Future Software Requirements for S2I2 - HEP

Neil Ernst

Senior Researcher

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

© 2016 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.

Copyright 2016 Carnegie Mellon University

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

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.

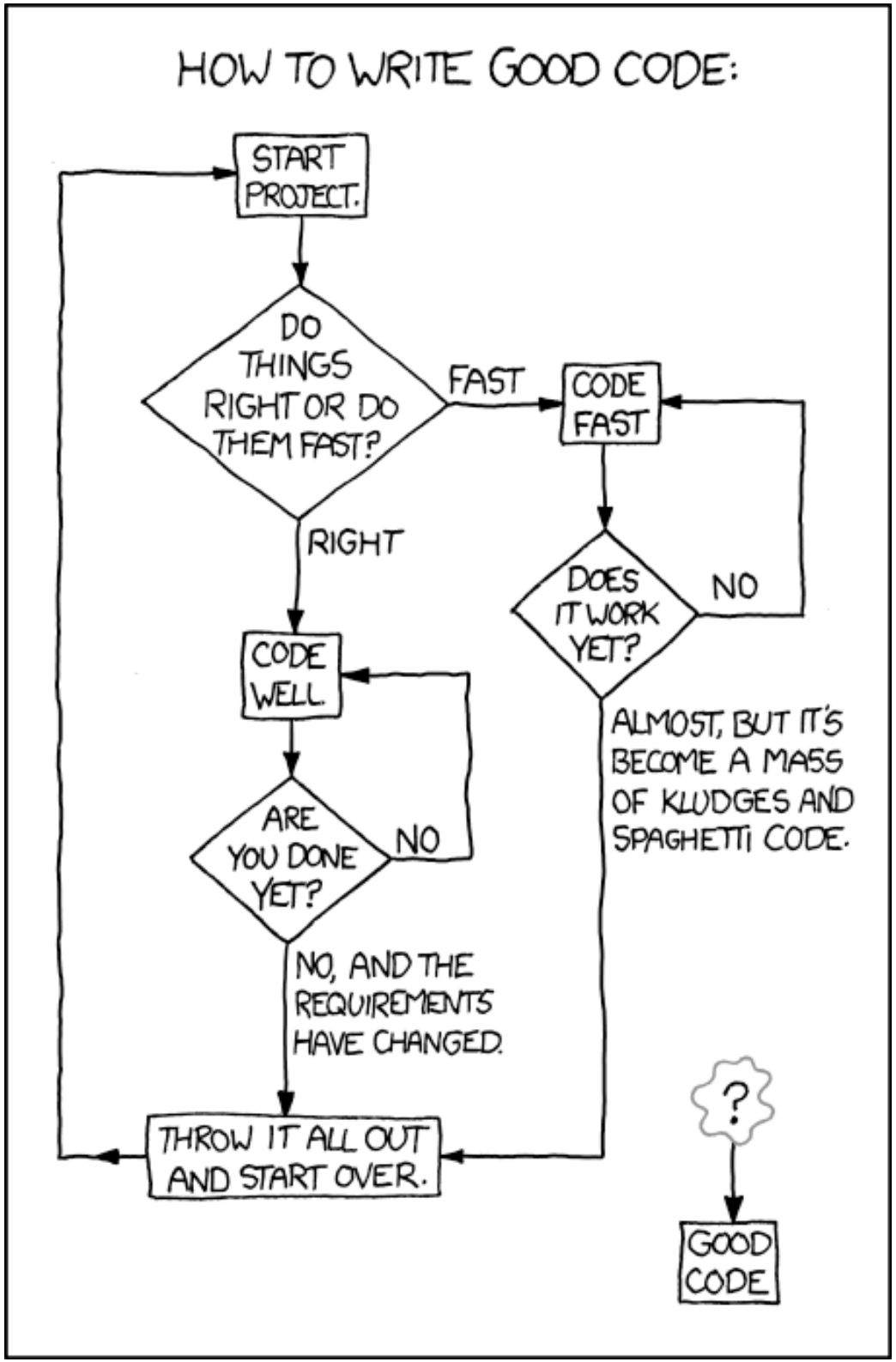
This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

Architecture Tradeoff Analysis Method® and ATAM® are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

TSP℠

DM-0004278

http://xkcd.com/844/



What is “CS”

Physics has a long and storied multi-century legacy

→ CS started formally post-WW2 (see Babbage, Turing, von Neumann as exceptions)

CMU School of CS

- Robotics
- Software Research
  - *Requirements*
  - *Architecture*
- Language Technologies
- Computational Biology
- Machine Learning
- Human Computer Interaction

CS Department

- AI
- Cybersecurity
- Graphics
- Programming Languages
- Systems (Cloud, DB, networks)
- Theory (P/NP etc)

© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.

What is the Software Engineering Institute

- A US federally funded research and development centre (FFRDC)
  - (see also JPL, RAND, Los Alamos)
- Sponsored by Department of Defense and hosted by Carnegie Mellon University in Pittsburgh
- Created in 1985 to respond to DoD software crisis
- Exemplar work:
  - Capability Maturity Model (CMM) (spun off)
  - Early work in s/w architecture (Bass, Kazman, Clements)
  - Ultra large scale systems report (Northrop 06)
  - Team Software Process (Watts Humphrey)
  - Software Product Lines
  - CERT (cybersecurity emergency response) first ‘CERT’ in US

© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.

The Requirements Evolution Problem

Requirements Engineering: acquire, analyse, model and select requirements for system design.

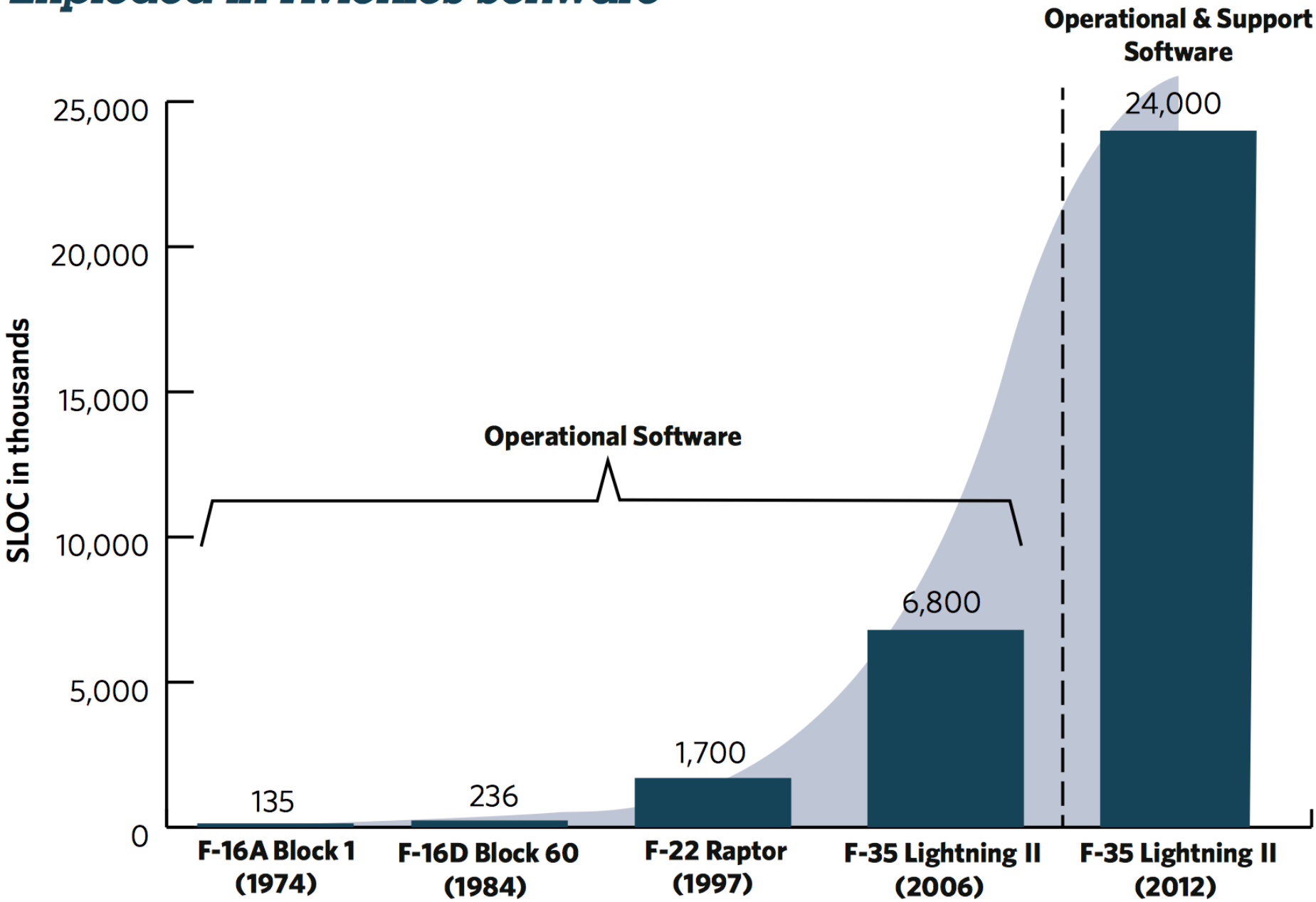
Requirements Problem: given the requirements discovered, find software components that together with domain constraints, satisfy the requirements.

Requirements Evolution Problem: plan to satisfy requirements we may not know we have → transfer Rumsfeld's "Unknown Unknowns" into "Known Unknowns".

© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.

Figure 1. The Number of Source Lines of Code (SLOC) Has Exploded in Avionics Software

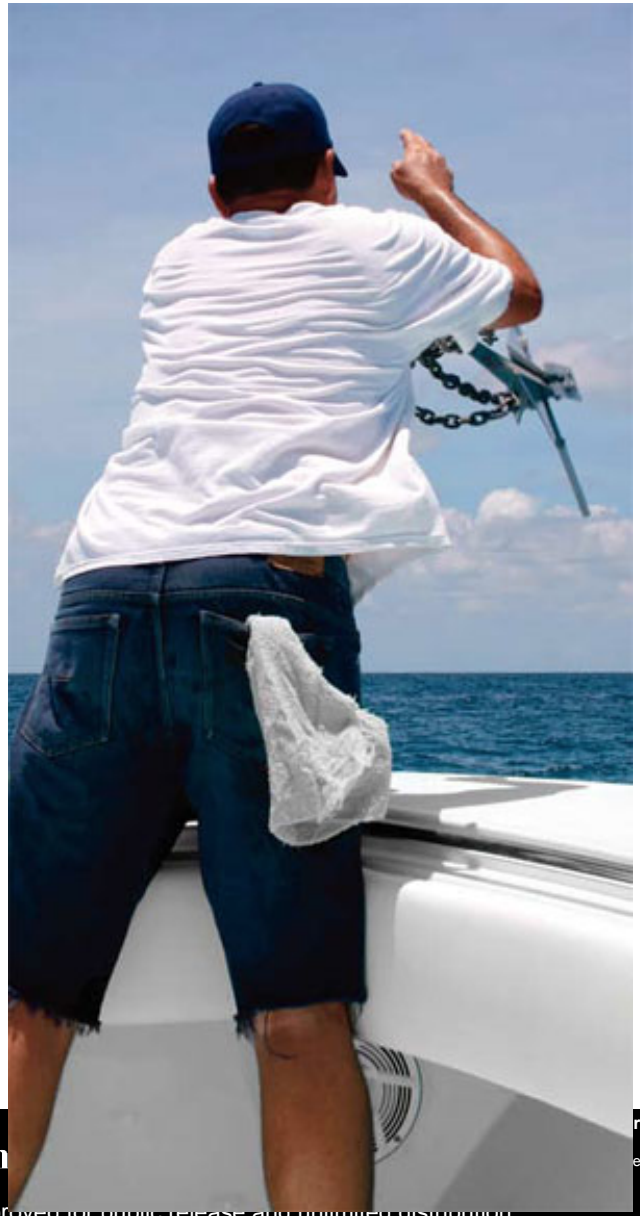


© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.

“Traditional” RE

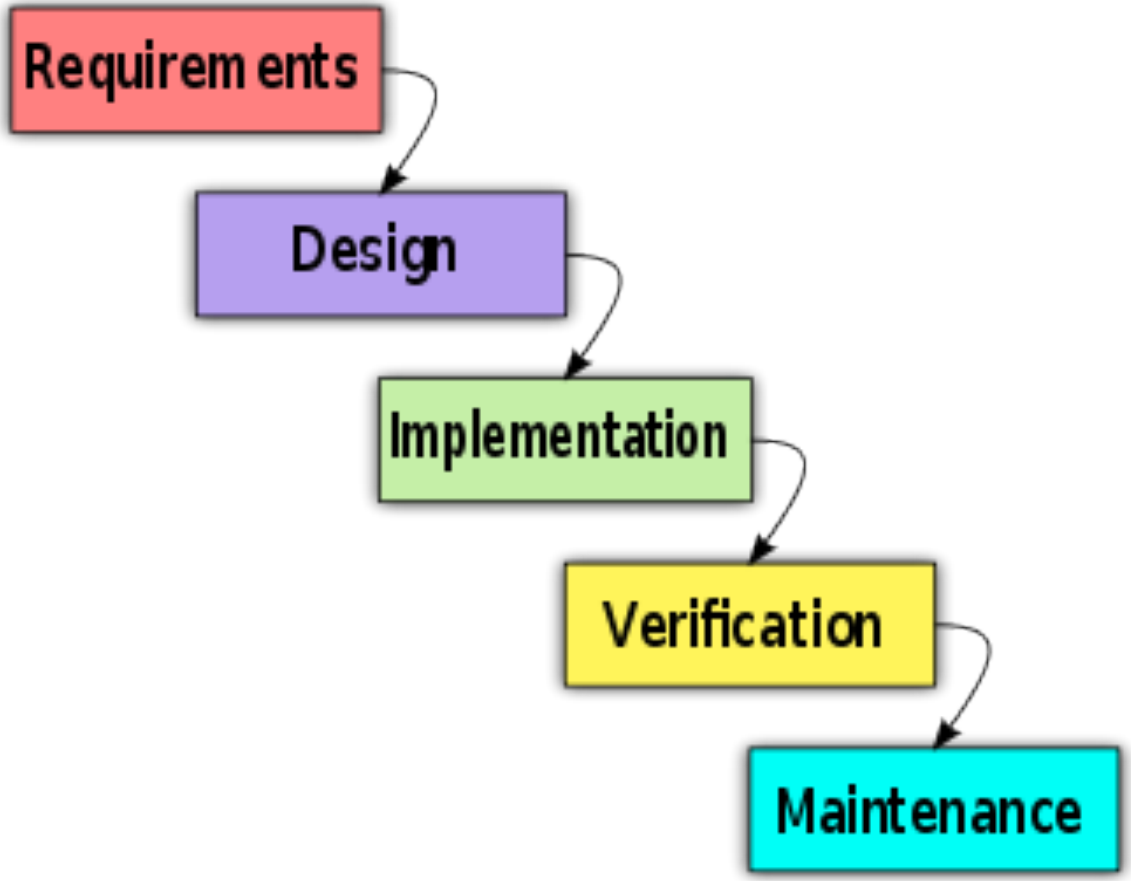
Requirements team separate and siloed, “over-the-transom-style” handoffs



[Distribution Statement A] This material has been approved for public release and unlimited distribution.

“Traditional” RE

Typically (if not ideally) done once, at inception



© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.

“Traditional” RE

Store artifacts in management tool

blueprint

Rational. software

© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.

Pejoratively called:

Big Requirements Up Front

© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.

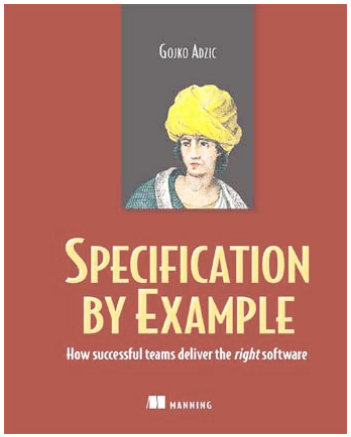
Just-In-Time RE

assume change and react, rather than plan

RE is ongoing and continuous

lightweight and iterative

developers talk to "Product Owner"



e.g. specification by example, behavior-driven development (BDD), feature driven development, user stories, acceptance testing.

Understanding change

Just-in-time is a response to this

Context: evolving and agile software development

Note that in some cases

Funders insist on documentation

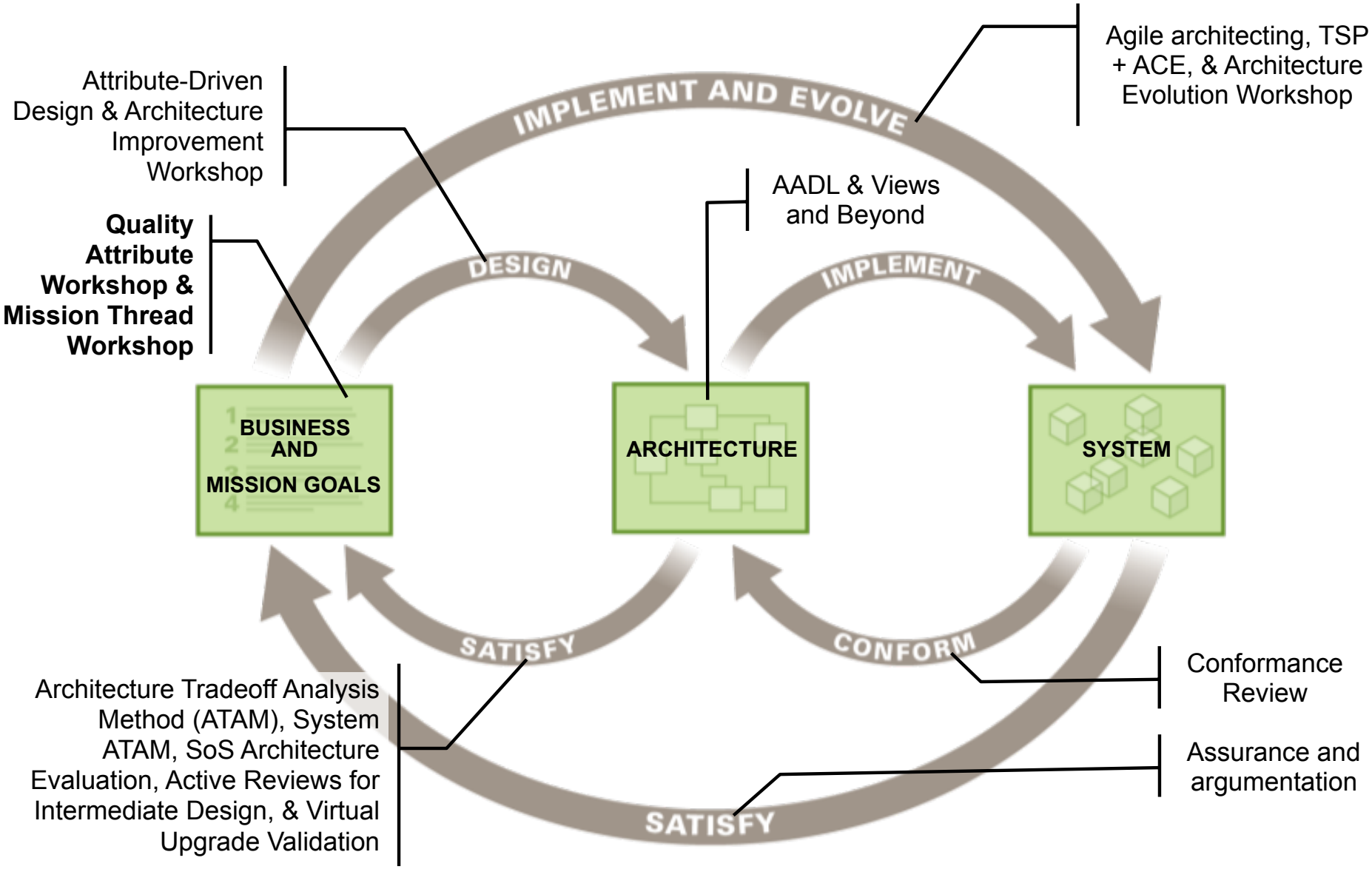
Planning ahead can be done (probabilistlically)

Safety and budget allow for expensive requirements analysis

© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.

Architecture And Requirements



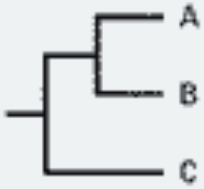
© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.

Figure 7-2 Approaches for Monitoring and Updating Strategies Based on the Level of Residual Uncertainty

1. A clear enough future

. Traditional strategic planning and decision-making processes



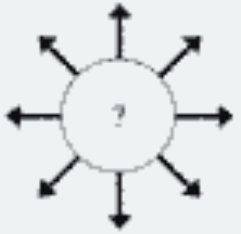
2. Alternative futures

. Contingent road maps

3. A range of futures

. Contingent road maps (if limited number of uncertainties with clear resolution paths over time)

. Six-region option portfolio management framwork



4. True ambiguity

. Strategic evolution principles
- scanning
- experimenting
- monitoring
- committing
- supporting organizational norms

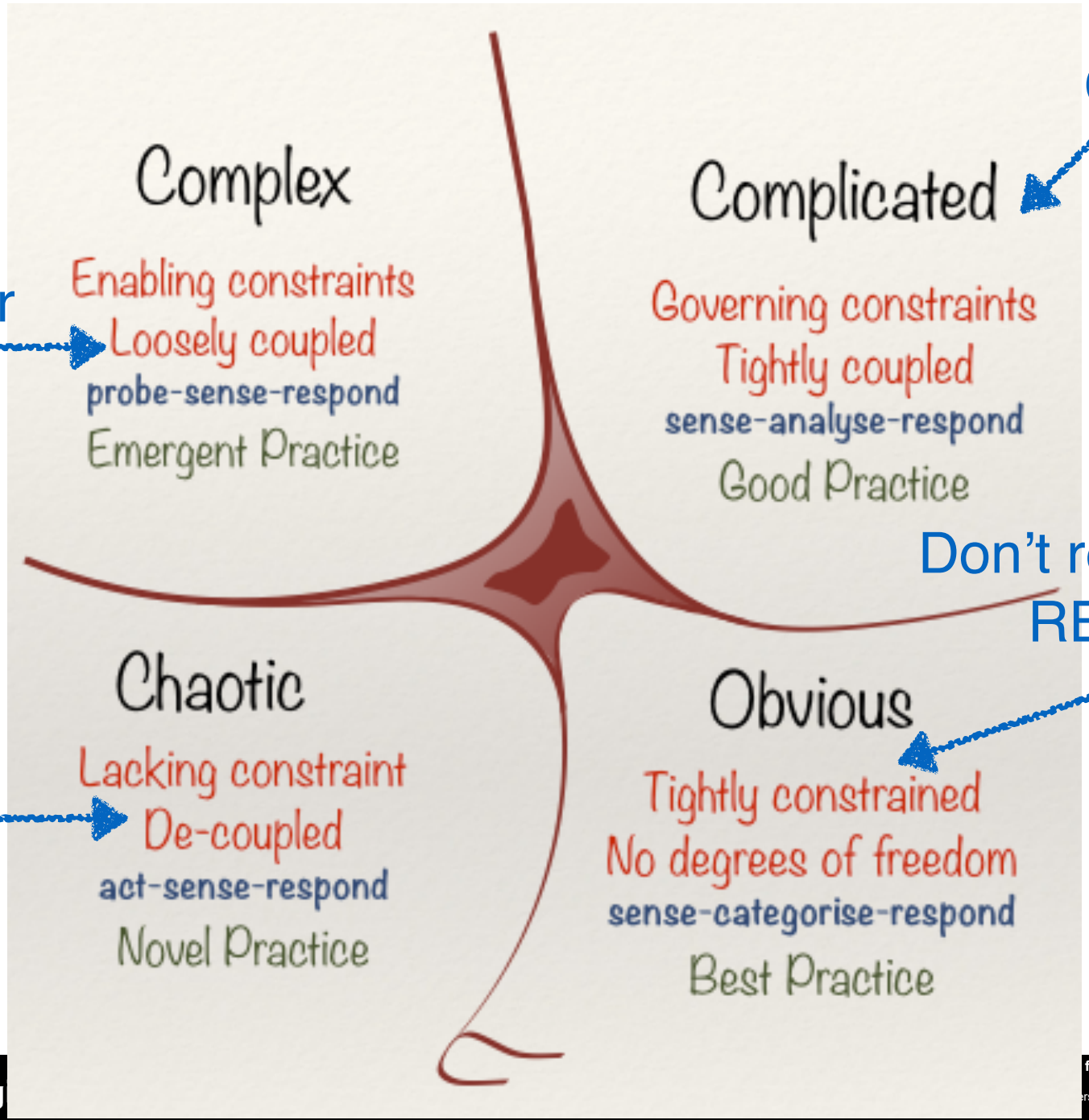
http://www.youtube.com/watch?v=GiPe1OiKQuk

© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.


HISTORY

Cynefin (Kih-Neh-Vin)



Get these right!

Cannot gather upfront

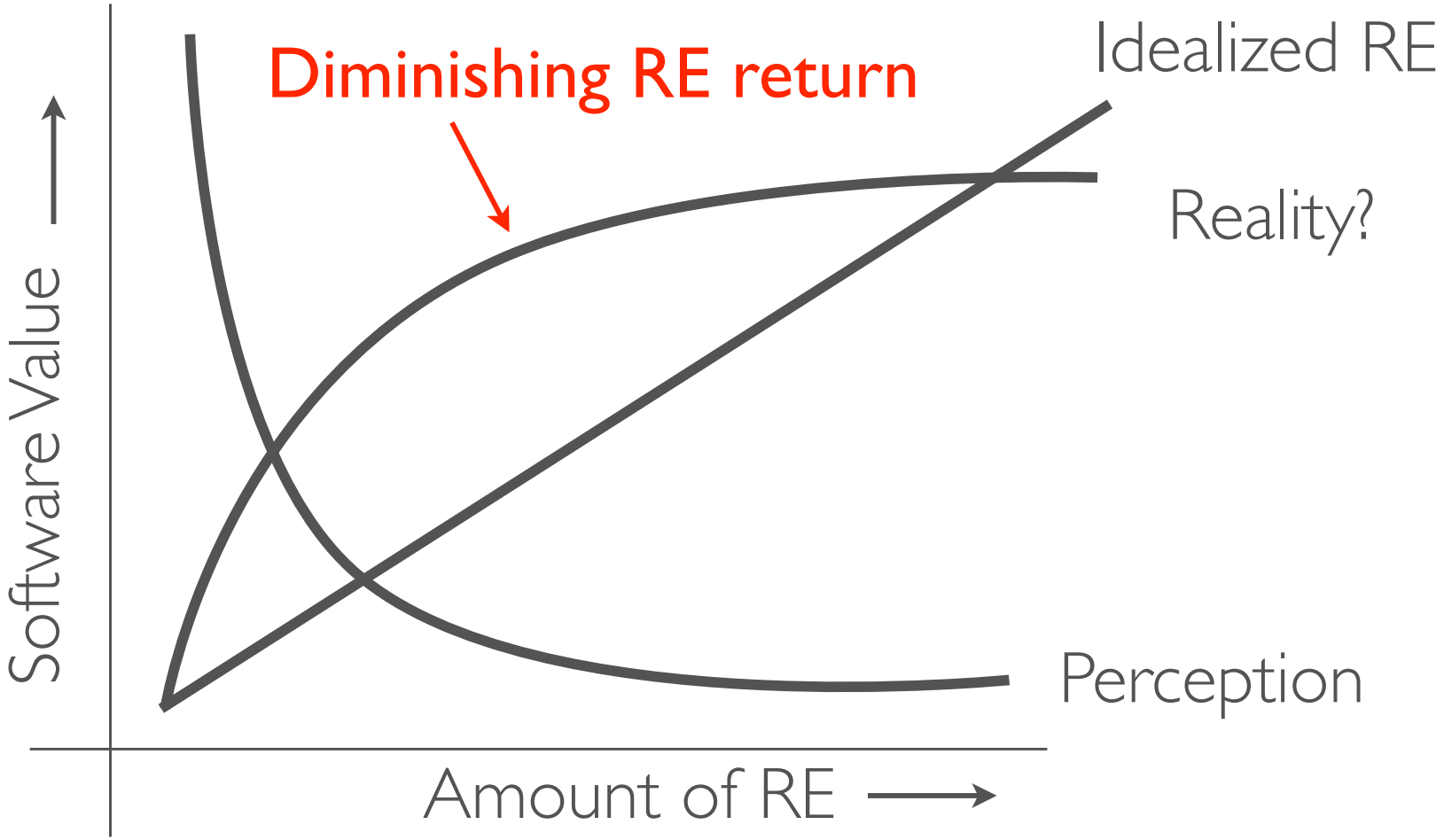
Don't really need RE here

Creativity

http://cognitive-edge.com/library/more/video/introduction-to-the-cynefin-framework/

[Distribution Statement A] This material has been approved for public release and unlimited distribution.

Ultimate Empirical Question



© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.

Software ecosystems

Software in a ‘dissolving system’ world

Increasingly about ‘ecosystem’ and not central control. Unlikely to be many BDFLs. HSF seems like a good start

Case study: Robot Operating System ecosystem

- Central ‘groundskeepers’
- Low barrier to entry
- Use it if you want
- (But not perfect)

In addition to technical requirements, think about the collaborative, social requirements:

- where to keep code
- versioning and dependency management
- how to set guidelines
- encourage diverse and non-traditional audiences

© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.

Technical Debt

"a design or implementation choice that is expedient in the short term, but sets up a technical context that can make a future change more costly or impossible."

Does HEP software have Technical Debt?

"Most of the current software, which defines our capabilities, was designed 15-20 years ago: there are many software sustainability challenges."

Decisions are being made today that you will have to live with for 20+ years.

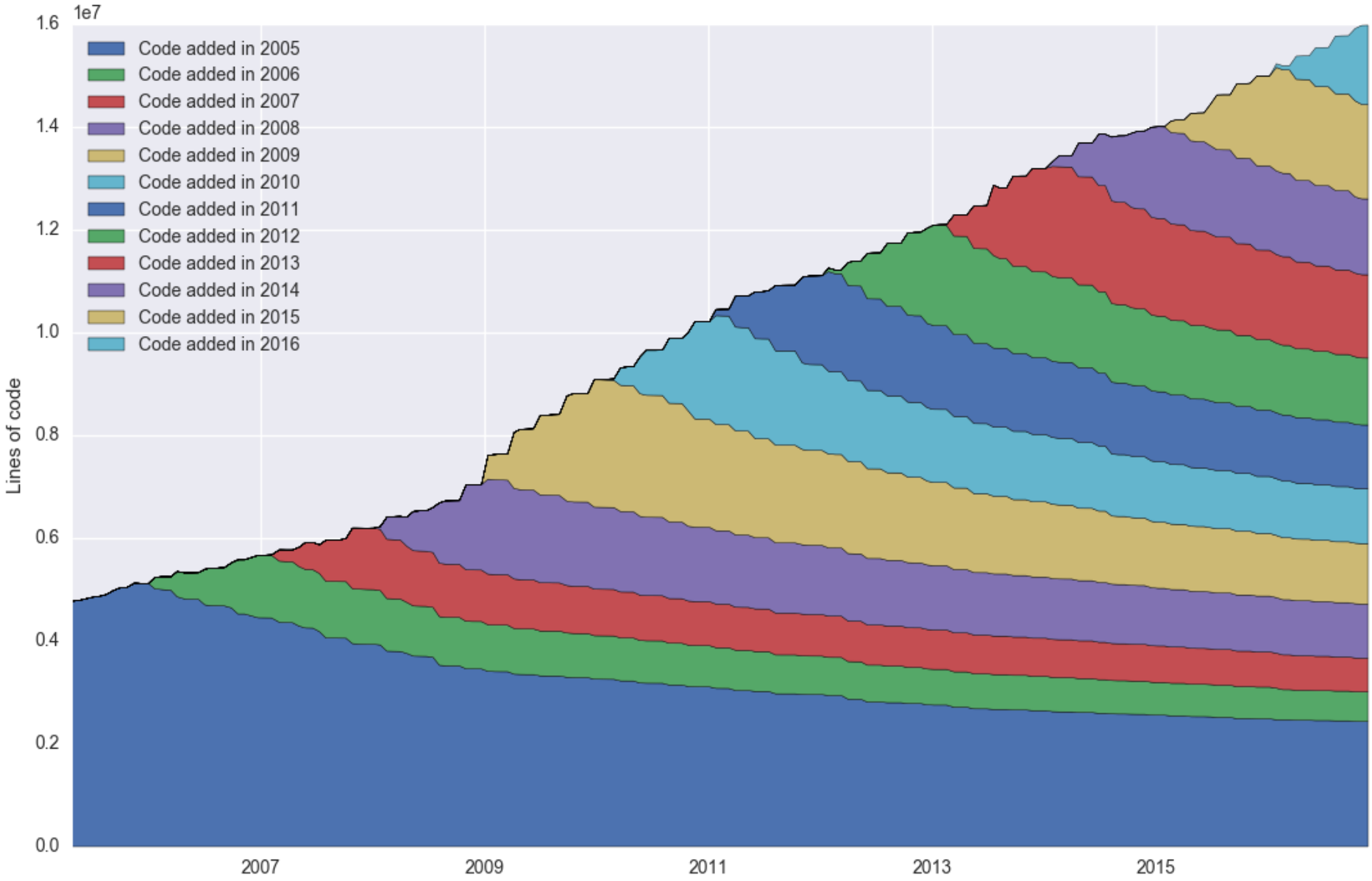
Can't always "know the unknowns", but can plan for periodic refactoring, new requirements, and labeling shortcuts as such.

© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.

source: https://erikbern.com/2016/12/05/the-half-life-of-code.html

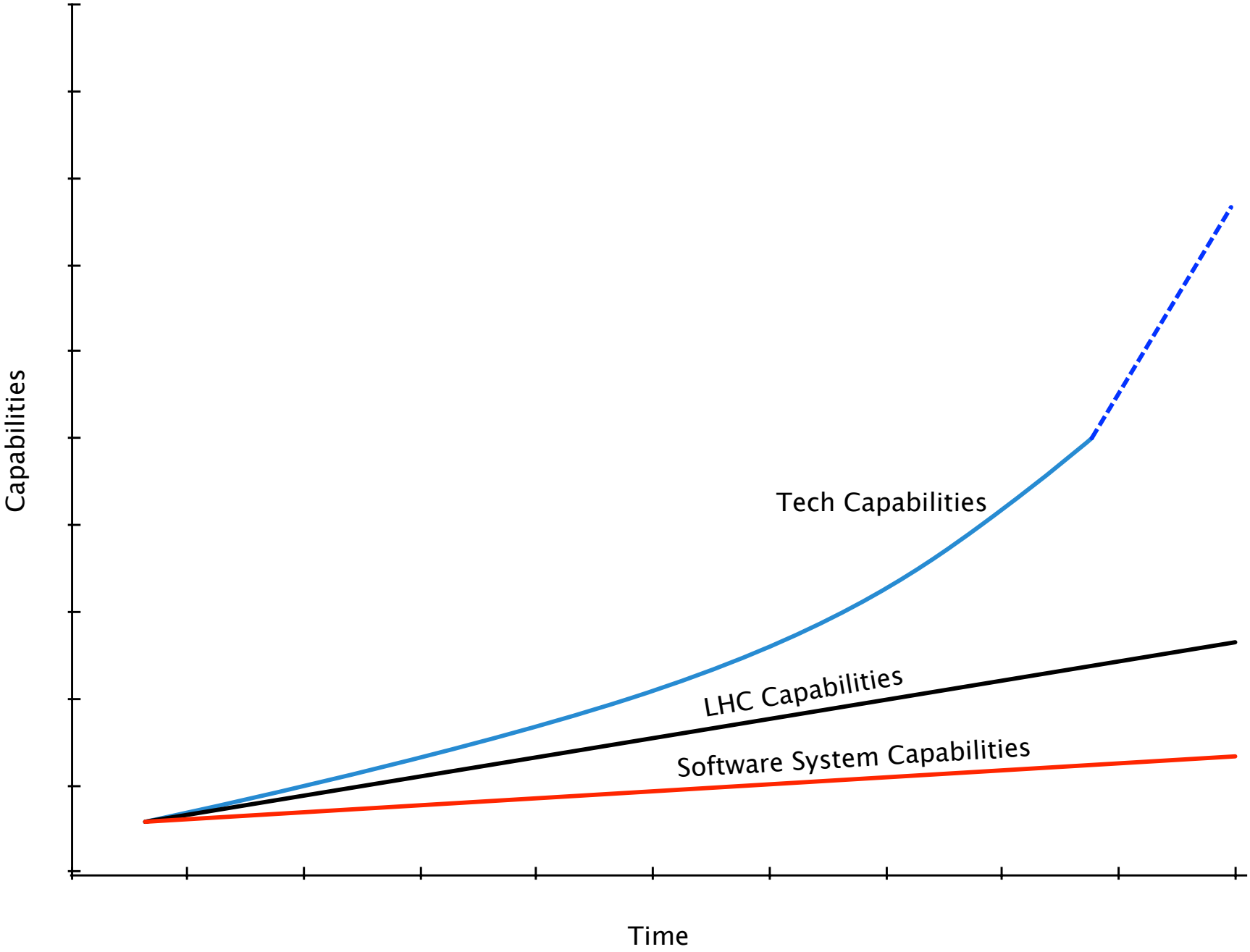
Software Will Not Go Away



Linux Kernel, additions by year

© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.



© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.

Fundamental Constraint

- When system go-live happens in 10 years, you WILL be running on outdated software and hardware
  - Tech cycle is always faster than system dev. cycle (exceptions: FB, Twitter, Amazon)
  - What percentage of software is ‘owned’ vs outsourced
- Many organizations refuse to believe this! (we are not all FB)

© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.

Future-Proofing Approaches

- Modularize for evolution.
  - *Tradeoff: integration risk.*
- Modularize for release.
  - *Tradeoff: duplication*
- Abstract for architecture and for release.
  - *Tradeoff: leaky abstractions*
- Defer decisions until Last Responsible Moment. Iterate and Prototype ‘Just-in-time’.
  - *Tradeoff: schedule impact, duplicated work*
- Evaluate architecture approach regularly with business goal scenarios.
  - *Tradeoff: cost, process buy-in.*

@neilernst – nernst@sei.cmu.edu

© 2015 Carnegie Mellon University

[Distribution Statement A] This material has been approved for public release and unlimited distribution.