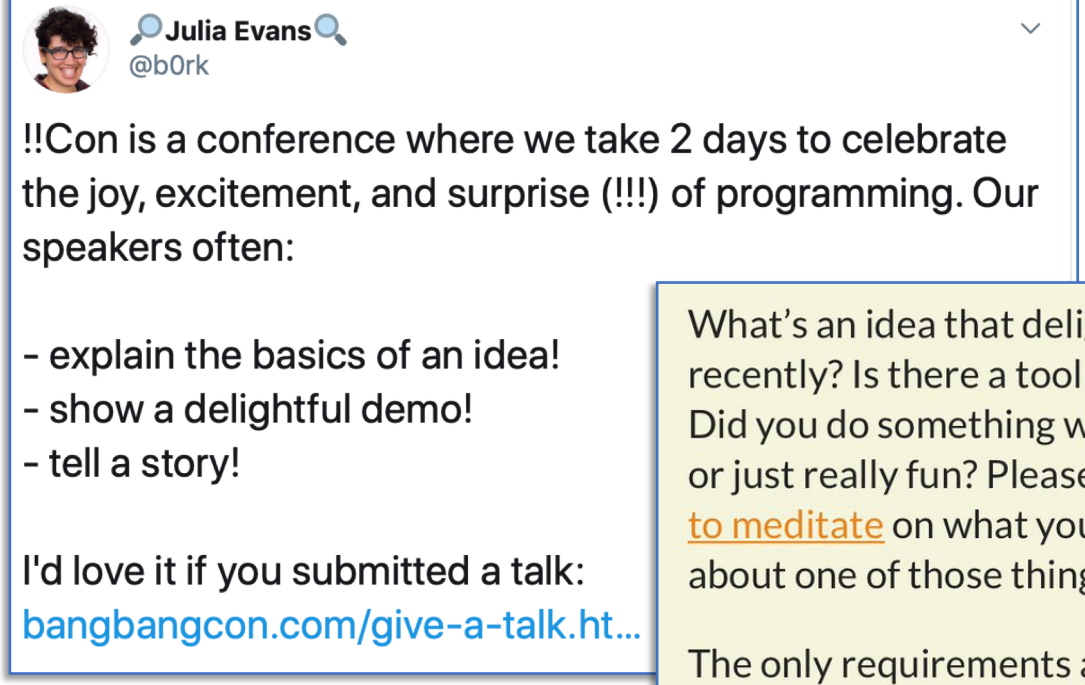
Lean Architecture (I)

Arie van Deursen



What's an idea that delights you? Did you learn something surprising recently? Is there a tool you love that you've been telling everyone about? Did you do something with computers that seems impossible or amazing or just really fun? Please take this call for talk* proposals as an invitation to meditate on what you love about computing, and then submit a talk* about one of those things!

The only requirements are that your talk*:

- be computing-related!
- be about something you think is interesting and cool!
- **have at least one exclamation mark in the title!**

Casper Boone 12:55 PM

1

Commented on lguerchi's message: **Hi, when are you planning to provide us the github repository to for..**

> As soon as you've made a group and chosen a project (i.e. your proposal on Brightspace got approved), we will add you to the repository on GitLab. Note that you don't have to (and actually cannot I believe 😅) fork the repo 😉
>
> (edited)
>
> 💪 2

Casper Boone

3 hours ago

We added you to the GitLab repository: https://gitlab.ewi.tudelft.nl/in4315/2019-2020/desosa2020/. Please take a good look at the README to see how everything's set up. Your project has already been added to the repository. You can already get started by writing an introduction that introduces the open source project you chose. Want an example? Take a look at the React Native or Pandas example projects in the repo.

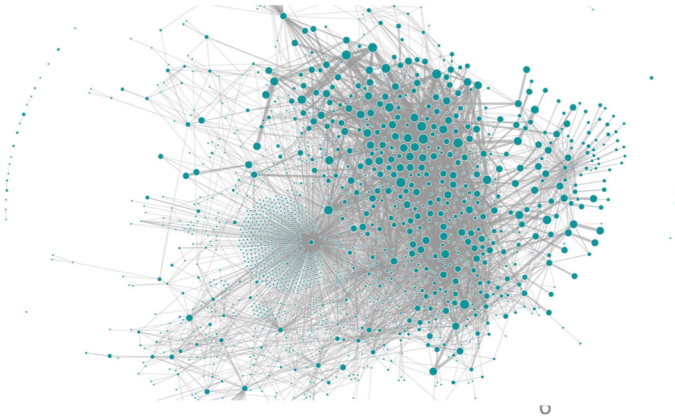
Projects Picked So Far

- Micronaut-core (micro-service architectures)
- VSCode (building on 2017)
- Openpilot
- ArduPilot
- Ripple

Date	Time	Activity	Teacher	Topic
Wed Feb 12	15:45	Lecture 1	Arie van Deursen	Course structure (slides)
Fri Feb 14	15:45	Lecture 2	Arie van Deursen	Lean Architecture I
	17:00		Grady Booch, IBM	*Ask Me Anything* on SATURN 2016 keynote
Wed Feb 19	15:45	Lecture 3	Arie van Deursen	Lean Architecture II
	17:00		Engin Bozdag, Uber	*Ask Me Anything*; privacy by design (slides)
Fri Feb 21	15:45	Lecture 4	Xavier Devroey	Architecting for Configurability
Wed Feb 26	15:45	Lecture 5	Marco Di Biase, SIG	Architecting for Maintainability
Fri Feb 28	15:45	Lecture 6	Ayushi Rastogi	Architecting as a Team Activity
Wed Mar 4	15:45	Lecture 7	Luis Cruz	Architecting for Sustainaility
Fri Mar 6	15:45	Lecture 8	Bert Wolters, Adyen	Architecting for Scalability
Wed Mar 11	15:45	Lecture 9	Ferd Scheepers, ING	Architecting for the Enterprise
Fri Mar 13	15:45	Lecture 10	Steffan Norberhuis	Architecting for Operations
Wed Mar 18	15:45	Lecture 11	tbd	tbd
Fri Mar 20	15:45	Lecture 12	Daniel Gebler, Picnic	Architecting with or wihout Microservices
Wed Mar 25	15:45	Lecture 13	tbd	tbd
Fri Apr 3	full day		All student teams	Team presentations
	17:00	Drinks	Everyone	

Engin Bozdag:

- *Senior privacy architect at Uber*
- Privacy by design in monoliths versus micro-services
- Preparation next lecture:
  - Slides of his Usenix/Enigma 2020 presentation
  - Uber privacy statements
  - Propose questions in #ama channel on MM



Transparency	Accuracy	Data Minimization	Privacy Rights
Lawfulness	Purpose Limitation	Storage Limitation	Privacy by Default

Assignment

1. Selecting an open source system: The system needs to be sufficiently complex, under active development, and open to external contributions.
2. Writing four essays, covering
    1. the product vision, including required capabilities, roadmap, product context, and stakeholder analysis.
    2. architectural decisions made, including system decomposition, tradeoff points, as well as architectural styles and patterns adopted.
    3. quality control and assessment; and
    4. a deeper analysis based on the lectures or other relevant material specific to the system of choice;
3. Contributing changes to the open source system selected (via pull requests submitted on GitHub)
4. Preparing a final poster and presentation
5. Reviewing work from other teams, to learn from them, and to give them feedback

Deadlines

Date	Time	Writing	Coding	Reviewing
Mon Feb 17	17:00		Teams have selected project	
Wed Feb 19	17:00		Top-level decomposition	
Mon Mar 9	17:00	Team essay 1	–	–
Mon Mar 16	17:00	Team essay 2	First pull requests and plan	Essay 1
Mon Mar 23	17:00	Team essay 3	–	Essay 2
Mon Mar 30	17:00	Team essay 4	Pull request report	Essay 3
Mon Apr 6	17:00	–	–	Essay 4, poster

Students will receive grades based on the following:

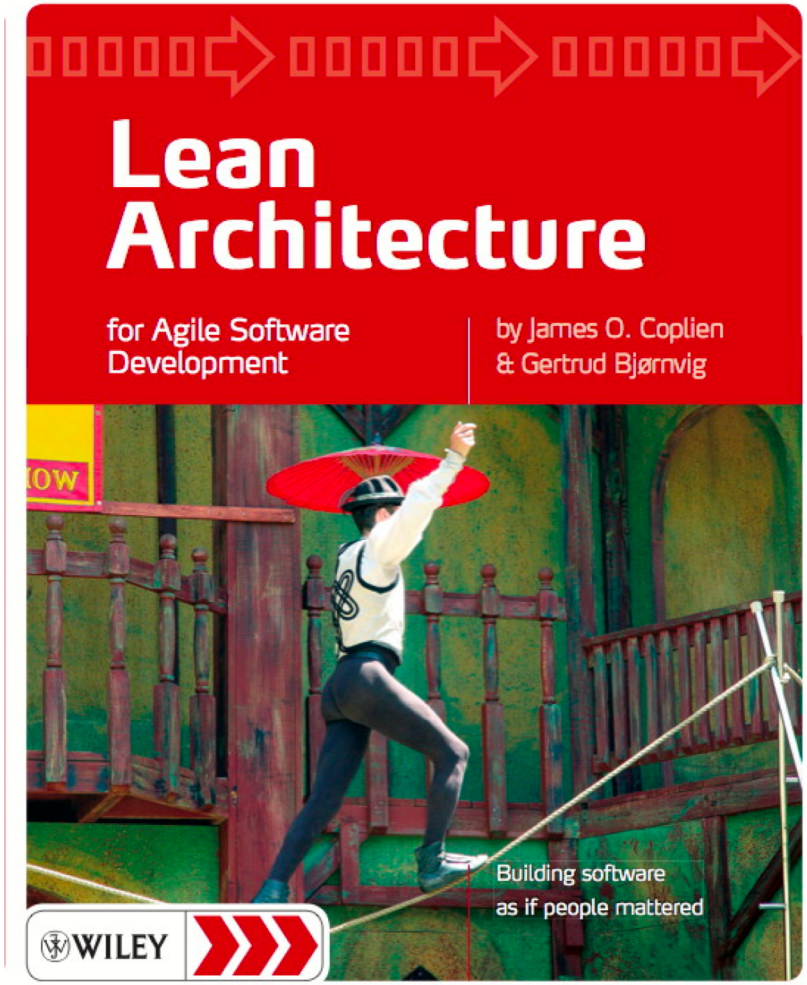
- `E`: Team performance for each of the four essays (1-10), composed form the average of the four essays E1..E4.
- `C`: Team performance for code contributions (1-10)
- `P`: Team performance for poster presentation (1-10)
- `R`: Individual performance in per review reviews (-1, 0, 1) – zero by default
- `A`: Individual performance in participation (-1, 0, 1) – zero by default

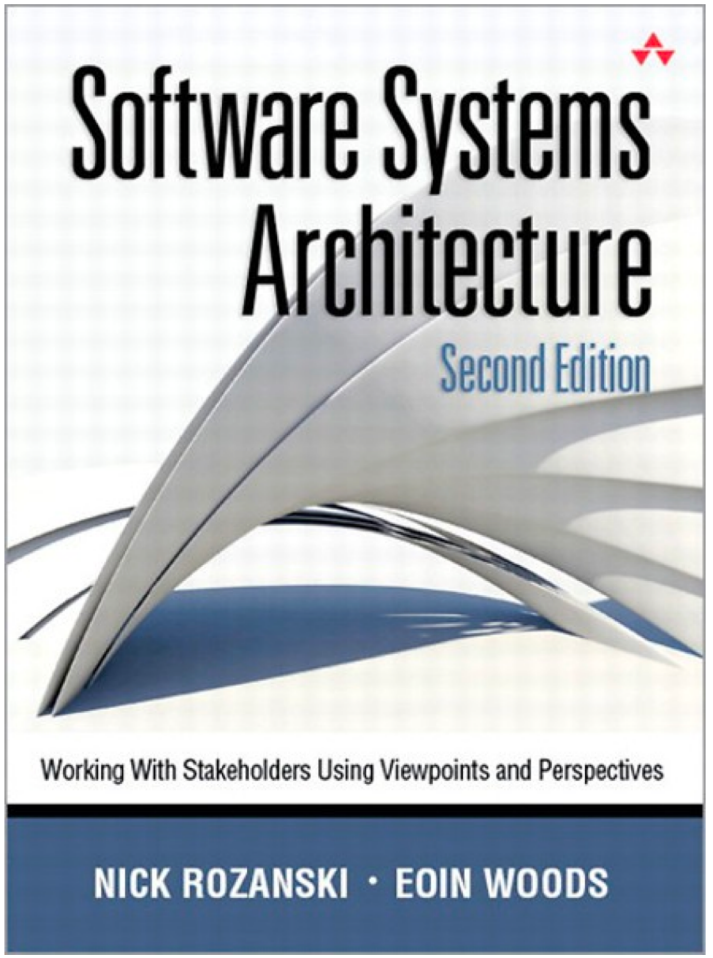
The *team grade* is the weighted average of the team activities:

```
T = (3*E + C + P)/5
```

The *individual grade* then is the team grade to which a bonus can be added (or subtracted) for exceptionally (top/bottom X%) strong results.

```
I = T + 0.5 * (R + A)
```


Lean
Architecture
for Agile Software
Development
by James O. Coplien
& Gertrud Bjørnvig
Building software
as if people mattered
WILEY


Software Systems
Architecture
Second Edition
Working With Stakeholders Using Viewpoints and Perspectives
NICK ROZANSKI · EOIN WOODS

Lean Architecture	Classic Software Architecture
Defers engineering	Includes engineering
Gives the craftsman "wiggle room" for change	Tries to limit large changes as "dangerous" (fear change?)
Defers implementation (delivers lightweight APIs and descriptions of relationships)	Includes much implementation (platforms, libraries) or none at all (documentation only)
Lightweight documentation	Documentation-focused, to describe the implementation or compensate for its absence
People	Tools and notations
Collective planning and cooperation	Specialized planning and control
End user mental model	Technical coupling and cohesion

The System Vision (Essay E1)

- You can only architect a system if you know what it is supposed to achieve
- What (business) value does / will it generate?
- Which *defining* properties should the system realize?
- What resources are needed to realize the system?
- What resources are needed to operate the system?
- To what system roadmap does this lead?

As an architect, think of anything that is “architecturally significant”

End Users' Mental Model

System architecture should reflect the **end users' mental model** of their world:

1. System **form** relates to the user's thought process when viewing the screen, and to what the system *is*
2. System **functionality** relates what end users *do* – interacting with the system – and how the system should respond to user input.

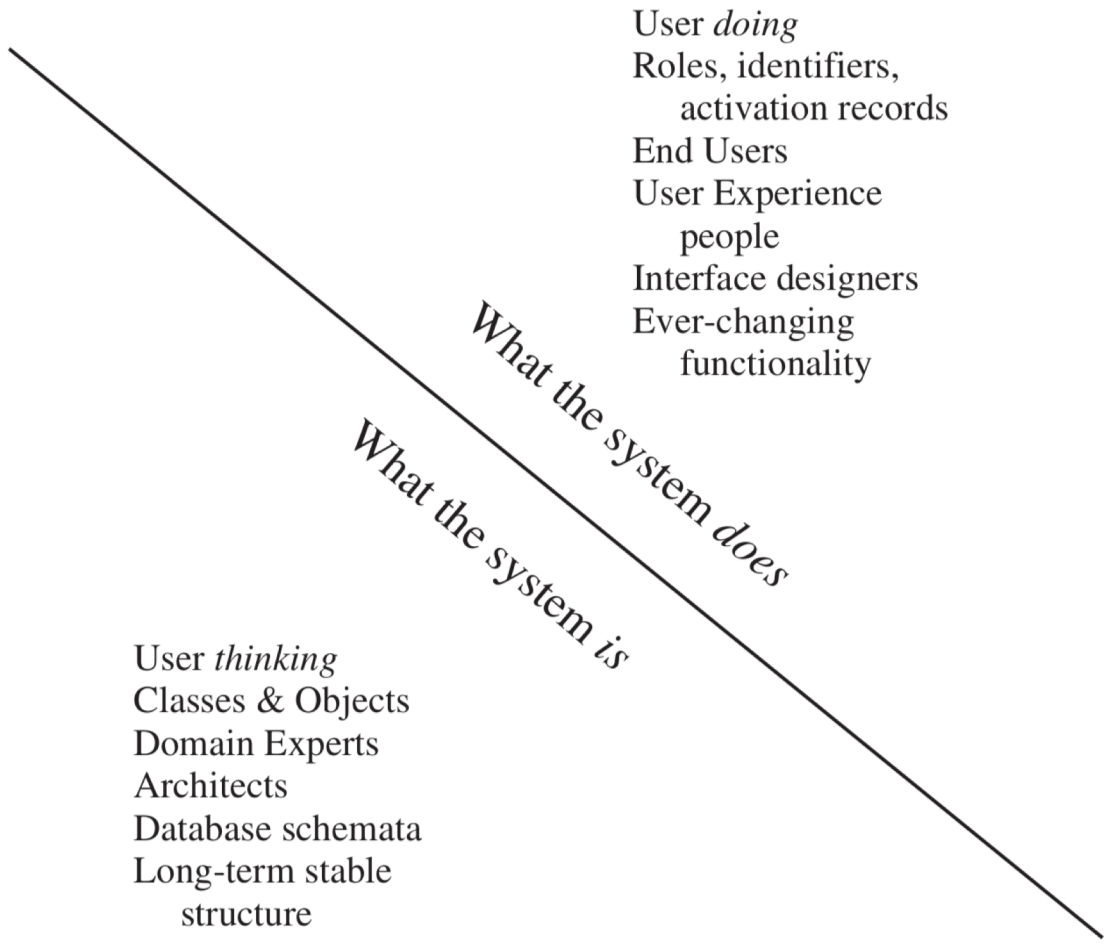
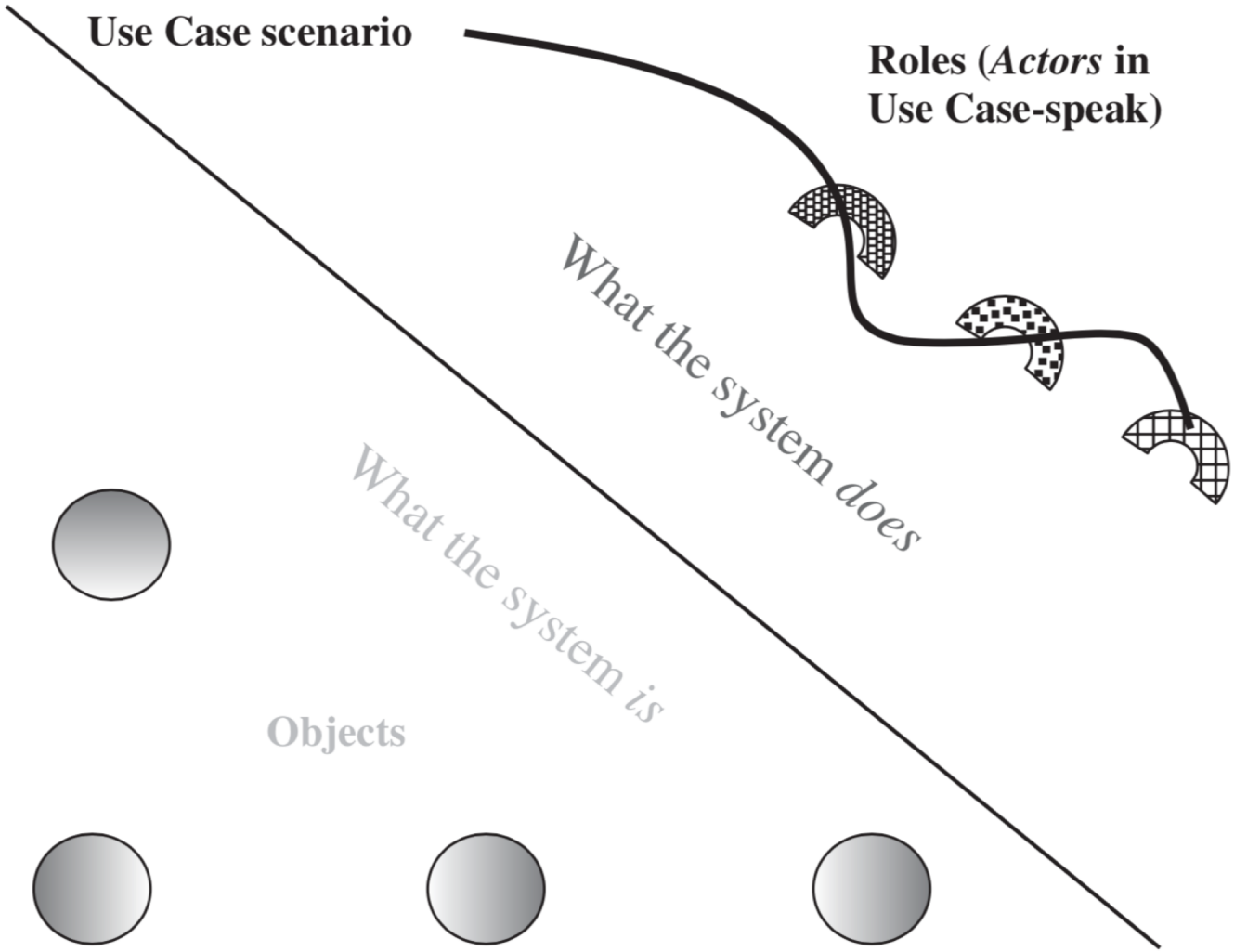


Figure 2-1 What the system *is*, and what the system *does*.


Use Case scenario
Roles (*Actors* in Use Case-speak)
What the system *does*
What the system *is*
Objects

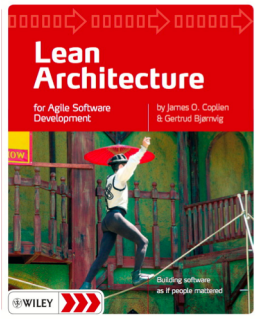
Exploring Form and Function

- To explore both form and function requires up-front engagement of all stakeholders, and early exploration of their insights.

- Deferring interactions with stakeholders, or deferring decisions beyond the responsible moment slows progress, raises cost, and increases frustration.

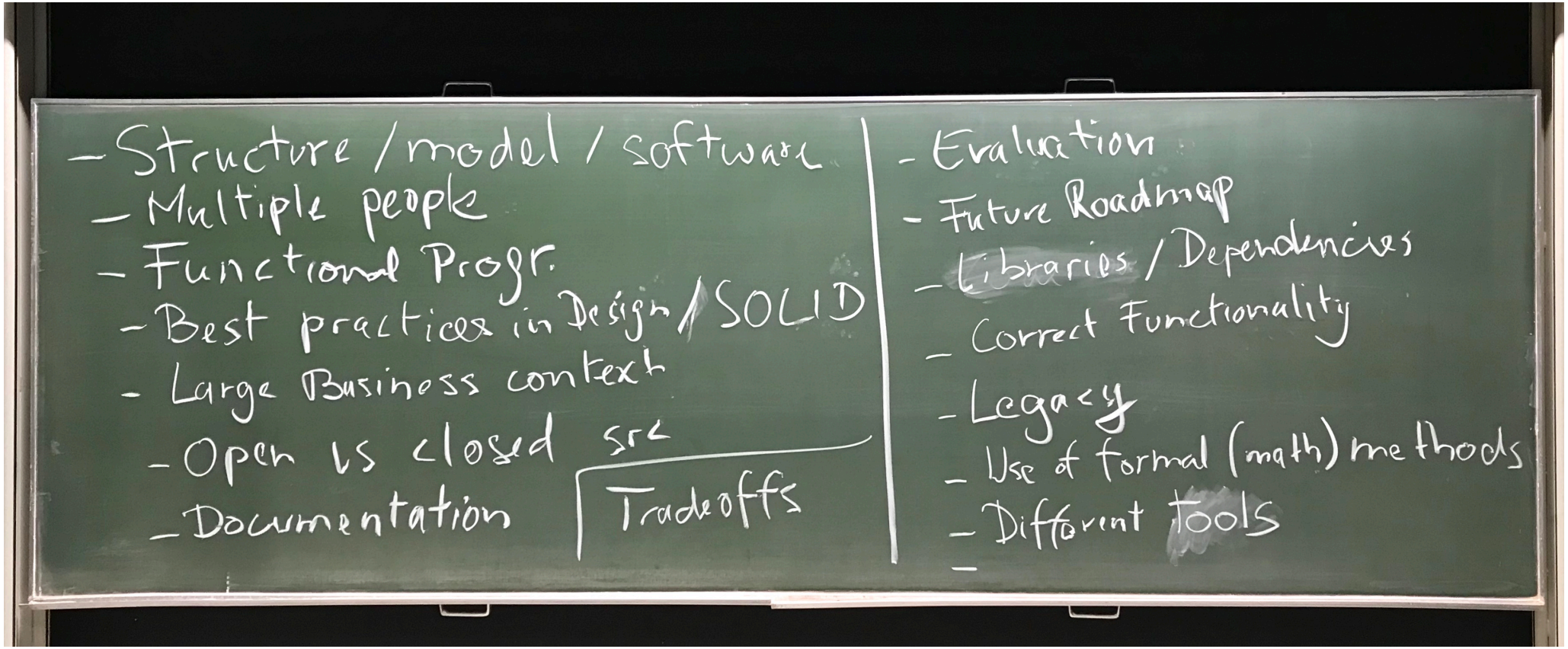
- A team acts like a team from the start.

Stakeholder *Engagement*



- “Maybe half of software development is about nerd stuff happening at the whiteboard and about typing at the keyboard.”
- “The other half is about people and relationships. “
- There are few software team activities where this becomes more obvious than during architecture formulation.

[Your Learning Objectives]



The Lean *Value Stream*

- Continuous, end-to-end process, of people delivering value to end users
- Avoid stress, overtime, cutting corners on the process to meet deadline
- “Lean production organizes around value streams instead of production steps”

Lean Architecture

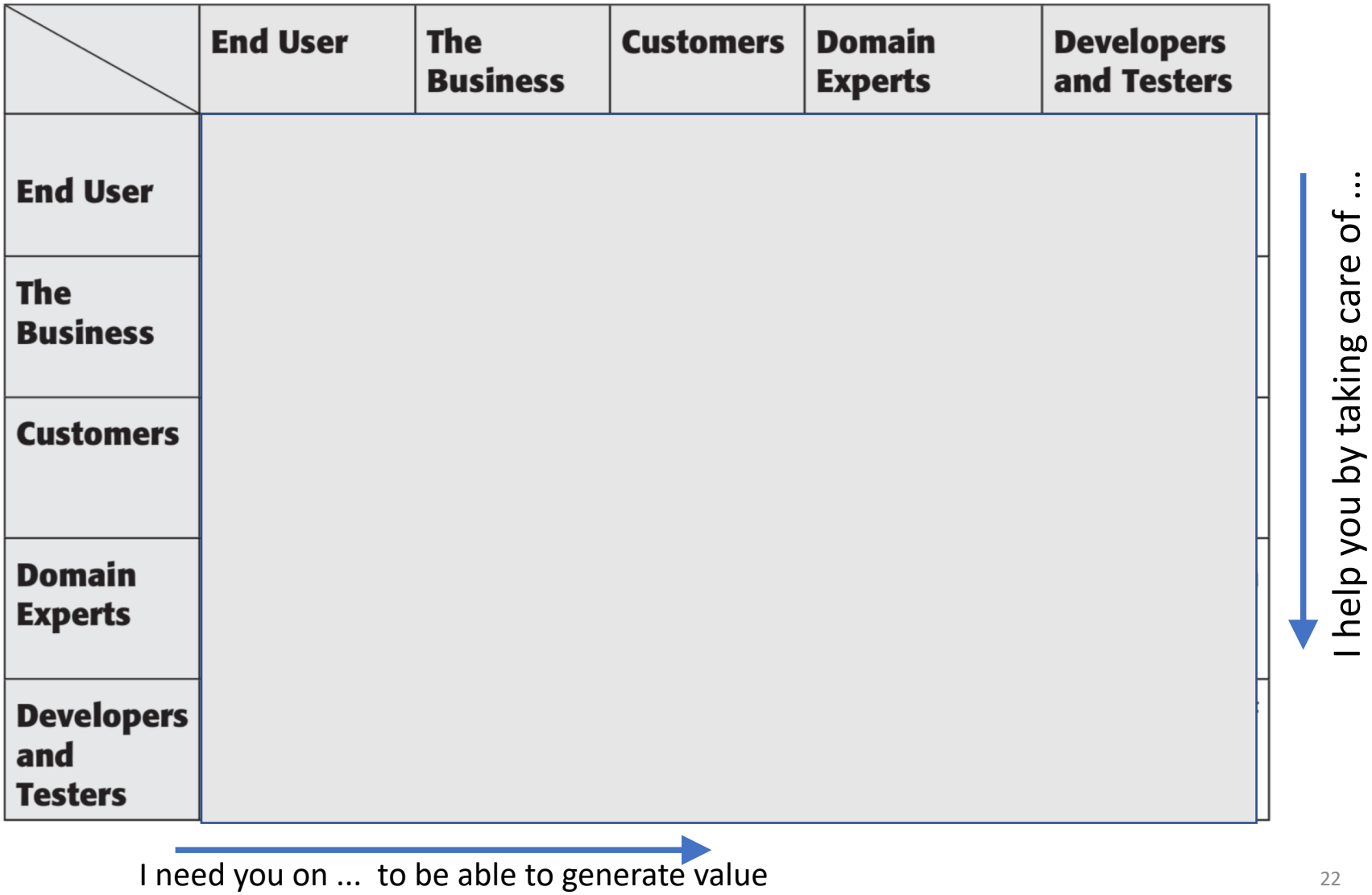
- Shorten time between
  - understanding user needs
  - and delivering a solution
- Maximize likelihood that we will deliver what the customer expects
- Realize cost savings (that can be passed on to the user):
  - Reduce distance between end user world model and code structure
  - Reduce *waste* (features not needed, bugs, waiting)
  - Avoid *rework* (re-doing work you could have done once)
- Maintain a consistent vision that helps system parts fit together

Stakeholders

- “Many different stakeholders derive value from your product”

- Major stakeholder areas include:
  - End users
  - The business
  - Customers
  - Domain experts
  - Developers and testers

A true software architect is one who is a domain expert, who knows how to apply the domain expertise to the design of a particular system, and who materially participates in implementation


End User
The Business
Customers
Domain Experts
Developers and Testers
End User
The Business
Customers
Domain Experts
Developers and Testers
I help you by taking care of ...
I need you on ... to be able to generate value

	End User	The Business	Customers	Domain Experts	Developers and Testers
End User		Feature priorities, scope	Purchase conve-nience	Product/ feature feasibility	Quality and proper functionality
The Business	Feasibility	Create standards	Process require-ments	Feasibility	Source of revenue
Customers	A market	Products and services	Create standards	Compliance with standards	Source of revenue
Domain Experts	Range of product variation	Workplace well-being	Need for standards	Domain synergies and conflicts	Constraints on technology
Developers and Testers	Requirement Ccarification	Workplace well-being	Advice on delivery process	Guidance, APIs, *poka-yoke*	Clarification of how existing code works

I help you by taking care of ...

I need you on ... to be able to generate value

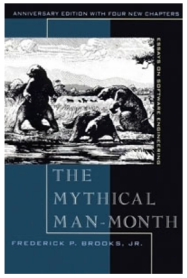
End Users

- What the customer *wants* or *needs?*
- What the customer expects!

- Beyond user stories: Elicit an end-user *cognitive model*
  - *"*Users carry models of the internals of the program they are using "

- Form: (Stable) domain structure
- Function: Domain behavior / use cases

Fred Brooks: Conceptual Integrity



- The quality of a system where all the concepts and their relationships with each other are applied in a consistent way throughout the system.

- Conceptual Integrity is the most important consideration in system design.

- It is better to have […] one set of design ideas,
  than [...] many good but independent and uncoordinated ideas.

The Business

- Funds software development
- Needs return on investment

- Has stake in well-being of its employees
- Management, sales, marketing, ...

- Owns decisions about project scope
- Buy-versus-build decisions

Customer

- Middleman *paying* for the product
- Investor before there are any users

- See the costs, but don't feel the benefits
- Focus on delivery times and rationalizing the development process

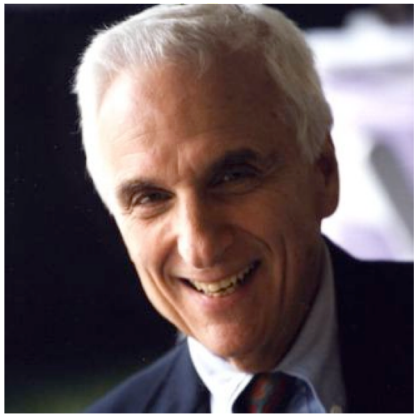
Domain Experts

- "*Over the years domain experts have integrated the perspectives of multiple end user communities and other stakeholders into the forms that underlie the best systems.*"
- "Domain expert engagement is to architecture as end user engagement is to feature development"
- "Innovation in the solution domain goes hand-in-hand with long-term experience of solution domain experts"

Developers and Testers

- Developers are the prime oracles of technical feasibility
- Developers should be *active* experts:
  - Gather empirical data about architectural trade-offs
  - *genchi genbutsu* (現地現物): go look and see for yourself.
- Developers should own the development estimates
- Developers and testers “should be friends”
- Architecture defines your test points
- Usability testing is done *before* coding begins

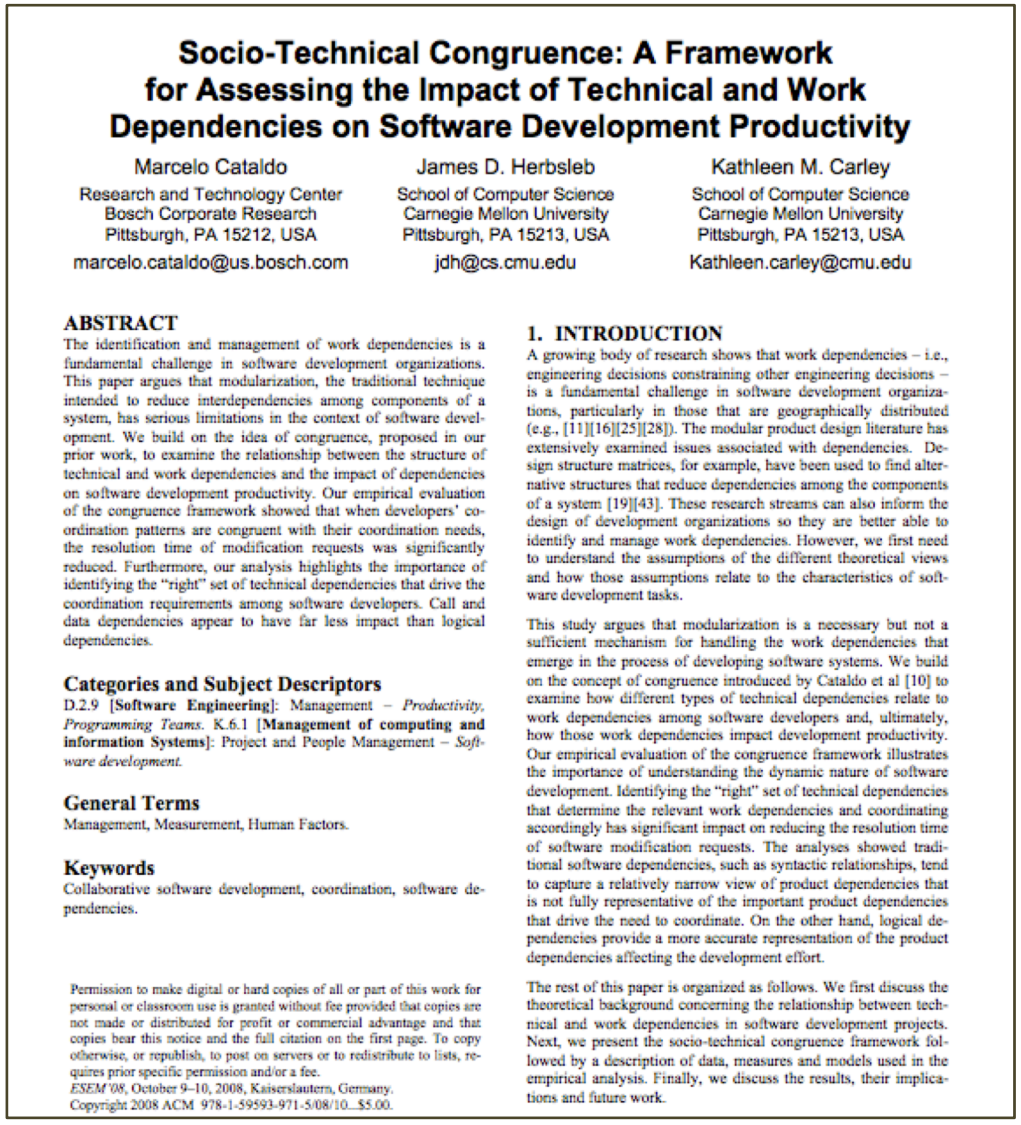
Conway's Law



"Organizations which design systems ...
are constrained to produce designs
which are
copies of the communication structures
of these organizations"

Melvin Conway, 1968

Socio-Technical Congruence



Socio-Technical Congruence: A Framework for Assessing the Impact of Technical and Work Dependencies on Software Development Productivity

Marcelo Cataldo
Research and Technology Center
Bosch Corporate Research
Pittsburgh, PA 15212, USA
marcelo.cataldo@us.bosch.com

James D. Herbsleb
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA
jdh@cs.cmu.edu

Kathleen M. Carley
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213, USA
Kathleen.carley@cmu.edu

ABSTRACT

The identification and management of work dependencies is a fundamental challenge in software development organizations. This paper argues that modularization, the traditional technique intended to reduce interdependencies among components of a system, has serious limitations in the context of software development. We build on the idea of congruence, proposed in our prior work, to examine the relationship between the structure of technical and work dependencies and the impact of dependencies on software development productivity. Our empirical evaluation of the congruence framework showed that when developers' coordination patterns are congruent with their coordination needs, the resolution time of modification requests was significantly reduced. Furthermore, our analysis highlights the importance of identifying the "right" set of technical dependencies that drive the coordination requirements among software developers. Call and data dependencies appear to have far less impact than logical dependencies.

Categories and Subject Descriptors

D.2.9 **[Software Engineering]**: Management – *Productivity, Programming Teams.* K.6.1 **[Management of computing and information Systems]**: Project and People Management – *Software development.*

General Terms

Management, Measurement, Human Factors.

Keywords

Collaborative software development, coordination, software dependencies.

ESEM'08, October 9–10, 2008, Kaiserslautern, Germany.

1. INTRODUCTION

A growing body of research shows that work dependencies – i.e., engineering decisions constraining other engineering decisions – is a fundamental challenge in software development organizations, particularly in those that are geographically distributed (e.g., [11][16][25][28]). The modular product design literature has extensively examined issues associated with dependencies. Design structure matrices, for example, have been used to find alternative structures that reduce dependencies among the components of a system [19][43]. These research streams can also inform the design of development organizations so they are better able to identify and manage work dependencies. However, we first need to understand the assumptions of the different theoretical views and how those assumptions relate to the characteristics of software development tasks.

This study argues that modularization is a necessary but not a sufficient mechanism for handling the work dependencies that emerge in the process of developing software systems. We build on the concept of congruence introduced by Cataldo et al [10] to examine how different types of technical dependencies relate to work dependencies among software developers and, ultimately, how those work dependencies impact development productivity. Our empirical evaluation of the congruence framework illustrates the importance of understanding the dynamic nature of software development. Identifying the "right" set of technical dependencies that determine the relevant work dependencies and coordinating accordingly has significant impact on reducing the resolution time of software modification requests. The analyses showed traditional software dependencies, such as syntactic relationships, tend to capture a relatively narrow view of product dependencies that is not fully representative of the important product dependencies that drive the need to coordinate. On the other hand, logical dependencies provide a more accurate representation of the product dependencies affecting the development effort.

The rest of this paper is organized as follows. We first discuss the theoretical background concerning the relationship between technical and work dependencies in software development projects. Next, we present the socio-technical congruence framework followed by a description of data, measures and models used in the empirical analysis. Finally, we discuss the results, their implications and future work.

A Good Problem Definition (Ch. 4)

- It is written down and shared.
- It is a difference between the current state and some desired state of the organization or business.
- Its achievement is measurable, usually at some mutually understood point in time.
- It is short: one or two sentences in clear, simple, natural language.
- It is internally consistent: that is, it does not set up an over-constrained problem.

Jerry Weinberg: Ask yourself: Who *owns* the problem?

Mapping Between Problems and Solutions

- The relationship between problem and solution is rich and complex.
- You can’t just start with a problem definition and methodically elaborate it into a solution
- The mapping from problems to solutions is many-to-many
- Multiple seemingly unrelated solutions might be required to solve what you perceive as a single problem
- Some problems seem to defy any mapping at all

Essay 1: The Product Vision

The starting point for your architectural analysis is a description of the *vision* underlying your project and its future success. Aspects to take into account include:

1. A concise, inspirational characterization of what the project aims to achieve
2. A description of the end-user mental model that is central to the system
3. A characterization of the key capabilities and properties the system should provide
4. An analysis of the stakeholders involved in the project, and how the resulting system is beneficial to them
5. A description of the current and future context in which the system operates
6. A product roadmap, laying out the main future directions anticipated for the upcoming years