The *Communications* website, http://cacm.acm.org, features more than a dozen bloggers in the BLOG@CACM community. In each issue of *Communications*, we'll publish selected posts or excerpts.

twitter
Follow us on Twitter at http://twitter.com/blogCACM

DOI:10.1145/3474351 http://cacm.acm.org/blogs/blog-cacm

Finding the Art in Systems Conversions, Naming

Doug Meil considers a third distinct type of development, while Mario Antoine Aoun ponders alternate names for ACM.

Doug Meil

The Art of Speedy Systems Conversions

https://bit.ly/2TpmcsG
June 1, 2021

Building a software system de novo is the baseline way that software engineering is taught and understood. Use cases are identified, architectures and patterns are designed, and then software is implemented and deployed. Users are onboarded. This kind of green-field development can be exhilarating opportunity to create anew. Upgrading an existing system is a second and more frequent type of development as for any given system there is only one initial release but many subsequent releases. While upgrades primarily focus on incremental improvements, it is arguably a more complex case as upgrades are primary risks of outages and functional regressions, whereas with the baseline case there is nothing else in place at the time of initial release.

But what if there is a prior operational system in place? Specifically, one that is being replaced.

System conversions represent a third distinct type of development. The project scope now includes all of the effort of an initial software release plus an entirely new set of complexities. The prior system is often caught in a downward spiral: technical constraints may exist that make upgrades difficult, which in turn can diminish the organizational will to improve the system, which in turn reduces system performance and viability. The prior system, however, must be kept alive long enough to transition the functionality as well as support the data conversion to a new platform. This can become an anxiety-inducing software "race against time." As an example of life imitating art, the 1994 action movie *Speed* with Keanu Reeves offers some surprisingly insightful lessons and how this situation can be managed.

Lesson #1: The Bus Couldn't Slow Down

In the movie, a transit bus is wired with a bomb and cannot go below 50 MPH without dire consequences. From a software standpoint, if an existing system is highly utilized and still running critical functions but not well maintained, it can feel like this. There may be multiple factors all pulling on the existing system to slow it down: an outdated and non-scalable architecture, an outdated codebase, and perhaps even a lack of developers to support the aforementioned items. Ignoring the current system, though, only makes the problem worse.

Lesson #2: A Second Bus Was Required

To save the initial bus, a second bus had to be obtained. In the software world, the "second bus" represents the new system and the development team to create that system. This could either be managed as one team with two major responsibilities (support old system, build new system) or two teams, but one thing is clear: there is effectively twice as much work. A key mistake of system conversion development is only budgeting for the "new" development.

Lesson #3: The Second Bus Was Accelerated to Catch the First Bus

Achieving functional parity is one of the most difficult aspects of system conversions especially when the first bus has a 100-mile head start, metaphorically speaking. The "chasing system" needs a long-enough runway both in terms of time and budget, complicated by the fact that the prior system may still continue to evolve at the same time and not be a static target. Even the most well-intentioned projects can get tripped up on this. This type of development could take multiple fiscal quarters or years, and one of the biggest issues is executive expectation management.

Lesson #4: The Passengers Are Rescued

In the movie, the passengers are rescued in dramatic fashion, and anyone that has lived through a large system conversion will recognize this is pretty much what it feels like. To rescue the passengers, both buses must be operating not just at high speed, but also close proximity, re-emphasizing the importance of feature parity. Having a second bus running 50 MPH but 5 miles distant and receding doesn't help.

Additionally, software to assist in conversions—particularly large-scale data migrations—is required and is a special art. Such software still needs to adhere to software engineering best practices, but also needs to be fast (as conversion windows are always under a time crunch), explainable (as conversions are always being asked to explain exactly what happened), and automatable (as the best conversions are always heavily practiced).

The management of conversions is an important aspect of software engineering and not for the faint of heart. The process represents the bridge from the old to the new.

Lesson #5: The First Bus Was Retired

In the movie, the first bus exploded spectacularly after the passengers were rescued. In real life, such kinetic outcomes are not generally desirable. Shutdown processes informed by contractual or regulatory provisions are important considerations, such as saving the existing system state for a required period of time and potentially leaving the system online in a read-only state. If a system state is saved as a backup, confirming that the backup can actually be restored is advised.

Conclusion

System conversions are a hard problem and will be ever-present in the software world as today's blue-sky development efforts become tomorrow's legacy code. Reasons for system-rot are myriad: technological obsolescence of frameworks or languages are one set of causes, but more than a few systems with reasonably current architectures have been undercut by boom-and-bust budgeting behaviors as systems are deployed with an enthusiastic initial release and then lay fallow. Technology leaders must actively manage every system in a portfolio. It's a lot of work to do this, but the alternative is worse.

Mario Antoine Aoun
The Name Game

https://bit.ly/3eYRBKN
May 19, 2021

There was a recent discussion in *Communications*' Letters to the Editor section regarding a name change for ACM. Editor-in-Chief Andrew A. Chien even encouraged sending him ideas or suggestions for new ways to rethink the letters A-C-M. I, too, thought of an interesting name change for ACM, but after careful consideration, I realized I adore the current name for its longstanding value and history.

Concerning previous suggestions made by others (see *Communications* June 2020 and September 2020), we must be careful that our association is not dedicated only to its registered members. That is, it is dedicated for advancing computing machinery as science and profession, and not just for members contributing to its mission or benefitting from it. For instance, articles are published in *Communications* or other ACM periodicals by authors who are not ACM members. Also, people (like my lovely wife, for instance) may read ACM proceedings or attend ACM conferences with attendees who are not members of the association.

I liked Andrew Chien's comment concerning name change and the idea of recursion. For that reason, I suggest the following list of potential substitutions for Association for Computing Machinery:

- Association of Computing Minds
- Association for Computing Minds
- Association for Computing Minds and Machinery
- Association of Computing Minds for Computing Machines
- Association of Computing Minds for All Computing Machines

My personal favorite is *Association for Computing Minds* because it encapsulates many meanings and its hold on ACM's mission is twofold: it works toward the advancement of computing in terms of machinery, and it works toward the advancement of computing for scientists and professionals (as per ACM's motto, "Advancing Computing as a Science & Profession"). Besides, it reminds us of Turing's paper "Computing Machinery and Intelligence," thus it implicitly offers tribute to him and explicitly to the evolution of computers while highlighting the mind and intelligence (natural and artificial).

What is interesting is 'Computing Minds' can refer to both a human and a computing machine. On one side, it gives legacy to the evolution of computers from their early invention as pure mechanical programmable calculators, as well as today's intelligent decision makers and knowledge discoverers. On the other side, it inspires programmers, software engineers, database designers, and computer scientists by calling them 'computing minds' as they create computing solutions by transforming thoughts into computing codes. In this way, we elevate 'machinery' to 'mind,' and at the same time we considered every person interested in this stuff as a computing mind, too.

Moreover, 'Association for Computing Minds' is new, novel, and unusual!

Still, as I said at the outset, I still adore 'Association for Computing Machinery' for its originality, value, and history.

What do you think?

Doug Meil is a software architect at Ontada. He also founded the Cleveland Big Data Meetup in 2010.
Mario Antoine Aoun is an ACM Professional member who has been a Reviewer for *ACM Computing Reviews* since 2006. He has 25 years of computer programming experience and holds a Ph.D. in Cognitive Informatics from the Université du Québec à Montréal. His main research interest is memory modelling based on chaos theory and spiking neurons.

© 2021 ACM 0001-0782/21/9 $15.00