Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 151 (2019) 884–890

Procedia Computer Science

www.elsevier.com/locate/procedia

The 6th International Workshop on Computational Antifragility and Antifragile Engineering (ANTIFRAGILE)
April 29 – May 2, 2019, Leuven, Belgium

No More Snake Oil: Architecting Agility through Antifragility

Barry O'Reilly*

Black Tulip Technology 3973 Djurhamn

Abstract

The confusion surrounding the role of architecture when aiming for Agility isn't simply a labored talking point – it's part of the reason Agile initiatives fail, and architecture teams are losing influence. In a recent survey conducted by IASA Global of 260 organizations, greater than 75% were implementing some form of Agile practice, 50% of the 260 were implementing agile at scale. Of those who responded, less than 50% have integrated architecture into their agile process. As it stands, it appears agile & architecture are struggling to find a fit. This paper considers the possible impacts of a third way – Agility through Antifragility. Rather than aiming to control, or to remove control, we seek to build systems, both technical and business, that aim to be Antifragile to change. This allows the production of business and technical architectures that actually enable Agility through design rather than process or 'mindset'. Taking ideas from Systems Engineering, and Complexity Science, and data from IASA's 2018 Architecture Survey, we explore how the inherent interconnectedness of architecture and Agility can be leveraged to make the management of complexity something all organizations can do.

A previous version of this article appeared in Cutter Business Technology Journal, www.cutter.com July 2018 Architecture + Agile: The Yin & Yang of Organizational Agility

Keywords: Antifragility;Design;FMEA;Decomposition

---

* Corresponding author.
E-mail address: barry@blacktulip.se

1. Enterprise Software and VUCA: The Need for a New Approach

The modern business environment is a strange place, if visited by the manuals and best practices of yesteryear. The end of Taylorist management science [1] is, according to some, clearly in view. [2] Indeed, the complexities of the modern world refute the join-the-dots MBA business playbook. The world of VUCA (volatility, uncertainty, complexity, ambiguity) [3] requires a new approach. Disintermediation, globalization, market upheaval, disruption, and technological advance all combine to produce an effect that is difficult to mitigate, impossible to predict, and arduous to detect. The software crisis, [4] first defined in 1968, is entering a new phase, and the consequences of continued shoulder shrugging are becoming ever more serious.

Witness the growth of the Agile industry, with its ceremonies, high priests, and rituals. It has, quite rightly, found the zeitgeist: the decline of management science and the pseudo-scientific pretense of order in the domain of complex human systems. This is what causes Agile mysticism; we know that waterfall will not work, so we reject it based on past experience but do not replace it with anything demonstrably better. This creates the gap for "snake oil." The diagnosis of the multiple failings of waterfall is completely correct; yet the results of the Agile cure do not seem to bear the weight of investigation.

A 2017 report of 300 UK/US-based CIOs demonstrates the problem: 21% of Agile projects end in complete failure (i.e., nothing delivered), and 68% of CIOs want to see more architects involved in Agile projects. [5] Moreover, the projected cost of Agile failure is 37 billion British pounds (US $48 billion). Yet, a recent IASA Global survey reveals that over 75% of 260 responding organizations are implementing some form of Agile practice, and 50% are implementing Agile-at-scale. However, less than 50% of all respondents have integrated architecture into their Agile process.

In an environment where both inflexible and unstable software can lead to business failure, modern businesses need both the flexibility espoused by Agile practitioners and the rigor of more structured systems engineering methodologies. This contention is the source of much debate and confusion between the Agile and architecture camps and requires an alternative architectural approach. Thus, we propose that by *architecting for antifragility*, businesses can gain real agility and deliver systems with a higher level of quality. Risk analyst and scholar Nassim Nicholas Taleb describes an antifragile system as one that gains from disorder; a system that becomes stronger when exposed to stressors (even unpredictable or unknown stressors). [6] An antifragile system is by definition agile and resilient.

2. Accepting Complexity

Complex systems, under which most contemporary business-critical systems would be classified, are not merely complicated. They are systems that cannot be assumed to behave in a certain way and have nonlinear responses to changes in input. Consider the concept of the Platonic fold, [7] which tells us that the act of modeling the world simplifies it to the point where any decisions made based on that model are misinformed due to details omitted for the sake of hiding complexity. This is also called 'Hidden Intelligence Syndrome" [8]. Thus, dynamic real-world problems twist and bend, while the static solution cannot keep up (referred to as Horning Syndrome) [8], causing the demise of quality. In software, this leads to a multitude of problems, including shortened life span, patching, and quality issues.

When humans build complex systems, they tend to fail, often catastrophically, because of Platonic folding. The solution to the Platonic fold requires accepting complexity as something we can neither predict nor control, along with accepting the limitations of modeling and risk management. Instead of pursuing correctness in these areas, we should aim to build systems that are antifragile to fluctuations in the VUCA elements (i.e., the system becomes stronger as the business environment warps and changes with time).

3. Antifragility in Software

Due to extensive research being carried out on the subject of computational antifragility, many solutions to this kind of problem will emerge in the future. [9] It is important to realize that the degree of fragility of a system is often a function of its internal structure. The ability of a system to change under stress is governed by the interconnectedness of its parts, how strongly they are tied to each other, and how much change ripples through the system. Therefore, there is a need to ensure that we match the level of interconnectedness of a system's components with the effort required to reorganize them in the face of change. This is something that architects are well qualified to do.

For many years, the decomposition of software systems has been held captive by the latest technological trends, vendor interests, and a slow-shifting mindscape. Many students of software engineering still hold fast to ideas of elegance and reuse, often making software unnecessarily complex in the process. There has, however, been a broad library of dissent against these methods, dating back to 1972. Software engineering pioneer David Parnas' ideas on nonconventional decomposition [10] tell us that we can build better systems by focusing on what will change rather than what will happen functionally, while software architect Juval Löwy's important distinction between functional- and volatility-based decomposition via the IDesign approach [11] provides some ideas and techniques that make this easier.

Each of these methods relies on focusing on the elements that can change, rather than on concrete requirements. By building a system where the primary requirement is the ability to handle change, a very different piece of software is constructed than would happen otherwise. This need for change in design philosophy — away from building to specific requirements and toward building systems that are antifragile — has been expressed elsewhere, including at NASA. [12] Kjell Jørgen Hole's book Anti-Fragile ICT Systems illustrates that systems demonstrating high levels of antifragility have the following four properties: [13]

- Modularity (consisting of separate, linked components)
- Weak links (a low level of interconnectedness between components)
- Redundancy (the presence of more than one component to cope with failure)
- Diversity (the ability to solve a problem in more than one way with different components)

4. Antifragile Systems Design

The Antifragile Systems Design process guides the architect to optimize and balance the four antifragile properties mentioned above with the VUCA elements present in a project. With a few days of analysis and design work, we can shift any project in the direction of antifragility, without incurring a great deal of overhead. The Antifragile Systems Design process mixes ideas from complexity science and systems engineering to create a method to guide the design effort.

This process embraces the complexity in building dynamic systems. Following the advice of Taleb, Parnas, Löwy, and others, we need to focus on what we do not know before focusing on what we do know — accepting our limitations and our inability to predict the future. Indeed, the Antifragile Systems Design process is not fixed but can grow and change with every project. With this new architectural approach, the intention is not to create yet another framework or silver bullet, but to provide a starting point for a new type of design process. This process follows several simple steps and requires no more tooling than an Excel spreadsheet.

4.1. Who Takes This On?

The steps outlined below require a mix of skills within business, business architecture, and software engineering. However, this is not simply a business activity or a software design activity and cannot be divided into different tasks for different silos; each step in the process creates feedback loops to ensure that answers arrived at are coherent. Antifragile Systems Design requires an organization to move as one toward solving the problem of complexity, which means changing the perspective from "us vs. them" (IT vs. business) to simply "us" (business). Business leaders, business/ enterprise architects, and software architects all need to engage with the process to make it work. This requires a new approach from both architects and business leaders.

Architects need to work with the business to describe the VUCA environment, translate the impacts on the software decomposition, and even assist in business- level mitigations. Currently, few architects span this range; therefore, a business architect and a software architect often must work together to guide the process. However, it is possible for a single architect (business/ architecture-focused or software-focused) who combines business understanding and software engineering knowledge to guide the process.

Business leadership plays an important role in enabling the architects and the project to embrace this approach. By employing Antifragile Systems Design at a high level, business leaders can learn to ask the right questions of their software teams and quickly assess the stability of an initiative.

4.2. Step 1: VUCA Analysis

In the first step, we describe the VUCA environment for this particular initiative, listing the VUCA elements with regards to the business model, and begin to sketch our architecture. We design the system to cope with fluctuations based on the VUCA elements identified in the business model, meeting each challenge with a change in one or more of the four antifragile properties of the system.

This exercise starts at the business level, with input from business leaders. It identifies VUCA elements in the business model and clarifies what business mitigations, if any, are in place or need to be in place. This step can actually help improve the business processes or organizational structure behind the initiative. This kind of work is usually carried out by the business, but rarely shared in detail with architects. VUCA analysis requires the following actions:

- Represent the initiative's business model using the Business Model Canvas [14] and its standard building blocks.
- Perform a VUCA analysis, noting everything considered volatile. For example, what can change? What happens if a partner is acquired or ceases trading? What happens if a cost escalates? This is a useful exercise for the organization and can educate the architect in how the wider market works.
- Run through everything that is uncertain. For example, what do we not know? What is purely guesswork? What impact can a lack of knowledge have on the system?
- Run through all complexities (processes that have nonlinear responses to input) and ambiguities. Explore the impact of being wrong about something and what would need to change to accommodate the error.
- Record this in a spreadsheet with a list of VUCA elements and the corresponding mitigations.
- Choose the most appropriate mitigation for each VUCA element, excluding those too expensive or unrealistic.

Note that this exercise does not involve trying to predict the future, but rather having an awareness of the types of change that can happen to a system. We cannot predict all change, but we can work with what we know.

4.3. Step 2: System Decomposition — Flow First Design

The next step is to propose a system design. Here, we use Flow First Design, a design process for distributed systems, described briefly below:

- Describe the software as a series of data flows enabling the functional requirements.
- Create a component decomposition for each flow that is completely decoupled from all others and all data sources; the flow is its own system. This creates a system with very low levels of interconnectedness.
- Subject each data flow to the fluctuations described in the VUCA analysis.
- Ensure that the mitigations listed in the VUCA analysis are represented in the software.
- Consolidate different flows, reducing the level of interconnectedness; aim for minimum disruption when each VUCA element changes, as described by Parnas. [15]

This allows the architect to refine system decomposition by measuring the system's ability to meet changes likely to happen based on the VUCA analysis. The system decomposition now relates to both functionality and system behavior. This step establishes the right level of modularity and weak links, the first two properties of systems demonstrating high levels of antifragility and connects them to the VUCA elements identified previously.

This step requires knowledge of software engineering patterns and the management of coupling; however, it does not require a detailed knowledge of software development. It is enough to be able to ascertain that a VUCA fluctuation will have a minimal level of impact on the system.

4.4. Step 3: Design Testing

In this step, we present the architecture to various stakeholder groups through an exercise such as the Architecture Trade-Off Analysis Method (ATAM). [16]

This ensures that all concerns have been addressed and that the VUCA analysis was accurate and promotes confidence in the role the architect has played by providing a sense of rigor and demonstrating a potentially robust and resilient system.

4.5. Step 4: Modified FMEA

Failure Mode Effects Analysis (FMEA) [17] is a Six Sigma technique that helps manage quality in a system by investigating how the system will cope with failure. Using FMEA, we can investigate system behavior and adjust the architecture to be resilient to failure during operations. However, in this step, we do not attempt to prioritize or predict risks or criticality, as this pro- vides little benefit when dealing with complex systems. FMEA includes the following actions:

- Create a FMEA spreadsheet listing the different ways each component can fail.
- Record how failure is detected and mitigated and the impact of component failure.
- Aim for a high level of automation.
- Change the system design to accommodate mitigation of these failures.
- Repeat the process for any number of failure modes until the mitigations become repetitive.

This step in the process tunes the system to have the right balance of redundancy and diversity (the last two properties of systems demonstrating high levels of antifragility), pushing the system toward antifragility. This step also protects against the risk that too many mitigations can produce an overcomplicated system. In such a case, FMEA will struggle to mitigate all known errors at a reasonable cost and will send the architect back to the VUCA analysis for a more realistic take on what can change or to the decomposition step to redraw the system scope.

5. Why This Process Works

To make this process work, we can leverage the idea of *exaptation*, [18] where an element of a system developed for one purpose can have serendipitous effects for another purpose. Building a wall in your house, for example, allows spreading the load of the roof, but also provides the basis for rooms, stops noise traveling between rooms, and gives privacy. A wall also stops fire from spreading, provides somewhere to hang paintings, and a place to bang your head against when dealing with Agile coaches. When we combine two separate mitigations, say the wall and the fact that we added a space in the wall for insulation, we suddenly create the conditions for dealing with something we did not see coming — hiding electrical wires in the wall!

In working through the list of VUCA elements, tweaking the design, and adding mitigations, with each mitigation the system becomes antifragile to that particular VUCA element. The first 10 are usually tricky, but after 50 mitigations, a pattern emerges: many of the VUCA elements in the list are resolved by previous mitigations and the effect of mitigations can be said to be nonlinear. By following this process, the system trends toward antifragility, which is the only possible good result in a complex environment that we do not control. When this process repeats as part of the FMEA step, the likelihood of future exaptation increases. The VUCA analysis also builds confidence among stakeholders that the system will be "robust," but, as architects, we know that we are doing much more than that: we are providing the bedrock for antifragility! We call this pattern *nonlinear system responsiveness*.

Once a system is in place, the Antifragile Systems Design process becomes iterative. Every failure is considered feedback and the system should be strengthened by the team by rerunning the process. The best example of this kind of system is Microsoft's Azure or Amazon's AWS cloud platforms — outages are used to strengthen the platform, with these two platforms becoming some of the most resilient in the world.

While the idea of nonlinear system responsiveness seems intuitive, it has as of yet no proven mathematical basis and is not guaranteed to occur. However, by aiming to induce it, we at least make the system less fragile and provide the basis for a positive, nonlinear response. The actual degree of exaptation can never be predicted and never be complete (all systems will die someday), but this process actively encourages exaptation as the premier focus of the design effort.

6. Concrete Actions for Business Leaders

Going forward, business leaders should consider the following actions:

- Understand that complexity is the key cause of software failure.
- Don't waste time and take unnecessary risks by trying to predict and control the unpredictable and uncontrollable.
- See software execution as a business task with varying results that requires constant monitoring beyond status reports.
- Use VUCA analysis to understand the stability of IT delivery. "What happens if?" questions tell you all you need to know about a software project's quality. Bring the architect into the core business team and make VUCA analysis a natural part of your execution.
- Enable your architects to embrace antifragility as the key to real agility.
- Understand that current industry trends around Agile cannot deliver in the face of complexity. Use the VUCA analysis process to have a voice and influence in the direction of software projects and ensure quality is there from the start.
- Demand traceability in architectural decision making.
- Ensure that technical decisions are grounded in a shared understanding of the VUCA environment and are FMEA-tested.

7. Concrete Actions for Architects

Going forward, architects should consider the following actions:

- Practice VUCA analysis on the initiative's business model. A thorough grounding in business basics is required, which can be a challenge for technically focused solution architects. This is a necessary evolution of the role of the architect and cannot be avoided.
- Become an expert in software decomposition.
- Learn different methods for software decomposition, the difference between service-oriented architecture and microservices, the IDesign Method, and Flow First Design. Learn how modern cloud applications are composed and the major components involved.
- Learn to use modified FMEA to improve system designs.

8. Conclusion

The result of this work is a business with a better understanding of its own fragility and a software system capable of bending and meeting the needs of the changing business environment. This kind of process calls for a new type of architect and a new type of architecture. It requires a solid understanding of the business environment, the effects of change on the business architecture, and a thorough understanding of how software can be decomposed, rather than written. This cross-set of skills can allow architecture to contribute by designing antifragile systems that enable agility and answers the business question of how to become resilient to the VUCA world.

There is no guaranteed result from this process, so the Taylorist approach of measurement, prediction, and comparison will not provide any benefit here. Over time, this approach will succeed for some and fail for others, and this lack of certainty may cause many to resist the approach. The alternative — to do nothing and wait for machine learning and complexity science to solve problems — is not a viable option for today's enterprises.

Acknowledgements

Many thanks to Dr. Riccardo Bennett-Lovsey and Tanya O'Reilly for their valuable comments and suggestions on the drafts of this article.

References

[1] "Taylorism." *Encyclopaedia Britannica* (https:// www.britannica.com/science/Taylorism).
[2] Stacey, Ralph D. *Complexity and Organizational Reality: Uncertainty and the Need to Rethink Management After the Collapse of Investment Capitalism*. 2nd edition. Routledge, 2010.
[3] Bennett, Nathan, and G. James Lemoine. "What VUCA Really Means for You." *Harvard Business Review*, January-February, 2014 (https://hbr.org/2014/01/what-vuca-really-means-for-you).
[4] "Software crisis." Wikipedia (https://en.wikipedia.org/wiki/ Software_crisis).
[5] Porter, Chris. "An Agile Agenda: How CIOs Can Navigate The Post-Agile Era." 6point6, April 2017 (https:// cdn2.hubspot.net/hubfs/2915542/White%20Papers/ 6point6-AnAgileAgenda-DXWP.2017.pdf).
[6] Taleb, Nassim Nicholas. *Antifragile: How to Live in a World We Don't Understand*. Allen Lane, 2012.
[7] Taleb, Nassim Nicholas. *The Black Swan: The Impact of the Highly Improbable*. 2nd edition. Random House, 2010.
[8] De Florio, Vincenzo Software Assumptions Failure Tolerance: Role, Strategies, and Visions https://repository.uantwerpen.be/docman/irua/979199/2471.pdf
[9] De Florio, Vincenzo. "Antifragility = Elasticity + Resilience + Machine Learning Models and Algorithms for Open System Fidelity." *Procedia Computer Science*, Vol. 32, 2014 (https:// www.sciencedirect.com/science/article/pii/S1877050914006991).
[10] Parnas, David L. "On the Criteria to be Used in Decomposing Systems into Modules." *Communications of the ACM*, Vol. 15, No. 12, 1972 (https://dl.acm.org/citation.cfm?id=361623).
[11] Löwy, Juval. "Volatility-Based Decomposition." IDesignIncTV, 22 November 2013 (https://www.youtube.com/watch? v=VIC7QW62-Tw).
[12] Jones, Kennie H. "Engineering Antifragile Systems: A Change in Design Philosophy." *Procedia Computer Science*, Vol. 32, 2014 (https://www.sciencedirect.com/science/article/pii/ S1877050914007042).
[13] Hole, Kjell Jørgen. *Anti-Fragile ICT Systems.* Springer, 2016.
[14] "The Business Model Canvas." Strategyzer, 2018 (https:// strategyzer.com/canvas/business-model-canvas).
[15] Parnas (see 9).
[16] Kazman, Rick, Mark H. Klein, and Paul C. Clements. "ATAM: Method for Architecture Evaluation." Technical Report, Software Engineering Institute/Carnegie Mellon University, August 2002 https://resources.sei.cmu.edu/library/ asset-view.cfm?assetid=5177).
[17] "FMEA — Failure Mode and Effect Analysis." Six-Sigma.se, 2007 (http://www.six-sigma.se/FMEA.html).
[18] Exaptations." Understanding Evolution, 2018 (https:// evolution.berkeley.edu/evolibrary/article/exaptations_01).