Resilience Methods within the Software Development Cycle

Acklyn Murray, Marlon Mejias, Peter Keiller

Department of Electrical Engineering and Computer Science
Howard University
Washington, DC 20059

Abstract

Resilience, attuned to software development, is the ability of a component such as an Operating System (OS), server, network, data center or system of storage to rapidly recover and continue with operations even when certain failures or errors occur with limited disruption that could otherwise affect the progress of operations running in the computer. In the following paper, we will discuss the development cycle in relation to resilience methods, resilience factors that may contribute to unscheduled disruptions and upcoming possibilities in the examination of resilience methods.

Keywords— Resilience; Software methods; Dependability, Fault Tolerance

1 Introduction

Many of today’s business models are unique, advanced and highly dependent on technology when compared to the models of earlier organizations. The increasing use of technology in business today has changed the nature of business operations [11,16] and has made software an indispensable aspect of operations. In the past, businesses have applied software to support operations during business hours from 9 A.M. to 5 P.M., Monday to Friday. In such operations, computers would process orders, reconcile Point of Sale (POS), cash transactions, and other activities. As technology and operations advanced new models emerged where integrated systems were used for batch processing to complete tasks such as consolidation, reconciliation, and data transfer overnight. Today however, most organizations use software to process transactions on a real-time basis, and for decision support in global operations[1]. This development means that small failures can cripple operations and have far reaching impact. The level of reliance on software today is not comparable to the level of reliance any time before in history. This increased reliance on software has not been met with appropriate improvements in the software design process to ensure that software products are resilient. While resilience is a main stay of ‘high-risk’ industries with physical products such as in aviation, chemical manufacturing, healthcare technology, and traditional manufacturing, much can be learnt and applied to the field of software development [23].

The study of software resilience is relatively new and there are only limited studies or publications [1]. As with any emerging field, it is important to review recent literature and suggested best practices so that conclusions and future areas of research can be identified. This paper presents a review of recent articles on software engineering and methods of software resilience. Relevant analysis and discussion are provided before conclusions are presented.

2 Defining Software Resilience

Generally, in engineering disciplines, resilience is measured by the ability of a system to persevere or work through a major fault in a critical part of the system [5]. It is the ability of a system to “take a hit to a critical component” and recover in a reasonable manner or acceptable duration [2]. Resilient software should have the capacity to withstand a failure in a critical component but still recover in an acceptable predefined manner and duration. Failures in software can arise from intentional activities/attacks or unintended faults. Either way, a resilient system should have the capacity to recover after such failures. Software that does not recover on its own or applications that recover after a long duration are not resilient.

2.1 Factors affecting software resiliency

There are several factors that affect the resiliency of software. Factors include complexity, globalization, interdependency, rapid change, level of system integration and human factors [2]. These factors are discussed further.

Today’s organizations have complex networked systems that come from many integrations and interdependencies. For example, a telecommunication company providing a mobile money service through a third-party application has a complex system with multiple points of failure. When one critical system fails, the entire platform cannot deliver the required service [18]. Software that relies on the internet and ever-changing, dynamic networks increases its chances of failure. Net-centricity can introduce complexities leading to greater chances of errors [6]. Combining systems introduces higher chances of failure due to complexity. This complexity makes it extremely hard to provide a platform with consistent levels of resilience. The greater the number of integrated systems, the greater the chances of lower resilience [8].

Another important aspect of highly integrated complex systems to consider is that organizations could be using third-party systems. Failures in such applications affect local systems. This often leads to considerations of complexity as a factor in software resilience.

Today, systems rely on multiple other technology platforms to provide services [10]. For instance, a firm can use a combination of locally implemented software and other cloud-based architectures. In such a scenario, failures in the cloud become failures in the local systems. The use of open source software could also complicate matters since more complexity is
introduced. COTS, or Commercial off-the-shelf, products could have different designs and features compared to open source software. For instance, when firms integrate a combination of ‘off-the-shelf’ and open source software, resilience is reduced because of the complexities and different levels of net-centricity. Modern businesses operate a range of interdependent, interconnected, and interrelated systems, hence the failure of one easily becomes the failure of the many.

The other factors considered important include globalization and rapid changes. In terms of globalization, the new software development models and software supply chains are seen to focus more on cost reduction rather than quality. They focus more on reducing their costs of production rather than the resilience of the systems. Further, changing requirements and frequent software releases lead to dynamic and complex outcomes. For instance, when new software releases are being provided to the market, new changes may be incorporated and resiliency may be reduced compared to earlier releases [11].

Additional research needs to focus on the influence and impact of human error in software resilience. It is important to consider failure in complex systems as a multifaceted problem where several factors can be identified. It is not proper to attribute failure to one root cause but to the wider array of interrelated factors. The manner in which users or developers react can affect the resilience of systems. From a socio-technical point of view, it is possible that safety, operational errors, and other considerations of the man-machine interface can affect the resilience of systems [17].

Lapses, rule violations, and process confusion can affect the ability of systems to remain resilient. Such issues can be related to the manner in which operators handle the systems. However, if we attribute the resilience of entire systems to human error, it could lead to an insufficient approach to combating failure. Failure to human factor alone could be insufficient in providing solutions to resiliency problems. Therefore, human error should be taken as the starting point as we move towards an examination of the wide array of factors that can cause the problems.

3 Resilience in new methods of Software Engineering

One of the obvious approaches to ensuring resilience is adequate testing at each phase within the software development cycle. When software is being completed, a series of tests are necessary to ensure that all aspects of the software satisfy the design requirements [12]. Adequate testing can ensure improved integrity and resilience in the new software. Most specifically, clear and regular testing should be concentrated on the coding stage as poor coding standards, a lack of peer reviews and follow up regression testing can make the system prone to resilience problems. An untested error routine is a major cause of later system failure and the disability to easily recover after a disruption. Inadequate testing could lead to disastrous consequences.

A second method of resilient software development is designing the software for resilience when the software specifications are being developed. Some systems can fail because relevant resilience routines or features were not included in the initial design. Designing for resilience calls for a clear understanding of the requirements of the software and business process before starting the entire design process. This can be daunting, especially in the age of agile development. When designing technology systems, it is important to ensure that the design captures important aspects related to failure or resilience [11] such as recovery from hardware failure, incorrect input into the system, human factors and graceful degradation. When such features are included early in the design, it becomes easier to design for resilience even at advanced stages of development [10]. However, when such features are absent at early stages, it becomes a burden to include them at the advanced stages of development. It is important that human factors are considered early in design phase. Inappropriate use of software can lead to failure when applications are not used for the correct purposes. This could result into routine failures or process failure leading to the failure of the entire application. It is also important that designers consider fault-tolerant hardware, load balancing, high-availability clustering [15], back-up and recovery options. The application should also be designed with fault tolerance and failure recovery at the forefront of the design. High fault tolerance ensures that the application can protect itself from further damage when an error or fault occurs [20]. Hardware checking, preparation for recovery and the recovery process are being addressed through the new general algorithm fault tolerance (GAFT) [22].

With improved design methods and planning to mitigate the consequences of potential failure, the ability to create software and a business process with a good ability to recover is enhanced for the benefit of the entire organization.

Change management is also a vital stage in the software development life cycle. Strategies that handle change in a scheduled manner are preferable and should occur as early in the software development cycle as possible. The change process should be highly controlled in order to increase the likelihood of security, resiliency and integrity in the software development cycle. The dependencies of all systems/components should be documented so that when changes are scheduled to occur they are discussed with relevant stakeholders. Unscheduled change management strategies are extremely detrimental to the system especially when the system is already in production.

Generally, systems are becoming more integrated and development is becoming more complex at a worldwide basis. Reduced complexity in the software design is a good resilience method. The software should be designed modularly and each module should have a method of monitoring its own status. This makes the system less complex to trouble shoot in the event of failure. Despite this, the system should be able to meet all the needs that were outlined in the objectives while still maintaining complexity. With increased complexity, the inability of the final system to recover after various disruptions also increases.

4 Software Engineering process for improved resilience

In systems that are already deployed code refactoring and optimization is important. Improving a system’s resiliency calls for the collection of detailed measures of the software process and product quality. To design for improved resilience, engineers should focus on several concerns. These concerns may include planned and unplanned downtime, system repair time, disaster recovery, and uptime requirements [9]. In the system optimization phase engineers define key performance measure for
system uptime or availability, estimation of functional availability, acceptable downtime and assessments of possible failures [3]. After defining these metrics the system should be reworked from its core to ensure resilience.

Another approach, as suggested by J. Allspaw, is to ensure that organizations learn from failure. When software fails, the organization should consider this as a lesson meaning that new resilience features should be introduced in the systems. Companies should consistently pursue strategies to identify the systemic vulnerabilities present in their software. One valid approach is to identify such vulnerabilities when software fails so that new features can be developed. Even though such exercises are very complex and lengthy, the result of a successful exercise is worth the time and resources invested. Indeed, postmortems are opportunities for organizations, users, and developers to learn the underlying vulnerabilities in the systems [17]. This often improves the ability to anticipate and resolve failures in future. When systems fail, most organizations launch investigations to understand the root cause of the failures (14). In such endeavors, companies may even discover vulnerabilities that they didn’t know existed. Even though the system may be working optimally, it is prudent to identify and resolve issues to curb any future failures.

Organizations should develop features, routines, and processes for recovery and restoration after failure [12]. Failure recovery should be governed by processes and activities that have been identified and documented. Recovery procedures should outline other fail-over systems where the application can redirect operations when there is a failure [9]. This is often provided for by backup systems located on-site or off-site. Recent developments in cloud systems have provided another option for fail-over techniques [2]. This does not necessarily point to software features. Organizational work procedures and processes can include several activities aimed at ensuring that systems recover quickly from failures [5]. Such an approach is important since modern architectures operate in diversified environments where technological and human factors interplay. This is in agreement with the insights of J. Allspaw, that human factors are important in system failure [17]. The human-machine interface determines how systems operate and their resilience might rely on how such interfaces are planned and implemented.

5 Economics of Resilience

Normally, all aspects of business depend on economics. Organizations must weigh between costs and the need for resilience. The strategic importance of the application should also be considered. Ultimately, organizations must strike that delicate balance between reducing costs and ensuring that applications are resilient and reliable [2]. Therefore, with costs in mind, organizations should understand the balance between backups, redundancy, fail-over systems, and the economics of their operations [4]. One approach is the consideration of the “Greedy approximation”. “Greedy approximation” adds a new redundant component at each step, yielding the highest gain in reliability on each iteration. The result is to induce the redundancy around the system from the baseline forward. Similarly, the apportionment process spreads the redundancy within the system, but in a top down fashion. In general, the two techniques may yield a different set of sub optima; most of

the time, both will yield good solutions yet feasibly economically divergent. The apportionment approach has a number of advantages, including the following: (a) it fits in well with the philosophy of system design, which is generally adopted by designers of large systems; (b) its computations will in general be simpler; and (c) it may provide more insight into when the suboptimal solutions it generates are good.[21]

In some cases, it might not be economically feasible to improve the resilience of systems. However, organizations will always operate better when they rely on resilient systems especially when such systems form the core or backbone of organizational processes.

6 Selected software resilience projects in literature

The following projects are being conducted through the Center for Resilient Software at MIT. The general notion in resilient features is adding functionality to create resilient software yet, with a minimizing functionality to make it more feasible to formally verify software.

“The Cloud Intrusion Detection and Repair (CIDAR) project is developing a system that observes normal interactions during the secure operation of the cloud to derive properties that characterize a secure operation. The project is funded under the DARPA Mission-Oriented Resilient Clouds (MRC) program with MIT at the helm. If any part of the cloud subsequently attempts to violate these properties, the system intervenes and changes the interaction by adding or removing operations, or changing the parameters that appear in operations, to ensure that the cloud executes securely and survives the attack while continuing to provide uninterrupted service to any legitimate user.

The approach revolves around a new technique: input rectification. Applications process the vast majority of inputs securely. Malicious attacks find success in input that the application regards as errors and capitalizing on the malformed, novel features. Input rectification research observes inputs that the application processes to derive a model in the form of constraints over input fields. A defined boundary is created for the applications with a set of inputs that the application can process error free. When the application encounters an input that is outside the boundary, the rectifier uses the model to change the input to move the input into the corrective boundaries of the application. The results show that this technique eliminates security vulnerabilities in a range of applications, leaves the overwhelming majority of safe inputs unchanged, and preserves much of the useful information in modified atypical inputs.[19]”

The SPICE project, another collaborative project with promise from IARPA under the STONESOUP program, has devised and is fine tuning a method to “eliminate vulnerabilities triggered by a variety of low level errors in stripped x86 binaries. A combined dynamic and static type inference system analyzes reads and writes to infer how the application structures the flat x86/x64 address space. The information is then utilized to preserve the integrity of the execution. Through this project one can neutralize malicious attacks that require the exploit of buffer overflow vulnerabilities and uses a configurable security policy to modify the execution to eliminate the vulnerability and enable continued safe execution.
An additional method derived from the SPICE project is a precise taint tracing system. This system combines static and dynamic analysis to assist minimizing overhead. The taint information enables one to detect the unsafe direct use of untrusted input fields at vulnerability field locations such as command invocation, SQL and memory allocation sites. Secondly, one is able to track memory allocation information to properly eliminate buffer overflow attacks for ongoing monitoring.

A third project, under STONESOUP, is focused on eliminating bugs in a particular language and set of libraries. Headed by Kestrel and MIT, the VIBRANCE project is developing techniques to eliminate vulnerabilities in Java applications triggered by unchecked inputs. VIBRANCE is attempting to provide a taint tracer with overhead small enough for routine production use that combines precise dynamic taint tracing with static analysis to reduce the tracing overhead [26]. Inputs monitored will quell SQL, command, ldap, and xquery injection attacks also resource allocation vulnerabilities and numeric overflow and underflow vulnerabilities.

The International Workshops on Software Engineering for Resilient Systems (SERENE) is now in its 9th conference bringing together researchers and industry practitioners to discuss advances in engineering resilient systems.

7 Conclusion

The topic of software resilience is relatively contemporary. Several studies have been conducted and the results indicate that the software engineering field has been lagging behind compared to other engineering sectors. The studies also noted that there are some important factors that affect resilience; complexity, globalization, hybridization, interdependency, rapid change and net-centricity are important considerations. Further, some design related factors include human factors, inappropriate use, poor designs, change management and inadequate testing. To ensure that systems are resilient, software engineers should ensure that their development process considers core elements or features in the design for resilience. Fault tolerance, fail-over systems, recovery options, and backup systems are important considerations. Further, human factors should be considered in the design as well as the relevant economic considerations. Ultimately, organizations perform better when their core software systems are resilient and reliable. Considering the importance of this area of study, it is important for more research to be conducted especially in the areas of resilience features, new products and features, and relevant organizational case studies.

8 References


