Software Patterns

L12: Collaboration and Management Patterns

Prof. W. Maalej – @maalejw

Overview

1 **Introduction**

2 Work Allocation Patterns

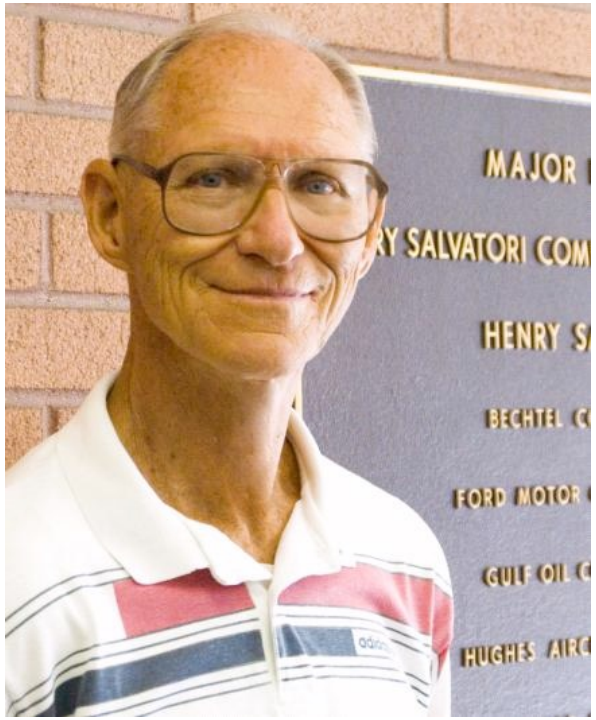
3 Collaboration Patterns

4 Management Anti-Patterns

Software projects are getting larger

"The more members that are added to a team, the more time is consumed in ***communication*** *with other team members. This decrease in programmer productivity is called a* ***diseconomy of scale*** *in economic terms."*

[Böhm, 1981]



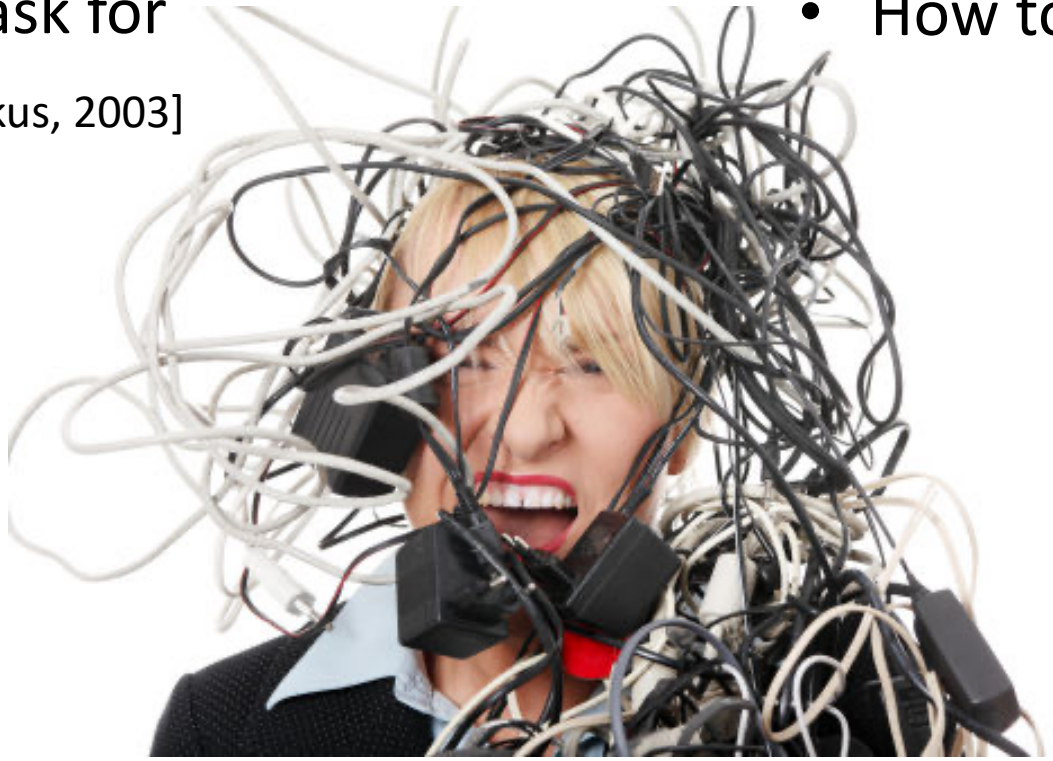
Communication challenges in software projects

- Communication overhead
- What to ask for

[Herbsleb & Mockus, 2003]

- Whom to contact
- How to communicate

[Maalej et al 2015]



Miscommunication is a major reason for budget overruns, quality problems, project delays!

[Hoch, 2000]

Software projects are getting global

Software development projects are often distributed across multiple sites and countries

Distribution increases communication overhead

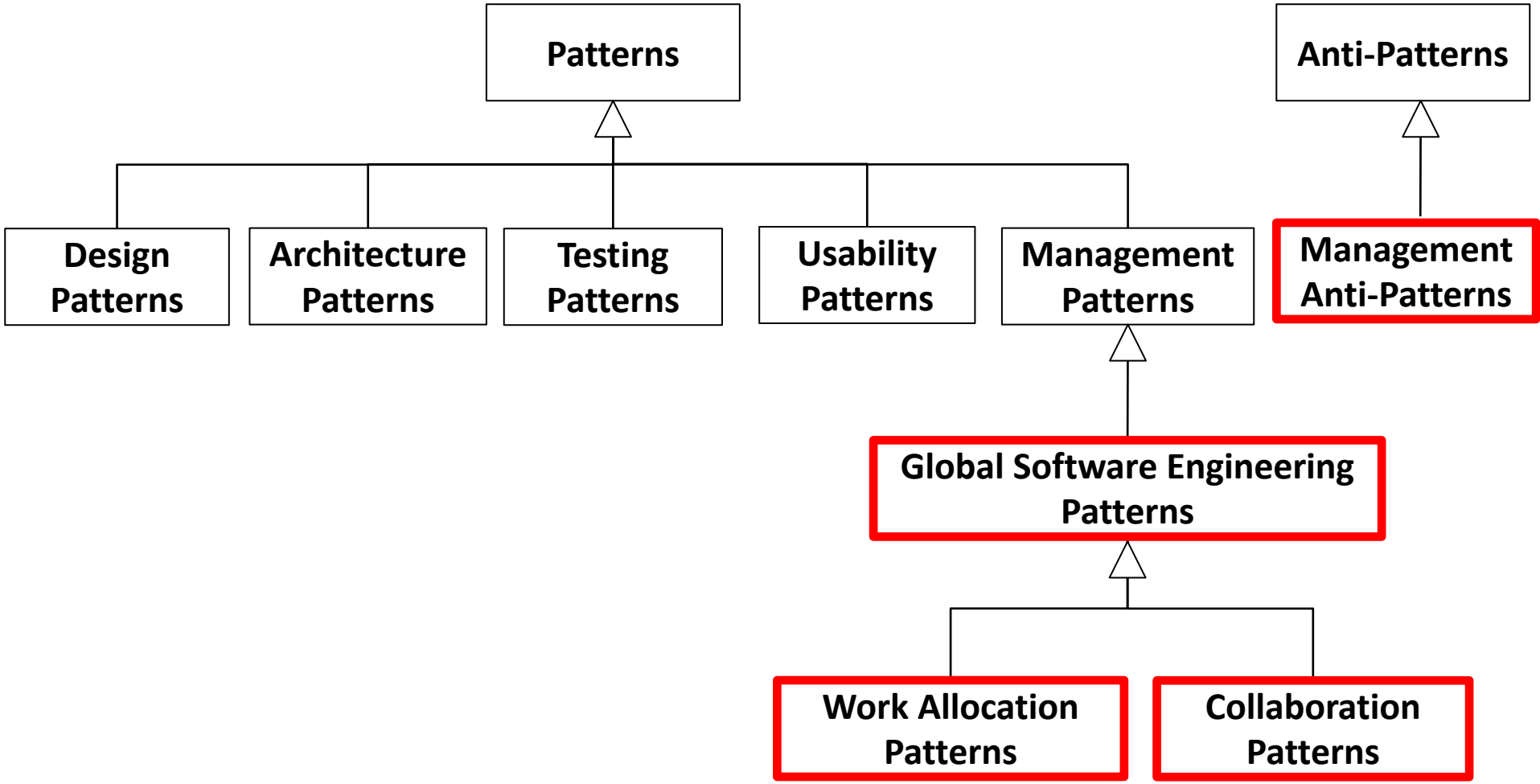
Collaboration challenges of global software projects

Physical distance

Time distance

Cultural distance

Management patterns in distributed software engineering projects



Overview

1. Introduction
2. **Work Allocation Patterns**
3. Collaboration Patterns
4. Management Anti-Patterns

Organizational models in global software engineering

James D. Herbsleb

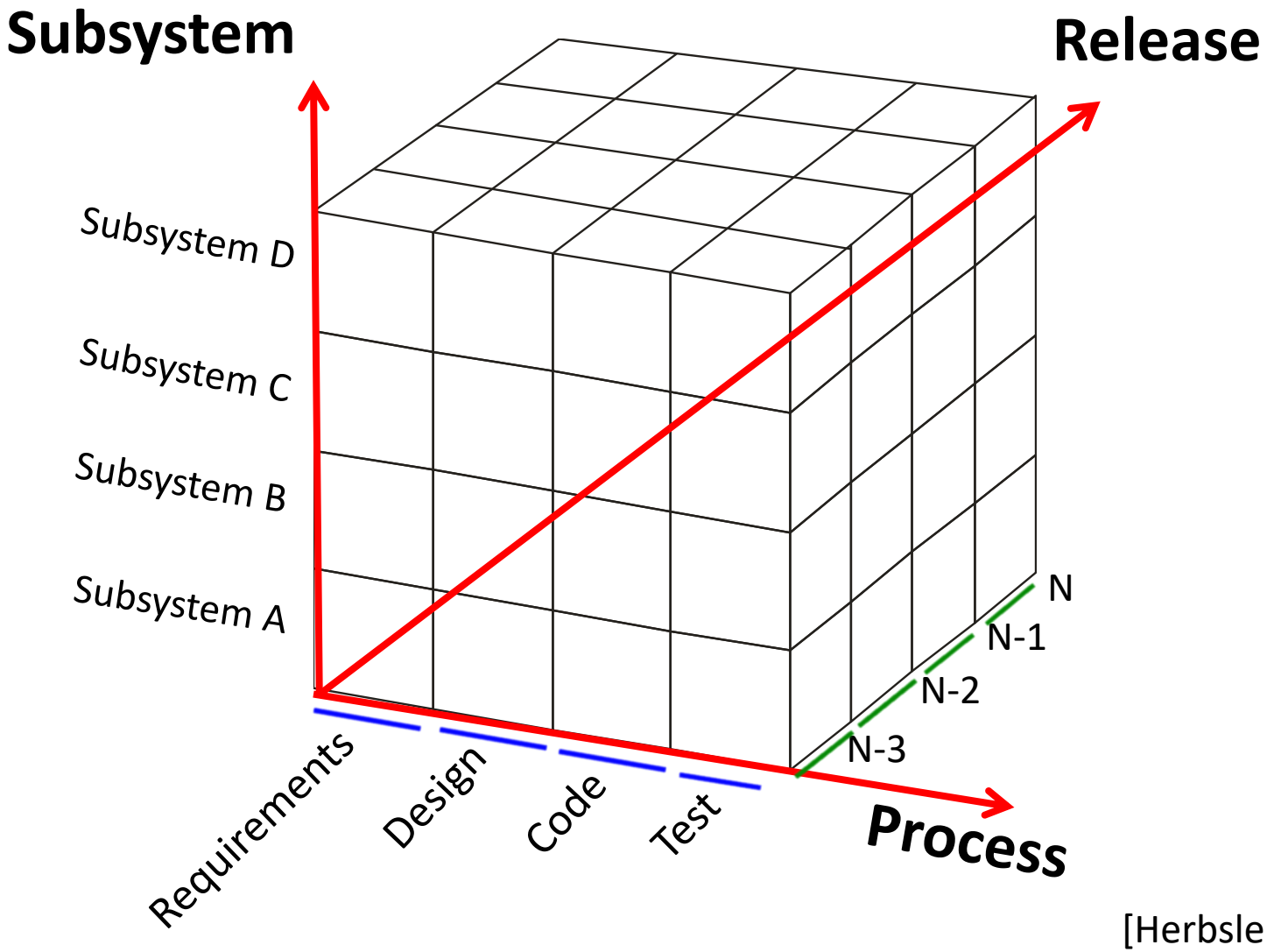
Professor, Institute for Software Research Carnegie Mellon University

Renowned expert in research

Most Influential Paper Award ICSE 2013

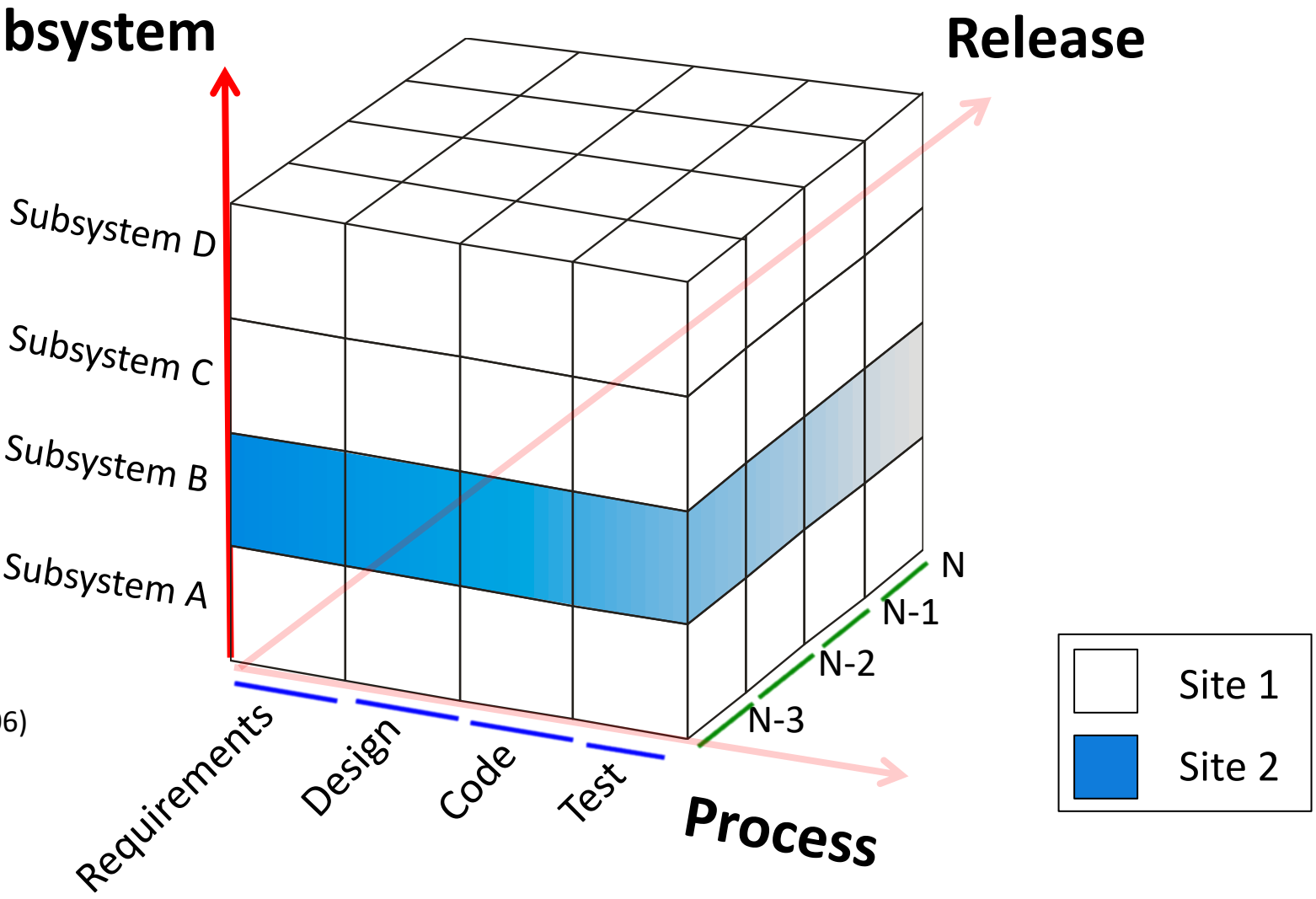
- Patterns in Organizational Models, [Herbsleb, 2006]
- Analyzed organizational structures and their evolution
- More than 150 interviews at 14 sites on 3 continents

Dimensions of organizational models



[Herbsleb, 2006]

Separate by subsystems



(Herbsleb, 2006)

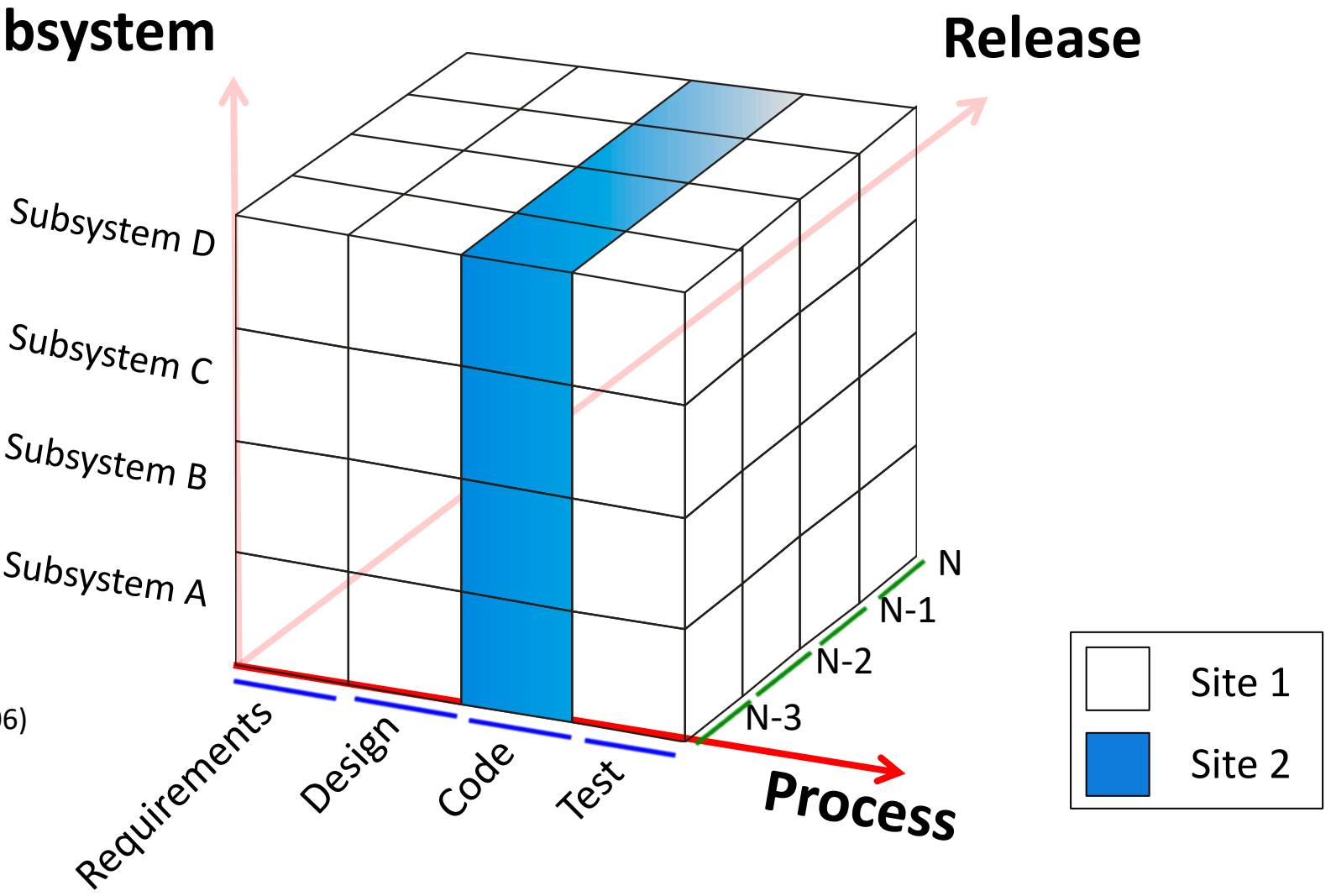
Separate by subsystems

☑ Modular product, clear, and stable interfaces

- **Consistent modification** of subsystems with low impact on other subsystems (low **coupling**, high **cohesion**)
- Sites can use different development processes, **synchronization milestones sufficient**

- **Integration** may become problematic
- **Central components**, cross-cutting features and non-functional requirements difficult to implement

Separate by process steps



(Herbsleb, 2006)

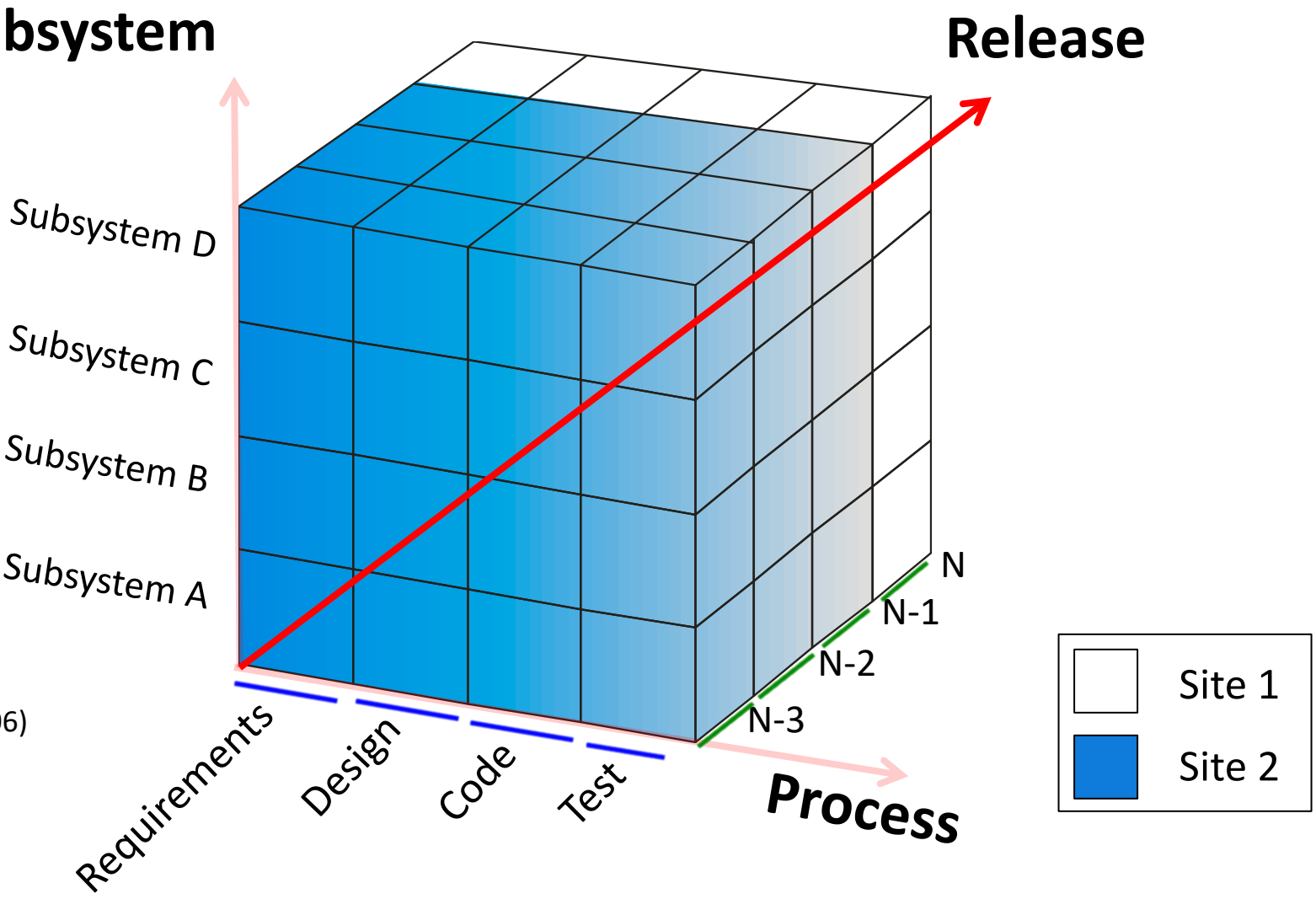
Separate by process

- ☑ Limited availability of technical resources (e.g. test equipment)
- ☑ Specific expertise at a given site (requirements, design, code, test)

- Leverage time zones to speed up test and fix cycle

- High cross-site communication, e.g. when fixing defects
- Inflexible to changing plans (availability of resources)

Separate by releases



(Herbsleb, 2006)

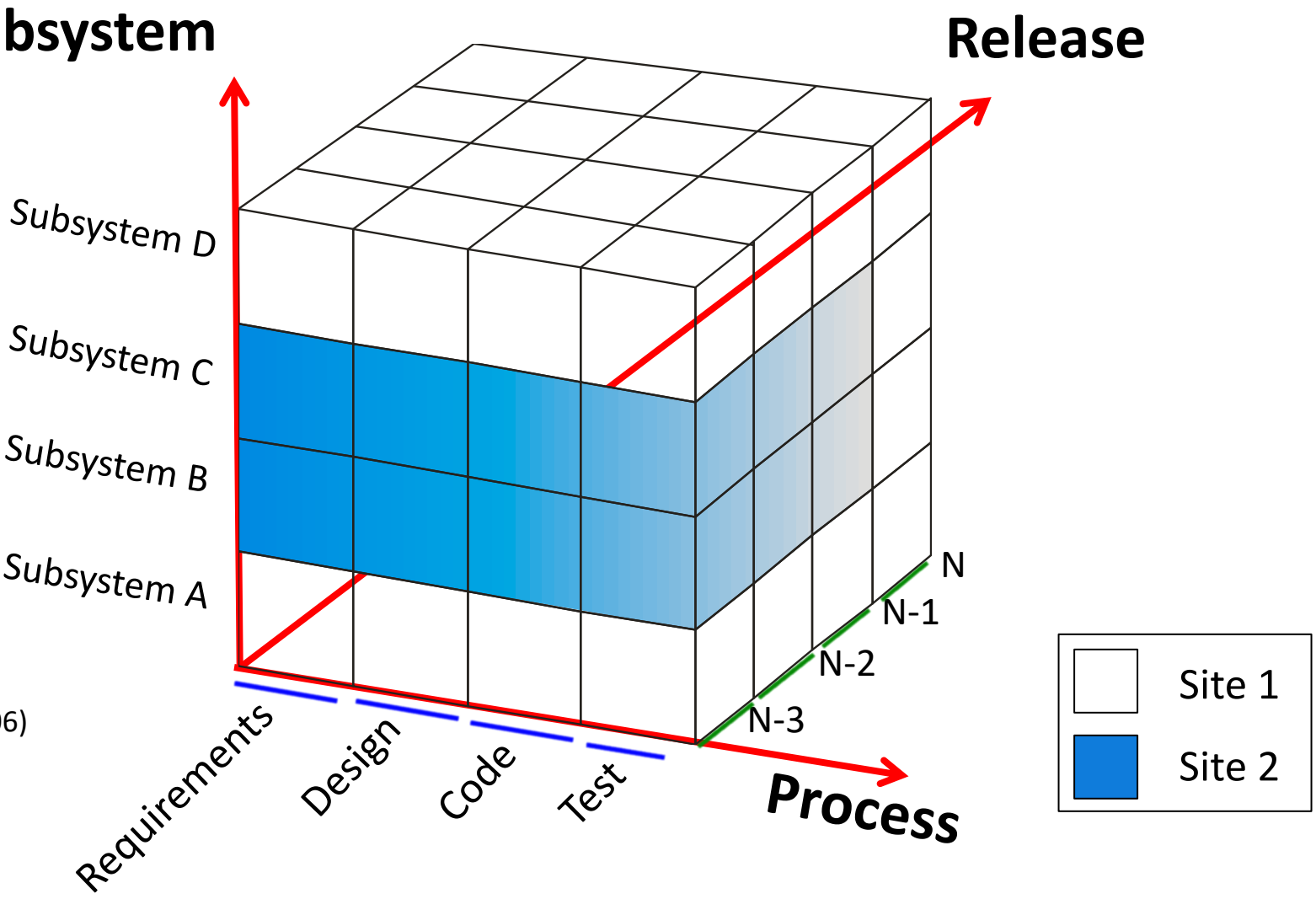
Separate by releases

- ☑ Previous releases will remain stable (no new functionality, few bug fixes)
- ☑ Current release is much more critical to business than old releases
- ☑ Want to expose new site to product

- Current release still developed within one site
- N-1 site can learn about whole product

- Cross-site communication between maintainers and original developers
- Maintainers need to learn about the whole product at once
- High effort for synchronizing bug fixes
- Maintainers likely to feel disrespected

Hybrid gradual subsystem split



(Herbsleb, 2006)

Separate by releases

- ☑ Interim step to take over responsibility for part of product
- ☑ New site is inexperienced with product
- ☑ Long term goal: subsystem responsibility

Release / Subsystem Hybrid:

- Move part of product, but not for current release
- In future, move subsystem responsibility for all releases

??

??

Conway's law

How Do Committees Invent?

Melvin E. Conway

Conclusion

The basic thesis of this article is that organizations which design systems (in the broad sense used here) are constrained to produce designs which are copies of the communication structures of these organizations. We have seen that this fact has important implications for the management of system design. Primarily, we have found a criterion for the structuring of design organizations: a design effort should be organized according to the need for communication.

Overview

1 Introduction

2 Work Allocation Patterns

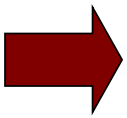
3 **Collaboration Patterns**

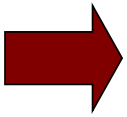
4 Management AntiPatterns

How to build one global team?

- Team not familiar with technologies necessary for the project
- No experience in working together
- Common understanding of the system to be developed needs to be created

- Need to establish better communication relationships
- Important domain or project knowledge only available at certain sites
- Issues in collaboration need to be identified and resolved





Kick-off and Teambuilding

TAILORED TRAINING

CO-LOCATED ANALYSIS PHASE

ONSITE MANAGEMENT VISITS

CROSS-SITE DELEGATION

UNFILTERED LINE MANAGER COMMUNICATION

Project Communication

[Lescher 2006]

Distributed team challenges

- **Requirements:** how to keep the scope and priorities known and understood
- **Architecture:** how to jointly develop and communicate the architecture
- **News**: how to keep the whole team informed
- **Access to expertise:** how to ensure access to technical or domain expertise
- **Team dynamics:** how to build trust and make each individual feel like an equal value contributor

Source: Lise Hvatum, EuroPLoP and Cutter IT Journal [8,9,10]

Collaboration patterns (handouts!)

- Tailored Training
- Co-located Analysis Phase
- Distributed Pair Programming
- Onsite Management Visits
- Cross-Site Delegation
- Unfiltered Communication
- Selected
- Prepared
- Culture Awareness
- Team Mentor
- Early Bonding
- Short Engagements
- Team Space
- Together
- Iteration Connect
- Completion United
- Smart Meetings
- Team Connector
- One Project
- Communication Strategy
- Common Information Infrastructure
- Living Process
- Common Development Environment
- Flexibility
- Full Credit

Overview

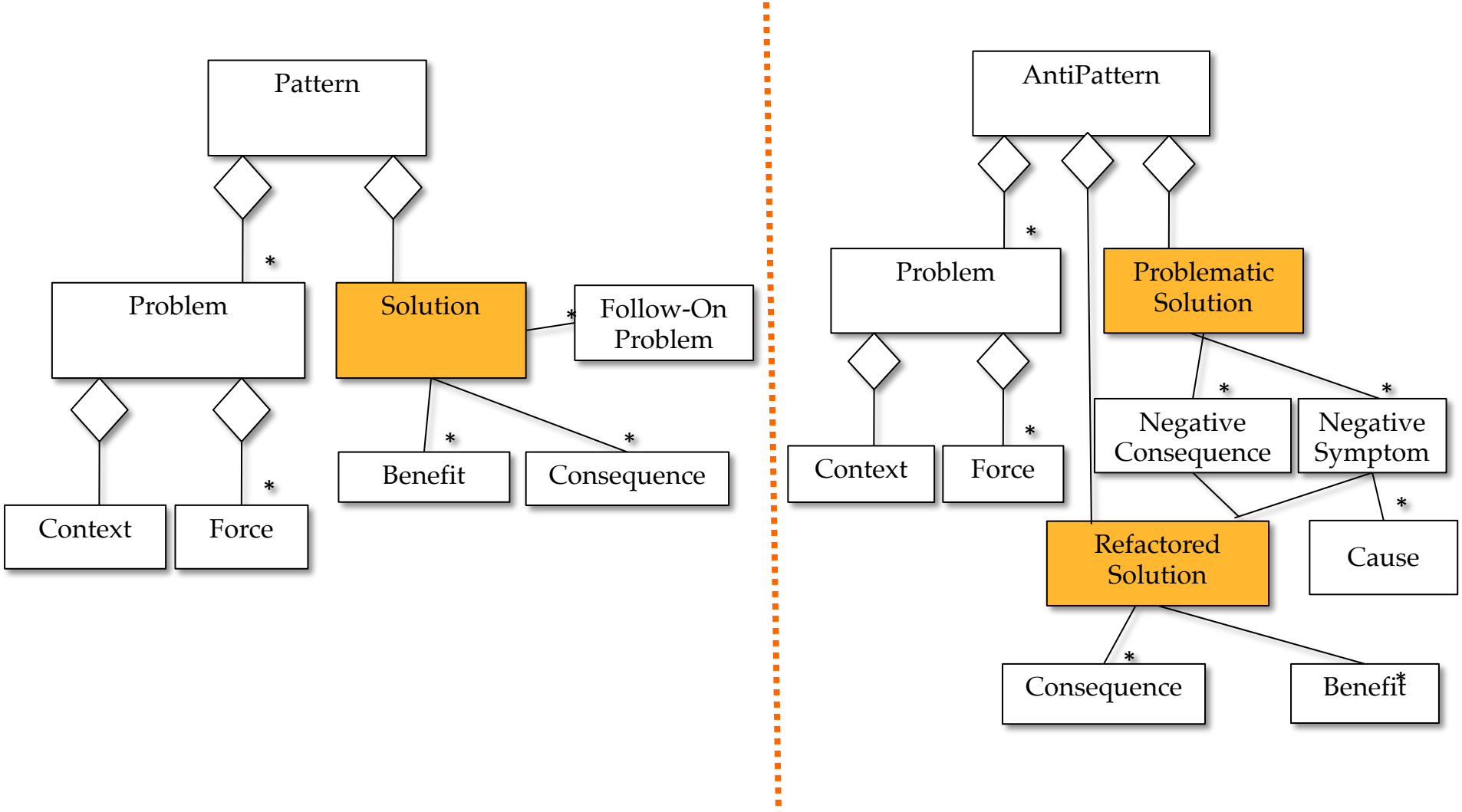
1 Introduction

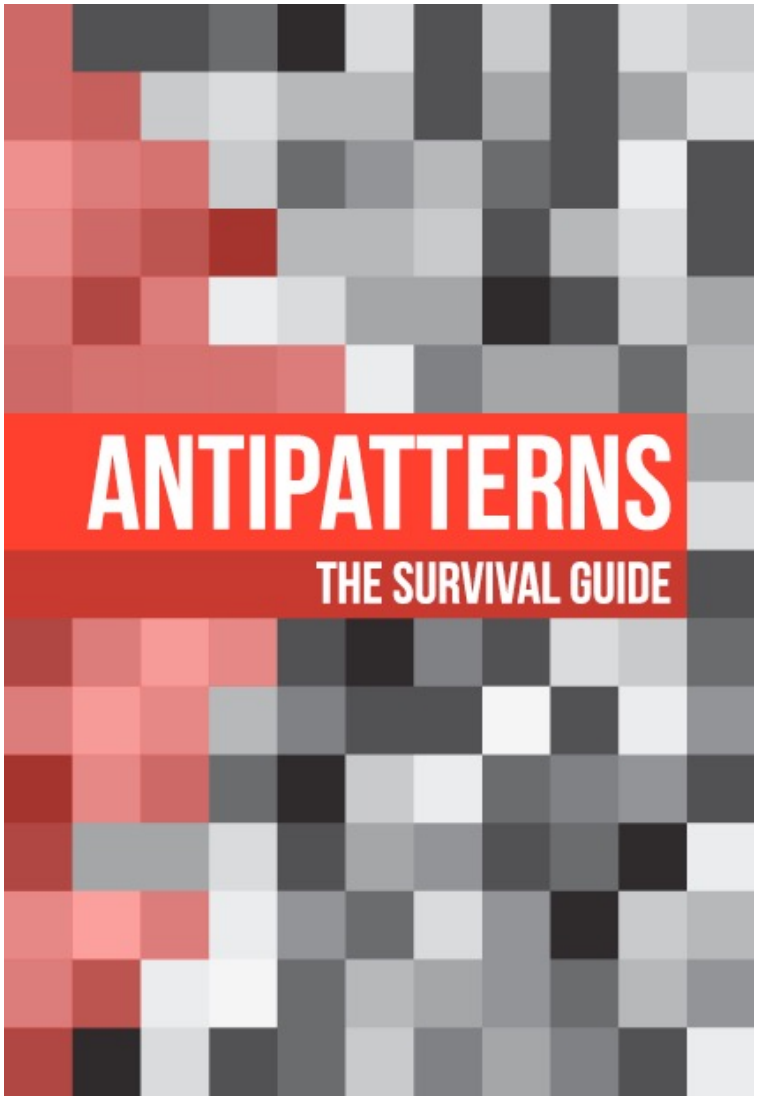
2 Work Allocation Patterns

3 Collaboration Patterns

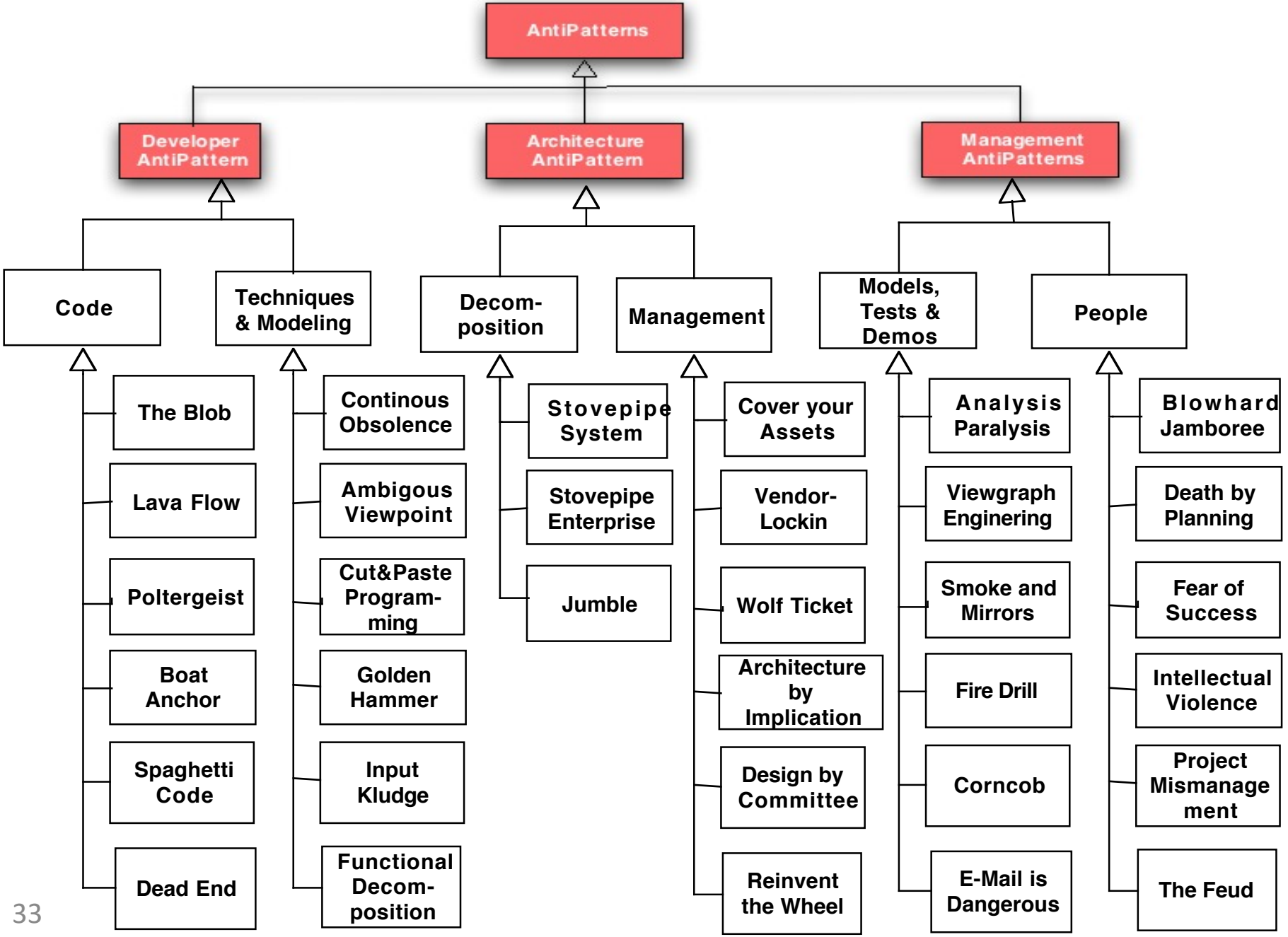
4 **Management Anti-Patterns**

Pattern vs Anti Pattern





http://sourcemaking.com/antipatterns/


AntiPatterns
Developer AntiPattern
Architecture AntiPattern
Management AntiPatterns
Code
Techniques & Modeling
Decom-position
Management
Models, Tests & Demos
People
The Blob
Lava Flow
Poltergeist
Boat Anchor
Spaghetti Code
Dead End
Continous Obsolence
Ambigous Viewpoint
Cut&Paste Program-ming
Golden Hammer
Input Kludge
Functional Decom-position
Stovepipe System
Stovepipe Enterprise
Jumble
Cover your Assets
Vendor-Lockin
Wolf Ticket
Architecture by Implication
Design by Committee
Reinvent the Wheel
Analysis Paralysis
Viewgraph Enginering
Smoke and Mirrors
Fire Drill
Corncob
E-Mail is Dangerous
Blowhard Jamboree
Death by Planning
Fear of Success
Intellectual Violence
Project Mismanage ment
The Feud

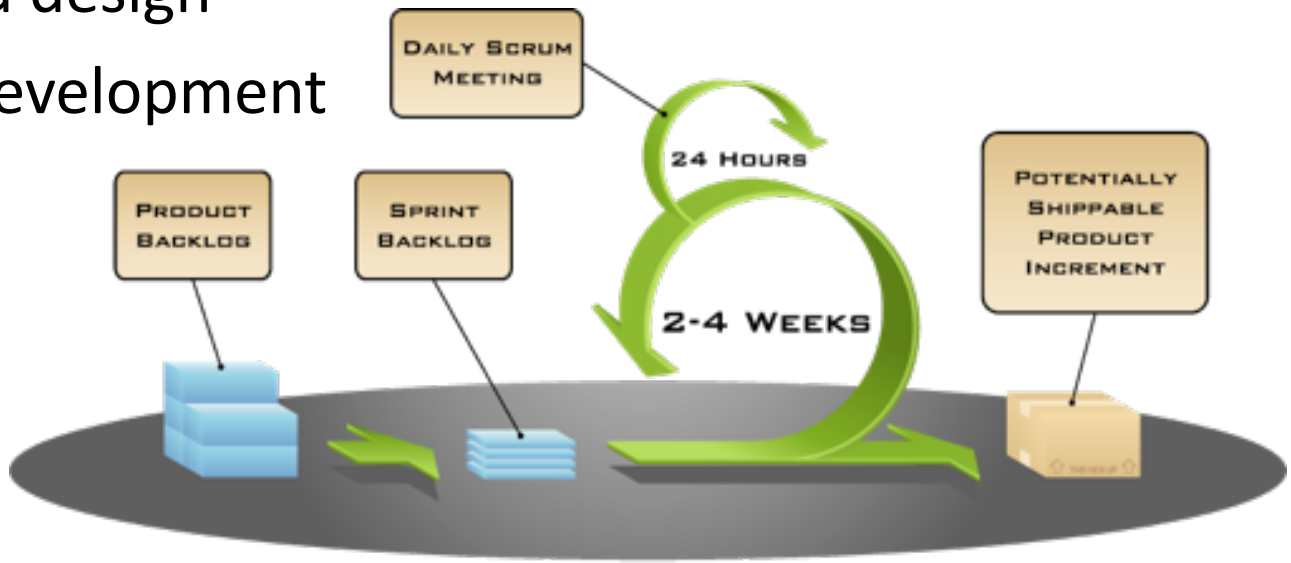
Analysis Paralysis

Analysis paralysis

- **Context**
  - Goal: achieve perfection and completeness of the analysis phase
  - Generation of very detailed models
  - Waterfall assumptions
    - Everything about the problem is known a priori
    - Detailed analysis can be successfully completed prior to coding
    - Analysis model will not be extended nor revisited during development
- **Symptoms and Consequences**
  - Cost of analysis exceeds expectation without a predictable end point
  - Analysis documents no longer make sense to the domain experts
- **Typical Causes**
  - Management assumes a waterfall progression of phases
  - Goals in the analysis phase are not well defined

Refactored solution

- Iterative Development
- Incremental development assumes that details of the problem and its solution will be learned in the course of the development process
- Vertical prototyping
- Scenario based design
- Sprint based development



Email is dangerous

Email is dangerous

- **Context**
  - E-mail is an important communication medium
  - But inappropriate for many topics and sensitive communications:
    - Confrontational discussions
    - Context sensitive discussion
    - Complex topics
- **Symptoms and Consequences**
  - Productivity and morale of project members degenerate
  - Your email inbox explodes, you receive in-actionable information
  - You are explicitly waiting for emails
- **Typical Causes**
  - Lack of clear communication guidelines and understanding of medium
  - Low professional writing skills

Refactored solution

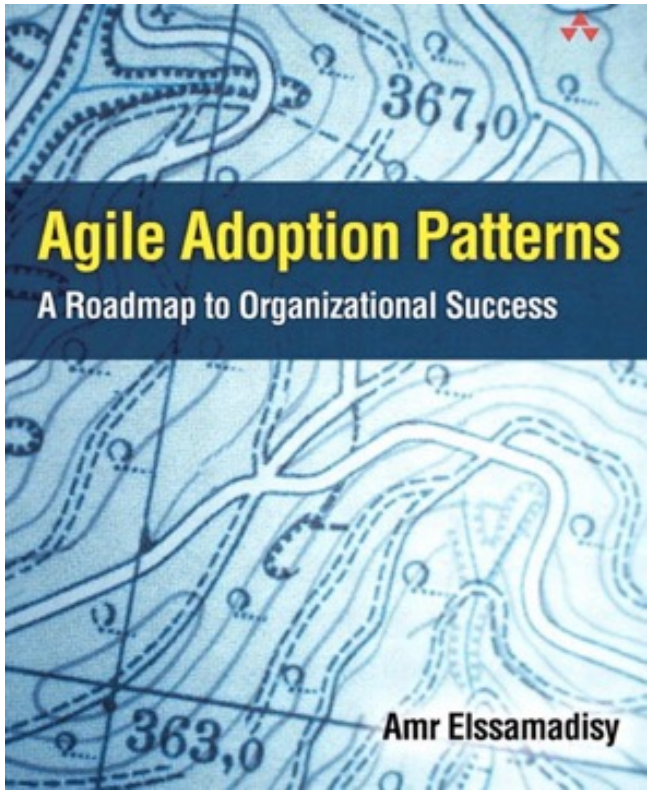
- Use e-mail cautiously
- Do not include emotions in email
- Select carefully the recipients and subject
- Make communication guidelines explicit
- Avoid using e-mail for:
  - Confrontations
  - Criticisms
  - Sensitive information
  - Politically incorrect topics
  - Legally actionable statements
- Use other media if there is any doubt

Business & process smells

"Smells are indicators that business value is not being delivered where it should be. "

— Amr Elssamadisy

- Indicator that something has gone wrong
- Business smells can be perceived by the customer
- Process smells are only visible to the development team
- *Smells are often a better incentive than proactive cleanliness.*



Example of process smells

- **Customer? What Customer?**
  - *Direct and Regular Customer Input Is Unrealistic*
  - Backlog, Release Often
- **Management Is Surprised**
  - *Lack of Visibility*
  - Done State, Demo, Backlog, Stand-Up Meeting
- **Bottlenecked Resources**
  - *Software Practitioners Are Members of Multiple Teams Concurrently*
  - Self-Organizing Team, Collective Code Ownership
- **"Hardening" Phase Needed at End of Release Cycle**
  - Automated Developer Tests, Functional Tests, Continuous Integration

"Adding manpower to a late software project makes it later."

— Frederick Brooks, *The Mythical Man month*