Chapter 13
Summary and Future Work

We have come to the end of the book, which has investigated different aspects of anti-fragile information and communications technology (ICT) systems. Taleb [10] introduced the concept of anti-fragility to show that it is not enough for large natural or man-made systems to be robust to predictable events with a large impact. In an unpredictable world, systems must be able to handle randomness, volatility, and unforeseen large-impact events. Learning from artificial and real incidents is necessary to remove vulnerabilities and prevent systems from developing fragilities over time. This chapter summarizes the book's main insights into the development and operation of anti-fragile ICT systems, discusses the design of future systems, and outlines the need for anti-fragile processes, especially to handle attacks on the confidentiality, integrity, and availability of ICT systems.

13.1 Achieving Anti-fragility

While many commentators find Taleb's concept of anti-fragility both interesting and useful, other commentators believe it is very similar to the well-known concepts of robustness and resilience. To determine whether Taleb's work [10] really brings any new insight into the development and operation of large ICT systems, the author has investigated different aspects of anti-fragility. The main insights are summarized in the following.

Stakeholders of complex adaptive ICT systems must embrace hardware and software failures because they are inevitable. Local failures should, at worst, result in degraded performance, not systemic failures such as unplanned system downtime. The book introduced four design principles—*modularity*, *weak links*, *redundancy*, and *diversity*—to isolate the impact of local failures and one operational principle—the *fail fast* principle—to quickly detect vulnerabilities by inducing artificial failures. The collective goal of the five principles presented is to limit the impact of failures by failing early, isolate the impact of local failures, and learn from small

K.J. Hole, *Anti-fragile ICT Systems*, Simula SpringerBriefs on Computing 1,
DOI 10.1007/978-3-319-30070-2_13

failures how to maintain the desired performance as a system and its environment change. Netflix's pioneering work shows that stakeholders can build and maintain web-scale applications in the cloud with a degree of anti-fragility to system downtime. Analyses of telecom infrastructures and electronic government systems confirm that the cloud facilitates anti-fragility to downtime.

A series of analyses outlined how to gain anti-fragility to the spreading of malware with unknown and time-varying spreading mechanisms. It was first found that application stores utilizing compilers with diversity engines in the cloud could generate enough software diversity to halt frequent malware outbreaks from spreading over huge networks of computing devices. Imperfect malware detection/removal was then added to this simple diversity-enhancing technique to keep the fraction of infected devices low over time. The resulting halting technique scales to prevent the spreading of frequent malware outbreaks on networks with millions of devices. While more work is needed to verify the practicability of the halting technique, the approach demonstrates that it is advantageous to model huge networked computing systems as complex adaptive systems and then apply results from network science to analyze the models' fragility, robustness, and anti-fragility to different classes of impacts.

If we cannot detect failures in a system, then it becomes impossible to determine and remove vulnerabilities. Hence, we must be able to monitor a system's behavior to ensure anti-fragility to a particular type of impact. We have argued that cloud-based software solutions should have a service-oriented architecture (SOA) with microservices implemented by virtual machines. While it is hard to monitor and analyze the internal behavior of applications with strongly connected modules, the simplicity of weakly connected microservices makes it possible to monitor and understand their individual behaviors. If the graph defining the dependencies between the microservices is not too large and dense, then it is also possible to analyze the consequences of the dependencies.

We need a technique to detect anomalies in streaming data because cloud infrastructures typically stream metric data about the status of virtual machines. Hawkins' learning algorithm is an interesting choice for anomaly detection in streaming data. The algorithm is based on a theory of how the brain learns, called hierarchical temporal memory (HTM). While the HTM learning algorithm may not always provide the best anomaly detection, it is very flexible and can be applied to many different metric streams. HTM automatically builds online data models, removing the need to store huge amounts of data in a database. Since the HTM algorithm is able to quickly detect anomalies, it facilitates corrective actions in real time. An application called Grok utilizes the HTM learning algorithm to detect anomalies in virtual machines running on the Amazon Web Services (AWS) cloud. Grok is able to detect anomalies that are hard for humans to see in the raw feed of AWS metric data. The application has also been successfully applied to user-defined metric streams.

In conclusion, many of today's ICT systems with strongly connected modules are too fragile to downtime and other large-impact events. While anti-fragile ICT systems have no absolute guarantee of avoiding the intolerable impact of all possible swan events, it is practicable to build systems that handle the impact of surprising events much better than many current systems. In a world where people are becoming increasingly dependent on ICT, we need to build anti-fragile systems to avoid rare but hugely negative events affecting whole populations.

13.2 Future Anti-fragile ICT Systems

Only complex adaptive ICT systems need to be anti-fragile to different classes of impacts because only complex systems are vulnerable to swans in the form of highly surprising, global failures with intolerable impact. We have concentrated on how to create anti-fragile systems in the cloud because its pay-as-you-go pricing model makes it economically feasible for even startups and other small companies to build anti-fragile solutions. Further investigation into the anti-fragility of ICT systems should consider whether additional design and operational principles, as well as anti-principles, are needed to ensure anti-fragility to different classes of impacts. A new principle should only be introduced if it is valid for many types of systems. The introduction of highly overlapping principles should be avoided.

Simplicity is an obvious candidate to become a general design principle for anti-fragile systems. We have already promoted simplicity by recommending the use of SOA with microservices. The single purpose of a microservice makes it easy to understand what each part of a system does. Furthermore, weak links between the services limits the effect of local failures and makes it easier to understand a system's overall behavior. However, more work is needed to understand the full meaning and impact of simplicity in the context of anti-fragile systems. New architectural patterns facilitating anti-fragility to classes of incidents would be particularly welcome. We also need to better understand the effort required to monitor anti-fragile systems.

Another obvious candidate to become a design principle is openness. Indirectly, we have also promoted openness by considering the anti-principle of closedness. Openness can undoubtedly reduce the negative impact of coincidental events. The advantage of openness is less clear when, for example, nation states are attacking each other's vital ICT infrastructures. More work is needed to fully understand the implications of openness in the context of anti-fragile systems of critical national or international importance.

The study of real systems is necessary to gain more insight into the concept of anti-fragility. While it is hard for independent scientists to obtain information on electronic payment systems, studies of such systems are of particular interest because of their great importance to society. It is particularly interesting to better understand how fraud detection can be exploited to achieve a degree of anti-fragility to financial losses.

The HTM theory discussed outlines how a small part of the neocortex learns sequences and predicts future inputs. While we applied HTM to detect anomalies in streaming metric data from the cloud, HTM theory has wider applications. The neocortex itself is an anti-fragile system (or system of systems) because it continuously learns new sequences and forgets old sequences in a way that is highly tolerant to noise, damaged cells, and broken connections. Future HTM theory is likely to provide important insights into the development of anti-fragile machine learning systems.

13.3 Future Bio-inspired System Designs

While superficial comparisons between complex adaptive ICT systems and biological systems should be avoided, it is useful to view distributed ICT systems as ecosystems or communities of autonomous entities interacting with each other and a changing environment. To encourage the reader to consider bio-inspired system designs in the future, we argue that ICT systems of simple and weakly connected modules avoid much of the fragility associated with strongly connected legacy systems based on old technologies that are hard to maintain and upgrade.

It is impossible to change an ICT system in a controlled manner if we do not understand its functionality. While a large ICT system has very diverse functionality, humans can only focus on one task at the time. When the human mind is forced to focus on multiple difficult tasks simultaneously, it tries to switch between tasks in rapid succession, making the effort of completing the tasks much more difficult. In particular, it is hard for software developers to understand the functionality of an ICT system consisting of many large, complicated software modules. As the original developers of a software system move to other projects and new developers start to modify the original code, the initial design is often violated. Many of these design violations occur unintentionally because the new developers do not fully understand the original system design and its implementation. Over time, the many changes to the original code generate more and stronger dependencies between the modules, resulting in a strongly connected system that is vulnerable to failure propagation causing systemic failures.

In general, the more tasks an engineer or developer has to consider at the same time to create some kind of module, the more complicated the module is. The development of simple modules, each with a single limited responsibility, helps explain why the class of microservice architectures first discussed in Chap. 5 is becoming increasingly popular. Since each microservice fulfills a single responsibility, one developer can understand the functionality of the microservice without undue strain. Furthermore, the developers in a team creating a microservice solution can concentrate on a single task at a time, making it more pleasurable to develop the solution.

The class of microservice architectures with weak links is the result of years of engineering work in building web-scale applications with very high availability, scalability, and performance. A microservice solution mimics nature. The whole system is constantly evolving, without the limited availability associated with monoliths. In particular, microservices come and go. Microservice solutions are "living software" that remove much of the fragility of legacy software because it is easy to remove old services and create new ones. In fact, developers often write a completely new microservice rather than modify an old service, because the limited functionality makes it easy to write a service from scratch.

Experience with microservice solutions suggests that building systems with simple and weakly connected modules can significantly reduce the fragility associated with legacy systems. The functionality of a system should be divided over many modules such that a single human can understand what each module does without undue effort. Weak links, redundancy, and diversity should then be used to limit the impact of local failures.

13.4 The Need for Anti-fragile Processes

While we have mainly considered how to design and implement anti-fragile ICT systems, another related approach to anti-fragility is to consider systems and their stakeholders as adaptive complex processes. This view provides us with a very general approach to the study of anti-fragility. In fact, a process may very well be anti-fragile to a particular type of undesirable outcome without any technology involved at all. Hence, we could study processes in the area of computer science, as well as society in general, to better understand fragility, robustness, and anti-fragility to particular impacts.

The international research community in cryptography has long deployed anti-fragile processes to develop new cryptographic solutions. A universally accepted cryptographic primitive such as a cipher or a hash function is the result of a competitive process in which some researchers suggest new primitives and other researchers try to determine if the primitives have exploitable vulnerabilities. After several rounds of modified suggestions and attacks, a new primitive emerges that is very hard to compromise because the cryptographers have learned from their own and others' earlier mistakes. However, even after this long and hard selection process, cryptographers know that the only way to ensure a primitive's strength over time is to keep attacking it on a regular basis. The same is true for cryptographic protocols.

One example of this never-ending anti-fragile process is the many evaluations of the Transport Layer Security (TLS) protocol, leading to new and more secure protocol versions. However, the serious Heartbleed Bug incident demonstrated that large-impact incidents can still occur. While many have evaluated the TLS design through the years and mitigated vulnerabilities, it is also necessary to carefully validate the implementations of TLS. The Heartbleed Bug disclosed in April 2014 involved an improper input validation in the OpenSSL cryptography library, a much used implementation of the TLS protocol. The missing bounds check allowed theft of the servers' private keys and users' session cookies and passwords. While many large companies used the OpenSSL cryptography library, it seems that none of them had carefully validated the code. This serious incident demonstrates the danger of trusting software libraries without evaluating the security of the code. Trust by default must be replaced by an understood degree of trust.

To improve the security of complex adaptive ICT systems, it is necessary to develop anti-fragile processes to maintain a high level of confidentiality, integrity, and availability. As argued in Chap. 2, risk management processes based on the prediction of incidents are not enough because the risk of a complex ICT system may very well be dominated by swans, which are notoriously difficult to predict. Hence, we need anti-fragile security processes that limit the impact of inevitable security incidents and learn from these incidents how to create more secure systems. Everybody with the ability to change the security of an ICT system should have "skin in the game," that is, they should share the responsibility for the consequences of a successful attack, not to be punished but to ensure a subsequent period of learning and mitigation to improve security and to stop similar attacks in the future.

13.5 Challenge to Readers

The author wrote this book to educate himself and the reader about anti-fragile ICT systems and to argue that it is both possible and desirable to develop and operate such systems. Some of the book's specific proposals will undoubtedly be replaced by better solutions, while others will hopefully survive. Since our knowledge of anti-fragile ICT systems is still limited and fragmented, more work is needed to better understand these systems. Improvements to the outlined solutions are very welcome, as well as brand new solutions with anti-fragility to different types of impacts. To model real systems, create anti-fragile processes, and discover ways to improve and extend the contents of the book, the interested reader will do well to first study the many surprising perspectives, interesting ideas, and important insights introduced by Taleb [8–12], Dekker [17, 18], and Geer [28–33].