Software Architecture:
The Dismal Science

Eric M. Dashofy
Principal Director, Development
Enterprise Information Services
The Aerospace Corporation

What is Software Architecture?

- A software system's architecture is the **set of principal design decisions** about a system
  - Implications:
    - Every system has an architecture (but not all architectures are equally good)
    - Some decisions are more important than others
      - Have broader or deeper effects on the properties of the resultant system
    - Stakeholders decide which decisions are "architectural"
      - What is "architectural" for one system may not be for another

Why so dismal?

- Good news: we can build more powerful software than ever!
- Bad news:
  - Trends in software engineering are eroding the abilities/ opportunities to make and enforce principal design decisions
    - Many of the principal design decisions about your systems are being made by not-you
      - By people who don't know you. Or like you.
    - Abstraction layers are leaking and affecting software design
      - Abstraction is a key method for architects to maintain intellectual control

A Rational Design Process

- Identify key stakeholders
- Agree on most important functional, non-functional requirements (-ilities)
- Choose an architectural style (set of high-level design rules) that will help you achieve those –ilities
- Identify a development and deployment platform
- Select (or develop, or enhance) architectural framework to bridge gap between style rules and platform
- Iterate through requirements refinement, design refinement, implementation, testing

Challenging Trends

- Frameworkapalooza
  - Increasing reliance on frameworks, coarse-grained software components and services
- Domain-specific megaplatforms
  - Is this even software engineering?
- Agile methods
  - Encourage deferring commitment to the 'last responsible moment'
- Leaking abstractions
  - DevOps and Microservices and Accelerators, oh my!

The Actual Design Process

- 10 Identify some stakeholders
- 20 Mock up UI in Balsamiq
- 30 Pick a hot framework that your developers will use
  - 35 So AngularJS
- 40 Pick 2-3 other hot frameworks that do the stuff your primary framework won't
  - 45 So Bootstrap and maybe jQuery
- 50 Put user stories in JIRA
- 60 Pick about 2 weeks worth of user stories off the front of the queue
- 70 SPRINT SPRINT SPRINT
- 80 GOTO 60

Frameworkapalooza

Middleware and Frameworks

- Software between your application and your underlying programming language/operating system to provide desirable services that are not provided by your PL/OS
  - Related: Platform, "Stack"
- Why middleware?
  - To make common but awkward or inelegant programming tasks easier
  - To provide selected desirable services
  - To (help) enforce architectural rules or constraints that elicit known benefits
  - Because some people really want to write one language in a different language

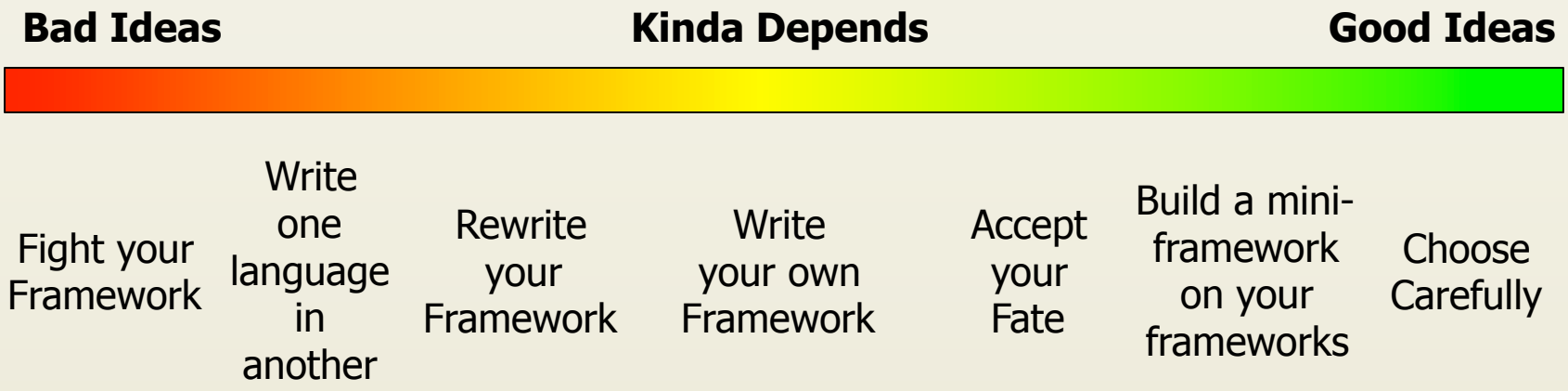
Relationship between frameworks/ middleware and architecture

- Middleware/frameworks *induce an architectural style* (Di Nitto and Rosenblum)
  - Sometimes intentionally, sometimes accidentally
- Architecture frameworks (mostly from research community) start from styles and then implement the style decisions
  - Most frameworks start from services and style decisions are a side effect
  - …but these are few and far between
- Point is: your framework designer makes a key set of principal design decisions for you without your help

Key issues

- Framework selection occurs very early in development, often before you have a chance to really understand your system's functional & design requirements
  - Once you choose a framework, changing is prohibitively expensive
- Extrinsic factors (adoption, sustainability) strongly affect framework choice
- Framework mismatch with your intended architecture or top-level quality goals
- Attempt to integrate multiple conflicting frameworks
- Attempt to integrate components and services built for a different framework (or none at all)

So what can you do?



Domain-Specific Megaplatforms

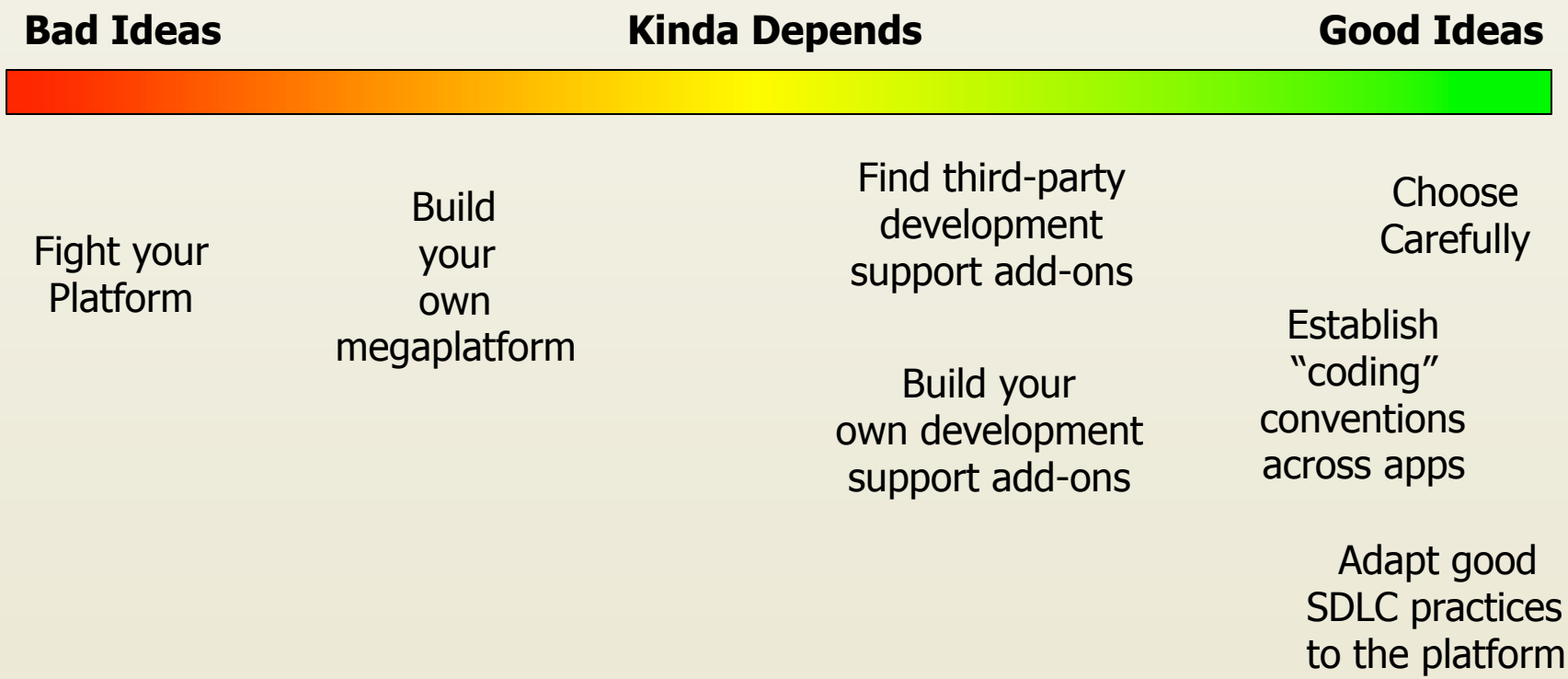
Megaplatforms and Software Architecture

- Software built on megaplatforms
  - Has the same lifecycle needs as traditional software (requirements, design, implementation, testing, maintenance)
  - Is built around first-class domain objects
    - Example, for business apps: forms, tables, reports, workflows, external data integrations
  - Is often implemented by configuration and code
  - Can (sometimes) be done significantly faster than "on the metal" coding (even with frameworks)

Key issues

- All the key issues you have with frameworks, but worse
  - Licensing, lock-in issues more prevalent
- Big steps backward in support for SDLC processes
  - Configuration management, deployment, testing, integrated development environments…
  - Developers in these environments often have no/little SE background
- Integration with software outside the megaplatform environment
- Cloud vs. on-premises tradeoffs
  - Security, performance, accessibility of internal network resources…

So what can you do?



Agile Methods

Agile Development and Software Architecture

- Common threads in agile development
  - Dynamic backlog of features to implement
    - Short development cycles with demonstrable delivered value/functionality at end of each cycle
  - Deferred decision making "until last responsible moment" (point where cost of not making decision exceeds the cost of making it)
    - Local vs. global decision making
    - YAGNI principle
    - Designs as emergent rather than constructed
  - Continuous refactoring

Key Issues

- Easy for top-level designs to get lost (or top-level decisions not made at all)
- Focus on local decision making can lead to architectures that are agglomerations instead of cohesive wholes
  - Possible missed opportunities for abstraction if you're not careful
  - High-level qualities/-ilities might get lost or difficult to imbue into the product
- Skimping on any part of agile tends to make other parts dangerous

So what can you do?

YAGNI without refactoring

Adapt agile processes to incorporate traditional design steps

Add up-front design to agile processes

Continuous refactoring to maintain conceptual integrity

End-to-end integration testing

Leaking Abstractions

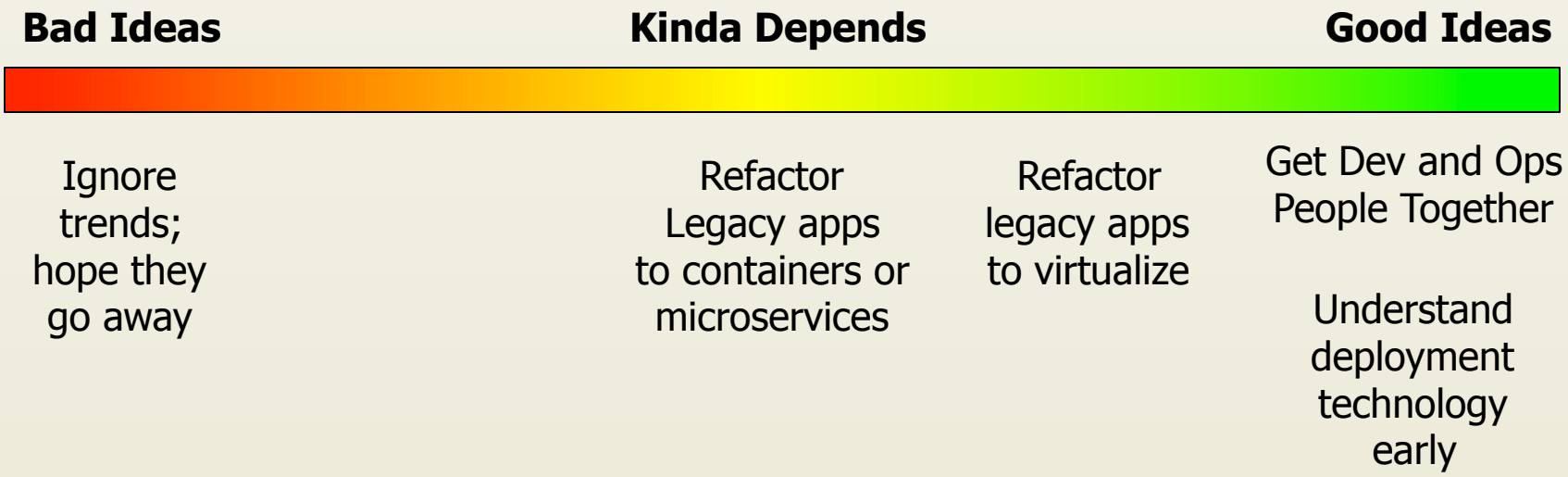
Leaking Abstractions

- Physical architecture influence over logical architecture is increasing
  - Virtualization → DevOps → Containers → Microservices
    - DevOps technologies are influencing
  - Specialized hardware (e.g., GPUs, other accelerators) require tighter connection between software and hardware

Key Issues

- Some –ilities can be addressed at new layers
  - E.g., reliability, performance through high-availability and load balancing at the container level
- Virtualization or containerization of legacy applications
  - Implications not always easy to understand
- Usual issues with emerging technology issues
  - These will likely settle out over time
- Conflict between virtualization and accelerator technologies

So what can you do?



Takeaways

- Architecture remains important, but top-down architecture may diminish
  - Architecture "in the large" → "in the small"
  - Architecture prescriptions → emergent architecture
  - Maintain architectural quality through
    - Conceptual integrity
    - Continuous refactoring
    - Applying best practices even when the support is lacking

Disclaimers

- The trademarks, service marks and trade names contained herein are the property of their respective owners.