Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 00 (2016) 000–000

Procedia Computer Science

www.elsevier.com/locate/procedia

4th International Workshop on Computational Antifragility and Antifragile Engineering (ANTIFRAGILE 2017)

Towards Antifragile Software Architectures

Daniel Russo[a,b,c,*], Paolo Ciancarini[a,b,c]

[a]*Department of Computer Science & Engineering - University of Bologna, Mura Anteo Zamboni 7, 40126 Bologna, Italy*
[b]*Consiglio Nazionale delle Ricerche (CNR), Institute of Cognitive Sciences and Technologies*
[c]*Consorzio Interuniversitario Nazionale per l'Informatica (CINI)*

Abstract

Antifragility is a rising issue in Software Engineering. Due to pervasiveness of software in a growing number of mission critical applications, traditional resilience and recovery systems may not be sufficient. Software has taken over many functionalities which are of vital interest in today and future world. We relay a lot on software applications which may be faulty and cause immense damages. To cope with this scenario, claiming to develop better software is not enough, since unexpected events a.k.a. Black Swans, may disrupt and overcome our system. The purpose of this paper is to propose a new architectural design that responds to the need to build antifragile systems for contemporary complex scenarios. We suggest a system which enhances its strength through experience and errors. It is a self adaptive system architecture improving when facing errors. The most relevant aspect of this approach is that architectures are not only resilient, they extract the intrinsic value of faults. This paper suggests that a fine grained architecture is the key issue to build antifragile systems.

Keywords: Complex Systems; Software Engineering; Antifragility; Architecture.

1. Introduction

The Jeep hacked remotely through the internet was not, *per se* a big event[24]. However, the awareness that it was actually possible to compromise familiar mission critical devices, rose crucial questions. If someone can compromise remotely, just with an internet connection, a mission critical application, like a self driving car, potential impact on human life could be tremendous. Any car, and any mission critical application, could be a potential weapon which endangers people. This is leveraged by the fact that such kind of applications are rising, due to the wider use of software in nearly all human-related domains. However, also traditional critical industry, like the banking one, expressed concerns regarding the reliability of their systems[23].

The rise of ever tougher challenges in the domain of open and complex systems deserves solutions which are not built with a traditional incremental approach. There is the clear need for disruption. Technology in general has incredibly increased its capability in the last decades. Nevertheless, there are still big issues (both engineering and

* Corresponding author. Tel.: +39 051 2094861 ; fax: +39 051 20 94510.
E-mail address: daniel.russo@unibo.it

philosophical) that need to be tackled. Among them, the costs in money and time of designing, testing, delivering, operating and maintaining large systems is accelerating at an unsustainable rate. Moreover, these systems often do not perform as intended. Even if they perform as expected for some time, when failures occur, they are often difficult and expensive to fix. The architectural complexity increases also the probability of intermittent faults. There is the need for new engineering and philosophical designs to develop new systems to achieve those performances that traditional methods are unable to guarantee.

According to this view, mission critical organizations (e.g., NASA) are actively working to solve this big challenge. Instead of on designing systems to meet known requirements that will always lead to fragile systems at some degree, systems should be designed wherever possible to be antifragile: designing cognitive cyberphysical systems that can learn from their experience, adapt to unforeseen events they face in their environment, and grow stronger in the face of adversity[17]. Apparently, US federal research and engineering organizations are actively searching new solutions, since "addressing future challenges [...] is not simply to do what we know how to do now better: we need to do things we currently do not know how to do"[17].

The rest of this paper is outlined as follows. Section 2 explains the emergence of the new antifragile paradigm with respect to the differences with resilience. In section 3 major insights about design and implementation of such systems is given. Finally, a roadmap for an antifragile architecture is proposed in section 4 and in section 5 we briefly sum up our main findings.

2. A new emerging paradigm

The emergence of a new paradigm is the direct reaction to give the community a reference set for Antifragility.

Etymologically, *resilience* (from the Latin *resilire*, to rebound) means springing back. Traditionally, in computer science and, in particular in the field of dependable computing systems, resilience has been intended as fault tolerance[2]. The definition evolved to self-adaptive and automated systems, where the "resilience of a system as its ability to maintain service delivery that can justifiably be trusted in spite of changes in its internal and external contexts or in the interface between these (i.e., the avoidance of failures that are unacceptably frequent or severe, even when facing changes)"[1]. Generally, self-adapting systems are expected to become more resilient as they adapt themselves to the context scenario. However, no definition refers to resilience as an aptitude to mature, learning from errors. Learning from errors with respect to contest to adapt to the context (i.e., being robust to errors) appears to be a much powerful approach.

"Antifragility is beyond resilience or robustness. The resilient resists shocks and stays the same; the antifragile gets better"[26]. In general, an antifragile system is able to evolve its identity in order to improve itself systematically in the context.

Main characteristics of antifragility are[7]:

- Systems are not necessarily more "resilient" than change tolerant or elastic objects and systems. As it is the case for those entities, also antifragile systems are characterized by a yielding point (i.e., a point which beyond they would fail or become untrustworthy).
- The identity of systems itself may change over time. This means that the behaviors of antifragile software may outlay the system's specifications.
- Systems may have some form of awareness of their current and past environment fitness. In particular, their risk model of adapt and evolve aspects of their system identity may be considered.

3. Design & Implementation

From an architectural perspective, by *fragility* is meant that any changes to a system, either during the integration of the system (e.g. software errors, changes in requirements), or during the maintenance of the system when additional functionality is to be implemented, will generally produce the result that some part of the application will not comply with the original architectural design[12].

Taleb extended such definition to any possible condition, where:

For the fragile, the cumulative effect of small shocks is smaller than the single effect of an equivalent single large shock.[26]

Generally speaking, according to Taleb's view, a fragile entity is hurt a lot more by extreme events than by a succession of intermediate ones. A common life example is that of the difference between being hit by a large stone and a thousand pebbles. The difference between a thousand pebbles and a large stone of equivalent weight is a potent illustration of how fragility stems from nonlinear effects. Clearly, throwing a ten-pound stone to someone will cause more than twice the harm of a five-pound stone, more than ten times the harm of a one-pound stone, and so on[26].

Similarly, for fine grained architectures the cumulative effects of small shocks (of any type, intrinsic or extrinsic ones) is smaller than the single effect on a monolith. If the monolith is not robust enough, the whole service may collapse. While, in a fine grained architecture, only single services may be hurted, not system's integrity.

From what is happing around us, we can deduce a non-linear relationship. Since the response of such attacks are not straightforward (i.e., not a straight line), if a doubling of the number of attacks the outcome is a lot more or a lot less than double the effect. If we draw a line on a graph, with harm on the vertical axis and the effect of e.g., malwares on the horizontal axis, it will be curved, not a straight line. That is a refinement of asymmetry.

In order to elaborate his theory, Taleb had to invent the neologism *antifragile* since, according to the author, there is no word for the exact opposite of fragile[26]. Antifragility is beyond resilience or robustness. The resilient resists shocks and stays the same, while the antifragile entity gets better. For Taleb, this property is behind everything that has changed with time (e.g. evolution, culture, ideas, revolutions, political systems, technological innovation). The antifragile *loves* both randomness and uncertainty. One of the most interesting proprieties is dealing with the unknown, to do things without understanding them and actually succeeding in doing them.

Antifragility is the opposite of fragility because it makes understand fragility better. Just as we cannot improve health without reducing disease, or increase wealth without first decreasing losses, antifragility and fragility are degrees on a spectrum. So, understanding fragility means to understand it and to decrease it, whenever and as so much as possible. Antifragility means to benefit from fragility reduction and, then, gain from the external (context) exposure. In general, anything that has more upside than downside from random events (or certain shocks) is antifragile. Anything that acts reversely is fragile. So:

For the antifragile, shocks bring more benefits (equivalently, less harm) as their intensity increases (up to a point).[26]

Antifragile systems are non-linear and, in particular, they are a convex function, i.e. fragility is a concave function (more pain than gain) and antifragility is a convex one (more gain than pain).

Both fragility and antifragility are non-linear functions. In many domain, also in software engineering we often assume, for simplicity, the linearity of many process, even though this is often an oversimplification. Work (especially that of skilled people) is a typical example of non - linearity. Programmers rarely code effectively 8 hours a day. We know that, on average, programmers spend only few hours per day on effective coding[19]. In other words, coding is a non-linear activity, since the productivity varies over day time. So, the definition itself of man - hours seems to be a linear oversimplification, since they are not the same. Such entities are not comparable. It is an average.

3.1. Antifragility: Jensen's inequality

Convexity and concavity (negative convexity) are two terms quite known in financial economics. A convex function is a non-linear function which depend on the second derivative. Intuitively, if a system's element changes, the system's respond does not change linearly.

Taleb used as an example the case of options. An option can be seen as an assurance that protects from market's asymmetry. In fact it is a contract which gives to the buyer the right (not the duty), to buy or sell an asset at a specified price or date. An option has a convex function, since potential downsizes are predictable and computable (the price paid for the option). Whereas, potential profits of options are unlimited (depending on the asset), since there is no duty to buy it. The buyer buys only the possibility (right) to buy/sell the asset if it is for him convenient. The option price is typically given by the intersection of the two concave and convex functions.

This example is useful also in software engineering, where we have to design reliable architectures. Since an option's price is given by the intersection a positive (convex) and negative (concave) function and you have the right to buy it, this minimize your exposure to external phenomena. So, it maximizes system's efficiency (with potential unlimited gain), minimizing risk of system's failure.

The mathematical propriety behind antifragility is Jensen's inequality[26]. This is a strong intuition why linear model are so inefficient with respect to convex ones.

For a real convex function φ, numbers $x_1, x_2, ..., x_n$ in its domain (i.e. antifragile elements), and positive weights a_i, we have the following:

$$\varphi\left(\frac{\sum_i a_i x_i}{\sum_i a_i}\right) \leq \frac{\sum_i a_i \varphi(x_i)}{\sum_i a_i}$$

if weights a_i are homogeneous and equal 1, we have that:

$$\varphi\left(\frac{\sum_i x_i}{n}\right) \leq \frac{\sum_i \varphi(x_i)}{n}$$

With a linear payoff there is need to be more efficient. With a convex payoff, your tolerance is much higher, represented by the Jensen's inequality. The efficiency of convexity is also represented by an over performing in random (standard) condition with respect to a linear one.

If your function is very convex, your position to gain, respect to loose is magnified by randomness. Randomness magnifies the probability to outperform with respect to the linear dimension. An example of the effectiveness of randomness with respect to simple linearity can be describe with family of gossiping algorithms[9]. It is shown how the member that chooses to broadcast votes at random already outperforms the linear case.

If you have convexity, then in the long run, you will outperform the average in the presence of uncertainty. The more uncertainty, the more role for optionality to kick in, and the more you will outperform.

4. A roadmap for Antifragility

To build an antifragile system there are three main concepts to follow. Since antifragile means to benefit more than to loose (positive asymmetry), the first step is to reduce possible losses. This is the typical condition of attackers, which have little to loose but great benefits (e.g. access to personal and sensitive data). Reducing potential losses means to reduce the exposure to negative disruptive elements (negative Black Swans[25]).

The second step is to avoid disastrous scenarios by hedging correctly risks. Due to systemic asymmetry and variability, assuming concepts like "average" risk is misleading. There is no average solution, due to the possibility of misinterpreting risks. Risks, according to this model, are only high (fully exposed to the public) and near-null (close system). Once the most relevant vulnerabilities are safe, all others may be exposed to variability, to learn from the environment. Antifragility assumes an ongoing learning process by external variability of non-functional elements. If the core of a system is relatively safe, the non-core parts of the system may profit from external stimuli and solicitation to enhance antifragility.

The last step is to embed adaptive fault tolerance[18]. Statically embedded fault tolerance is intrinsically fragile, as it embeds a fault model as an assumption[8]. Such an architecture implies a dynamic approach, where the systems takes advantages from errors, improving constantly. Just a static fault tolerance is not considerable antifragile, because there is no advantage from experiencing more faults. Since the system has to learn from errors (i.e., bugs), automatic bug fixing (i.e. a method for automatic software repair) have to be part of the architecture[11]. This because each fixed bug means a change in the code to improve the system on and on. At the end, dealing with continuous improvements of the system itself means to be antifragile, since they are originated from bug fixing capabilities.

As stated before, a fine-grained service architecture i.e., Microservices is a viable suite for Antifragility. Well established companies exploit it sucessfully, like Netflix[28 27]. Microservices are innovative architectural structures which offer a great degree on flexibility[10]. However, Microservices *per se* are not the solution, just enablers. They do not "learn" from errors, they are just resilient.

Therefore, some authors propose also a fault injection approach, to increase the numbers of errors to enhance the learning capabilities[16]. The major insight is that a software system using a fault injection methodology in production is antifragile, since it decreases the risk of miss, mismatch, and rot of error-handling code by continuously exercising it.

With regard to the process there are also some proposals in literature[16]. One of the most suited methodology is the test driven development (TTD)[3]. Since developers write automated tests for each feature they write, they are much more confident over their artifact, implementing a good refactoring. Moreover TTD allows continuous deployment, so short release cycle. In fact, with slow releases errors have smaller impacts and, when found, fixed very rapidly. Actually, also the Development and Operations methodology (DevOps) works in a similar way and can be considered an antifragile methodology[20],[22]. DevOps is a growing software engineering methodology to assure a better integration of the Development, Quality Assurance and Operations Department, in order to optimize the quality and the number of releases. Data quality and automatic comprehension is here also an important issue to support complex systems[4]. This is, for instance, crucial in a dynamic environment where features have to change very fast (i.e. mission and security critical environments), optimizing the releases and decreases the Design Debt. Moreover, dedicated software processes may be useful to enhance Antifragility within mission critical software development. With respect to code reuse, legal issues are also to be taken into account since possible litigations may cause negative side effects[5],[6].

We are aware that there is a great need to build a systematic framework around Antifragility. Clearly, a single contribution can not build enough consensus around a topic which needs both wide practice and validation. To stimulate discussion within the software engineering community we proposed an "Antifragile Software Manifesto"[1] [21]. A Manifesto is an affirmation of some principles concerning a domain. For instance, the Agile Manifesto wants to persuade software engineering professionals about how to build software in more effective ways to be implemented in their organizations. As many agile software development methodologies refer to and got inspiration from the Manifesto, so we started proposing an Antifragile one.

We suggest that Antifragile can be seen as an extreme way of doing Agile. Therefore we got direct inspiration from the Agile Manifesto, developing 12 principles. The reason of this is that according to our view, the Agile way of thinking is essential to build an antifragile organization. In this regard, Agile education has to be supported to improve these crucial skills[14],[15],[13]. In this instance, from a software engineering perspective, Agile is a prerequisite on which antifragile software development is build on. However, we do not see Antifragility as a sub domain of Agile. There are, at least, two motivational dimension to distinguish Antifragility:

- *intrinsic*: it is related to the level of awareness of organizations, due to a paradigm changing mentality, as proposed by Jones (2014). The learning leverage gained by external stimuli (i.e., errors) means much more than having an open and direct discussion. It means to build structurally not only a culture of trust but to accept proactively failures, in order to improve the whole system or organization. The learning by doing process enables people to gain from effective experience and not just from hypotheses.
- *extrinsic*: both Agile and Antifragile have horizontal approaches, trying to involve different stakeholders. However, while Agile's main goal is to build in an efficient and effective way a software system; Antifragility pays great attention to the context as learning source. For instance, the most relevant aspect is that the main goal of Antifragility is to learn from errors to become stronger. This can not happen without a constant evaluation of the larger environment and different inputs evaluation. Instead, Agile is much more focused on development and deployment, according to the client's expectations. This last aspect is also a part of Antifragility, as suggested, but it is given for granted, it is a pre requisite.

Finally, this is not a final statement for a complete antifragile architecture. This proposal serves to gain a better understanding of the topic and to launch a signal to the community to engage in such an approach. We strongly believe this will become a future trend in software engineering within the next years, alongside the diffusion of software in distributed and mission critical applications.

5. Conclusion

This paper is a proposal for a novel software architecture which responds to the high criteria of reliability. We pointed out why antifragile software is more effective than traditional approaches, interpreting the new paradigma

[1] www.antifragilesoftwaremanifesto.org

of Antifragility by Taleb. An invite to systematize such an approach was carried out, proposing and inviting to the contribution of an "Antifragile Software Manifesto".

Future work will go in two main directions. Firstly, foundations of this new approach needs to be validated and recognized by the community. Secondly, this high level approach needs to be practically validated by real life experiments and case studies.

6. Acknowledgment

This work was partially funded by the Consiglio Nazionale delle Ricerche (CNR) and the Consorzio Interuniversitario per l'Informatica (CINI).

References

1. R. Almeida and M. Vieira, *Benchmarking the resilience of self-adaptive software systems: perspectives and challenges*, Software Engineering for Adaptive and Self-Managing Systems (SEAMS '11) 6th International Symposium on, May 2011.
2. P. A. Alsberg and J. D. Day, *A principle for resilient sharing of distributed resources*, Software Engineering (ICSE '76) 2nd International Conference on, October 1976.
3. D. Astels, *Test Driven Development: A Practical Guide*, Prentice Hall Professional Technical Reference, 2003.
4. P. Ciancarini, F. Poggi, D. Russo, *Big Data Quality: a Roadmap for Open Data*, 2nd IEEE International Conference on Big Data Service (BigDataService '16), Oxford, UK, March-April 2016.
5. P. Ciancarini, D. Russo, A. Sillitti, G. Succi, *A Guided Tour of the Legal Implications of Software Cloning*, 38th International Conference on Software Engineering (ICSE '16), Austin, USA, May 2016.
6. P. Ciancarini, D. Russo, A. Sillitti, G. Succi, *Reverse Engineering: a Legal Perspective*, 31st Annual ACM Symposium on Applied Computing (SAC '16), Pisa, Italy, April 2016.
7. V. De Florio, *On resilient behaviors in computational systems and environments*, Journal of Reliable Intelligent Environments, 1(1), pp. 33–46, 2015.
8. V. De Florio, *Software assumptions failure tolerance: Role, strategies, and visions*, Architecting dependable systems VII, Springer Berlin Heidelberg, pp. 249–272, 2010.
9. V. De Florio, C. Blondia, *The algorithm of pipelined gossiping*, Journal of Systems Architecture, 52(4), pp. 235–256, 2006.
10. N. Dragoni, S. Giallorenzo, A.L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, L. Safina, *Microservices: yesterday, today, and tomorrow*, arXiv preprint arXiv:1606.04036, 2016.
11. C. Le Goues, T. V. Nguyen, F. S. and W. Weimer, *GenProg: A Generic Method for Automatic Software Repair*, IEEE Transactions on Software Engineering, 38(1), pp.54–72, 2012.
12. C. D. Locke, *Software Architecture for Hard Real-Time Applications: Cyclic Executives vs. Fixed Priority Executives* , The Journal of Real-Time Systems, 4(1), pp. 37–53, 1992.
13. M. Missiroli, D. Russo, P. Ciancarini, *Agile for millennials: a comparative study*, 1st International Workshop on Software Engineering Curricula for Millennials (SECM '17), Buenos Aires, Argentina, May 2017.
14. M. Missiroli, D. Russo, P. Ciancarini. *Una didattica Agile per la programmazione*, Mondo Digitale, 15(64), 2016.
15. M. Missiroli, D. Russo, P. Ciancarini, *Learning Agile Software Development in High School: an Investigation*, 38th International Conference on Software Engineering (ICSE '16), Austin, USA, May 2016.
16. M. Monperrus, *Principles of Antifragile Software*, eprint airXiv:1404.3056, 2014
17. K. H. Jones, *Engineering Antifragile Systems: A Change In Design Philosophy*, Procedia Computer Science, 32(1), pp. 870–875, 2014.
18. Z. T. Kalbarczyk, R. K. Iyer, S. Bagchi, and K. Whisnant, *Chameleon: A software infrastructure for adaptive fault tolerance*, IEEE Transactions on Parallel and Distributed Systems, 10(6), pp. 560–579, 1999.
19. R. S. Pressman and B. R. Maxim *Software Engineering: A Practitioner's Approach*, 8th ed. Mc Graw Hill Education, New York, 2015.
20. D. Russo, *Benefits of Open Source Software in Defense Environments*, Software Engineering for Defence Applications (SEDA) 4th International Conference in, May 2015.
21. D. Russo, P. Ciancarini, *A Proposal for an Antifragile Software Manifesto*, Procedia Computer Science, 83, pp. 982–987, 2016.
22. A. Messina, F. Fiore, M. Ruggiero, P. Ciancarini, D. Russo, *A new Agile Paradigm for Mission Critical Software Development*, CrossTalk, 29(6), pp. 25–30, 2016.
23. D. Russo, P. Ciancarini, T. Falasconi, M. Tomasi, *Software Quality Concerns in the Italian Bank Sector: the Emergence of a Meta-Quality Dimension*, 39th International Conference on Software Engineering (ICSE '17), Buenos Aires, Argentina, May 2017.
24. M. Schellekens, *Car hacking: Navigating the regulatory landscape*, Computer Law & Security Review, 32(2), pp. 307–315, 2016.
25. N. N. Taleb, *The Black Swan: The Impact of the Highly Improbable*, 1st ed. Random House, Inc., New York, 2007.
26. N. N. Taleb, *Antifragile: Things That Gain From Disorder*, 1st ed. Random House, Inc., New York, 2012.
27. J. Thones, *Microservices*, IEEE Software 32(1), pp. 116–116, 2015.
28. A. Tseitlin, *The antifragile organization* Communications of the ACM 56(8), pp. 40–44, 2013.