The Past, Present and Future of Technical Debt

Learning from the Past to Prepare for the Future

Eoin Woods
Endava
125 Old Broad Street
London EC2N 1AR, UK
eoin.woods@endava.com

ACM Reference format:
Eoin Woods. 2018. The Past, Present and Future of Technical Debt. In *Proceedings of TechDebt '18: International Conference on Technical Debt , Gothenburg, Sweden, May 27–28, 2018 (TechDebt '18),* 1 pages.
https://doi.org/10.1145/3194164.3194181

1 EXTENDED ABSTRACT

While technical debt has emerged as a formal concept relatively recently [2] we have had technical debt from the earliest days of software development, it has simply evolved in nature. So what can we learn from past types of technical debt to allow us to prepare for its future forms?

When we look back over recent software history, we can see five identifiable evolutions of software systems [5], each one roughly aligning with a decade.

Before and through the 1980s, software systems were largely *monolithic* and tended to run on single computers, with software being developed as monolithic "programs". As we moved into the 1990s, *distributed* systems became mainstream and the standard style for an enterprise system became three-tier client server. The Internet became a mainstream technology in the late 1990s, and organisations developed *Internet-connected* systems, which were "always on" rather than just "online" and could support difficult and unpredictable quality properties. In the current era, we are building *Internet-native* systems, where "the Internet is the system". These systems are built from a combination of open source components, remote Internet connected services and custom code, and their services often form part of the Internet via publicly accessible APIs.

Following current trends, it seems that the next phase of evolution will be to *Intelligent-Connected* systems, as artificial intelligence (machine learning in particular) becomes mainstream [1], users expect context specific assistance, and fast, reliable networks allow us to connect "things" (devices) to our systems as well as traditional computers [3].

Software engineering practice evolves in response to new challenges and each era of computing has introduced new techniques and technology but each has also introduced its own types of technical debt too.

In the *monolithic* era, the focus was structuring a single program, with "spaghetti code", poor naming and unrestricted use of the "goto" emerging as examples of the earliest types of technical debt. As we moved into the *distributed* era, we ended up tangling presentation and business logic code in our client/server user interfaces, while in the *Internet-connected* era we distorted our systems to meet performance and scalability concerns at all costs and often ended up with poorly-designed automated tests being an inflexible technical debt of their own. More recently *Internet-native* systems often introduce a mishmash of microservices with poorly understood choreography and diverse internal implementations, references to external APIs that became unsupported or difficult to use and public APIs with many versions, all of which have to be maintained "forever" due to callers who would not migrate to new versions.

So what types of technical debt do we expect in the future?

In the *intelligent-connected* era, amongst other things, applications will have machine learning features and we'll need large datasets to train machine learning models and provide context-specific user experiences and we'll have lots of non-computing devices connected to our systems, providing data. So we'll probably get machine learning debt [4], ML models that we can't explain, models that we can't improve because people rely on their quirks (even if wrong). We'll also have large inflexible data sets which our systems and models rely on, and we'll have unknown and unpredictable collections of "things" connecting to our services, which we can't change because other people own them.

While this sounds like a daunting set of challenges, the intelligent-connected era is only just beginning, so we have not yet incurred significant amounts of technical debt. By looking to the past as our guide to the future we can be forewarned and start find solutions to our future technical debt before we have become overwhelmed by it!

TechDebt '18, May 27–28, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5713-5/18/05.
https://doi.org/10.1145/3194164.3194181

REFERENCES

[1] Jacques Bughin, Eric Hazan, Sree Ramaswamy, Michael Chui, Tera Alllas, Peter Dahlström, Nicolaus Henke, and Monica Trench. 2017. *Artificial intelligence the next digital frontier?* Discussion Paper. McKinsey Global Institute.
[2] Ward Cunningham. 1993. The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger* 4, 2 (1993), 29–30.
[3] Denise Lund, Carrie MacGillivray, Vernon Turner, and Mario Morales. 2014. Worldwide and regional internet of things (iot) 2014–2020 forecast: A virtuous circle of proven value and demand. *International Data Corporation (IDC), Tech. Rep* 1 (2014).
[4] D Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems*. 2503–2511.
[5] Eoin Woods. 2016. Software architecture in a changing world. *IEEE Software* 33, 6 (2016), 94–97.