idealized influence (admirable behaviors intended to arouse follower emotions and identify with the leader), inspirational...
motivation (behavior focused on communicating an inspiring and appealing vision), individualized consideration (the degree to which a leader provides support and encouragement to followers, and coaches and mentors them), and intellectual stimulation (the degree to which a leader increases awareness and helps followers challenge assumptions and take risks). These have been shown to relate to both individual- and organizational-level outcomes. Thus transformational leadership is positively related to organizational commitment, justice perception and organizational citizenship behaviors, follower motivation, organization performance, and leader effectiveness (Cho and Dansereau 2010; Judge et al. 2004).

C. Virtual Teams

A virtual team is a common form of work structure in contemporary organizations. According to Malhotra and colleagues (2007), virtual teams are defined as “teams whose members are geographically distributed, requiring them to work together through electronic means with minimal face-to-face interaction” (p. 60). As such, in virtual team settings, it is inevitable that leaders and team members must work cooperatively and interdependently in the face of some degree of physical distance. According to Antonakis and Atwater (2002), physical distance in leadership refers to “how far or how close team members are located from their leader” (p. 684). Because physical distance among members is necessary, in some cases employees are asked to work with and for individuals whom they have never met face-to-face.

Virtual teams have been preferred in many situations due to their well-known advantages, such as efficiency, flexibility, connectivity, and task-focused project management, and the advancement of new communication technologies plays a crucial role in alleviating the barriers of time and space. At the same time, this structure creates new challenges for leaders, who could have less influence over followers due to the relatively scarce opportunities to demonstrate their effective leadership behaviors directly to those followers. Reliance on electronic communication technologies leads to a lack of visual and auditory cues (Purvanova and Bono 2009), which might otherwise help effective leaders to appeal to followers. Dissemination of messages through these communication channels could end up being impersonal and purely task-specific, which makes it difficult for members to develop emotional closeness, inspiring relationships, and social networks.

In reflecting on this theoretical prediction, previous studies have reported that physical distance has a negative effect on the quality of the communication exchange and follower satisfaction; reduces the leader’s influence and social interaction; and reduces follower performance and citizenship behaviors (Antonakis and Atwater 2002).

III. THE CONCEPTUAL MODEL DEVELOPMENT

As mentioned earlier, transformational leadership has a meaningful impact on individual- and team-level job performance as well as on contextual performance (Judge et al. 2004). One of the main driving forces for positive outcomes can be found in transformational leaders’ emotional and behavioral characteristics that appeal to followers’ values and beliefs (Bass 1985). By eliciting their followers’ emotional arousal, which needs to be in agreement with the importance and meaningfulness of their work, transformational leaders tend to inspire as well as to motivate their followers for better performance.

Although little is known about the effect of transformational leadership in software testing teams, some preliminary evidence has addressed the potential positive effect of transformational leadership in virtual communication contexts. For example, Purvanova and Bono (2009) have reported that transformational leadership has a stronger effect on team performance in virtual teams than in traditional face-to-face teams, based on their laboratory experiment involving undergraduate students (N = 301).

In keeping with that finding, we predict that transformational leaders’ attributes and behavioral characteristics can enhance virtual software testing team performance both at the individual level (turnover, process improvement) and at the group level (team loyalty, team satisfaction, team cohesiveness). Specific attributes and behavioral characteristics are as follows:

1) **Idealized influence**: Transformational leaders instill pride, faith, and respect in testing team members by establishing positive and unique norms in virtual communication (e.g., a short social networking time at the beginning of virtual meetings as a team ritual, systematic knowledge and information sharing, progress feedback, formal and verbal recognition about “a job well done,” virtual get-togethers) and through leading by example, thereby causing followers to increase their loyalty to the team and the leader.

2) **Inspirational motivation**: Transformational leaders enhance team members’ confidence not only in their ability to discover errors in a more systematic and effective way, but also in their ability to enhance the quality of the software products, by articulating an appealing vision and meaningfulness of their tasks and by expressing
high levels of expectation and optimism about the members’ ability to conduct a series of testing activities.

3) **Individualized consideration**: Transformational leaders coach or mentor team members for achieving testing effectiveness and provide individualized support, while listening to the concerns and needs of testing team members.

4) **Intellectual stimulation**: Transformational leaders stimulate team members’ creative problem-solving skills by challenging them to address old problems using new perspectives and by soliciting the members’ ideas and cooperation for the better achievement of high quality in testing activities.

These transformational leadership behaviors allow team members to feel loyalty, satisfaction, and cohesiveness (defined as the degree to which members like one another and remain united in the pursuit of a common goal) toward their team per se. In addition, such positive team spirit and perceptions help the members to increase process improvement and engage in voluntary extra-role behaviors, and to decrease turnover in the team.

Figure 1 exhibits the preliminary conceptual model of this study.

![Figure 1. The Preliminary Conceptual Model](image)

**IV. FOLLOW-UP RESEARCH**

This initial study can be further developed to identify more sophisticated research questions that espouse solid theoretical relationships between transformational leadership and virtual software testing team performance. For example, the preliminary conceptual model can be enriched by adding the potential mediators or moderators that could answer the following questions: (1) How and why are transformational leaders effective in virtual software testing teams? and (2) In which circumstances does transformational leadership lead to better outcomes?

To investigate these issues, data will be collected using interviews and surveys of team leaders and team members. After data collection, the preliminary model as well as more sophisticated conceptual models will be tested using various statistical techniques, including hierarchical regression, structural equation modeling (SEM), and multilevel SEM. Those statistical tools have been validated as powerful and effective means to assess the hypothesized relationships in the field of management.

The findings of the present study have both theoretical and practical implications. To our knowledge, this study represents the first attempt to systematically explore the effect of transformational leadership in virtual software testing teams. Therefore, the results can shed light on the importance of transformational leadership behaviors in software testing teams, and this knowledge can then be used for further theoretical grounding for future research. In addition, the findings of this study draw both management’s and testing team managers’ attention to the importance of transformational leadership development and training in virtual settings. On the one hand, management should provide transformational leadership development programs for their software testing team managers, who should be encouraged to act in ways that inspire and motivate their followers using transformational leadership behaviors. On the other hand, testing team managers should note that their team performance can be enhanced by their inspiring and individually considerate behaviors.
References


