Why (and How) Should Technical Debt be Managed?

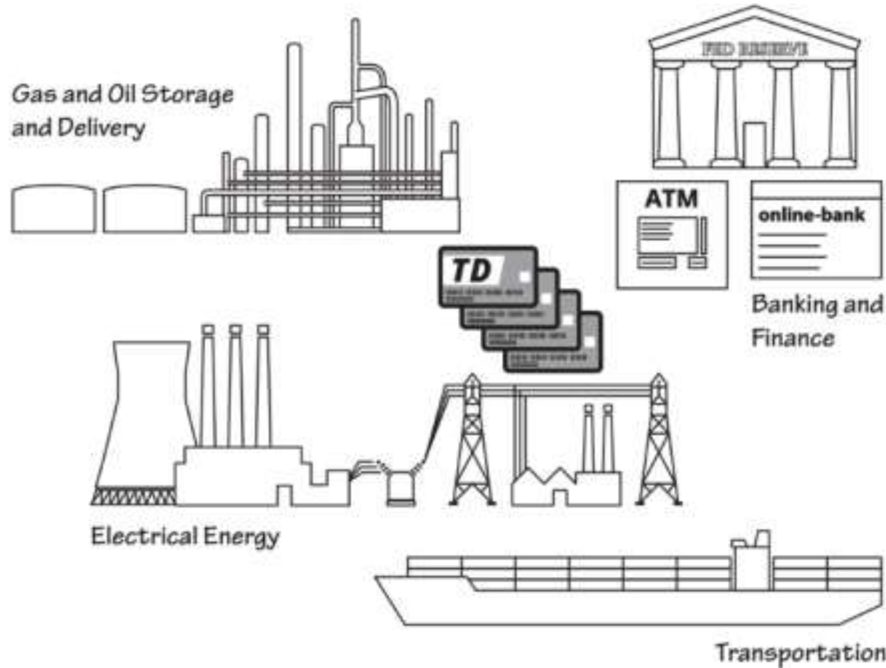
Ipek Ozkaya

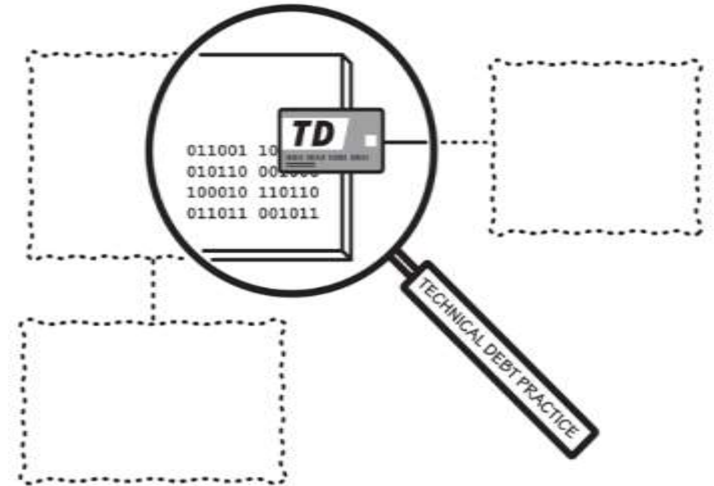
ozkaya@sei.cmu.edu

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

[Distribution Statement A] Approved for public release and unlimited distribution.

ALL SYSTEMS HAVE TECHNICAL DEBT!





© 2022 Carnegie Mellon University
[Distribution Statement A] Approved for public release and unlimited distribution.

Agenda

What is technical debt?

- Its definition and relationship to defects and vulnerabilities
- Misconceptions of what technical debt is

Detecting and making technical debt visible

Resolving technical debt items

Technical debt within the acquisition lifecycle

© 2022 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Context Matters

© 2022 Carnegie Mellon University
[Distribution Statement A] Approved for public release and unlimited distribution.

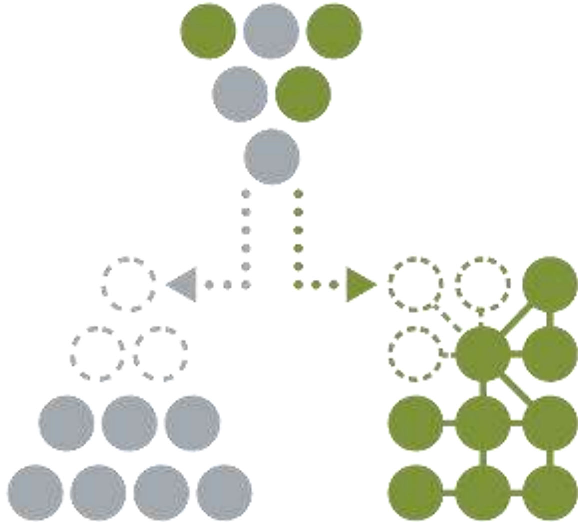
Common Consequences of Technical Debt

- Integration of products built by different teams reveals that incompatibilities cause many failure conditions and lead to significant out-of-cycle rework.
- Progress toward milestones is unsatisfactory because unexpected rework causes cost overruns and project-completion delays.
- Recurring user complaints about features that appear to be fixed and other systematic issues make the system less usable.
- Outdated technology and platforms require lengthy convoluted solutions and added complexity in maintaining or extending the systems.

© 2022 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Technical Debt: A Definition



In software-intensive systems, technical debt consists of design or implementation constructs that are expedient in the short term but set up a technical context that can make future changes more costly or impossible.

Kruchten, P.; Nord, R.L.; & Ozkaya, I. *Managing Technical Debt: Reducing Friction in Software Development.* Pearson Addison-Wesley. 2019. ISBN-10: 013564593X.

© 2022 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Technical Debt is Common

Unmanaged technical debt costs organizations time and money!

- Government's old technology deficit that needs to be replaced is estimated to be up to $7.5 billion in 2016 (https://federalnewsradio.com/reporters-notebook-jason-miller/2016/09/39000-shows-modernization-effort-matters-much/)
- U.S. Department of Veterans Affairs spending 75% of its technology budget to maintain outdated legacy systems (https://techcrunch.com/2017/06/19/how-the-presidents-american-tech-council-should-tackle-reforming-government-tech/)
- Major government programs often in the news for lack of their technical debt management (https://www.military.com/daily-news/2016/02/17/f35-deficiencies-decreasing-hundreds-remain-program-manager.html)
- High-profile industry failures are often associated with technical debt (for example, United Airlines network connectivity failure and New York Stock Exchange glitches of July 8, 2015: https://venturebeat.com/2015/07/09/3-takeaways-from-3-big-tech-outages-nyse-united-and-wsj/)

© 2022 Carnegie Mellon University
[Distribution Statement A] Approved for public release and unlimited distribution.

A Typical Technical Debt Example

A decade ago, processors were not as powerful. ***To optimize for performance, we would not insert code for exception handling when we knew we would not divide by zero or hit an out of bounds memory condition****. These areas now are hard to track and have become security nightmares.*

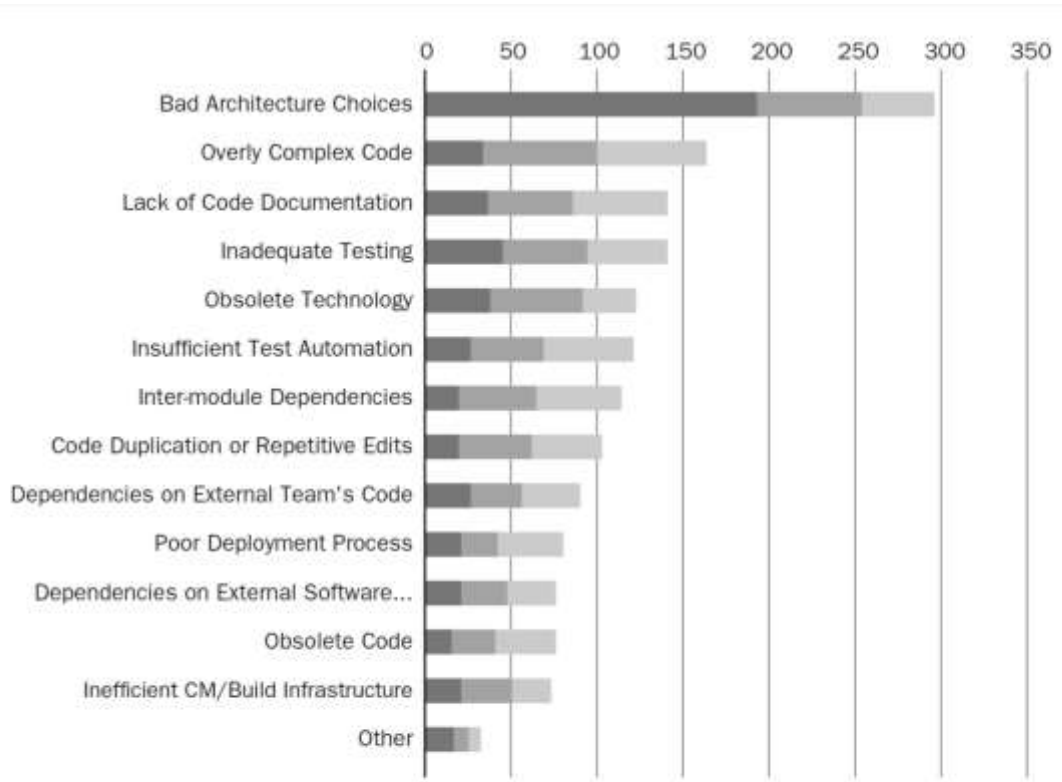
Technical debt is a **software design decision** made to solve a problem but may not stand the test of time and will cause rework. Technical debt, therefore

- Exists in an executable system artifact, such as code, build scripts, data model, automated test suites
- Often is traced to several locations in the system, implying issues are not isolated but propagate throughout the system artifacts
- Has a quantifiable and increasing effect on system attributes (e.g., increasing defects, negative change in maintainability and code quality indicators)

© 2022 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Software Architecture and Design Tradeoffs Matter



Results from over 1800 developers from two large industry and one government software development organizations reinforce that unattended architecture decisions and practices are at the root of technical debt.

Ernst N.; Bellomo, S.; Ozkaya, I.; Nord, R.; & Gorton, I. Measure it? Manage it? Ignore it? Software Practitioners and Technical Debt. In *Int. Symp on Foundations of Software Engineering*. 2015.

© 2022 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

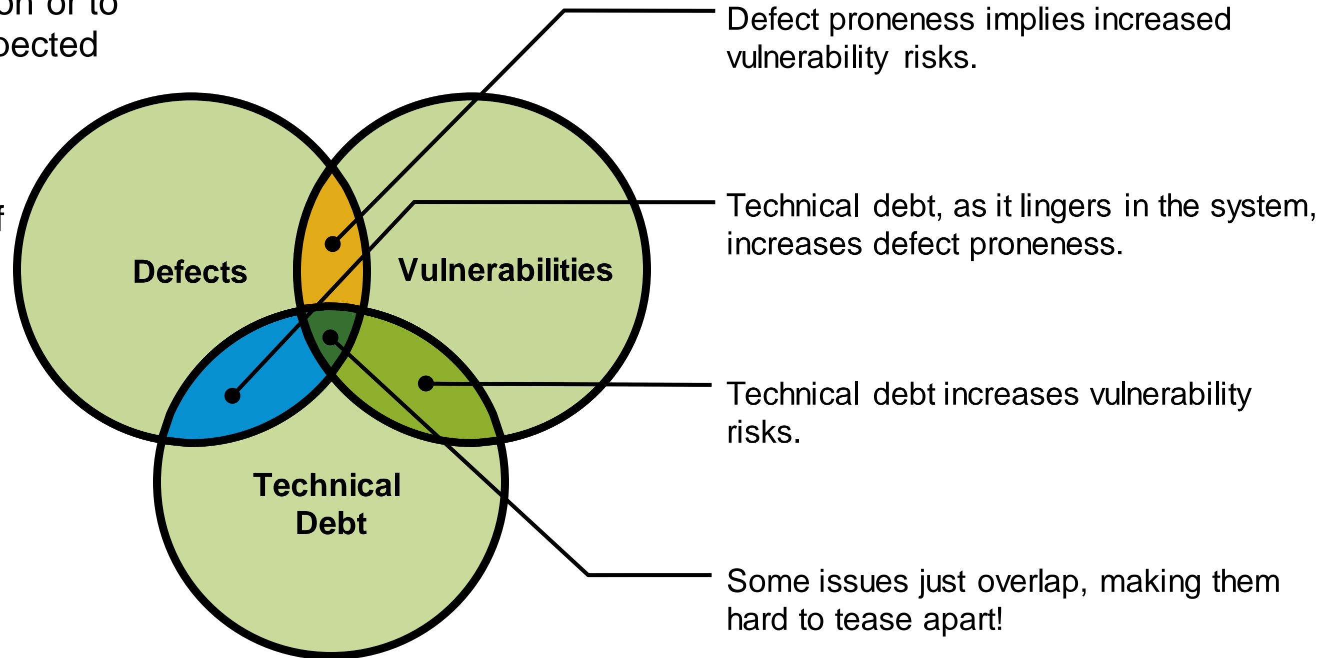
The Need to Identity Technical Debt

Defect – error in coding or logic that causes a program to malfunction or to produce incorrect and/or unexpected results

Vulnerability – system weakness in the intersection of three elements:

- system flaw
- attacker access to the flaw
- attacker capability to exploit the flaw

Technical debt – design or implementation construct traced to several locations in the system, that make future changes more costly



© 2022 Carnegie Mellon University
[Distribution Statement A] Approved for public release and unlimited distribution.

Common Misconceptions

Technical debt is not simply bad quality.

Lack of process is not technical debt.

New features not yet implemented are not technical debt.

Everything not yet done, e.g. the backlog, is not technical debt.

Code analysis tools cannot find all technical debt, nor is everything that they find technical debt

© 2022 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Agenda

What is technical debt?

- Its definition and relationship to defects and vulnerabilities
- Misconceptions of what technical debt is

Detecting and making technical debt visible

Resolving technical debt items

Technical debt within the acquisition lifecycle

© 2022 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Detecting Technical Debt

1. *Detect technical debt from code*, where code-level conformance to establish quality rules and structural analysis indicate concerns related to the structure of the system and the codebase.
2. *Detect technical debt from symptoms* that signal architecture issues.
3. *Detect technical debt from architecture* during design reviews and analysis of decisions.
4. *Detect technical debt from development and deployment infrastructure*, which are not typically part of the delivered system but may impact its delivery, security, and quality.

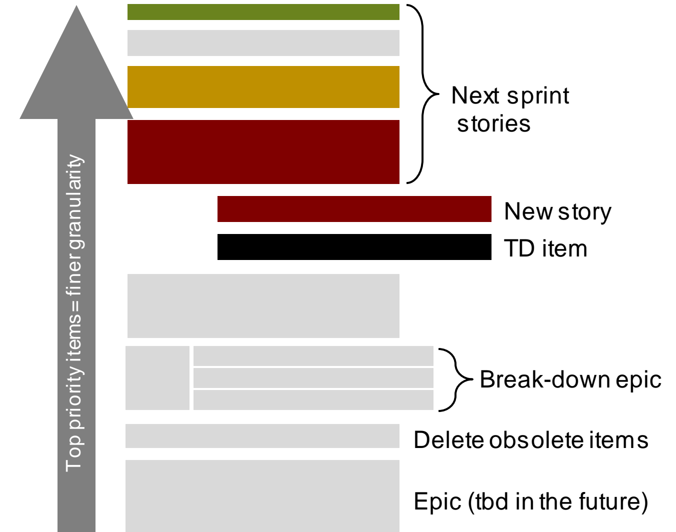
© 2022 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Making Technical Debt Visible

Making technical debt visible implies communicating and tracking technical debt in a manner that

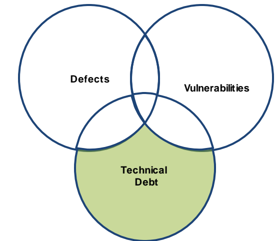
- Is timely
- Concretely identifies what and where
- Includes experienced and potential consequences
- Involves all relevant stakeholders



© 2022 Carnegie Mellon University
[Distribution Statement A] Approved for public release and unlimited distribution.

Technical Debt Examples – Detect from Code

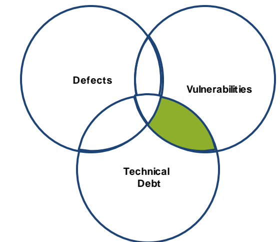
Name	Connect #Gateway-1631: Remove empty Java packages
Summary	The re-architecture of the source code to support multiple adaptor specifications has introduced a new Java packaging scheme. Numerous empty Java package folders across multiple projects.
Consequences	No impact to functionality; however, re-architecture may lead to confusion for users implementing enhancements or modifications to the source code.
Remediation approach	New and existing classes have been moved into these new package folders; however, the previous package folders have been left in place with no class files.
Reporter / assignee	Gateway developers



© 2022 Carnegie Mellon University
[Distribution Statement A] Approved for public release and unlimited distribution.

Technical Debt Examples – Detect from Symptoms

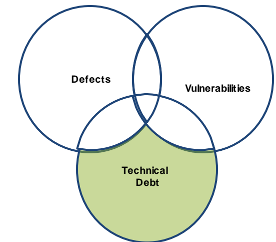
Name	Unexpected crashes due to API incompatibility
Summary	The source code uses a very large negative letter-spacing in an attempt to move the text offscreen. The system handles up to -186 em fine, but crashes on anything larger. A similar issue was fixed with a patch, but there were several other similar reports. My sense is that if we patch it here, it will pop up somewhere else later.
Consequences	We already had 28 reports from seven clients. And it definitely leaves the software vulnerable. Finding the root cause can be time consuming given that existing patches did not resolve the issue.
Remediation approach	The external web client and our software likely has an API incompatibility, but further analysis is needed. The course of action is to verify where the root of this problem is and see if we can fix it on our side. If the external web client team needs to fix it, we would need to negotiate.
Reporter / assignee	DevSecOps Team / External Web Client Team



© 2022 Carnegie Mellon University
[Distribution Statement A] Approved for public release and unlimited distribution.

Technical Debt Examples – Detect from Architecture

Name	Publish/subscribe design likely to not meet real-time timing requirements as message load increases
Summary	We are not able to guarantee delivery of messages in priority order with the current pub/sub design. We already need to deal with increasing number of messages, which is slowing down performance. We may likely have missed messages as well.
Consequences	We will not be able to meet the real-time requirements, hence not pass the tests in the lab, which will delay production.
Remediation approach	Given the size of the code base, switching to a different messaging approach is too risky. However, we can supplement our existing pub/sub architecture with a message prioritization approach with some local architectural changes
Reporter / assignee	Reported during system integration test /assigned to the architecture team



© 2022 Carnegie Mellon University
[Distribution Statement A] Approved for public release and unlimited distribution.

Agenda

What is technical debt?

- Its definition and relationship to defects and vulnerabilities
- Misconceptions of what technical debt is

Detecting and making technical debt visible

Resolving technical debt items

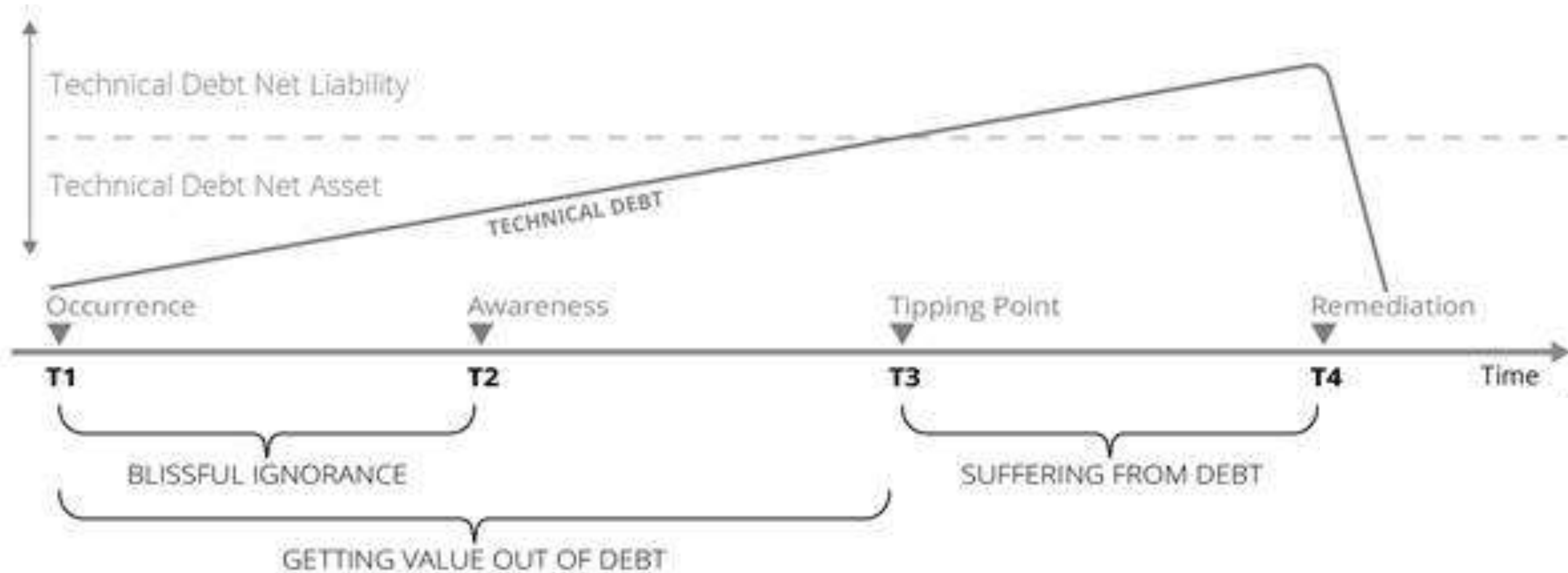
Technical debt within the acquisition lifecycle

© 2022 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Manage the Technical Debt Timeline

Each technical debt item will have a different timeline.



© 2022 Carnegie Mellon University
[Distribution Statement A] Approved for public release and unlimited distribution.

Recommendations for Managing Technical Debt Items

Create a dedicated technical debt item label in your issue tracker.

Ensure that you are only identifying technical debt symptoms through static analysis within the CI/CD tool chain, but also uncovering the technical debt related to the symptoms.

Add technical debt identification at multiple points in the software development lifecycle, including architecture reviews, cybersecurity analysis, and code quality analysis.

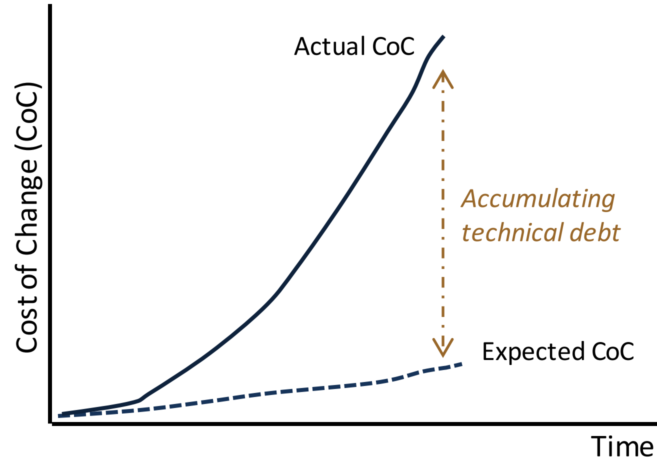
Monitor technical debt based on cost of rework, new technical debt items identified, and high-priority technical debt items that need to be resolved.

Triage identified technical debt items regularly based on their priority and decision to when and if to resolve.

© 2022 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

The Cost of Accepting Technical Debt



For each instance of technical debt make informed tradeoff decisions about remediation.

- Understand range of consequences of keeping and resolving the debt
- Measure what you can
- Qualitatively assess what you can't
- Reconcile data with assessments
- Develop resolution strategies

© 2022 Carnegie Mellon University
[Distribution Statement A] Approved for public release and unlimited distribution.

Agenda

What is technical debt?

- Its definition and relationship to defects and vulnerabilities
- Misconceptions of what technical debt is

Detecting and making technical debt visible

Resolving technical debt

Technical debt within the acquisition lifecycle

© 2022 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Technical Debt Management and the Acquisition Lifecycle

Include language on how technical debt will be managed in contracts, including

- Percentage of resources to be withheld until high-priority technical debt is resolved
- Data to be shared throughout the development lifecycle
- Ongoing analysis to be conducted and its results shared
- Incentives to share technical debt the contractor takes on

Include technical debt discussions as part of assessments, and request use of both appropriate software quality tools and architecture reviews.

Request evidence from contractors and continuously assess where you are on the technical debt timeline.

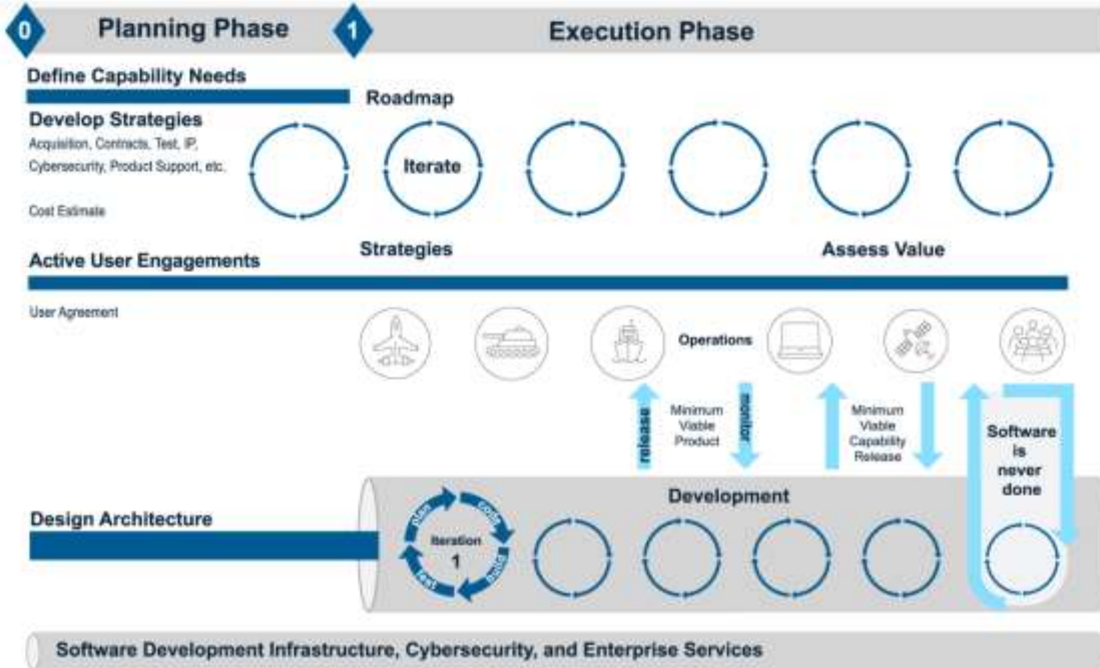
- Helpful data includes commit histories, defect logs, testing results, architecture conformance measures, and software quality analyses.

© 2022 Carnegie Mellon University

[Distribution Statement A] Approved for public release and unlimited distribution.

Acquisition Pathways – Software

An iterative and incremental, architecture focused process which includes proactive technical debt management is recommended.



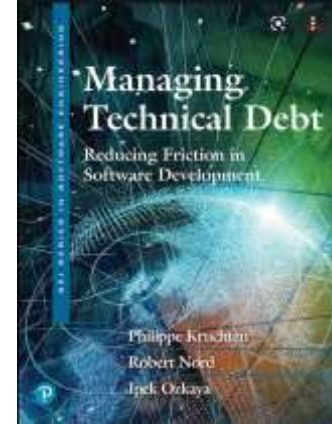
Programs will maximize use of automated software testing and security accreditation, continuous integration and continuous delivery of software capabilities, and frequent user feedback and engagement. **Programs will consider the program's lifecycle objectives and actively manage technical debt.** ….

(https://aaf.dau.edu/aaf/software/)

© 2022 Carnegie Mellon University
[Distribution Statement A] Approved for public release and unlimited distribution.

Further resources

Kruchten, P.; Nord, R.L.; & Ozkaya, I.
Managing Technical Debt:
Reducing Friction in Software Development.
Pearson Addison-Wesley. 2019. ISBN-10: 013564593X.



© 2022 Carnegie Mellon University
[Distribution Statement A] Approved for public release and unlimited distribution.

Copyright 2022 Carnegie Mellon University.

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

DM22-0206