O'REILLY®

Antifragile Systems and Teams

Dave Zwieback

Antifragile Systems and Teams

Dave Zwieback

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo O'REILLY®

Antifragile Systems and Teams

by Dave Zwieback

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editor: Mike Loukides

April 2014: First Edition

Revision History for the First Edition:

2014-04-21: First release

2015-03-24: Second release

ISBN: 978-1-491-94796-8

[LSI]

Table of Contents

Antifragile Systems and Teams

Impermanence

Systems break because of change, and we often go to great lengths to prevent or manage change with heavy-handed, bureaucratic change-management processes. What we forget is that systems also *function* because of change.

For instance, computer systems function through state transition: 0s changing to 1s and back again. More fundamentally, computers work because 0s *can* change into 1s, because computer systems are *changeable*.

This is a somewhat subtle point, one that's easy to overlook in search of more tangible conditions required for systems to function or malfunction. But it's worth repeating: the fundamental reason that systems start, stop, or continue working—the one root cause of all functioning systems and all system outages—is their changeable, impermanent nature.

More broadly, impermanence is a fundamental property of all compounded things, i.e., those that consist of two or more parts. All of the systems that we work with certainly have two or more parts (how about five billion parts for the upcoming Xbox chip?).

So how can this theoretical and philosophical understanding of impermanence help us? How can we make impermanence usable and useful?

First, impermanence is useful because it reminds us that *all* functioning systems will eventually break down. Understanding impermanence frees us from looking for the "single root cause" of outages, and

from the mistaken belief that there is none. (Sidney Dekker proclaims that "What you call 'root cause' is simply the place where you stop looking any further."[1]) The single root cause of all outages—and all functioning—is impermanence, the changeable nature of all systems.

Second, having accepted impermanence, we (as engineers) cannot accept that things break or function entirely randomly. If we mixed two parts hydrogen with one part oxygen and sometimes got water and at other times ice cream, we might as well give up our chosen profession! While we may not be able to identify all the conditions required to build or keep the system running or all those that result in malfunctions, we can certainly identify *some* of the conditions. Engineering, above all, is the discipline of studying conditions and their effects.

Most important, we can identify some of the conditions that we can actually impact. While a butterfly fluttering its wings in Africa may be one of the conditions of a hurricane resulting in a power outage in Northern Virginia, we don't have any control over the butterfly, but at least we have some say in whether our data centers have backup power generators.

Fragile, Robust, and Beyond

While all systems break eventually (because they're all *breakable*, impermanent), they differ in their robustness to stress. For instance, imagine you have a very important document that you want to protect and share widely. Let's say you decide to store it on a physical hard disk (the spinning platters variety) in a server connected to the Internet via a single network connection. This particular system would be fragile, since a failure of any single component (disk, server, network, etc.) would make the file unavailable, possibly forever. Moreover, if the file became widely popular, its popularity would negatively impact its availability, as any of the subsystems could easily be saturated.

As we can see, fragile systems dislike and are harmed by even small amounts of stress (a.k.a. volatility, randomness, disorder, or impermanence). One way to reduce fragility is to increase redundancy: we could build a more robust system by removing single points of failure by mirroring the disks, setting up multiple servers in high-availability

1. Dekker, Sidney. *The Field Guide to Understanding Human Error: Second Edition*. Farnham, Surrey, UK: Ashgate Publishing, 2006.

configurations and distributing them geographically, adding multiple higher-bandwidth Internet links, etc. While this system would be more robust and increase the availability of the document, it would remain vulnerable to somewhat rare but catastrophic conditions like the leap second bug (because all servers ran the same version of the OS) or equipment seizure by the government. Moreover, this is a considerably more complex system than a single-server one, and it will at times function and break in unexpected and unpredictable ways.

We usually think of robust systems as the opposite of fragile ones because they don't to care too much about (i.e., neither like nor dislike) stress. In fact, robust systems are merely less fragile. The true opposite of a fragile system would be one that actually *benefits* from stress. Nassim Nicholas Taleb calls such systems *antifragile*.

In the case of the aforementioned file, imagine we stored it in a system that made it more available the higher the demand for it was. In fact, BitTorrent is precisely this type of system: the more our file is requested, the more robust to failure and available it becomes because parts of it are stored on a progressively larger number of computers. In fact, the best-case scenario for a file stored on BitTorrent is that every person on earth would want to download it and then share it. This scenario is exactly the absolute worst case for both fragile and robust systems. Note also that our cost of distributing this file would remain constant—not so for the cost of making systems more robust to anticipate higher demand or improve resiliency.

Antifragility

The main property of antifragile systems is that the potential downside due to stress (and its retinue) is lower than the potential upside, up to a point. Taleb defines this asymmetry as follows: you are antifragile to event intensity between x^{low} and x^{high} if you are better off after the event than before (up to x^{high}). The rare, entirely unpredictable event (a.k.a. the Black Swan event) in which every person on earth wants to download a copy of a file distributed on BitTorrent would actually make the file maximally available, robust to failure, and practically impossible to delete.

BitTorrent is similar to another human/social system with antifragile properties: information shared via gossip. The more someone (e.g., a government or a self-righteous group) tries to suppress, criticize, or disprove some information (e.g., an idea or a book), the more wide-

spread and persistent it becomes. Critics of Miley Cyrus did her a great favor with their outraged responses to her 2013 VMA performance—a smarter strategy would have been to ignore the train wreck entirely. In fact, Brendan Behan's famous observation that "there's no bad publicity except an obituary"[2] provides a nice summary of antifragility: the downside to gossip and negative publicity (e.g., damage to one's reputation) is smaller than the upside (e.g., selling more records), up to a point (e.g., death).

The longevity and popularity of some books is due in part to being banned at one time or another; in fact, 46 of the 20th century's top 100 novels were targets of ban attempts. From my childhood in Soviet Russia, I remember my parents and their friends staying up nights to read a samizdat version of *The Gulag Archipelago*, despite the fact that those found in possession of the banned manuscript faced long prison sentences. This illustrates another often overlooked property of antifragile systems: antifragility by layers. Being banned may help books become antifragile, while reading banned books can make individuals fragile.

More generally, complex systems contain myriad components, sometimes organized in hierarchies or layers. As in the above example, sometimes antifragility is achieved at a certain layer of a complex system at the cost of fragility of another. Another example of antifragility by layers is vaccination: while it appears to benefit populations, vaccines are known to cause serious side effects for some individuals. That is, vaccination makes a population antifragile because the downside (a small number of individuals having negative side effects) is small in comparison to the upside (an entire population gaining immunity to a disease). However, vaccination certainly makes the specific individuals who wind up having side effects or die from it fragile.

DevOps and Antifragility

DevOps is a modern organizational philosophy, which can be applied to improve the antifragility of complex systems. ("Systems" includes people as well as computers and software.) Let's explore how the layers of DevOps (Culture, Automation, Measurement, and Sharing) contribute to the overall antifragility of organizations that adopt this philosophy.

2. McCann, Sean. *The World of Brendan Behan*. London: Twayne Publishers, 1966. 56.

Culture, Part 1: Managing the Downside

Etsy is known for deploying to production 40+ times per day. Netflix routinely terminates production instances at random or introduces latency into their application. An architect working on HP printer firmware "would purposely make changes that would break the code until it was architecturally correct."[3] What makes all these examples acceptable or even possible within organizations? Fundamentally, these organizations have realized that the potential downside of frequent changes (otherwise known as volatility, variance, etc.) is smaller than the potential upside. This knowledge has become part of the culture at these companies. This is also the classic asymmetry of pain versus gain required to make these companies and their systems antifragile.

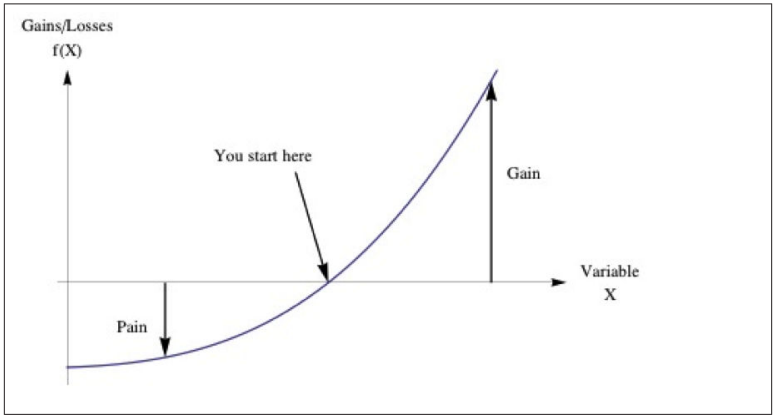

Figure 1. The Antifragile asymmetry between potential downside and upside

The major upside of embracing higher volatility is that customers receive higher-quality products and services (i.e., value) faster and at a lower cost than is possible with traditional, risk- and volatility-averse approaches. The often overlooked and critical point is that this is only possible not only because these companies have identified potential

3. Gruver, Gary, Mike Young, and Pat Fulghum. *A Practical Approach to Large-Scale Agile Development: How HP Transformed LaserJet FutureSmart Firmware*. Boston: Addison-Wesley, 2012.

upside, but also because they have become experts at minimizing the downside:

> At Etsy "this isn't license to break stuff, quickly. *Engineer-driven QA and solid unit testing are integral parts of the process.*"[4]
>
> Netflix "is known for being bold in its rapid pursuit of innovation and high availability, but not to the point of callousness. It is *careful to avoid any noticeable impact to customers from these failure-induction exercises.*"[5]
>
> At HP "we could tell when [the architect] had found and plugged a hole because a *particular test area would have a huge drop in passing tests or the build would break completely.*"[6]

It is the robust testing, quality assurance, and (continuous) deployment practices that enable these companies to reduce the possibility of any individual change causing a catastrophic event. Both Etsy and Netflix have become experts at quickly going back to a previously working version of software should the newly deployed one be found defective. Amazon takes this up a notch, performing "rollbacks" automatically in the event that a deploy causes problems. This is one of the reasons that only about 0.001% of software deployments at Amazon cause outages, even with the mean time between deployments a staggering 11.6 seconds! At the same time, in all these companies, some of the potentially risker changes (e.g., database schema or OS updates on core routers) are certainly *not* performed frequently or continuously.

We should be careful *not* to attribute the antifragility of the above companies entirely to effective testing or QA, or to the frequency and the skill with which they release to production. For comparison, consider Knight Capital, which was a large global financial services firm until trading losses forced it to be acquired in a fire sale. Knight *appears* to have had reasonable testing and QA processes and an agile-like, iterative software development and deployment approach similar to the one practiced by Etsy and Netflix:

1. Initial development phase with no market interaction.

4. *http://slidesha.re/1fsqzHB*
5. *http://bit.ly/1e9i40t*
6. Gruver, Gary, Mike Young, and Pat Fulghum. *A Practical Approach to Large-Scale Agile Development: How HP Transformed LaserJet FutureSmart Firmware*. Boston: Addison-Wesley, 2012.

2. Production test phase at a limited range from the trading desk. New strategy could be tested on one order in one symbol.
3. Wider testing, possibly of a larger order in a single stock and then further analysis of what happened versus what was expected.
4. Once the strategy can conduct multiple orders in a variety of securities across different markets, potential beta users are recruited.
5. When the algorithm has passed all tests, it's deployed as a new strategy and a performance review is done.

And yet, this turned out to be little more than "risk management theatre". On August 1, 2012, "It took only *one defect* in a trading algorithm for Knight Capital to lose $440 million in about 30 minutes. That $440 million is three times the company's annual earnings. The shock and sell-off that followed caused Knight Capital's stock to lose 75 percent of its value in two business days."[7]

Hidden beneath the seemingly sensible processes was the possibility for one massive, negative Black Swan event—rare, unpredictable, and catastrophic—to wipe out the company. In hindsight, it appears that Knight Capital did not manage their downside effectively, and the potential downside was much larger than the upside, which made the company fragile and eventually resulted in its demise.

Culture, Part 2: Skin In the Game

Historically, developers designed and produced software, then "threw it over the wall" to operations people, who deployed and supported the software in production. It was often the operations people who would be responsible for the nonfunctional requirements such as security, scalability, resilience, backup and recovery, and so on. When things broke, as they invariably did (see: impermanence), it was the operations folks who would be summoned to fix them at the proverbial midnight hour.

This is an example of the transfer of fragility, in this case from developers to operations. In organizations where this is possible or even encouraged, developers essentially have no *skin in the game* in operating the software that they write; in fact, having transferred all the

7. *http://bit.ly/1e9i7t4*

downside (and fragility) to operations while keeping all the upside (e.g., promotions, bonuses, and uninterrupted sleep), they become (temporarily) antifragile to writing and deploying defective software!

This transfer of fragility results in the all-too-familiar "downward spiral," the depressing antipattern that is so brilliantly documented in *The Phoenix Project* by Gene Kim et al.:[8] the more frequently bad code makes it into production, the less willing the ops people are to deploy it. With less frequent deploys, more (bad) code must be deployed each time, increasing the complexity of deployment and the probability of outages, and making ops even more reluctant to deploy. This ultimately results in fragility of the entire system and organization, paralyzing or even causing some companies to go out of business.

It's telling that the very opposite of this schism between developers and operations is encoded in the word "DevOps," which originally focused on bringing developers and operations together. (Sadly, this is still the colloquial understanding of the term, although DevOps does not only apply to dev and ops.) More than just collaboration, what DevOps relies on—and what gives DevOps organizations the possibility of becoming antifragile—is skin in the game. It's no surprise that in companies like Etsy, Netflix, and Facebook, developers themselves deploy much of the code and are called upon to fix things when they break.

More broadly, DevOps aims to align entire organizations (not just operations and developers) toward common goals, which ensures that local optimization (the absence of skin in the game) doesn't make the overall organization fragile. This alignment is possible at more than a superficial level only when unnecessary organizational barriers and silos are removed, and when information is flowing openly and freely.

Not surprisingly, so-called generalists (a.k.a. multi-disciplinarians, polymaths, Renaissance men and women, comb-shaped people, or erudites)[9] gravitate toward and thrive in DevOps organizations and become their glue. Attracted by the fluidity of roles and the fast pace of change, they connect disparate parts of the organization and are unafraid to dive into new and unknown areas of knowledge. In contrast, overspecialized individuals, who've spent years or decades hon-

8. Kim, Gene, Kevin Behr, and George Spafford. *The Phoenix Project*. IT Revolution Press, 2013.

9. *http://bit.ly/1fsqBPH*

ing a narrow set of skills, are typically put off by what they perceive as chaos.

Finally, it's worth noting that generalists themselves are antifragile. Because they can do (and learn) many things and typically are not attached to becoming super-experts in any particular field, they are able to take advantage of a diverse set of opportunities during their careers. Generalists do have a small downside of not being "qualified" for rare, highly compensated, and super-specialized jobs. But their upside is much bigger than the downside: the more wide and diverse their knowledge, the more opportunities they have. In contrast, specialists may be more highly prized and compensated (especially in large organizations) and are also subject to catastrophic events in their careers, such as when the database they spent 20 years dissecting suddenly falls out of favor.

Culture, Part 3: Tinkering (or Continual Experimentation and Learning)

Much time is spent on planning in traditional organizations, often by company managers, in a process that excludes non-management employees. Planning is certainly not without benefit; however, it's worth noting two facts: 1) many of the most important breakthroughs and discoveries in science and technology happened by accident (e.g., penicillin)[10] as opposed to being results of planned research, and 2) many of the worst human-made disasters resulted from centralized, top-down planning (e.g., the devastating famines in Soviet Russia and North Korea, which were both centrally planned economies).[11]

Of course, it's impossible to predict groundbreaking discoveries in advance—they are positive Black Swan events. However, it is possible to increase the likelihood of stumbling on these breakthroughs, namely through continual experimentation and learning (otherwise known as the Third Way of DevOps). Taleb calls this "antifragile tinkering," where

> …mistakes are small and benign, even reversible and quickly overcome. They are also rich in information. So a certain system of tinkering and trial and error would have the attributes of antifragility. If

10. *http://bit.ly/1e9i807*
11. *http://bit.ly/1fsqC6v*

> you want to become antifragile, put yourself in the situation [that] "loves mistakes" …by making these numerous and small in harm.[12]

For example, when Google experiments continuously, and not just for user-visible changes, they're doing so in an antifragile way, because the costs of failure are small. Famously, Gmail, Google Maps, Adsense, Google Talk, and many other products were born from Google's (now supposedly defunct) "20% time", which encouraged engineers to tinker and experiment on projects they found interesting. The potential downside was limited to 20% of the company's time, but the payoff has been immense: a reported 50% of Google's products originated from this unstructured, unplanned research. Adsense alone has been responsible for over 25% of Google's revenue! As we've seen before, this favorable asymmetry of potential losses and gains is key to antifragility.

Continual tinkering is not just random, aimless wondering. As Taleb says, "We use randomness to spoon-feed us with discoveries—which is why antifragility is necessary." Failures provide meaningful, useful signals: having failed, we know what does not work and what not to do. So much the better to fail quickly—and inexpensively—and eventually stumble onto a positive Black Swan.

Automation and Measurement: A Cautionary Tale

Both automation and measurement are key components of implementing DevOps. As tools of the trade they are concrete, easily graspable artifacts of DevOps culture, and they receive the bulk of attention from the public. Sometimes this results in equating the DevOps philosophy with implementing the tools; of course, running Chef or Puppet (the most popular open source configuration management tools, often labeled as "DevOps tools") will not magically transform an organization into a DevOps (or an antifragile) one.

Certainly the automated deployment and configuration management tools increase both the speed and ease of deployment, and the uniformity of systems. They may also enable anyone in the company to continuously deploy into production thousands of times per day, supporting the kind of antifragile tinkering discussed above. Similarly,

12. Taleb, Nassim Nicholas. *Antifragile: Things That Gain from Disorder*. New York: Random House, Inc., 2012. Kindle locations 675—678.

measuring and analyzing the results of experiments is absolutely required in order to determine what's working and what's not.

Overreliance on measurement and automation can make systems fragile. For instance, airline regulations now require the use of autopilot in most flying situations. Such an extensive use of automation may have contributed to make this the safest aviation period on record. However, the FAA has recognized that "pilots' 'addiction' to automation…has eroded their flying skills to the point that they sometimes don't know how to recover from stalls and other mid-flight problems."[13] Such events are rare and unpredictable, but their potentially catastrophic outcomes (compared to their benefits) make obvious that excessive dependence on automation is fragile. (Ironically, when fatal crashes do happen, air travel actually becomes safer for those not killed in the accident—air travel, as a whole, is antifragile.)

Similarly, if the only way that software can get deployed or systems configured is through automation tools, we are subject to a catastrophic Black Swan event when automated systems are unavailable when we attempt to deploy or configure and manual ways of accomplishing these tasks are unknown or not immediately available. Moreover, one of the biggest benefits of automation tools is that they make systems uniform, which can also be one of the conditions for a serious outage, as illustrated by the leap second bug mentioned previously. Artificially constraining or enforcing uniformity is a recipe for fragile systems.

With respect to measurement, due to a host of cognitive biases, humans are prone to constructing stories that feel good, but are untrue. We may find correlation or causation where there is none, or get stuck in analysis paralysis, always looking for more and Bigger Data:

> Yes, it's true that a team at Google couldn't decide between two blues, so they're testing 41 shades between each blue to see which one performs better. I had a recent debate over whether a border should be 3, 4 or 5 pixels wide, and *was asked to prove my case*…I've grown tired of debating such minuscule design decisions. There are more exciting design problems in this world to tackle.[14]

13. *http://bit.ly/1e9i9kR*

14. *http://bit.ly/1fsqFPI*

Sharing

Sharing is a critical part of the DevOps philosophy. Within a DevOps organization, sharing is the key element of the Second Way, made visible by an increasing "tribal knowledge," trust, and a greater level of collaboration between development and operations and throughout the entire organization.

Within the wider community of DevOps practitioners, there is an emphasis on sharing the tools and configuration (via open source) and knowledge (e.g., DevOps architectural patterns and antipatterns via articles and gatherings such as devopsdays). Most encouraging is the fairly recent tendency among DevOps organizations to openly share detailed postmortems after outages, some of which are potentially embarrassing. (Compare this to the fact that, to this day, what actually happened at Knight Capital is not public record and remains a matter of speculation. This is despite the publication by the SEC of "Release No. 70694", which has some of the details of the incident, but is not a full postmortem.) This sharing is antifragile: the benefits of sharing such knowledge far outweigh the potential downside for all those involved.

Is DevOps Antifragile?

Just because an organization states that it has adopted DevOps, that doesn't necessarily mean that it is antifragile as a whole, or even that it exhibits antifragile properties on some level. However, as we've shown previously, there is significant overlap in practices of DevOps organizations and those that seek the benefits of antifragility. It is also possible that the reason why DevOps works is that, instead of attempting to constrain volatility in an unnatural and harmful way, DevOps embraces and makes use of disorder, randomness, and, ultimately, impermanence.

Acknowledgments

First and foremost, this paper is based on—and would be impossible without—Nassim Nicholas Taleb's superb and life-changing book, *Antifragile: Things That Gain from Disorder*, published in 2012 by Random House. I blatantly stole so many of his ideas that it would be impossible to reference them all. Any misunderstandings and misrepresentation of Taleb's ideas are solely my own.

In addition, the following individuals have reviewed and materially contributed to this paper: Damon Edwards, James Turnbull, James Urquhart, Jan Schaumann, Jeff Sussna, Jez Humble, John Allspaw, John Willis, Oleksiy Kovyrin, Stephen Nelson-Smith, Todd Deshane, and Walter Blaurock.

About the Author

Dave Zwieback has been managing large-scale, mission-critical infrastructure and teams for 17 years. He is the VP of Engineering at Next Big Sound. He was previously the Head of Infrastructure at Knewton, managed UNIX Engineering at D.E. Shaw and Co. and enterprise monitoring tools at Morgan Stanley, and also ran an infrastructure architecture consultancy for seven years. Follow Dave @mindweather or at *http://mindweather.com*.