DOI:10.1145/3152722

Stephen J. Andriole

Viewpoint
The Death of Big Software

We are past the tipping point in the transition away from 20th-century big software architectures.

WHY WOULD ANYONE undertake a multiyear software project today? Or upgrade an in-house-hosted legacy application? Or build—or use—anything that behaved like a monolithic software application? Big software project failure data is legendary.[11] There are myriad horror stories with titles like "9 VERY Scary ERP and ERP System Implementation Statistics."[12] The Standish Group actually labels their annual technology project analyses as "Chaos Reports."[14] They reported that 66% of all technology projects completely or partially failed in 2015.

So assuming that management is reasonably well informed, it knows that big software projects are likely to fail. Yet in the 1990s and early 21st century there were still companies willing to try their hand with big software and prove they were unlike the others who failed so spectacularly. In spite of this unjustifiable optimism, many of these companies also failed. Even the U.S. Defense Department failed spectacularly.[6]

So that no one thinks that failure only plagues ERP applications, the data suggests all kinds of big software projects fail.[13] Big customer relationship management (CRM) projects fail. Big database management systems (DBMS) projects fail. Big infrastructure projects fail. Big communications projects fail. In fact, most software projects designed to address enterprisewide problems with single, integrated platforms fail. Failure crosses vertical and functional areas as well, including retail, government, financial services, and even science.[10]

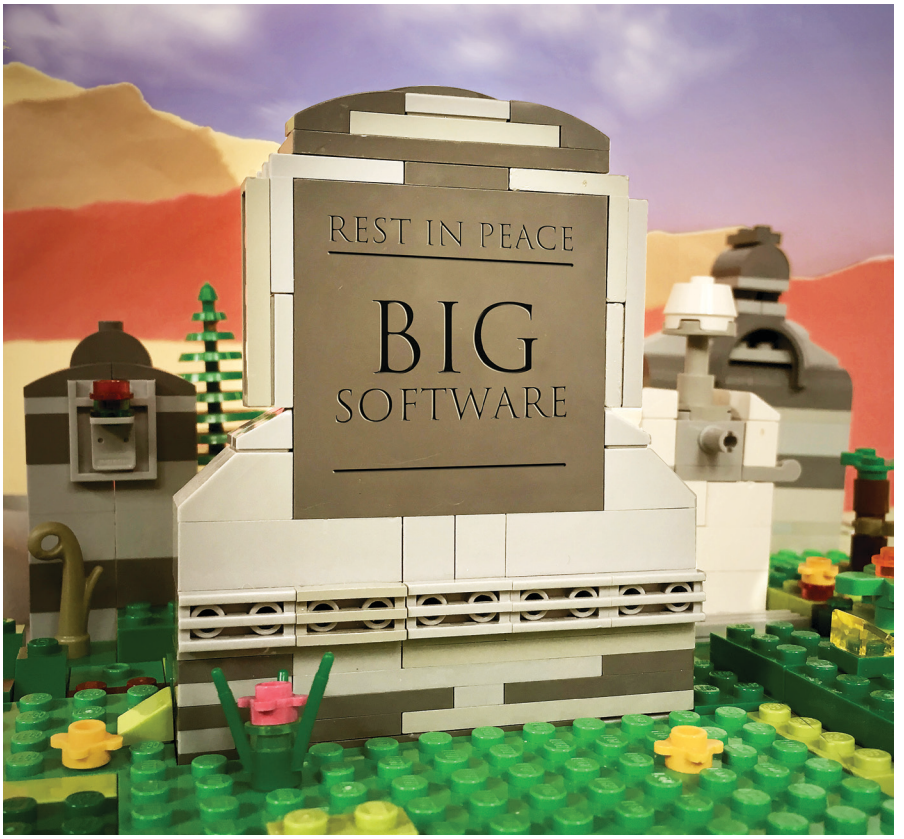

IMAGE BY ALICIA KUBISTA/ANDRIJ BORYS ASSOCIATES

The high rate of failure helped kill big software. But there were other causes of death.

Causes of Death

Big software is dead. There were lots of assassins. Some were all business, some were hiding in the governance trenches, some were up in the clouds and some were architectural. Let's look at the assassins in a little detail.

One of the *business* assassins was control. When a company embarks on a multiyear journey with a big software vendor it cedes significant—if not total—control to that vendor and the business processes embedded in the

code. For example, ERP modules were originally designed to eliminate process chaos. Remember when there were no intra- or intercompany (or industry) standardized processes? Remember when software applications never integrated? Remember when 1970s and 1980s "legacy" software was a barrier to scalability, not to mention how expensive it was to customize and maintain? ERP vendors came to the rescue by controlling the mess that homegrown applications created. But one of the side effects was the loss of process control to the vendors who defined supply chain management, financial reporting, and other business processes for the companies (and industries) they de facto managed.

While tightly bundled standardized software made some sense back in the day, it makes little or no sense in the era of *digital transformation* where disruptive business processes and business models are seen as necessary paths to competitiveness: *disruption and standardized big software are not birds of a feather*. Of course, in 1995 would have seemed heretical. Companies were desperate to end the chaos of uncoordinated business processes and rules. Standardized processes incarnated in software were the vitamin pills everyone needed. But in retrospect it is not clear that everyone understood exactly what they were consuming. When business models moved slowly in the 20th century, slow-and-steady worked, but when whole new "disruptive" business models began to appear in the 21st century (fueled by new and more powerful digital technologies), slow-and-steady became a clear threat to competitiveness.

Governance also killed big software. Big software projects that are "standardized"—that is, *required*—by corporate technology groups also usually failed, not because they did not work as advertised (which they often did not) but because of the governance that *forced* a one-size-fits-all approach to technology use. Huge off-the-shelf software packages—like ERP, CRM and DBMS packages—or even large custom in-house developed applications mandated by corporate IT—usually failed under the weight of their own governance which, to make matters worse, often resulted in increased "Shadow IT" spending.[1,2]

The *cloud* also killed big software. Years ago, companies would implement huge software systems in their own data centers. Armies of programmers would work with armies and navies of (happy) consultants to bring big systems to life. Some years later the software might "launch" with a "switch" that—according to the data—usually failed (at least the first time). So the armies and navies would go back to work to get it right (until they got it right). Implementation cost was also a killer. $10M often turned into $50M, which often turned into $250M and sometimes into billions: the Standish Group reports that big technology projects run anywhere from 40%–50% over budget—and deliver less than 50%–60% of the promised ROI.[14] Cloud delivery changed all that: it is now possible to access an enterprise application directly from the cloud from any number of providers.

While implementation pain was avoided through cloud delivery, process control was still ceded to the big software vendors who owned the embedded business processes in the cloud-delivered software (while some of the control went to the cloud provider who deployed the systems on behalf of their clients). While it was almost always cheaper (by total cost of ownership [TCO] metrics) to move from on-premise big software applications to cloud hosted applications, companies were still denied access to the transformational and disruptive playing fields.[4,18,a]

a TCO debates around on-premise-versus-cloud continue. There are all sorts of ways to compare costs across services, and many of the results will vary depending on individual services such as SaaS, IaaS, PaaS, and other cloud-based services. But a full comparison should include variables like agility, governance and long-term costs connected with training, testing, upgrades and security, among others. If a company is looking to get out of the technology business by moving its operational and strategic technology to the cloud, it will find ways to justify the cost model. If there is a bias toward keeping everything in-house then favorable on-premise cost models can be developed. The larger question is around core competency: Does the company want to be in the technology acquisition, deployment and support business—or not? The cloud offers opportunities to reassess core competencies and enables rational for various cost models, though ideally the models are based on empirical evidence.

Finally, some of the assassins were (sometimes unknowingly) *architects*. The overwhelming technical complexity and inflexibility of huge, standardized software systems also explain the death of big software. Enormous whole-company projects were often beyond the capabilities of even the most experienced project and program managers—especially when there is never 100% consensus about the need for a total enterprise project in the first place. High-level functional and non-functional requirements were nearly impossible to comprehensively define and validate; detailed requirements were even more elusive.

But perhaps the real architectural assassin was monolithic software design. Many of the big software architectures of the 20th century were conceived as integrated functional wholes versus decoupled services. Over time, monolithic architectures became impossible to cost-effectively modify or maintain and—much more importantly—became obstacles to business process change. The trend toward microservice-based architectures represents an exciting replacement to monolithic architectures (see below).

The Rise of Small, Cloudy Software

There are also small software cloud-based alternatives that scale, integrate, and share process control through customization tools deliberately built into smaller, more manageable platforms. Companies can find lots of incredibly inexpensive alternatives, from vendors like Zoho and Zendesk, among many others.[b]

While "small" software packages also embed business rules and processes, they are built in smaller, more integrate-able pieces, which provides much more flexibility to clients who want to mix-and-match (existing and new) functionality.

The major driver of software change is continuous digital transformation. Big standardized software systems conceived in the 20th century were not designed to adapt or self-destruct the moment a company or industry pivots.

Software architectures must be blank canvasses capable of yielding tiny pictures or large masterpieces.

Another way of thinking about all this is the relationship between micro and macro (or monolithic) services. *Big software begins with macroservices in monolithic architectures*.[3,5] Or we could just think about all this as small versus large programming.[8]

Architectural assassins argue that monolithic architectures are stiff, inflexible, and unyielding. They are also difficult and expensive to maintain primarily because functionality is so interconnected and interdependent. They also argue that monolithic architectures should be replaced by microservice-based architectures.[16,17] According to Annenko,[3] "the concept is rather easy, it's about building an application consisting of many small services that can be independently deployed and maintained, don't have any dependencies but rather communicate with each other through lightweight mechanisms and lack a centralized infrastructure. It is even possible to write these small (micro-) services each in its own language." Why microservice-based architectures? Annenko continues: "their benefits are undoubted, too: they easily allow for continuous deployment, and certain parts of an application can be changed, debugged or even replaced quickly and without affecting the rest. With microservices, you absolutely cannot break an application: if something goes wrong, it will go wrong only within its own microspace, while the rest of the application will continue working as before."

Was there any doubt that these architectural assassins would hit their target?

All of that said, SOA architecture dreams continue to develop.[9] The big data world, for example, has already defined an open source architecture that is fast, flexible, cost-effective—and always changing.[15] The tools enable low latency and real-time processing through Spark and Flink, among other open source tools. The details are specified in tools like Lamda, Kappa, and SummingBird. MapReduce moved us from parallel processing, and file systems have evolved from Google File Systems to Hadoop. Building on Hadoop, Spark and Flink provide real-time runtime environments. Even data streaming has been addressed with tools like Storm and Spark Streaming. But while SOA complements microservice-based architecture, they are different.[7] SOA is not the threat to monolithic big software that microservice-based architecture is; in fact, SOA often behaves like a big software vitamin supplement. Said differently, SOA is not a replacement for monolithic big software and is therefore not a big software assassin.[c] But candidly, SOA-based integration and interoperability have proved illusive in spite of continued promises and a growing library of open source application programming interfaces (APIs) and Web services. SOA is still more of a dream than an answer for continuous digital transformation. It might, in fact, be the wrong answer.

In addition, cloud delivery is becoming increasingly flexible. Container technology offered by companies like Docker offers freedom to companies who may need to pivot away from

b Zoho (www.zoho.com), Zendesk (www.zendesk.com). Also see: https://www.getapp.com/customer-management-software/a/zoho-crm/alternatives/ and https://www.crowdreviews.com/zoho-crm/alternatives

c Clark[7] describes the differences simply: "microservices architecture is an alternative approach to structuring applications. An application is broken into smaller, completely independent components, enabling them to have greater agility, scalability, and availability. SOA exposes the functions of applications as more readily accessible service interfaces, making it easier to use their data and logic in the next generation of applications."

The entire world of traditional big software design, development, deployment, and support is dead.

their cloud providers to another provider for any number of reasons. Containers enable clients to retain control over their applications just as emerging application architectures enable them to retain control over their software-enabled business processes.[19] This means that dependencies are shrinking. So the combination of microservice-based architectures and container technology may be the response to monolithic applications.

Will the big software vendors respond? Yes.

They will milk the current big enterprise revenue streams for as long as they can and then systematically make their offerings to look more and more like their small software competitors. Many of them, like SAP and Oracle, have already by necessity begun this process through small business and mid-market cloud offerings that are much cheaper than the gold-plated goliaths they sold for years. They began to cannibalize their own products because they too know that the days of big software are numbered. But they have not fundamentally rearchitected their applications. They have shrunken them.

The Death and Resurrection of Software

The entire world of big software design, development, deployment and support is dead. Customers know it, big software vendors know it and next generation software architects know it. The implications are far-reaching and likely permanent. Business requirements, governance, cloud delivery and architecture are the assassins of old "big" software and the liberators of new "small" software. In 20 years very few of us will recognize the software architectures of the 20th century or how software in the cloud enables ever-changing business requirements. C

References

1. Andriole, S. Who owns IT? *Commun. ACM 58*, 8 (Aug. 2015).
2. Andriole, S., Cox, T. and Khin, K. T*echnology Adoption & Digital Transformation*. CRC Press, 2017.
3. Annenko, O. Breaking down the monolithic: Microservices vs. self-contained systems. DZone, June 2016; http://bit.ly/2dEfFBG
4. Boisvert, G. Cost of Server Ownership: On-Premise Vs. IaaS. SherWeb, Sept. 2015; http://bit.ly/2z3Sg9l
5. Brown, S. What is agile software architecture, *Coding the Architecture*, 2013; http://www.codingthearchitecture.com/2013/09/03/what_is_agile_software_architecture.html
6. Charette, R.N. U.S. Air Force blows $1 billion on failed ERP project. *IEEE Spectrum*, Nov. 2012; http://bit.ly/2zim1El
7. Clark, T. Microservices, SOA, and APIs: Friends or enemies?: A comparison of key integration and application architecture concepts for an evolving enterprise. IBM DeveloperWorks, Jan. 2016; https://ibm.co/2zhsMWR
8. DeRemer, F. and Kron, H.K. Programming-in-the-large versus programming-in-the-small. *IEEE Transactions on Software Engineering*, 2 (June 1976); http://bit.ly/2xyLvZM
9. Erl, T. et al. *Next Generation SOA: A Concise Introduction to Service Technology & Service-Orientation*. Prentice Hall, 2015.
10. Gorton, I. Cyberinfrastructures: Bridging the divide between scientific research and software engineering, *Computer 47*, 8 (Aug. 2014); 48, 55; http://bit.ly/2yWVjkf
11. Kimberling, E. Key Findings from the 2015 Report. Panorama Consulting, Apr. 2015; http://bit.ly/2hpGwWo
12. Lee, J. 9 VERY scary ERP and ERP system implementation statistics. ERP/VAR, Oct. 2014; http://bit.ly/2yukxFj
13. Leibowitz, J. IT project failures: What management can learn. *IEEE IT Professional* (Apr. 2016); http://bit.ly/2ynZhlF
14. Lynch, J. The Chaos Report. The Standish Group, 2015; http://bit.ly/2zScMqv
15. Madan, A. 100 open source big data architecture papers for data professionals. LinkedIn, (June 2015); http://bit.ly/1UEZdRt
16. McLarty, M. Microservice architecture is agile software architecture. *Infoworld*, May 2016; http://bit.ly/24hvrnD
17. Proctor, S. From monolith to microservices: Big rewards from small software architecture. *IT World Canada*, (Aug. 2016); http://bit.ly/2iglbgk
18. Tomkins, B. SaaS solutions 77% cheaper than on-premises. *Information Week*, (May 2010); http://ubm.io/2z4wAd9
19. Townsend, K. Containers: The pros and the cons of these VM alternatives. *TechRepublic*, Feb. 2015; http://tek.io/2nfzjav
20. Wailgum, T. 10 famous ERP disasters, dustups and disappointments. *CIO Magazine* (Mar. 2009); http://bit.ly/2zv2mxK

Stephen J. Andriole (steve@andriole.com) is the Thomas G. Labrecque Professor of Business at the Villanova School of Business at Villanova University where he teaches and conducts research in emerging technologies, requirements modeling and business technology strategy. His most recent book is *Ready Technology: Fast Tracking Emerging Business Technologies* (CRC Press, 2014).

The author thanks the reviewers who significantly improved the article. With their help, the "death of big software" message was clarified especially regarding the discussion of microservice-based architectures.