Trustworthy Systems Manifesto

As a greater portion of mission, business, and safety critical functionality is committed to software-intensive systems, these systems become one of, if not the largest source of risk to enterprises and their customers. Since corporate executives are ultimately responsible for managing this risk, we establish the following principles to govern system development and deployment:

1. **Engineering discipline in product and process**
2. **Quality assurance to risk tolerance thresholds**
3. **Traceable properties of system components**
4. **Proactive defense of the system and its data**
5. **Resilient and safe operations**

Signatories indicate their willingness to develop policies and practices within their organizations to implement these principles, and to encourage their adoption in other organizations. The Manifesto should be used to initiate conversations about how to reduce risk to the business or mission created by software-intensive systems.

This manifesto is developed and maintained by the Consortium for IT Software Quality™ (CISQ™), a standards consortium managed by the Object Management Group® (OMG®). OMG is a member-driven, not-for-profit IT standards organization. CISQ is chartered to advance the trustworthiness of software-intensive systems by producing standards for automating the measurement of size and structural quality from software source code. CISQ conducts outreach activities to spread measures and techniques for improving the trustworthiness of software-intensive systems.

Purpose of the Manifesto

Motivation. As businesses and governments automate more of their business and mission processes, the risks to which software-intensive systems expose the organization grows dramatically. In an era of 9-digit glitches (incidents with damages over $100,000,000), senior executives outside IT are held accountable, and some have lost their jobs as a result. Since senior executives are rarely IT experts, they need guidance on how to govern the risks of untrustworthy systems.

Objective. This manifesto provides a list of five principles to guide senior executives in establishing policies to govern the development, deployment, and operation of business or mission-critical software-intensive systems. These principles jointly create an optimal environment for developing, deploying, and operating trustworthy systems. A trustworthy system is one that is secure from unauthorized users and actions, reliable and safe in its performance, resilient to unexpected conditions, and accurate in its computations. Each of the five principles is explained in the following pages. At a minimum this manifesto should be used to initiate conversations about reducing the risk to which software-intensive systems expose the business.

Implementation. These principles cannot be enacted overnight. To execute their governance responsibilities over system risk, executive management should establish policies requiring an improvement program whose status is reported at least quarterly. The program should execute a planned progression of improvements that implement the principles of trustworthiness throughout the development, deployment, and operational organizations. Since threats to trustworthiness multiply and change with new technologies and business models, the improvement program does not have an endpoint, but rather continually refocuses on new challenges. In any endeavor as complex as developing software-intensive systems, there will be tradeoffs that must be evaluated. To the extent possible, tradeoff decisions should be based, at least in part, on risk and cost analysis.

I. Engineering Discipline in Product and Process

1. **The principles and practices of software engineering must predominate other considerations in developing software-intensive systems**. Although much of software creation involves the craftsmanship of skilled developers, their work must adhere to rules of sound software architecture, design, and coding practice. These software product engineering principles are critical for developing trustworthy software that is free of severe flaws which cause damaging operational incidents or excessive ownership costs.

2. **Trustworthy systems do not emerge from haphazard development and deployment processes.** Trustworthy software-intensive systems must employ rigorous software engineering practices and methods. Frameworks from which trustworthy software development practices can be drawn include the NIST Cybersecurity Framework, IEEE 12207-2017, CMMI, COBIT, ITIL, Autosar, CERT Resilience Management Model, Scaled Agile Development (SAFe), the DevOps Handbook, and SEMAT Essence. With the growth of software-as-a-service and deployment to public clouds, software deployment and disaster recovery are increasingly software-based, requiring more rigorous infrastructure management processes such as in ITIL and IT4IT. Development processes should be periodically audited to ensure that all critical development, deployment, and operational processes are sustained and improved when necessary.

3. **The shorter the time, the greater the need for process discipline.** Modern development practices tend toward shorter development cycles. On shorter timescales, developers have little time to figure out their methods or correct voluminous mistakes. Consequently, agility requires discipline to achieve its potential for detecting mistakes earlier and delivering trustworthy system increments. Discipline is especially important for system components fast-tracked into production, since undetected defects can have severe consequences.

4. **Developers and operators must be supplemented by automated technologies that can reduce complexity and improve their visibility into systems and operations**. As the size of software-intensive systems grows exponentially, they are increasingly constructed from a stack of technologies often using interfaces between existing applications (APIs) and small reusable components (microservices). The complexity of these systems exceeds the understanding of any single developer or team. Since the IT infrastructure is increasingly built on software, manual methods are insufficient to monitor complex system operations. As the scale of interactions within and among systems grows exponentially, especially in the Internet of Things, trustworthy development will only be possible if aided by automation.

5. **Organizations must ensure that developers have the knowledge and skills needed to build and deploy trustworthy systems.** These skills include the organization's architecture and coding rules, development and deployment tools and processes, operational environment, and business domain. Developing a trustworthy workforce requires disciplined workforce practices. Suppliers should be also required to retain key personnel and staff skill levels.

II. Quality Assurance to Risk Tolerance Thresholds

1. **Executives must determine the risk that can be tolerated from each business or mission critical system**. Governments are issuing regulations such as General Data Protection Rights (GDPR) in the European Union (EU) to protect the public from cyber risks. Since it is impossible to produce flawless software in large complex systems, a trustworthy system is one that operates within the enterprise or regulator's tolerance for risk. Risk tolerance is specific to the purpose of each system. With finite limits on the time and effort available for quality assurance tasks, management must know when a system can be trusted to perform within risk tolerance thresholds. Risk can be evaluated by the cost of an outage or breach, the damage of faulty operations, the harm to an enterprise's reputation and customer loyalty, and other factors. Risk thresholds can be expressed as allowable downtime, maximum response time, number of lost transactions or customers affected, and similar outcomes.

2. **Quality assurance must ensure the system operates within risk tolerance thresholds**. All development and change processes should include quality assurance practices to determine if a system operates within threshold limits. Initially developers will only have an estimate that the system operates within risk thresholds. With continued use, developers can calibrate their development and quality assurance methods to assure these thresholds are achieved. These performance measures should be regularly reported to the business. Traditional testing primarily assures that system functions deliver correct results. Trustworthiness requires additional methods such as penetration testing for security, static analysis for structural integrity, stress and load testing for performance, user acceptance testing for validation, and usability testing for customer experience.

3. **Executives' policies should require that critical systems have evidence they can perform within risk thresholds before being released to operations**. The required evidence includes test case results, static and dynamic evaluation, and other demonstrations of threshold performance. The most frequent threat to risk tolerance thresholds is pressure to release systems on truncated schedules without sufficient time for testing and evaluation. If executives rush a system into operation with defects that exceed the risk threshold, they must accept responsibility for the additional risk created.

4. **Executives must protect time for remediating high priority defects**. Business and mission critical systems should maintain a backlog of defects to fix in priority order based on the severity of their impact on risk tolerance thresholds. Time for repairs is frequently sacrificed to demands for additional functionality from business or mission executives. A remediation policy should require a minimal proportion of effort devoted to remediation or refactoring in each sprint, release, or other time-based interval. If high priority defects are not eliminated, trustworthiness declines with the deteriorating quality of the software.

III. Traceable Properties of System Components

1. **Developing modern software-intensive systems requires managing a supply chain of component sources**. These sources include in-house developers, captive centers overseas, software vendors, outsourced developers, open source systems and libraries, software tool providers, and others. This broad, often opaque supply chain of system components involves numerous risks such as:
    - untrustworthy components,
    - components with missing features,
    - counterfeit components,
    - unpatched vulnerabilities, and
    - violated software licenses

    Trustworthy supply chains require time to establish since policies and practices must be spread across suppliers. Nevertheless, the enterprise deploying the system owns final responsibility for the trustworthiness of system components and violated software licenses.

2. **Evidence of provenance and trustworthiness should be carried forward with components and shared across the supply chain**. Component supply chains can be opaque, in that it is not clear where components originated, how they have been modified, or whether they have severe flaws. Evidence of trustworthiness, provenance, authenticity, license compliance, known vulnerabilities, and related matters should be prepared by each member of the supply chain producing or modifying system components. When developers incorporate open source components, external APIs, or microservices, they should document their source and related data for inclusion in a System Bill of Materials (SBOM). Where possible, evidence of provenance, trustworthiness, and patched vulnerabilities (CVEs from the Common Vulnerability Enumeration) should be audited to ensure accuracy and completeness.

IV. Proactive Defense of the System and Its Data

1. **Cybersecurity begins with an assessment of the attack patterns relevant to the system's purpose, environment, and use**. Protecting business or mission-critical systems starts with understanding cybersecurity issues unique to the system. These issues can include:
    - business or mission purpose of the system and how it will be used,
    - motivation for attacks and types of attackers,
    - probable attack patterns and surfaces,
    - risks in the system's operational environment, and
    - vulnerabilities and unremediated weaknesses

2. **Protection of the system and its data from malicious actors requires multiple layers of defense**. In a highly mobile, connected world, system perimeters are increasingly ambiguous. Cybersecurity plans and practices must not only protect the internal networked environment, but also business or mission assets that communicate or connect with it. Each business or mission-critical system and its connection to other systems should be protected from unauthorized access. Data transferred between systems should be vetted to ensure their trustworthiness and compatibility. Because malicious actors may have already penetrated a system, security-sensitive architectural layers within the system and data housed in them should be protected from unauthorized access and manipulation from other system layers. Layered security involves using multiple methods to defend different components of the system and its environment against different types of threats. The trustworthiness of interconnected systems and the environment in which they operate requires an effort beyond that devoted to the trustworthiness of constituent systems.

3. **System behavior should be continuously monitored to detect suspicious actions and data movements**. Security experts warn that the only safe assumption is that malicious actors have already penetrated perimeter defenses and are inside a system. Many malicious actors are sophisticated in avoiding detection once they have penetrated a system, so internal system security must be vigilant and sensitive to suspicious patterns over extended periods. The movement or modification of data should be monitored for suspicious or malicious patterns.

4. **Security practices must also cover the behavior of authorized system users to ensure system defenses are not circumvented**. Many unauthorized penetrations result from insecure user behaviors such as sharing passwords, leaving passwords on exposed notepads, succumbing to phishing attacks, or exposing confidential data on printouts or screens. Users should be trained in secure user behavior, and their use monitored for suspicious patterns.

V. Resilient and Safe Operations

1. **Failure modes should be enumerated for business, mission, and safety-critical systems**. The initial step in making systems resilient and safe is to understand how and why they can fail. Techniques such as Failure Modes and Effects Analysis (FMEA) can produce a list of risks for each critical system that should be addressed in system design, operational environment plans, business continuity and recovery methods, and/or safety practices. Priority should focus on designing resilience and safety features into software-intensive systems to minimize the probability of system failures and subsequent continuity operations.

2. **To sustain the business or mission, systems must continue operations in the face of unexpected events, or if interrupted, recover their operations efficiently**. If interrupted, systems should fail to a state that loses as few transactions as possible and protects the state of system data. Business and mission-critical systems must be designed to withstand such events as interrupted communications, power outages, unexpected processing or memory loads, and other disruptive phenomena. Resilience also requires backup systems and data stored in locations that are not subject to the same environmental conditions and impacts that affect primary system such as offsite facilities, third party repositories, or the cloud. Backup data should be updated sufficiently often to minimize business and mission disruption from data-related problems. Resilient operation should be subject to periodic verification.

3. **Failsafe properties of software-intensive systems should be designed in and verified**. Failsafe operations are required in life and safety-critical systems such as avionics, autonomous vehicles, nuclear power plants, and medical devices. In addition, those affected by system operations, including operators, should be protected from harm. This principle also applies to software-intensive systems controlling critical infrastructures where property can be damaged, or customer activities and interests harmfully disrupted. The risks of flawed system operation should be enumerated, and failsafe properties designed into the system to eliminate or mitigate them.