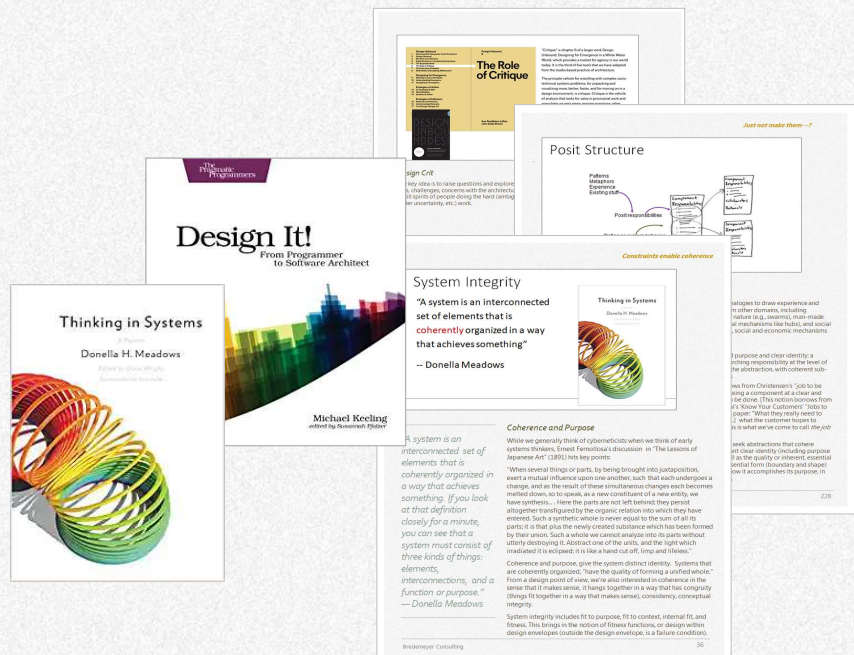


System Design and Software Architecture

Ruth Malan
Bredemeyer Consulting



October 2024

Copyright

Since this is teaching material, I weave together quotes and references to work by others. I have tried to take care to note the source of quotes and image sources, and these obviously belong to their original authors and creators.

Reasonable use of original work herein is permitted, but please give appropriate credit to Ruth Malan, and indicate if changes were made (though please do not do so in any way that suggests we endorse you or your use, unless you review it with us first and get said endorsement).

Copyright © 2024 Ruth Malan

Technical Leadership Workshops

Remote:

- December 4 and 11, 2024, 12pm-3pm Eastern Time (US/Canada).

System Design and Software Architecture Workshops

Remote:

- Feb 24-26 and Mar 3-5, 2025, 11am-3:30pm Eastern Time (US/Canada).

See ruthmalan.com and ti.to/bredemeyer/ for schedule and more information.

*discourse (n.): late 14c.,
"process of understanding,
reasoning, thought,"*

AGENDA

Setting the Scene

What: Concepts in Systems, Design and Architecture

How: Designing Systems

How: Frames and Practices



Who: Roles and Organizational Dynamics

Closing the Scene

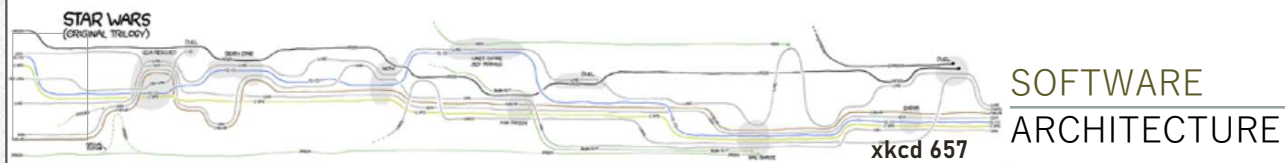


Image: by Randall Monroe, <https://xkcd.com/657/>

The XKCD 657 narrative map is used to suggest (but only loosely and figuratively; there's no intended or hidden meanings to be found by looking for parallels) we're all unfolding our own story, and our journeys through these several days, and through this space are all going to be unique, and challenging, and will have some familiarity and surprises. The slider is just a reminder of where we are. Now, we're at *a* beginning. Setting the scene. But we're also, no beginning is *the* beginning. There's all that came before, and each of us is well into our journey of learning in software and systems, and we bring so much experience and insight to the work we will do together.

The territory we span here, is vast. Choices had to be made. This is one path.

*"The map is not the territory," Snicket's chaperon advises him. "That's an expression which means the world does not match the picture in our heads."
— Lemony Snicket, Who Could That Be at This Hour?*

"the word is not the thing"
— Alfred Korzybski

AGENDA: Frames and Practices

	System in context	System (internals)	
Business Strategy	Business Strategy	Engineering Strategy	Engineering Strategy
Product Design	Product Design	Conceptual Architecture	Conceptual Architecture
System Properties	System Fitness	Execution Architecture	Physical Architecture
Platform Design	Platform Design	Logical Architecture	Logical Architecture

SOFTWARE *and* SYSTEM
ARCHITECTURE DESIGN

Agenda as Map

We will follow the “map” (or framework of frames and practices) in the slide. It reflects an orientation to system design where the design of the system internals is informed by what the system is and is becoming, and the various organizational, development and user contexts that place demands on the system. Of course, system design is not just evolutionary design, but highly nonlinear. For one thing, decisions interact, and we work across views, with a willingness to backtrack and rethink as we discover and learn and adapt to change (in our understanding, context, other parts of the system, ..). The various facets of system design are not evolved sequentially, though the material here has to be presented in some order. So, once we have explored the formative concepts of architecture and systems, this workshop will iterate between System-in-Context and System Internals views and related design practices.

Woods' Theorem: "As the complexity of a system increases, the accuracy of any single agent's own model of that system decreases rapidly."

Welcome!

approximately

today

11:00 – 11:50

Systems

Systems and
architecture

12:00 – 1:00

and Arch

1:00 – 1:30

Break

1:30 – 2:30

Strategy &

Strategy and
situational awareness

2:40 – 3:30

Sit. Aware.

Eastern Time

Co-Creators of this Experience

You! This workshop draws together what we have learned architecting, and from architects, and further, has the benefit of your years of experience in software development, and leverages that. We have come via different paths of experience and formal and self-education, building unique knowledge foundations and insights, so we each have much to offer each other, as co-teachers and co-learners. The format creates “containers” for collaboration, both as we practice the practices of system design and architecting (in team exercises), and in the large-group sessions where slides and notes create context for conversation. Learning is multi-faceted, but includes seeing anew and making sense of our experience and adding to it. We learn by doing, but also by adopting and trying out heuristics others have distilled from their experiences, and relating them to our contexts. Conceptual frameworks help organize both knowledge and practice, and foster (inter)connections. Various canvases, templates, and diagrams help convene the conversations where we collaborate on system understanding and system shaping, designing to “make things more the way they ought to be” (Herb Simon). Quotes, and our own insights and conversations, are among the matters we gather to think other matters with (Donna Haraway).

*'It matters what matters we use
to think other matters with; it
matters what stories we tell to
tell other stories with; it matters
what knots knot knots; what
thoughts think thoughts, what
descriptions describe
descriptions, what ties tie ties. It
matters what stories make
worlds, what worlds make
stories.'*

*— Donna Haraway, Staying
With The Trouble*

Let's Go!

Setting the Scene

Architecture: What?

Architecting: How?

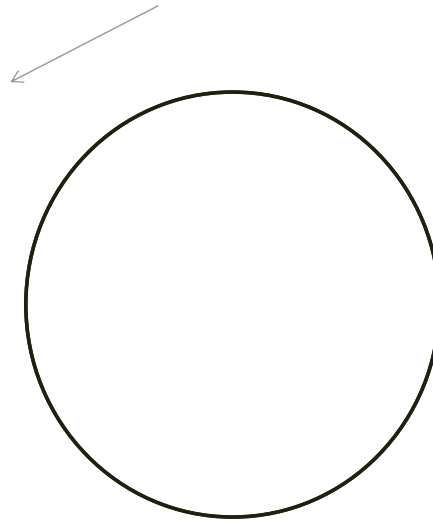
Architects: Who?

We Are Our Own Stories

Think of some system design or architecture work you did, that impacted how things worked out

- Jot down some notes about the work and the situation it addressed
- Draw a circle. Inside the circle: describe what you did to influence the situation
- Outside the circle: describe what others did, to shift outcomes

We are doing this



ARCHITECTS
AS LEADERS

You are your own stories. — Toni Morrison

Architecture Stories

We like to begin with a story, and we could begin with a story from history, or our field. But we are our own stories, too, and so let's begin there. Let's spend a few minutes reflecting on some situation we've been in, where we did some architecture work, and we like what we brought to it. Not that we think everything was perfect, but where we brought something to the situation that impacted outcomes and experiences.

Draw a circle. To one side, describe the situation briefly. Inside the circle, describe what you brought to the situation, to influence and impact "success" (effectiveness or achievement of desired outcomes). Don't shy away from noticing things to learn from, like what didn't go so well. Our stories are messy. Outside the circle, add what others brought to the situation, to impact (or impede) success. We can repeat this, reflecting on our experience, filling out more of the space, with situations and what we brought to them.

Stories are crucibles for learning. Our own stories too. In these stories, it is worth drawing out: what was the problem or challenge and what made it important or of value to solve? What role did we and others play? What did we and others bring to it?

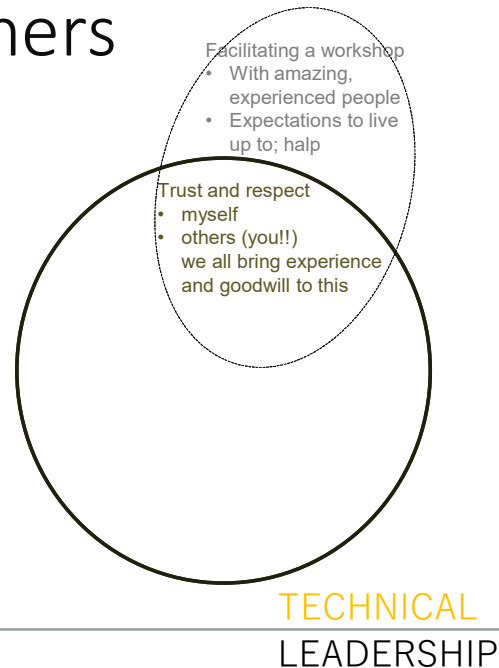
Source: Toni Morrison's Commencement Address to the Wellesley College Class of 2004

"Of course, you're general, but you're also specific. A citizen and a person, and the person you are is like nobody else on the planet. Nobody has the exact memory that you have. What is now known is not all what you are capable of knowing. You are your own stories and therefore free to imagine and experience what it means to be human without wealth. What it feels like to be human without domination over others, without reckless arrogance, without fear of others unlike you, without rotating, rehearsing and reinventing the hatreds you learned in the sandbox. And although you don't have complete control over the narrative (no author does, I can tell you), you could nevertheless create it." — Toni Morrison

We Are Because of Others

As we do so, what do we notice

- About situations needing leadership?
- About the role of others, when we're leading?
- About ourselves, when we're leading?



Ubuntu: "I am because we are"

Leading and Following

Kurt Lewin proposed the following heuristic equation:

Lewin's Equation: $B = f(P, E)$

Behavior is a function of a **P**erson interacting with the **E**nvironment (or situation)

Our leading in a context has various attributes, including our noticing what in the situation called for leadership, and following in the sense of actively pitching in to co-shape intent and the response to the situation, and get something done, that we couldn't have done alone. And this is ongoing, as we and the situation co-evolve.

"I am because we are, and since we are therefore I am" — John Mbiti

"We know from everyday experience that a person is partly forged in the crucible of community." — Abeba Birhane

Quote Source:

Nanette Monin and Ralph Bathurst, "Mary Follett on the Leadership of 'Everyman'," 2008

Abeba Birhane, "Descartes was wrong: a person is a person through other persons," 2017

"[Mary Parker] Follett argues that the primary responsibility of leadership is to discover the sense-making thread that structures understanding of the 'total situation', establish the 'common purpose' that emerges from this, and by leading, 'anticipating', make the next situation."

— Nanette Monin and Ralph Bathurst

Introduction to Systems and Software Architecture

Architecture

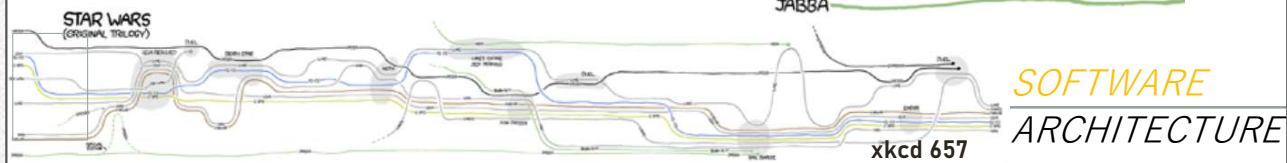
Systems

Design

Decisions

ARCHITECTURE ESSENTIALS

What is architecture?
What are (complex) systems?
What is design?
And design decisions?



Architecture

This section explores what architecture is.

Systems and Complex Systems

Systems are not just more, but other, than the sum of their parts (Jabe Bloom, adapting Russ Ackoff). We will explore systems, and complexity.

Design

Simply put, "we design to make things more the way we want them to be" (Herbert Simon). We will elaborate on design as it applies to, and is distinguished in, architecture.

Decisions

Design Decisions: we will introduce the facets of decisions (context, constraints, decisions as enable and constrain, tradeoffs, side-effects and consequences, alternatives and options) as a bridge to design decision making.

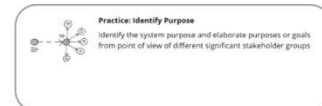
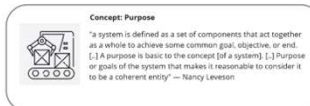
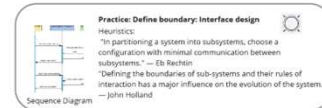
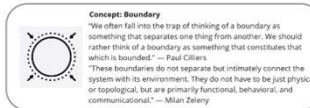
*"All architecture is design, but
not all design is architecture."*
— Grady Booch

*"Architecture is the thoughtful
making of space."*
— Louis Kahn

Gathering What We Know

“In school, you're taught a lesson and then given a test. In life, you are given a test that teaches you a lesson.”

— Tom Bodett*



* variations on this quote abound; I just picked one (ascribed to Bodett) that playfully fits our first team exercise
Card deck image top: Dave Gray, Visual Frameworks

Exercise: Concepts and Practices

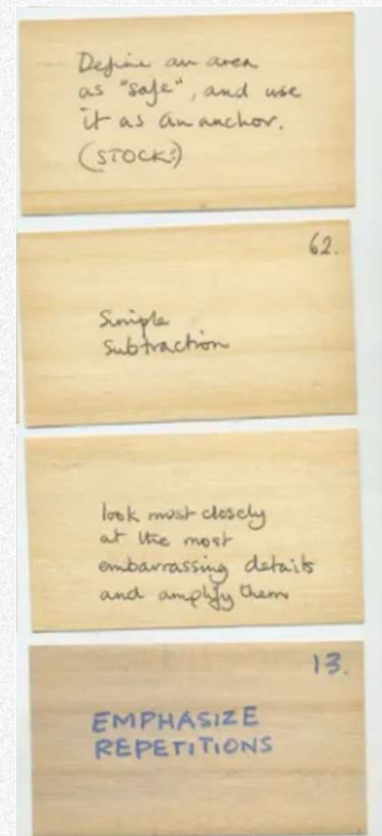
There are instructions on the Miro board, but essentially we’re creating concept and practice cards. In person, we’d use index cards (that have fronts and backs). We’re simulating those cards on the Miro board (there’s a template, but use freedom in adapting to your needs).

The most important part of this activity is to have some fun creating several cards that capture some of the key concepts in systems and architecture, that shape how we go about designing systems and software architecture, along with some practice guidance that you’re harvesting from your experience and investigative research or reading.

The activity has several veins to it, including starting to get to know each other a little, by working on something that draws on and draws out the vast experience we collectively have in this group. It has many opportunities to contribute (whether it’s ideas for a representative evocative image to go with the concept, or a quote you find useful in the way it shapes an insight just so, or a heuristic wrestled from experience, and so forth).

But it is time-bound, so we accept good enough. It’s not so much a test of what we can do in the time, as it is a test of our willingness to pitch in and work collaboratively, and then accept good enough, knowing that we will iterate over this space of ideas, and have many opportunities share and learn.

Oblique Strategies from Brian Eno and Peter Schmidt, 1974:



<https://www.themarginalian.org/2014/01/22/brian-eno-visual-music-oblique-strategies/>

What is Software Architecture? '92



The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss



Controversy Corner

What do software architects really do?

Philippe Kruchten*

Electrical and Computer Engineering, University of British Columbia, 2332 Main Mall, Vancouver, BC

ARTICLE INFO

Article history:
Received 24 June 2008
Received in revised form 19 August 2008
Accepted 19 August 2008
Available online 28 August 2008

Keywords:
Software architecture
Software architect
Antipattern
Time-management

ABSTRACT

To be successful, a software architect must balance between an external focus, developing a long-term vision, reflective focus: spending time to make Teams that stray too far away from it antipatterns of software architecture

1. Introduction

"—Mr. Beck, what is software architecture?" asked a participant at an OOPSLA workshop in Vancouver in the fall of 1992. "—Software architecture?" replied Kent, now famous for being the father of XP (eXtreme Programming, not the O.S.), "well, it is what software architects do." (Chuckles in the audience.) "—So then, what is an architect?" "—Hmm, 'software architect' it's a new pompous title that programmers demand to have on their business cards to justify their sumptuous emoluments."

1. Introduction

"—Mr. Beck, what is software architecture?" asked a participant at an OOPSLA workshop in Vancouver in the fall of 1992. "—Software architecture?" replied Kent, now famous for being the father of XP (eXtreme Programming, not the O.S.), "well, it is what software architects do." (Chuckles in the audience.) "—So then, what is an architect?" "—Hmm, 'software architect' it's a new pompous title that programmers demand to have on their business cards to justify their sumptuous emoluments."

In the following four years I was going to lead a rather large team of software architects and I often ask myself that very question: "what do architects really do?" and was also asked this by my management and my customers. Since then I have seen many architecture teams in many countries, in companies of all sizes and various domains, and I have witnessed a wide range of good, not-so-good, and really bad answers to this question.

So domain some what line in design must revisit

Let us assume for now that the architects' responsibilities are the part of both the design and the design decisions that have long-lasting impact on some of the major quality attributes of a software-intensive system: cost, evolution, performance, decomposability, safety, security, etc., and still able to support the functionality expected by its end user. Then this is what software architects should be focused on, this is what software architects should do: make design choices, validate

Preamble

How we approach architecture, shapes what architecture, in effect, is, at least for our system. That is, no matter what we say architecture is (for), architecture is (an outcome of) what we do. This is a POSIWID (the purpose of a system is what it does – Stafford Beer) kind of point, noting that what the thing does, trumps what we may think it is and ought to do.

Still, intentions influence behavior. How we conceive of architecture, influences what we do. If we think software architecture is a set of decisions, say, we might use Architecture Decision Records (or similar). If we think architecture is the organizing structure of the system, we might direct attention to diagrams or maps. If we think architecture is system design, we bring attention and intention to clarifying how they system ought to be. And so on.

In short, what we do in the name of architecture, shapes what it is; what we think it is, shapes what we do.

At any rate, what follows is an iterative elaboration towards a richer understanding of what software architecture is, to inform how we advocate approaching the architecture of systems we're design-evolving.

"There is after all [...] no point in claiming that the purpose of a system is to do what it constantly fails to do" — Stafford Beer, as quoted by David Benjamin and David Komlos

"Mr. Beck, what is software architecture?"
"Software architecture?"
replied Kent, "well, it is what software architects do."
"So then, what is an architect?"
"Hmm, 'software architect' it's a new pompous title that programmers demand to have on their business cards to justify their sumptuous emoluments."

Quote source: "What do software architects do?" Philippe Kruchten, 1992

What is it that Architects Do?

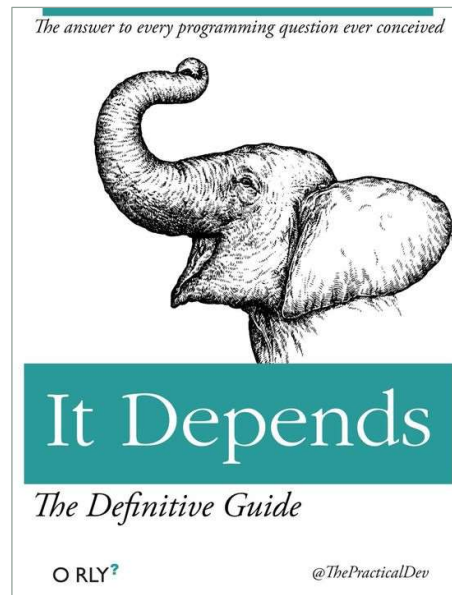
An architect is the person who says: "It depends!" to every question

Small print: A good architect tells you what it depends on.

Even smaller print: A top-notch architect* asks "What problem are we trying to solve?"

* In this case, Yvonne Lam

A little wry humor, as antidote to hubris and inappropriate certainty, and other confidence tricks.



Preamble

The "it depends" joke is used in various contexts. Another is:

A technical specialist knows more and more about less and less, until they know everything about nothing.

An architect knows less and less about more and more, until they know nothing about everything.

Behind the wry quips are insights about messy situations, extremes and balances, and context relevance. Part of what we're doing, is understanding the space, with all its forces, and interactions and "wickedness" (in the sense of wicked problems). Contextual sense and experience gives us great insights like Kelsey Hightower's heuristic guidance in the quote alongside.

First, we will explore *what* software architecture is, and then we will turn our attention to how we create software architecture and what that entails. This introduction sets the scene for the focus of this workshop on the *how* of system design and software architecting, and we will iterate through various system design views and decision sets. And finally, we'll explore what it means for architects and other system designers.



"Stick to boring architecture for as long as possible, and spend the majority of your time, and resources, building something your customers are willing to pay for."
— Kelsey Hightower

Software Architecture: What?



Ken Scambler

@kenbot@hachyderm.io

May 27



Software architecture hot tips:

- Good things are better than bad things, except when they're not
- Also nothing is good or bad
- It depends
- The answer to every question is "it depends", except for when it doesn't. It depends
- Name three things you like. You can't have them at the same time
- No.
- There are many definitions of software architecture, but none of them are correct
- There's no such thing as software architecture

tl;dr? "Name three things you like. You can't have them at the same time. " Shorter? "No."

There are many definitions of software architecture, but none of them are correct

There's no such thing as software architecture

Software Architecture: Wut?

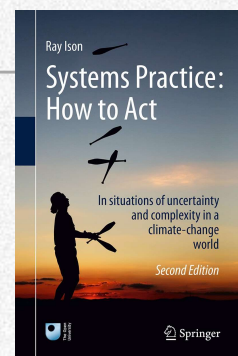
Ken Scambler's "hot tips" cover a lot of ground. Like, really a lot. We should return to them, but for now let's focus on the last pair.

Many definitions. None correct. All trying to do that hard thing, which is characterize software architecture in a way that helps the field know itself, and explain (and sell?) itself, and shape what it attempts. The emphasis shifts, as different characterizations of what software architecture is, attempt to bring focus to an area of concern, and software architecture defies these attempts – abstractions are leaky, especially when pressed. Harder when there is no such *thing* as software architecture.

There are (more or less useful and more or less complete) expressions of architectural intent, or renderings of architecture of the current system, and so forth. But... definitions do put stakes in the ground, so we can see better what we have to build on, and how we're doing relative to the landscape of concern. So we will take a characterization that gets a lot done, explore what it holds and unfolds, and where we'd like to add to or amend it.

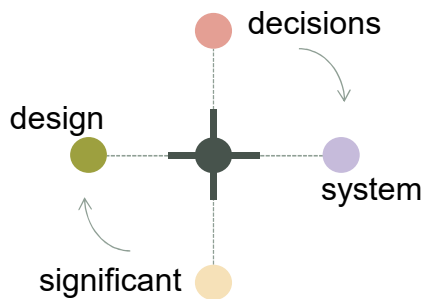
"In my experience definitions are constraining because (1) they are abstractions and thus a limited one dimensional snapshot of a complex dynamic and (2) we do not appreciate how definitions blind us to what we do when we employ a definition."

— Ray Ison



All models are wrong, but...

... some are useful? We'll use Booch's characterization as a flywheel...



“Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change.”

— Grady Booch

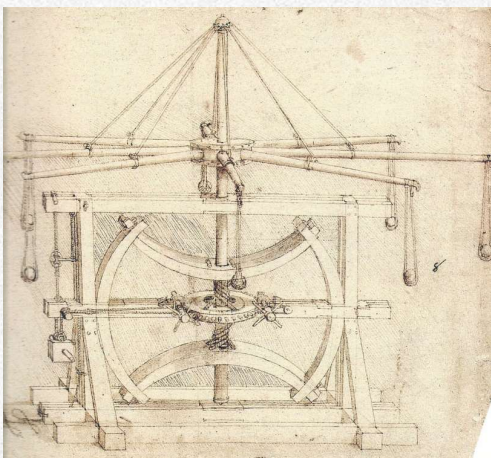
SOFTWARE
ARCHITECTURE

Introduction: What is Software Architecture?

To summarize what's ahead: each word in Grady Booch's characterization of architecture is worth exploring, because it informs what software architecture is (and hence what we're doing when we create and evolve system and software architecture). The definition provides a roadmap to this section.

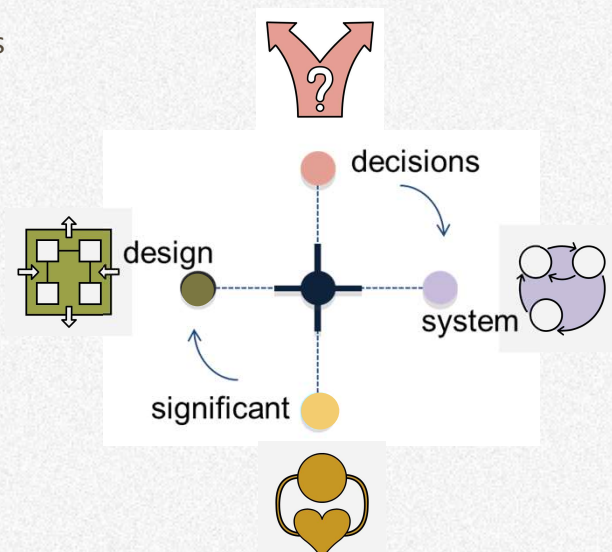
We will use that definition as a flywheel of sorts, giving impetus to an elaboration of our notion of architecture. We'll let it take us as far as it reaches, and then explore an adaptation to the last phrase, so that we include cost of change, but also other matters of architectural significance (strategic import, structural integrity and resilience, adaptability and sustainability in various terms, including economic).

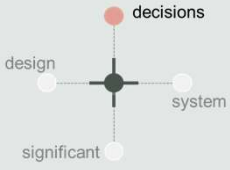
We will use a schematic “flywheel” – oriented to also bring a compass to mind – to serve as a “locator” and context provider on the slides. Since we keep cycling through the topics, the icons provide another visual cue of what topic we have returned to, to elaborate further.



A flywheel serves to store mechanical energy for later use: it is an accumulator that will deliver a surge in power output upon a drop in power


Image: Flywheel design, Leonardo da Vinci (Wikimedia commons)





- Decisions!

What are the implications?



Decisions!

“Architecture represents the significant design **decisions** that shape a system, where significant is measured by cost of change.”

— Grady Booch

Hold up. Architecture is decisions? We got this!

Decisions

Decision has an interesting etymology, deriving from “to cut off”:

decision (n.)

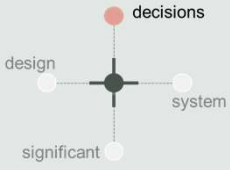
mid-15c., “act of deciding,” from Old French *décision* (14c.), from Latin *decisionem* (nominative *decisio*) “a decision, settlement, agreement,” noun of action from past-participle stem of *decidere* “to decide, determine,” literally “to cut off,” from *de* “off” (see *de-*) + *caedere* “to cut” (from PIE root **kae-id-* “to strike”).

Decisions (at least those we make intentionally) imply reasoning, and coming to a determination, weighing choices in a design space, which implies not just alternatives but (often interacting) tradeoffs weighed and constraints taken into account. There’s also matters of timing: when do we make these decisions? What do we know when, and what can we know/have confidence in? etc..


*“I love thinking about the word **DECISION** through the lens of “what am I cutting off?”*

It causes me a bit of a pause because in choosing a path of action, I’m not choosing other paths of action (at least at that time)

— Eb Ikonne



- Decisions!
- ADRs



Architecture Decisions

the template indicates the territory

Title: short noun phrase

Context: desired outcome(s) and constraints and forces at play (probably in tension)

Decision: recommended decision and supporting arguments

Alternatives: other approaches we considered

Consequences: describes the resulting situation, after applying the decision

'Are you having a moment of Deja Vu? When working on a project, look at a design or code and think, "Why did they do this? Why did they not consider option X?" I have been there. Decision Records shine here not just as a documentation tool but also to preserve the context of our design choices and why they were made.'

*— Indu Alagarsamy**

*<https://domainanalysis.io/p/document-your-product-and-software>

Architecture Decisions

So architecture entails decisions — which we can identify, make and document. An Architecture Decision Record (ADR) documents decisions in terms of: the statement of the decision, the outcome sought and the forces and other factors weighed in the making of the decision, along with consequences or implications of the decision. This template has echoes of the patterns template used by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides in their *Design Patterns* book (1994), that launched a genre (inspired by Christopher Alexander).

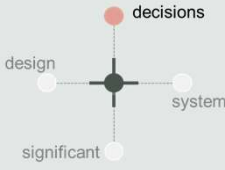
The Tyree/Akerman (IEEE paper) and Zimmerman (IBM) decision templates (that pre-dated Nygard's more simple version), also keep track of alternatives considered (but ruled out), and this is valuable too. We're persisting the reasoning that went into choices.

See also, Nat Pryce's ADR tools on Github:
<https://github.com/npryce/adr-tools>


Some examples:

- 18f TTS, <https://engineering.18f.gov/architecture-reviews/>
- Upmo, <https://upmo.com/dev/decisions/>





- Decisions!
- ADRs
- Examples



ADR Examples

6. Database

Date: 2020-09-10

Status [Approved](#)

Context

Data storage and management is a key component of the tta smarthub platform. At a fundamental level, a relational as well as NoSQL systems were considered. Because the platform's data is mainly structured and with a need for frequent queries and reports, a relational database management system was viewed as more suitable. With that in mind we looked at MySQL and PostgreSQL (Postgres) both open source and popular choices.

Decision

While both databases provide adequate storage and management, especially with updates provided by version 8 of MySQL, in the end Postgres was chosen. The main advantages of Postgres are implementations leading to better concurrency, specifically the MVCC (Multiversion Concurrency Control) without the need to use read locks, transactional ACID (Atomicity, Consistency, Isolation, Durability) support making the system less vulnerable to data corruption. Postgres also supports partial indexes, can create indexes in a non-blocking way, multiple cores, GIN/GIST indexing accelerating full-text searches.

Consequences

We believe Postgres will provide most if not all that is needed from a database. Couple of minor disadvantages of Postgres include a need for better documentation, however there are a lot of online resources for troubleshooting. Given Postgres advantages mentioned above, this is a rather minor point. Having an object relational database will fit well with the structure of the data anticipated for the TTA Smarthub Platform.

[Head-Start-TTADP / docs / adr / 0006-database.md](#)

4. Application Logging

Date: 2020-08-25

Status [Approved](#)

Context

There are a few different options for application logging.

1. Logging to file(s) used to be the norm. Logging to a file has several issues, however. You must be on the host to view the log file, or setup a system to ship the log file off the host. You must rotate log files or your host will eventually run out of disk space.
2. Logging to stdout and letting a different system handle the logging is a more modern approach to logging. In our case cloud.gov takes care of gathering logs sent to stdout/stderr. Logs can be viewed with a cloud.gov cli tool.

Decision

We will log to stdout/stderr. On development machines logs will be presented as human readable strings. In deployed environments (dev, staging and prod) logs will be formatted in JSON.

Consequences

Sending logs to stdout/stderr simplifies a lot of management tasks. We can take advantage of the logging system cloud.gov provides. On development machines having logs be human readable is important when developing/troubleshooting. JSON vs string output will be configurable with an environment variable (see ADR #3). Machine readable logs in deployed environments will allow the logs to be ingestible by log aggregation services in the future.

[Head-Start-TTADP / docs / adr / 0004-application-logging.md](#)
Nice, but...

Architecture Decision Records

Joel Parker Henderson has collected various resources around Architecture Decision Records, from templates and guidance, to links to the ADRs of various organizations, on github.

<https://github.com/joelparkerhenderson/architecture-decision-record>

Examples. <https://github.com/joelparkerhenderson/architecture-decision-record/tree/main/examples>

The Application Logging ADR of the HHS/Head-Start-TTADP project might be contrasted with the Metrics, Monitors and Alerts ADR (as an exercise, noting that the team's context and judgment factors):

- Application Logging
<https://github.com/HHS/Head-Start-TTADP/blob/main/docs/adr/0004-application-logging.md>
- Metrics, Monitors and Alerts
<https://github.com/joelparkerhenderson/architecture-decision-record/tree/main/examples/metrics-monitors-alerts#metrics-monitors-alerts>

The ADR template is vital. Still, too much of a good thing sinks itself under its own weight, so an Architecture Decision Record leaves itself begging the question — which decisions?

Many, but not all, ADRs focus on technology choices.

Example:

<https://github.com/joelparkerhenderson/architecture-decision-record/tree/main/examples/high-trust-teamwork>

README.md

Decision Record for High Trust Teamwork

Date: [Insert Date]

Participants: [Insert Names of Participants]

Decision Reach: [Unanimous/ Majority decision/ Individual Decision]

Decision Description

The decision to adopt High Trust Teamwork as a core value and belief in our team and organizational culture.

Alternatives Considered

We considered alternative approaches to teamwork, including low trust or no trust teamwork, as well as other frameworks such as cognitive diversity teams, centralized decision-making teams, and agile teams.

Benefits and Risks

The benefits of high trust teamwork include improved collaboration, better decision-making, increased motivation and engagement, higher job satisfaction and retention rates, and enhanced innovation and creativity. The risks associated with high trust teamwork include the potential for trust breaches, conflicts, and misunderstandings due to miscommunications or not adhering to the high trust principles.

Decision Outcome

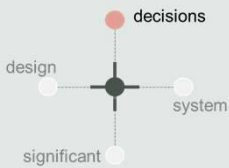
It was unanimously agreed upon to adopt high trust teamwork as the core value and belief in our team and organizational culture. We acknowledged the importance of building and maintaining trust through transparency, honesty, integrity, and respect, and recognized that this approach would help us to establish a positive and rewarding work environment.

Action Items

To implement high trust teamwork, we will take the following steps:

1. Define high trust principles and practices and communicate these to all team members
2. Establish regular feedback mechanisms and communication channels to keep the team updated on progress, challenges, and opportunities
3. Attend training on high trust teamwork and actively participate in team-building activities
4. Encourage and recognize behaviors that demonstrate high trust principles in action
5. Develop a plan to address potential trust breaches, conflicts, or misunderstandings and have a process to handle such situations effectively

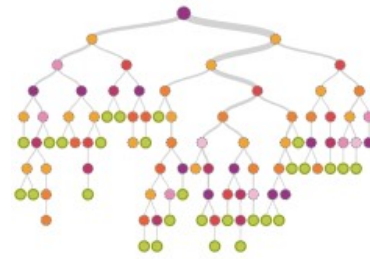
Which Decisions?



- Decisions!
- ADRs
- Examples
- Which decisions?



Decisions are central, and it is a great template, but you can just hear the captain in the cockpit yelling "pull up, pull up" — we'll run into a veritable forest of decision trees if we speed too far too fast down that runway just now. Which decisions?



So Many Decisions!

If we tried to document every decision using this template (or similar), we'd be overwhelmed, not only with the work of documenting them, but finding them. To be useful, we need to discern which decisions need this level of attention. It's in the title: Architecture Decision. But what counts as architecture? Yes, that's what we're exploring. And yes, judgment calls. But can we say more?

"All architecture is design, but not all design is architecture."
— Grady Booch

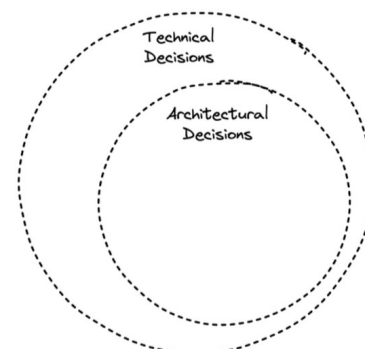
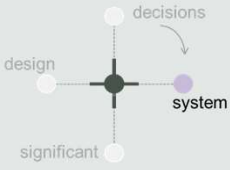



Figure 2-1: All architectural decisions are technical decisions, but not all technical decisions are architectural ones.



- Shape a system
- System?

What are the implications?



Decisions that shape a System

that shape a system!!! ... what's a system?

“Architecture represents the significant design decisions that shape a **system**, where significant is measured by cost of change.”

— Grady Booch

To address architecture (as a definition and as a design practice), we need to understand systems

What is a System?

While we generally think of cyberneticists when we think of early systems thinkers, Ernest Fernollosa's discussion in "The Lessons of Japanese Art" (1891) hits key points:

“When several things or parts, by being brought into juxtaposition, exert a mutual influence upon one another, such that each undergoes a change, and as the result of these simultaneous changes each becomes melted down, so to speak, as a new constituent of a new entity, we have synthesis... Here the parts are not left behind; they persist altogether transfigured by the organic relation into which they have entered. Such a synthetic whole is never equal to the sum of all its parts; it is that plus the newly created substance which has been formed by their union. Such a whole we cannot analyze into its parts without utterly destroying it. Abstract one of the units, and the light which irradiated it is eclipsed; it is like a hand cut off, limp and lifeless.”

Systems — wholes — become, not just “greater than the sum of” but something *other than* the parts they are made of; they give rise to emergent properties and integrated capabilities.

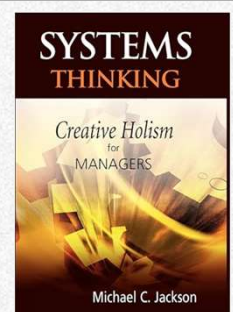
* This may be a translation or paraphrase of: “It has been said: The whole is more than the sum of its parts. It is more correct to say that the whole is something else than the sum of its parts”
Kurt Koffka, "Principles Of Gestalt Psychology," 1935

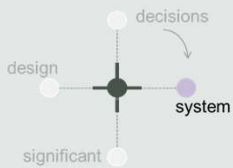
*"The whole is **other than** the sum of the parts"*

— Kurt Koffka*

"Simply defined, a system is a complex whole the functioning of which depends on its parts and the interactions between those parts"

— Michael C. Jackson





- Shape a system
- System?
- Whole that consists of parts



What's a System?

"A system is a Whole that consists of parts, each of which can affect its behavior or properties."

"The Parts of the system are interdependent"

— Russell Ackoff

Ackoff got you!



If Russ Ackoff had given a TED Talk...

SYSTEM meaning:

1. a set of connected things or devices that operate together

<https://dictionary.cambridge.org>

"The only thing added to the parts to make the whole greater than the sum of its parts is the interrelationships among them."*

— Eb Rechtin

* the whole is *other* than the sum

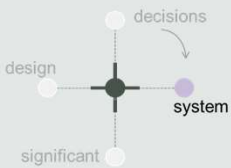
What Characterizes Systems

In this roughly 10 minute (starting at 1:24) talk (CW: limb loss), Russ Ackoff covers and illustrates key characteristics of systems. Notably, a system has properties that none of its parts have, on their own. When we take a system, decompose it into its parts, optimize the parts, and put them back together, we don't even necessarily get a working system. To see this, imagine you have the best automotive engineers in the world pick the best carburetor, the best fuel pump, distributor, and so on. Now ask them to assemble those best parts into a car. Most likely they can't because the parts don't fit, but even if they do, in all likelihood it doesn't work very well. And at any rate, we can't say anything about the properties, since they are emergent from interactions among the parts, and with the context (stopping on gravel versus pavement, etc.).

Without interrelationships, we have, as Wim Roelandts put it: "parts flying in formation, trying to be an airplane."

Obvious? Surely. Yet we need to act on this understanding. It is not enough to decompose a system into components or microservices or whatever the chunking du jour, minimizing interdependence, and proceed as if coherent systems will simply emerge from independent teams.

Whole that consists of Parts



- Shape a system
- System?
- Whole that consists of parts
- Elements and relationships
- Decomposition?

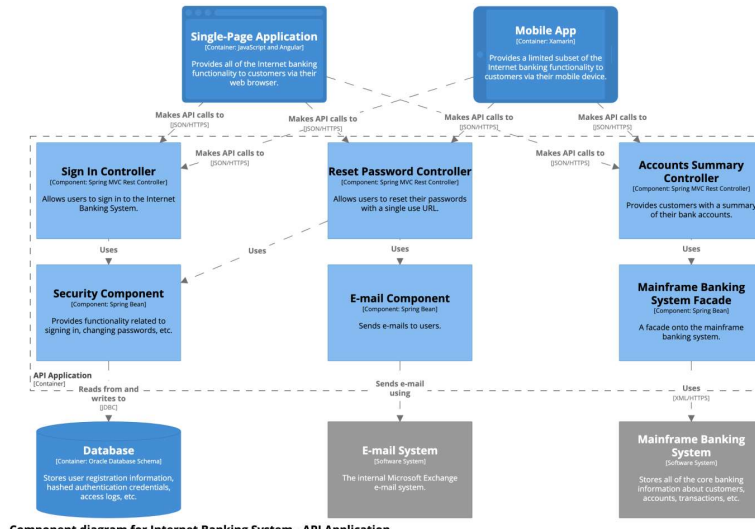


Image source: Simon Brown, <https://c4model.com>

So ... architecture is system decomposition??

Architecture is .. Block Diagrams?

And indeed, so central is modularity (and components) to architecture that when we think of expressions of software architecture that we would expect and recognize, there are the good old "block diagrams." These remain important as a means to reason about and express architecture as system structure, where elements are shown as boxes, and relationships as lines. We can use Simon Brown's C4 or UML or sysML or Archimate or something home grown, etc.. We're expressing the shaping design ideas of the system, visually.



Oliver Drotbohm • 1st
Staff 2 Engineer at VMware
6h • 🌐

The primary objective of system decomposition is to support change. To contain the effects of a change to as few as possible elements. If your decomposition strategy does not align with that, it's distracting at best and harmful at worst.



25

1 comment • 2 reposts

Going back to 1992, Perry and Wolf were defining (software) architecture as being concerned with "the selection of architectural elements, their interactions, and the constraints on those elements and their interactions." And, indeed, the contemporary go-to reference definition for software architecture (that being the definition in wikipedia from the SEI team/Clements et al book) is:

“Software architecture refers to the high level structures of a software system [...] Each structure comprises software elements, relations among them, and properties of both elements and relations.”

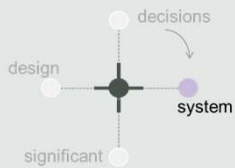
We're holding these two ideas about architecture in creative suspension — architecture is decisions about "the important stuff," and architecture is about the structure of the system.

Software architecture "includes how the system is divided into components and how the components interact through interfaces."

—Martin Fowler

"A complex system cannot be reduced to a collection of its basic constituents, not because the system is not constituted by them, but because too much of the relational information gets lost in the process."

—Paul Cilliers



- Shape a system
- System?
- Whole that consists of parts
- Elements and relationships
- Decomposition?
- Hierarchical structure?



Architecture Of Complexity

“Empirically, a large proportion of the complex systems we observe in nature exhibit hierarchic structure. On theoretical grounds we could expect complex systems to be hierarchies in a world in which complexity had to evolve from simplicity.”

— Herbert Simon

Source: “The Architecture of Complexity” by Herbert Simon, 1962

THE ARCHITECTURE OF COMPLEXITY

HERBERT A. SIMON*
Professor of Administration, Carnegie Institute of Technology
(Read April 28, 1962)

A NUMBER of proposals have been advanced in recent years for the development of “general systems theory” which, abstracting from properties peculiar to physical, biological, or social systems, would be applicable to all of them.¹ We might well feel that, while the goal is laudable, systems of such diverse kinds could hardly be expected to have any nontrivial properties in common. Metaphor and analogy can be helpful, or they can be misleading. All depends on whether the similarities the metaphor captures are significant or superficial.

It may not be entirely vain, however, to search for common properties among diverse kinds of complex systems. The idea that go by the name of cybernetics constitute, if not a theory, at least a point of view that has been proving fruitful over a wide range of applications.² It has been useful to look at the behavior of adaptive systems in terms of the concepts of feedback and homeostasis.

*The ideas in this paper have been the topic of many conversations with my colleagues, Allen Newell, George W. Cross, and several important improvements in logical content as well as editorial form. I am also indebted, for valuable comments on the manuscript, to Richard H. Meier, John R. Platt, and Warren Weaver. Some of the conjectures about the early decomposable structure of the nucleon-atom-molecule hierarchy were checked against the available quantitative data by Andrew Schone and William Wise. My work in this area has been supported by a Ford Foundation grant for research in organization and a Carnegie Corporation grant for research on cognitive processes. To all of the above, my warm thanks, and to the publisher, my appreciation.

¹See especially the yearbooks of the Society for General Systems Research. Prominent among the exponents of general systems theory are: L. von Bertalanffy, K. Boulding, R. W. Gerard, and J. G. Miller. For a more skeptical view—perhaps too skeptical in the light of the present discussion—see L. A. Simon and A. Newell, *Models: their uses and limitations*, in L. D. White, ed., *The state of the social sciences*, 66-83, Chicago, Univ. of Chicago Press, 1966.

²J. N. Wiener, *Cybernetics*, New York, John Wiley & Sons, 1948. For an imaginative treatment, see A. J. Lotka, *Elements of mathematical biology*, New York, Dover Publications, 1951, first published in 1925 as *Elements of physical biology*.

and to analyze adaptiveness in terms of the theory of selective information.³ The ideas of feedback and information provide a frame of reference for viewing a wide range of situations, just as do the ideas of evolution, of relativism, of axiomatic method, and of operationalism.

In this paper I should like to report on some things we have been learning about particular kinds of complex systems encountered in the behavioral sciences. The developments I shall discuss arose in the context of specific phenomena, but the theoretical formulations themselves make little reference to details of structure. Instead they refer primarily to the complexity of the systems under view without specifying the exact content of that complexity. Because of their abstractness, the theories may have relevance—application would be too strong a term—to other kinds of complex systems that are observed in the social, biological, and physical sciences.

In recounting these developments, I shall avoid technical detail, which can generally be found elsewhere. I shall describe each theory in the particular context in which it arose. Then, I shall cite some examples of complex systems, from areas of science other than the initial application, to which the theoretical framework appears relevant. In doing so, I shall make reference to areas of knowledge where I am not expert—perhaps as it does the whole span of the scientific and scholarly endeavor. Collectively you will have little difficulty, I am sure, in distinguishing instances based on idle fancy or sheer ignorance from instances that cast some light on the ways in which complexity exhibits itself wherever it is found in nature. I shall leave to you the final judgment of relevance to your respective fields.

I shall not undertake a formal definition of

³C. Shannon and W. Weaver, *The mathematical theory of communication*, Urbana, Univ. of Illinois Press, 1949; W. R. Ashby, *Design for a brain*, New York, John Wiley & Sons, 1957.

PROCEEDINGS OF THE AMERICAN PHILOSOPHICAL SOCIETY, VOL. 106, NO. 6, DECEMBER, 1962

467

"We find structure on all scales. In order to see how difficult it is to grasp these structures, it is necessary to look at the boundaries of complex systems, and to the role of hierarchies within them." — Paul Cilliers

"If you ask a person to draw a complex object—such as a human face—[t]he[y] will almost always proceed in a hierarchic fashion."
— Herbert Simon

Herbert Simon's Parable of the Watchmakers

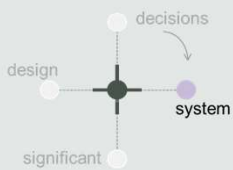
“Let me introduce the topic of evolution with a parable. There once were two watchmakers, named Hora and Tempus, who manufactured very fine watches. Both of them were highly regarded, and the phones in their workshops rang frequently -new customers were constantly calling them. However, Hora prospered, while Tempus became poorer and poorer and finally lost his shop. What was the reason?

The watches the men made consisted of about 1,000 parts each. Tempus had so constructed his that if he had one partly assembled and had to put it down-to answer the phone say-it immediately fell to pieces and had to be reassembled from the elements. The better the customers liked his watches, the more they phoned him, the more difficult it became for him to find enough uninterrupted time to finish a watch.

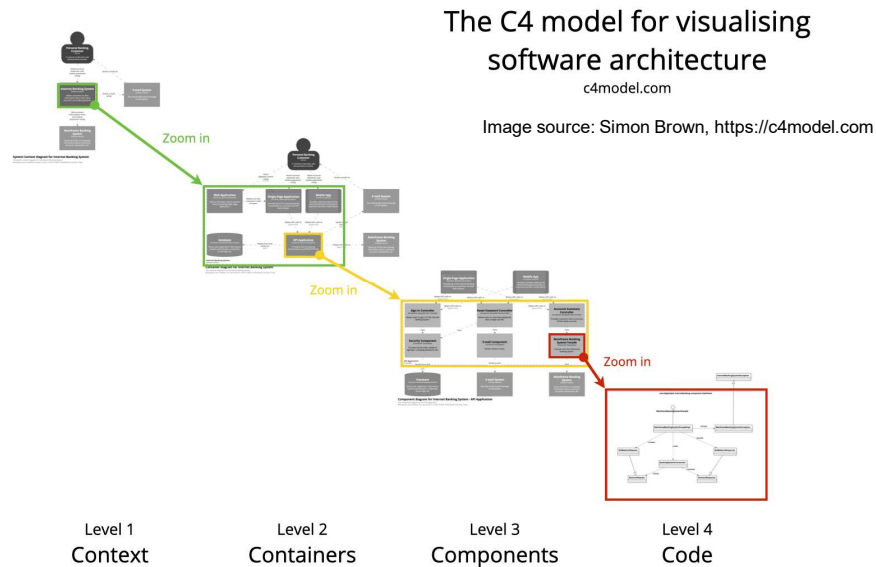
The watches that Hora made were no less complex than those of Tempus. But he had designed them so that he could put together subassemblies of about ten elements each. Ten of these subassemblies, again, could be put together into a larger subassembly; and a system of ten of the latter subassemblies constituted the whole watch. Hence, when Hora had to put down a partly assembled watch in order to answer the phone, he lost only a small part of his work, and he assembled his watches in only a fraction of the man-hours it took Tempus.”

Source: “The Architecture of complexity” by Herbert Simon

Hierarchical (de)Composition



- Shape a system
- System?
- Whole that consists of parts
- Decomposition?
- Elements and relationships
- Hierarchical structural?
- Zoom in

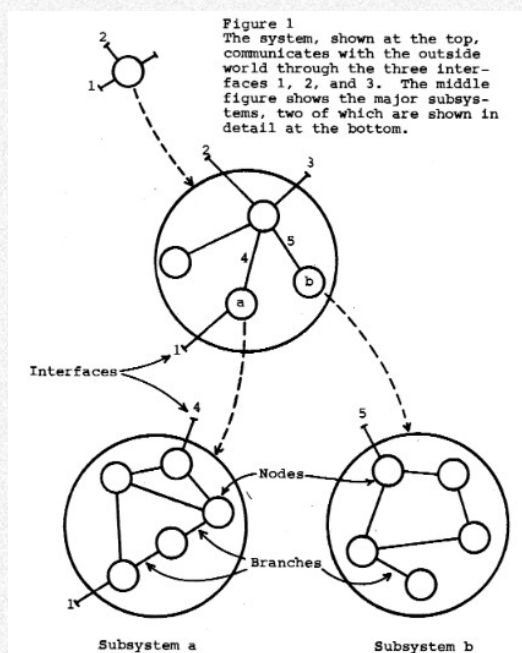


"Any system of consequence is structured from smaller subsystems which are interconnected. A description of a system, if it is to describe what goes on inside that system, must describe the system's connections to the outside world, and it must delineate each of the subsystems and how they are interconnected. Dropping down one level, we can say the same for each of the subsystems, viewing it as a system. This reduction in scope can continue until we are down to a system which is simple enough to be understood"

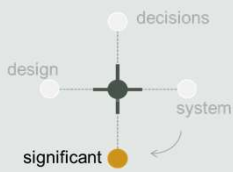
—Melvin Conway

Hierarchy: structural organization

Mel Conway, in the classic paper that articulated what became known as Conway's Law, illustrated a system at different "zoom levels": system; system composed of subsystems; and two of the subsystems in more detail. Simon Brown's C4, likewise, "zooms in" to more detailed decompositions within larger structures, from system in context, to containers, to components, to code.



Source: Melvin Conway, "How Do Committees Invent?", 1968



- Significant decisions!
- Significant?



Significant Decisions!

“Architecture represents the **significant** design decisions that shape a system, where **significant** is measured by **cost of change**.”

— Grady Booch

Which Decisions? Significant Decisions!

When I first read Martin Fowler’s “Who needs an Architect?” column, I playfully summarized it as:

Which decisions does the architect make?

Architecturally significant decisions!

What is architecturally significant?

The architect decides!

Yes, it’s a tautology. But this is an important insight for all its playfulness: judgment factors. That is, what is architecturally significant, what needs architectural attention, is a judgment call. And judgment is a matter of experience, of expertise, of wisdom. A system’s architects are in effect, those who perhaps accidentally, perhaps intentionally, perhaps both, make design-shaping decisions (in the code, too). That’s a broader set than those who play the role (with potential title) of architect. But the point of drawing attention to architecture as a focus of work and expertise, is making more of these significant decisions intentionally. With due consideration. Bringing insight and know-how, and know-what and know-when, to bear. In the making of architecture decisions, and in how we verify their effectiveness, and seek to learn and adapt to discovery of emerging or better understood needs and challenges, and possible improvements.

A matter of judgment. Can we say more?

“Architecture is a hypothesis about the future that holds that subsequent change will be confined to that part of the design space encompassed by that architecture.”

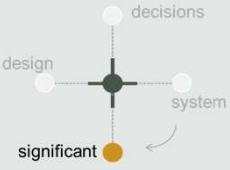
— Foote and Yoder

Source: Big Ball of Mud, Brian Foote and Joseph Yoder, 1997


<https://joeyoder.com/PDFs/mud.pdf>

“Wisdom = knowledge + experience + good judgment”

— Diana Montalion



- Significant decisions!
- Significant?



Significant? Cost of Change!

“Architecture represents the **significant** design decisions that shape a system, where **significant** is measured by **cost of change**.” effort

— Grady Booch

(ir)reversibility of decisions

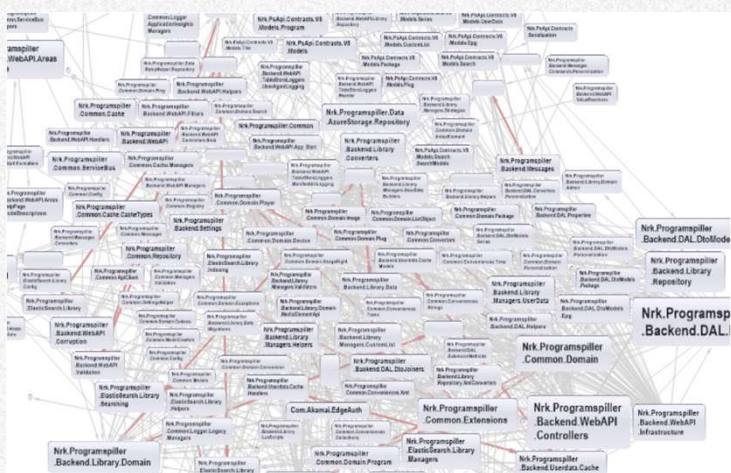
low(er(ed)) **cost of change**

high **cost of change**

Cost of Change

“Significant is measured by cost of change” has two thrusts: decisions that have high cost of change are (architecturally) significant. Also, decisions that (substantively) lower the cost of change, are architecturally significant. What makes decisions hard to reverse, is entanglement with assumptions, expectations, and other decisions and commitments (reified in code).

The opening sentence to the must-read classic “Big Ball of Mud” (by Brian Foote and Joseph Yoder, <http://www.laputan.org/mud/>), observes that the de-facto standard in software architecture is “the big ball of mud.” And a “Big Ball of Mud” (highly coupled; dependencies mean change ripples; etc.) architecture has high cost of change. What it looks like (image by Bjørn Bjartnes):



“If you think good architecture is expensive, try bad architecture”

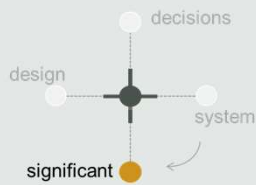
— Brian Foote and Joe Yoder

“You reach for the banana, and get the entire gorilla”

— Michael Stahl

In this sense, we’re seeking to reduce the cost of change, by reducing entanglement. This is not the only sense in which cost of change is important to architecting, but it is worth highlighting because it’s a big one.

High entanglement (coupling) leads to low comprehensibility, extensibility and evolvability, and further, is vulnerable to error propagation.



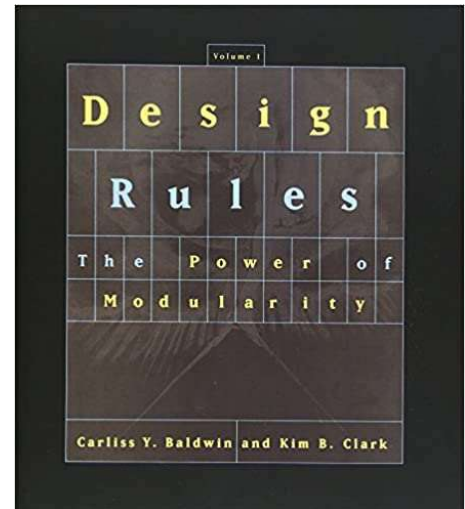
- Significant decisions!
- Significant?
- Modularity and cost of change



Modular Structure(s): ↓ Cost of Change!

- Isolate impact of change
- Isolate arenas of uncertainty and experiment and risk
- Increase reversibility, replaceability, deleteability
- Increase responsiveness and adaptability
- Scalability, scope
- Reduce complexity
- Separation of concerns (manageable cognitive load)

Interactions and coupling



Modularity: Containers for Change and Complexity

By contrast with an entangled "big ball of mud," a modular structure reduces cost of change by (and to the extent that it achieves) isolating change, shielding the rest of the system from cascading change. In a modular approach, parts of the system that are unstable, due to uncertainty and experimentation, can be shielded from other, better understood and more stable parts of the system.

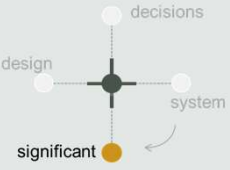
Parts can be plugged in, but removed if they don't work out, making for reversibility of decisions that don't pan out. They can be replaced with new or alternative parts, with minimal effect on other parts of the system, enabling responsiveness to emerging requirements or adaptation to different contexts.

Parts can be developed in parallel, engaging more teams.


Further, it's a mechanism to cope with, and hence harness, complexity. Partitioning the system, reduces how much complexity must be dealt with at once, allowing focus within the parts with reduced demand to attend (within the part) to complexity elsewhere in the system (caveats apply). We give a powerful programmatic affordance a handle with minimal understanding to invoke it, and can selectively ignore its internals (caveats apply). Modularity is a way we cope with our "bounded rationality" (Herbert Simon) and limit "cognitive load" placed on teams (*Team Topologies*, Skelton and Pais).

"we have to keep it crisp, disentangled, and simple if we refuse to be crushed by the complexities of our own making..." – Dijkstra

"if the features can be broken into relatively loosely bound groups of relatively closely bound features, then that division is a good thing"
– Tim Berners-Lee



- Significant decisions!
- Significant?
- Modularity and cost of change
- Change: it's the Law!



Lehman's Laws of Software Evolution

Change! It's the law!

1. Continuing Change" — [a system] must be continually adapted or it becomes progressively less satisfactory.
— M.M. Lehman

On Understanding Laws, Evolution, and Conservation in the Large-Program Life Cycle
M. M. Lehman
Imperial College of Science and Technology, London

Table 1. Five Laws of Program Evolution

1. CONTINUING CHANGE	A program that is used and that, as an implementation of its specification, reflects some other reality, undergoes continuing change or becomes progressively less useful. The change or decay process continues until it is judged more cost effective to replace the program with a recreated version.
2. INCREASING COMPLEXITY	As an evolving program is continuously changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain it or reduce it.
3. THE FUNDAMENTAL LAW (OF PROGRAM EVOLUTION)	Program evolution is subject to a dynamics which makes the programming process, and hence measures of global project and system attributes, self-regulating with statistically determinable trends and invariances.
4. CONSERVATION OF ORGANIZATION STABILITY (INVARIANT WORK RATED)	The global activity rate in a project supporting an evolving program is statistically invariant.
5. CONSERVATION OF FAMILIARITY (PERCEIVED COMPLEXITY)	The release content (changes, additions, deletions) of the successive releases of an evolving program is statistically invariant.

Lehman's Laws

Lehman's Laws recognize that complexity comes from (necessarily) adding value and adapting, AND it takes *work* and rigor to keep that complexity from being compounded by structural decay.

In particular,

1. a system must be continually adapted or it becomes progressively less satisfactory
2. as a system evolves, its complexity increases unless work is done to maintain or reduce it

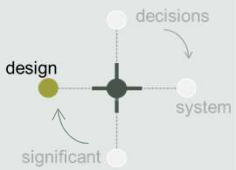
Lehman's laws of software evolution in "Programs, Life Cycles, and Laws of Software Evolution" — Meir Lehman, Proc. IEEE

Law of Stretched Systems

Sure, Lehman's Laws were from the 70's — that's still the early days of computing. What about an updated reference?


Law of Stretched Systems. Every system is stretched to operate at capacity. Improvements, regardless of aim, tend to be exploited for capacity and efficiency. (Woods & Hollnagel, Joint Cognitive Systems: Patterns in Cognitive Systems, 2006)

"the Law of Stretched Systems: every system is stretched to operate at its capacity; as soon as there is some improvement, for example in the form of new technology, it will be exploited to achieve a new intensity and tempo of activity." (David Woods and Sidney Dekker, Anticipating the Effects of Technological Change, 2000)



- Design!
- What is design?

What are the implications?



Design Decisions

What is design?

“Architecture represents the significant **design** decisions that shape a system, where significant is measured by cost of change.”

— Grady Booch

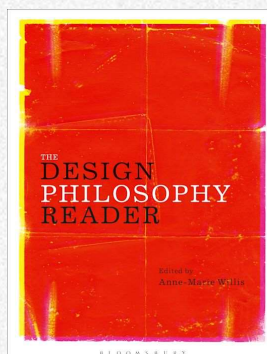
Design Decisions

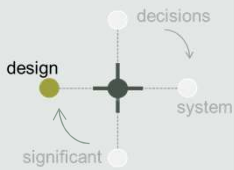
While this is useful, and points to a crucial focus of architecture (namely organizing structure and support for change), let's direct attention at *design*, and consider what else that brings into the characterization of architecture.

We might ask “What is design?” and given that design is used in various contexts, what is design in the context of architecture and systems?

“One common definition of design is to prefigure something that doesn't yet exist. This could be a totally new invention, the modification of an existing thing to a new use, or even a different way of organizing resources or people or workflow. The common feature across this variety of situations is that of seeking to bring about change, major or minor, and devising a means to do this.”

— Anne-Marie Willis





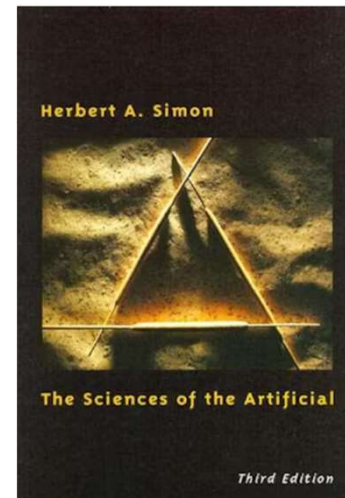
- Design!
- What is design?
- Existing to preferred



Design?

“Everyone designs who devises courses of action aimed at changing existing situations into preferred ones.”

— Herbert Simon



“The engineer, and more generally the designer, is concerned with how things ought to be — how they ought to be in order to attain goals, and to function.”

— Herbert Simon

Quote sources:

The Sciences of the Artificial,

Herbert Simon, originally published in 1968

“Ontological designing” by Anne-Marie Willis, 2006

We Design To Get More What We Want

What is design? In *The Sciences of the Artificial*, Herbert Simon notes “Everyone designs who devises courses of action aimed at changing existing situations into preferred ones.” This characterization is profound, for all its straightforward simplicity. It raises such questions as “whose preferences?”

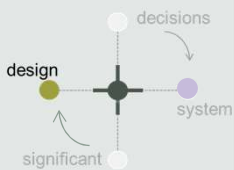
Dr. Jabe Bloom (<https://x.com/cyetaian/status/1427113866153103363>) has warned that Herbert Simon (in the quote alongside) ‘has managed to externalize the question, “what is the goal?”, “what is functional.” From here, with the goal given, Simon reduces design to the calculation of a transform from current to future state.’

We want to retain the openness of “changing existing situations into preferred ones,” where matters of preferred and “ought,” and whose preferences and oughts, are themselves among the design concerns.

“we design, that is to say, we deliberate, plan and scheme in ways which prefigure our actions and makings — in turn, we are designed by our designing and by that which we have designed”

— Anne-Marie Willis

Designing a System

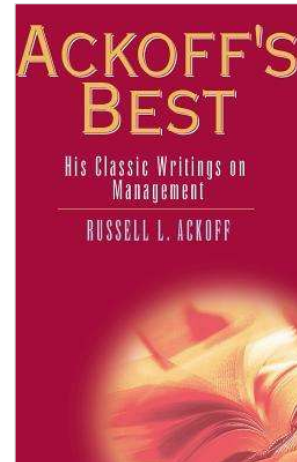


- Design?
- Existing to preferred
- Shape a system
- System?

Design... to reduce cost of change? So modules? Is that it?

“they [parts of a system] are designed to fit each other so as to work together harmoniously as well as efficiently and effectively.”

— Russell Ackoff



“When you ask what a system ought to be, then anybody who's affected has some relevant opinions. There is no such thing as an expert on an ought question. Everybody can participate.”

— Russ Ackoff

‘One popular definition of architecture is “stuff that's hard to change”. I'd argue that a good architect makes change easier— thus reducing architecture 😊’

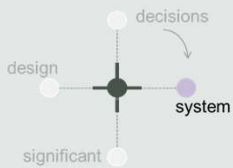
— Martin Fowler

Design of Systems

Architecture addresses the decomposition of the system into (architecturally significant) parts and composition or integration of parts into a system:

“To organize a system is to divide its labor functionally among its parts and to arrange for their coordination.” – Russell Ackoff

And yet our “ought” or “preferred” questions don’t stop at “reduce the cost of change” or “architecture is what’s hard to change, so less architecture is better” (paraphrasing Martin Fowler), nor even at “the parts fit and work together.”



- Shape a system
- System?
- Properties of the whole



A System has Properties

“The defining properties of any system, are properties of the whole, which none of the parts have. If you take the system apart, it loses its essential properties”



If Russ Ackoff had given a TED Talk...

— Russell Ackoff

“You have certain characteristics. The most important of which is life. None of your parts live. You have life. You can write. Your hand can't write. [...] An eye can't see. You can think. Your brain can't think. Therefore when the system is taken apart it loses its essential properties.”

— Russ Ackoff

“Synergy means behavior of whole systems unpredicted by the behavior of their parts taken separately.”

— Bucky Fuller

What Characterizes Systems

“A system is a set of two or more elements that satisfies the following three conditions.

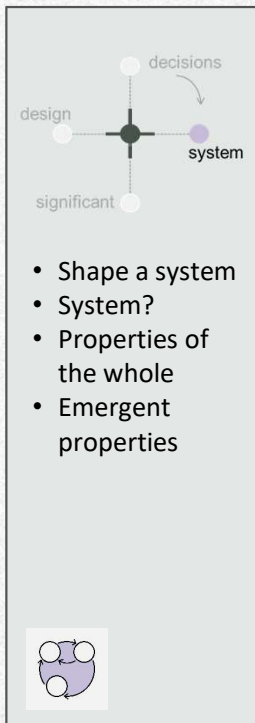
1. The behavior of each element has an effect on the behavior of the whole. [..]
2. The behavior of the elements and their effects on the whole are interdependent. [..]
3. However subgroups of the elements are formed, each has an effect on the behavior of the whole and none has an independent effect on it. [..]

A system, therefore, is a whole that cannot be divided into independent parts.”

— Russ Ackoff, *Ackoff's Best*

“You for example are a biological system called an organism, and you consist the parts. Your heart, your lungs, your stomach, pancreas, and so on, each of which can affect your behavior or your properties. [...] Therefore the way the heart affects you depends on what the lungs are doing, what the brain is doing. The parts are all interconnected. Therefore a system as a whole cannot be divided into independent parts.”

— Russ Ackoff, If Russ Ackoff had given a TED Talk



- Shape a system
- System?
- Properties of the whole
- Emergent properties

[Emergent] Properties

‘Roughly, by a complex system I mean one made up of a large number of parts that interact in a non-simple way. In such systems, the whole is more than the sum of the parts, not in an ultimate, metaphysical sense, but in the important pragmatic sense that, given the properties of the parts and the laws of their interaction, it is not a trivial matter to infer the properties of the whole’

— Herbert Simon,
The architecture of complexity, 1962

THE ARCHITECTURE OF COMPLEXITY

HERBERT A. SIMON*

Professor of Administration, Carnegie Institute of Technology
(Read April 26, 1962)

A NUMBER of proposals have been advanced in recent years for the development of "general systems theory" which, abstracting from properties peculiar to physical, biological, or social systems, would be applicable to all of them. We might well feel that while the goal is laudable, systems of such diverse kinds could hardly be expected to have any nontrivial properties in common. Metaphor and analogy can be helpful, or they can be misleading. All depends on whether the similarities the metaphor captures are significant or superficial.

It may not be entirely vain, however, to search for common properties among diverse kinds of complex systems. The ideas that go by the name of cybernetics constitute, if not a theory, at least a point of view that has been proving fruitful over a wide range of applications.¹ It has been useful to look at the behavior of adaptive systems in terms of the concepts of feedback and homeostasis,

and to analyze adaptiveness in terms of the theory of selective information.² The ideas of feedback and information provide a frame of reference for viewing a wide range of situations, just as do the ideas of evolution, of relativism, of axiomatic method, and of operationalism.

In this paper I should like to report on some things we have been learning about particular kinds of complex systems encountered in the behavioral sciences. The developments I shall discuss arose in the context of specific phenomena, but the theoretical formulations themselves made little reference to details of structure. Instead they refer primarily to the complexity of the system under view without specifying the exact content of that complexity. Because of their abstractness, the theories may have relevance—application would be too strong a term—in other kinds of complex systems that are colored in the social, biological, and physical sciences.

In recounting these developments, I shall avoid technical detail, which can generally be found elsewhere. I shall describe each theory in the particular context in which it arose. Then, I shall cite some examples of complex systems from areas of science other than the initial application, in which the theoretical framework appears relevant. In doing so, I shall make reference to areas of knowledge where I am not expert—perhaps not even literate. I feel quite comfortable in doing so before the members of this society, representing as it does the whole span of the scientific and scholarly endeavor. Collectively you will have little difficulty, I am sure, in distinguishing instances based on idle fancy or sheer ignorance from instances that cast some light on the ways in which complexity exhibits itself wherever it is found in nature. I shall leave to you the final judgment of relevance in your respective fields.

I shall not undertake a formal definition of

*N. Wiener, *Cybernetics*, New York, John Wiley & Sons, 1948. For an imaginative treatment, see A. J. Lotka, *Elements of Mathematical Biology*, New York, Dover Publications, 1956, first published in 1925 as *Elements of Mathematical Biology*.

²C. Shannon and W. Weaver, *The mathematical theory of communication*, Urbana, Univ. of Illinois Press, 1949; W. R. Ashby, *Introduction to the theory of machines*, New York, John Wiley & Sons, 1956.

PROCEEDINGS OF THE AMERICAN PHILOSOPHICAL SOCIETY, VOL. 106, NO. 5, DECEMBER, 1962

467

"The emergent is unlike its components insofar as these are incommensurable, and it cannot be reduced to their sum or their difference."
— G.H. Lewes (in 1875)

"Consciousness is an emergent property of the brain that cannot be predicted by examining a neuron."
— Paul Cilliers

Emergence

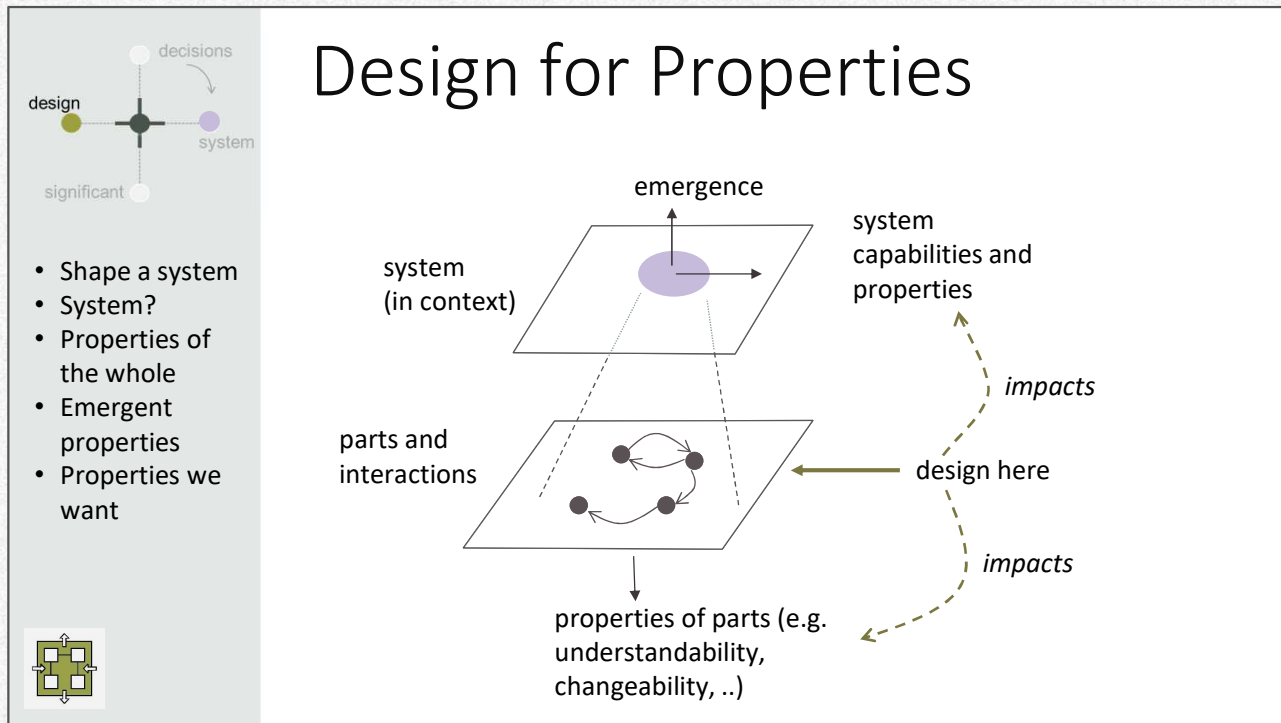
The capabilities and properties of the system, emerge from their components (parts) and their interactions. Jabe Bloom notes:
"Emergence can be of 2 types, regular and novel:

- An example of regular emergence is a BZ reaction. This will regularly emerge from a certain chemical reaction
- Novel emergence is the creation of new information/structure/systems... it unfolds over time. Time IS essential part of the context. Another way to say this is it takes time"

Paul Cilliers: "This is not the same as saying that complex systems are chaotic. Emergence is not a random or statistical phenomenon. Complex systems have structure, and, moreover, this structure is robust."

Emergent Properties

Richard Cook: "Safety is an emergent property of systems; it does not reside in a person, device, or department of an organization or system. Safety cannot be purchased or manufactured."



Patrick Hoverstadt on emergence:

"The motorbike as a whole has the property of speed, but take it to pieces and not only do none of the components have the property of speed on their own, if you hunt through the components, you will not find any 'speed component'. It isn't a component, isn't a thing in its own right and it isn't a property of any of the bits. The bike only has the property of speed once it is integrated into a system. [...] Systems engineering as a discipline is all about what's involved in designing parts so they do integrate so you get the emergent properties - like speed - that you want.

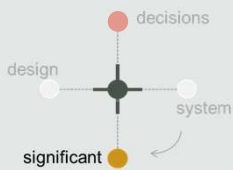
So far so commonplace, why then the mystery? I think partly because of the disconnect between a commonplace tangible measurable and, in the case of engineers, planned-for property like speed and the intangible nature of where it comes from. You can measure the speed of a motorbike, you can feel it and yet you can't see 'it' because it isn't a tangible thing. Take the bike apart and there's nothing but a pile of bits, there is no speed. There is an undeniably weird aspect to emergence; it's there, it's normal, it's tangible and in the case of a motorbike, for many bikers, speed is everything

and at the same time it is in a literal sense nothing. Emergence nearly always can play that trick of the mind on you - a strange shapeshifting, harlequin that despite its elusiveness is the point of everything.

For the systems engineering professors, their discipline is that integration to produce emergence and at a mechanistic level, this is the explanation of emergence. As Smuts put it: "A whole, which is more than the sum of its parts, has something internal, some inwardness of structure and function...some internality of nature that constitutes that 'more?'" The "inwardness of structure and function" is exactly what the systems engineers work with. It's about how the bike's engine connects to the gearbox connects to the back wheel connects to the road. Connect all that up differently or in a different order and the parts may stay the same, but the emergent will be totally different."

Source: Patrick Hoverstadt, *The Grammar of Systems: From Order to Chaos and Back*

Significant Decisions!



- Significant decisions!
- Significant?

“Architecture represents the
significant design decisions
that shape a system, where *that part!!*
significant is measured by
cost of change.”

— Grady Booch



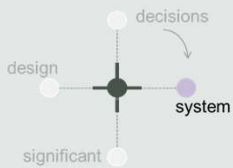
Decisions that Shape a System

While decisions that are hard to change is an important heuristic for identifying architectural significance, “significant design decisions that shape a system” has more (or other) direct implications for characterizing architecture decisions. We design (bring attention and intention and expertise) to achieve more the capabilities and system properties we want (where “we” and “properties” and “want” are design matters, too).

Lines of code don’t tell as about a system capability or property. We can’t tell by looking at a chunk of code, whether the system is changeable or performance is acceptable. (We do move up and down the scopes of the system, as we seek out what impacts performance — looking for a bottleneck, say.) Capabilities at one scope, rely on elements and interactions at more narrow scope, and system shaping at one level, impacts, and is impacted by, system shaping at other scopes.

Let’s return to characterizing systems, and draw out further implications for architecture when we direct our attention at system shaping decisions.

What else are we directing design attention and intention at, when we’re making decisions that shape the system of concern?



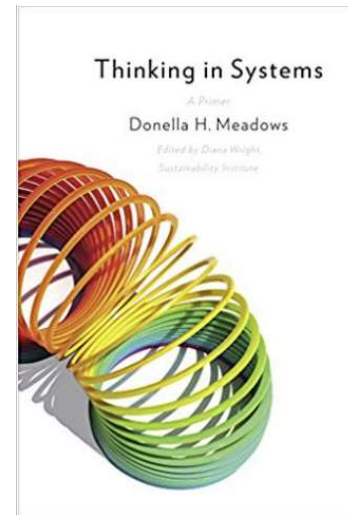
- Shape a system
- System identity and purpose



System Identity and Purpose

“a system must consist of three kinds of things: elements, interconnections, and a function or purpose.”

— Donella Meadows



“Theoretically, a system is defined as a set of components that act together as a whole to achieve some common goal, objective, or end. The components are all interrelated and are either directly or indirectly related to each other. So, a chemical plant, an airplane, an automobile, transportation in general, county government, and a television set are examples of systems. They all consist of a set of components working together to achieve a common goal. [...] A purpose is basic to the concept.”

— Nancy Leveson

Source: Nancy Leveson, “White Paper on How to Perform Hazard Analysis on a System-of-Systems,”
<http://sunnyday.mit.edu/SOS-hazard-analysis.pdf>

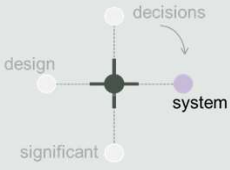
Identity, Coherence and Purpose

Systems don’t exist in isolation; they play some role or perform some function(s) that makes them viable in larger contexts. Purpose and coherence give the system distinct identity. Systems that are coherently organized, “have the quality of forming a unified whole.” From a design point of view, we’re also interested in coherence — the system has congruity (things fit together in a way that makes sense), consistency, conceptual integrity.


System integrity includes fit to purpose, fit to context, internal fit, and fitness. This brings in the notion of fitness functions, and design within design envelopes (where falling outside the design envelope, is a failure condition).

“A system is a whole that is defined by its function(s) in a larger system (or systems) of which it is a part and that consists of at least two essential parts, parts without which it cannot perform its defining functions.”

— Russ Ackoff



- Shape a system
- System identity and purpose
- Boundaries delimit



System Boundary

“In order to be recognisable as such, a system must be bounded in some way. However, as soon as one tries to be specific about the boundaries of a system, a number of difficulties become apparent. For example, it seems uncontroversial to claim that one has to be able to recognise what belongs to a specific system, and what does not. But complex systems are open systems”
— Paul Cilliers

International Journal of Information Management, Vol. 5, No. 3 (June 2001) pp. 135-147 © Inquest College Press

Boundaries, Hierarchies and Networks in Complex Systems

PAUL CILLIERS
Department of Philosophy
University of Stellenbosch
Stellenbosch 7600
South Africa
Pp@iisat.sun.ac.za

ABSTRACT
Models used in the understanding of complex entities, like organisations, are problematic in several respects. After an introductory discussion of this problem, this paper addresses the problems associated with the boundaries of complex systems, argues that although boundaries do exist, they have a peculiar status. Namely, it is argued that although boundaries exist, no separate part of the structure of complex systems, they are not clearly defined or 'sealed' as is often assumed. Hierarchies should also be regarded as problematic in a similar manner. Finally, the usefulness of network models are investigated. The conclusion is that although network models have a structure similar to that of complex systems, they are subject to the same limitations as models of complexity are faced with. A few implications for our understanding of organisations are mentioned.

Keywords: Complexity, hierarchies, boundaries, modelling, networks, organisations

Complexity

Complexity theory has been a bright new star in the academic firmament for a while now. It is being pursued eagerly in a number of disciplines (see Thrift 1999), generally with a fair amount of hype. Why the enthusiasm, and more particularly, why is there so much of it in the organisational sciences? My suspicion is that the reason has a lot to do with the hope that we are finally onto a method that will improve our understanding of, and therefore our control over complex systems like organisations. The argument may go like this: if we pay enough attention to flat hierarchies, networks of interaction, non-linearity and emergence, we may finally be able to develop a general theory of complex organisations. This will, of course, be a much sought after management tool, and it should come as no surprise that so many are looking for it. It should also not be a mystery that the Santa Fe style of approaching the problem – lots of chaos theory and mathematics – should be the most popular. We want to predict the behaviour of complex systems, and for that we need good models.

Of course complexity theory did not appear on the scene without antecedents. In many ways it is a continuation of what was done in cybernetics, general systems theory and chaos theory. These disciplines also generated lots of hype – and lots of results, of course – but could never quite deliver the theories and tools required for a general theory of complexity. There are a number of reasons for this, but two related reasons. I think, are central: they did not pay enough attention to the historical nature of complex systems, and consequently, did not pay enough attention to the radically contingent nature of a complex system. Complexity was taken to be symmetrical in time, a point of view no longer tenable after the work of Prigogine. (See also Despeignes 1997: 138, and Emmeche 1997: 48, 50).

On Boundaries

Paul Cilliers:

“In order to be recognisable as such, a system must be bounded in some way. [...] But complex systems are open systems where the relationships amongst the components of the system are usually more important than the components themselves. Since there are also relationships with the environment, specifying clearly where a boundary could be, is not obvious. Boundaries are simultaneously a function of the activity of the system itself, and a product of the strategy of description involved. [...] An overemphasis on closure will also lead to an understanding of the system that may underplay the role of the environment. However, we can certainly not do away with the notion of a boundary.”

Milan Zeleny:

“These boundaries do not separate but intimately connect the system with its environment. They do not have to be just physical or topological, but are primarily functional, behavioral, and communicational.”

Paul Cilliers:

“We often fall into the trap of thinking of a boundary as something that separates one thing from another. We should rather think of a boundary as something that constitutes that which is bounded. This shift will help us to see the boundary as something enabling, rather than as confining.”

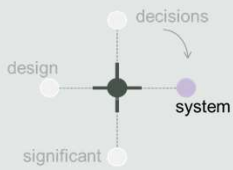
Donella Meadows:

“There are no separate systems. The world is a continuum. Where to draw a boundary around a system depends on the purpose of the discussion.”

“They mark the boundary of the system diagram. They rarely mark a real boundary, because systems rarely have real boundaries. Everything, as they say, is connected to everything else, and not neatly. There is no clearly determinable boundary between the sea and the land, between sociology and anthropology, between an automobile’s exhaust and your nose. There are only boundaries of word, thought, perception, and social agreement—artificial, mental-model boundaries.”

Source: “Boundaries, Hierarchies and Networks in Complex Systems,” Paul Cilliers

And: *Thinking in Systems*, Donella Meadows



Boundaries

- Shape a system
- System identity and purpose
- Boundaries delimit
- Boundaries as ideas, and promises



“There was a wall. It did not look important. It was built of uncut rocks roughly mortared. An adult could look right over it, and even a child could climb it. Where it crossed the roadway, instead of having a gate it degenerated into mere geometry, a line, an idea of boundary. But the idea was real. It was important. For seven generations there had been nothing in the world more important than that wall. Like all walls it was ambiguous, two-faced. What was inside it and what was outside it depended upon which side of it you were on.”

— Ursula K. Le Guin, *The Dispossessed*



More On Boundaries

Boundaries contain. Cells have cell membranes and cell walls. Animals have skin. Animal farms have perimeter enclosures. ... A car is distinct from its driver? Except that some responsibilities are shared.

“Ecotones are where two ecosystems converge, such as coastline, the edge of a forest, or a reed bed. They are transition areas between two habitats, where two biological communities meet and integrate.” (Tom Geraghty)

Boundaries are an important idea in systems. Sometimes this idea is reified as something physical or at least communicable, like a contract or promise. Sometimes it's more a transitional area, or an idea of some kind of separation. It could be a transitional zone created by nature's fluctuations. It might be sustained by contract and governance thereof, and/or maintained by convention and social mores.

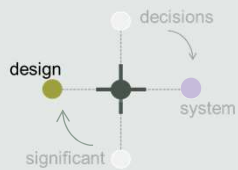
At some level, systems create and maintain boundaries as a mechanism to preserve (internal) coherence (and so emergent “wholeness”) and viability (allowing interactions across the boundary, to bring sustaining energy into the system, and to enable it to play its role in larger networks of relationships and interactions).

“Boundaries are simultaneously a function of the activity of the system itself and a product of the strategy of description involved”

— Paul Cilliers

“I use the word system to refer to any collection of elements that, through preferential interactions between them, generate a boundary with respect to other elements with which they can also interact in such a way that a totality results”

— Humberto Maturana



- Shape a system
- Design across boundaries



Boundaries by Design

System design is contextual design — it is inherently about boundaries (what's in, what's out, what spans, what moves between), and about tradeoffs. It reshapes what is outside, just as it shapes what is inside.

Not simply changes in interface and style; capabilities have been moved from driver to car



Images <https://stock.adobe.com/> (free trial)

Boundaries as Design Concerns

Whether we're talking about whole systems or abstractions within them, the notion of boundary (and the identity it shapes, and the consequences for relationships within and across boundaries) is a central one for us, as we (co-)design and co-evolve software-intensive and sociotechnical systems.

The system (of interest) plays some role in larger interacting systems of systems (variously identified, including value networks, environments, and ecosystems), and interfaces enable (and constrain and shape) interactions with the system.

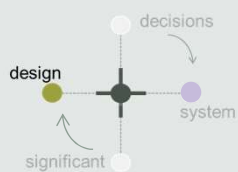
So we're interested in the larger (eco)system(s) our system fulfils a purpose within. We explore beyond the boundaries of the system we're designing, thinking about relevance, and what makes sense to explore and understand, as we shape the identity and purpose and capabilities and properties of the system we're design-evolving.

The matter of boundaries (or what is the focal system, really) is non-trivial, even when we might be tempted to think our code delineates that

boundary. For example, we might want our software-intensive system to be resilient (more than robust and reliable). For the kinds of systems we create, we might view adaptive capacity as being afforded by the larger socio-technical system that is design-evolving the system of interest. That is, for the system to be resilient and responsive to changes in the environment or context, we might draw our system boundary to include SREs and incident response on the one hand, and to include design learning, system adaptation and CI/CD capabilities on the other.

On the slide, we're indicating the evolution of the car as a system, where more capabilities have been shifted from driver to car (e.g. antilock brakes, cruise control, etc.) and this is indicated (suggested, but not fully evidenced) by shifts at the "user interface."

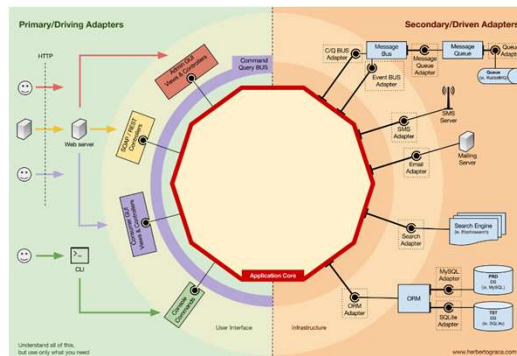
At any rate, part of system shaping is this shaping of system capabilities (derived from its purpose, but also evolving its purpose), as is designing how these capabilities will be afforded to its contexts (of use and value contribution).



- Shape a system
- Design across boundaries
- Design at the boundary

System Boundary

The Hexagonal (or Ports and Adapters) Architecture pattern, separates interactions at the system boundary from the core of the system



<https://herbertograca.com/2017/11/16/explicit-architecture-01-ddd-hexagonal-onion-clean-cqrs-how-i-put-it-all-together/>

System Boundary

The system boundary determines, and is determined by, interactions at, and across, the system boundary. For software intensive systems, we're ever balancing

- flexibility in what (capabilities or services, and properties) the system can offer its environment (users and their organization(s), other systems, "the business" and its various stakeholders, larger social contexts, etc.) and
- control, at some level, so the system meets its promises, and expectations in interactions with other systems (socio-techniocal, socio-economic, socio-political, etc.), sufficiently.

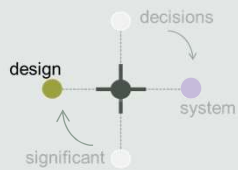
As we're determining the system boundary, we're making decisions about system capabilities and how to enable access (or expose) them. These capabilities define and are defined by the system identity or purpose. (That is, identity ripples up to the system level based on emergent capabilities, and identity or purpose is a shaping consideration, as we're exploring and designing what capabilities and properties we will build/evolve.). Users interact with capabilities via the UI (as designed and implemented) and external developers via APIs (platform design); and our system depends on and interacts with capabilities provided by other systems, etc.

We'll discuss Hexagonal Architecture in the architecture styles and patterns section. Here, we are using it as an example of boundary design. (The Hexagonal Pattern may be used at various scopes – system, subsystem or service, etc.) In the Hexagonal Pattern, interactions at the system boundary are separated from the core application logic via ports and adapters, and these are dedicated to maintain a further separation of concerns at the boundary. This separation supports change resilience (helping retain adaptive capacity, by partitioning and managing the impacts of change).

"Both the user-side and the server-side problems actually are caused by the same error in design and programming — the entanglement between the business logic and the interaction with external entities"

— Alistair Cockburn

<https://alistair.cockburn.us/hexagonal-architecture/>



- Shape a system
- Design across boundaries
- Design at the boundary
- Boundaries within the system



Boundaries within the System

Component design is contextual design — it is inherently about boundaries (what's in, what's out, what spans, what moves between), and about tradeoffs. It reshapes what is outside, just as it shapes what is inside.

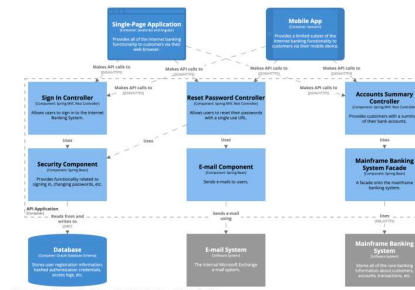


Image source: Simon Brown, <https://c4model.com>

Boundaries within the System

We've mentioned systems composed of elements (which may be systems, subsystems, components or modules), and this matter of boundaries and boundary as design concern comes up within the system too. That is, it is, in a sense, fractal. When it comes to component boundaries, we know this as interface design, but it's about the purpose of the system component and the capabilities or responsibilities of that component and how those are accessed, how the component handles surprises at the boundary, and what its dependencies are.

The repeated text on the slide, with a one word shift, serves to emphasize this fractal process. Within the system, we may talk about factoring and refactoring, but conceptually we're reminding ourselves that components have a role to play in the system, and if we change that role and its associated responsibilities or commitments, that has implications for other components and the system.

One approach comes from Domain Driven Design, where we look to the boundaries in the domain, to indicate boundaries both within the (software) system and within the (software development) organization design-evolving the system (gesturing in the direction of Conway's Law). More on all of this in a later section, of course.



Michael Plöd @bitboss · Nov 13, 2022

5. With (domain) modeling we need to find boundaries within which teams can make a high percentage of decisions without having to coordinate with others. Cross-team coordination will always slow you down, no matter what fancy tech you will use later on.

1 16 155



Michael Plöd @bitboss · Nov 13, 2022

6. You want to align domain modeling boundaries, with team boundaries and module boundaries in your software. This sounds straightforward but it isn't because you will realize that your first takes in the modeling may not turn out to be the best ideas.

The Blame Game

In *The Fifth Discipline*, Peter Senge describes the **Beer Game**. Invented by Jay Forrester at MIT in the 1960s, the Beer Game teaches systems thinking by demonstrating, through experience, the nonlinear nature of change in a system and how systemic outcomes are largely caused not by external "market pressures" but by the people in the system.

The game is simple to play but difficult to win. Players are split into four groups across the beer distribution chain. On one end is the Retailer team; they own a popular shop selling craft beer and kombucha. On the other end of the chain is the Brewery team, a small group of friends who brew cranberry craft beer. In between, the Distributors order beer from the breweries and sell it to Wholesalers, who keep the retailers supplied.

The teams communicate orders, Retailer → Wholesaler → a weekly order form. The game begins with an event: Taylor craft beer at the (American football) Super Bowl. Sudden The Beer Game players take turns ordering beer each week. Millions of people, from college students to experienced C Game. Most of those people lose.

They lose and meet Retailer's

Learning Systems Thinking
Essential Nonlinear Skills and Practices for Software Professionals



always a factor in systems! The Retailer doesn't know this, they just know their orders aren't being filled. So they increase their order, hoping to fix their problem. In frustration, each team along the chain, each week, orders more beer than they need. Until midway through, when too much beer starts flowing back downstream.

In the end, the customers have moved on to Taylor's new favorite beer, and the Retailer has a pile of unsold cranberry beer.

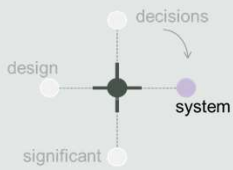
Structure produces behavior.

The hierarchical and linear structure of the ordering system encourages teams to play as four siloed parts. Each part operates from its own perspective. (We are out of beer!) The structure offers no dynamic information, shared across the system, in real time. Rather than operate as parts of one system, they operate as dissociated parts.

If the teams redesign their communication structure, they can win. But they don't.

Long-standing mental models about how business works, like perceptions about the competitive, dog-eat-dog world, also influence the teams' behavior. Even when players are given hints, told how to order optimally, they still make decisions based on their trust, or lack thereof, of other teams in the chain.

When the game ends and players are asked "What went wrong?" they don't blame the systemic structures or mental models. (Spoiler alert: that is why the system fails.) Instead, they blame one another.



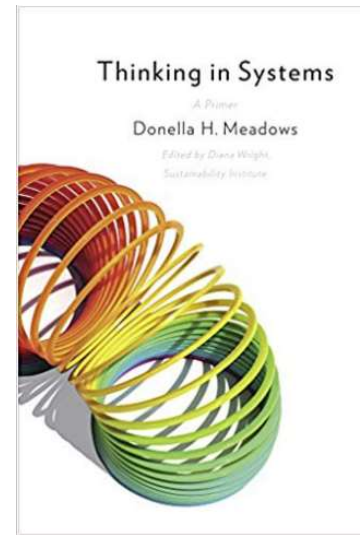
- Shape a system
- System?
- Properties of the whole
- Emergent properties
- Coherence



System Integrity

“A system is an interconnected set of elements that is **coherently** organized in a way that achieves something”

— Donella Meadows



“a system must consist of three kinds of things: elements, interconnections, and a function or purpose.”

— Donella Meadows

“A system is a whole that is defined by its function(s) in a larger system (or systems) of which it is a part and that consists of at least two essential parts, parts without which it cannot perform its defining functions.” — Russ Ackoff

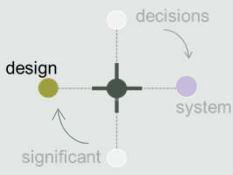
Coherence and Purpose

While we generally think of cyberneticists when we think of early systems thinkers, Ernest Fernollosa’s discussion in “The Lessons of Japanese Art” (1891) hits key points:

“When several things or parts, by being brought into juxtaposition, exert a mutual influence upon one another, such that each undergoes a change, and as the result of these simultaneous changes each becomes melted down, so to speak, as a new constituent of a new entity, we have synthesis... Here the parts are not left behind; they persist altogether transfigured by the organic relation into which they have entered. Such a synthetic whole is never equal to the sum of all its parts; it is that plus the newly created substance which has been formed by their union. Such a whole we cannot analyze into its parts without utterly destroying it. Abstract one of the units, and the light which irradiated it is eclipsed; it is like a hand cut off, limp and lifeless.”

Coherence and purpose, give the system distinct identity. Systems that are coherently organized, “have the quality of forming a unified whole.” From a design point of view, we’re also interested in coherence in the sense that it makes sense, it hangs together in a way that has congruity, consistency, conceptual integrity.

System integrity includes fit to purpose, fit to context, internal fit, and fitness. This brings in the notion of fitness functions, or design within design envelopes (outside the design envelope, is a failure condition).

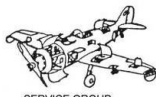


- System?
- Properties of the whole
- Coherence
- System integrity


System Integrity

We know it by its absence, like absence of balance


- Conceptual and design integrity (requisite cohesion in the context of requisite variety, ...)
- Structural integrity (resolves forces; in contexts of complexity, co-evolution, ...)
- Organization integrity (ethics, ...)




SERVICE GROUP



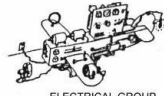
EQUIPMENT GROUP




ARMAMENT GROUP




WING GROUP



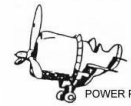
ELECTRICAL GROUP



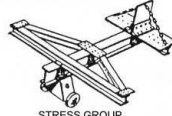
AERODYNAMICS GROUP



EMPENNAGE GROUP



POWER PLANT GROUP



STRESS GROUP

"Dream Airplanes" by C.W. Miller, Design Engineer at Vega Aircraft Corporation

"The essence of systems is relationships, interfaces, form, fit and function."

"The essence of architecting is structuring, simplification, compromise and balance."

— Eberhardt Rechtin

"The most important thing to remember about unity is — that there is no such thing. There is only unifying."

— Mary Parker Follett*

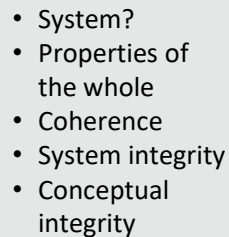
Integrity, Coherence and Purpose

To reiterate: From a design point of view, we're also interested in coherence, congruity and consistency — properties that have to do with conceptual integrity. Balance, too — the illustration indicates that overemphasis on any subset of stakeholder concerns and system properties they care about, unbalances the system; disturbs fit.

By counterexample, a failure-prone system has compromised integrity (hat tip: Arielle Paris). System integrity strives not just for internal integrity, but integrity in interactions with other systems: "When one complex system, with all its interactions, takes out other complex systems, you quickly get an avalanche of other failures" (quote from the pilot of Quantas Flight 32). We seek to balance building responsiveness and adaptive capacity and designing systems that don't fail in ways that are catastrophic.

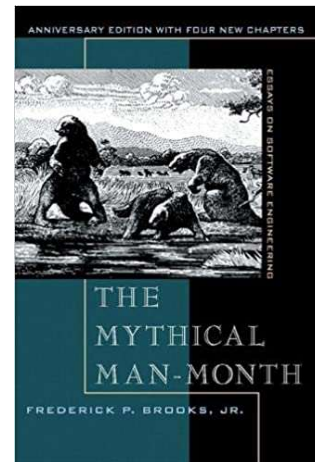
Structural integrity goes beyond conceptual integrity to include properties like reliability and robustness and recovery. System integrity would include resilience and sustainability, or adaptive capacity and coping mechanisms to deal with failures and with context shifts. Often we rely on people in the socio-technical system to add this capacity. Integrity is an ongoing project, bending the arc of the system towards resilience and integrity, recognizing that given complexity, uncertainty and change, we never reach some "ultimate" integrity.

* MPF, Co-Ordination, in "Freedom and Co-ordination"



“I will contend that conceptual integrity is the most important consideration in system design.”

– Fred Brooks



Architecture and Conceptual Integrity

According to Charles Betz (who researched this in writing his book), the first published use of architecture in a computing setting, was Fred Brooks in 1962:

“Computer architecture, like other architecture, is the art of determining the needs of the user of a structure and then designing to meet those needs as effectively as possible within economic and technological constraints.

Architecture must include engineering considerations, so that the design will be economical and feasible; but the emphasis in architecture is upon the needs of the user, whereas in engineering the emphasis is upon the needs of the fabricator." — Fred Brooks, "Architectural philosophy," 1962.

There already, Fred Brooks emphasized the importance of conceptual integrity:

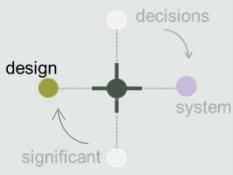
“The universal adoption of several guiding principles helped ensure the conceptual integrity of a plan whose many detailed decisions were made by many contributors.”

And Sharp, at the NATO Conference in Software Engineering in 1969:


“I think that we have something in addition to software engineering: something that we have talked about in small ways but which should be brought out into the open and have attention focused on it. This is the subject of software architecture. [...] Parts of OS/360 are extremely well coded. Parts of OS, if you go into it in detail, have used all the techniques and all the ideas which we have agreed are good programming practice. The reason that OS is an amorphous lump of program is that it had no architect. Its design was delegated to a series of groups of engineers, each of whom had to invent their own architecture. And when these lumps were nailed together they did not produce a smooth and beautiful piece of software.”

Conceptual integrity unifies the design; it gives the design ideas coherence – fit to purpose, fit to context, and fit to form a system. One that doesn't seem brute forced or unnaturally wrangled into a "frankenstein" whole.



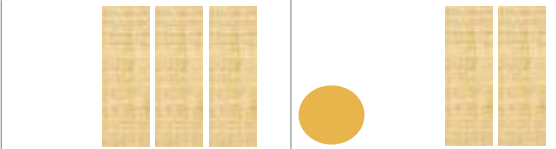



- System?
- Properties of the whole
- Coherence
- System integrity
- Conceptual integrity
- Fitness of Things




The Fitness of Things

- Artistry of Engineering: an innate sense of the fitness of things

Gordon Glegg design lecture, Department of Engineering, University of Cambridge



Gordon Glegg Design Lecture
https://www.youtube.com/watch?v=ezCp3Vy_01k&t=208s

Gordon Glegg on the Fitness of Things

"Now the artistry of engineering is an innate sense of the fitness of things. And let me try and describe by a rather disreputable example what I mean. It is something that commends itself to you without necessarily a rational background — you just say immediately instinctively, that's the way to do it.

There was a director of a firm up in Scotland which made an immense amount of plastic floor covering and many million pounds of it was stored in the warehouses there, and it was reported in a long series of board meetings that quite a large amount of it was being stolen. Now, we could not understand how anyone could steal plastic rolls of floor covering, two meters high, three quarters of a meter diameter, weighing an immense amount with these huge, strong steel doors, concrete floors. There was no sign of the doors being attacked. No signs of any exterior entry. No clues at all. The police couldn't discover a clue of any sort. How you got to those things mysteriously out of a heavily guarded factory until someone in the middle of the night spotted it being done.

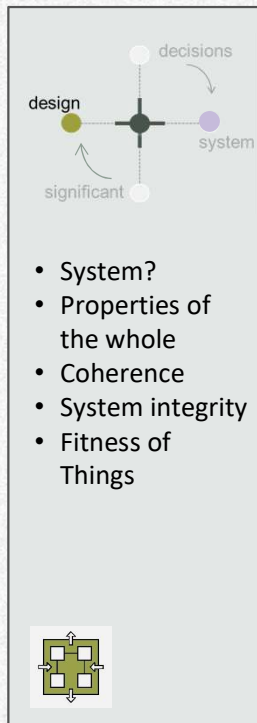
And one of the warehouse men each night before

he went home, he pushed over one of these plastic rolls and rolled it around 'til it was next to the door. He then proceeded to uncover the outside and stick the edge under the door.. came back in the middle of the night and just wound it up, you see.

Now why you laughed was there was a sense of the right way of doing it. The immediate impact was that if you're going to be a thief, this is a good style of thieving. This disreputable story is solely to produce that sort of sudden impact: That's a good idea. [chuckles] Even though it was a bad idea.

Now, this sort of impact happens in engineering design and is extremely valuable. And if you can develop it, it will censor out silly ideas at source. But there is a sense of paradox linked in with it. And that is this: that all new inventions are embodied to start with, in out of date technology.

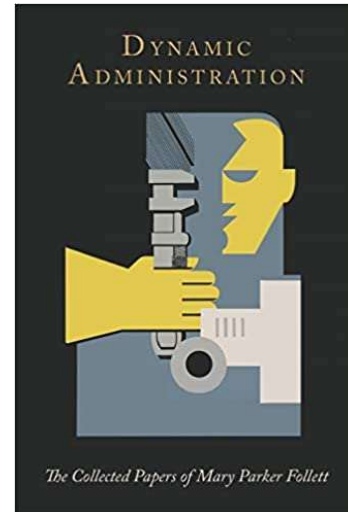
Technology always trots along behind the new invention. And therefore a new idea which is extremely good, may look extremely repellent when first produced because the technology is clumsy, awkward and unsuitable. And a sense of style sometimes needs the ability to look through the unsuitable technology to the idea beneath it."



The “Law of the Situation”

"Our job [...] how to devise methods by which we can best discover the order integral to a particular situation."

— Mary Parker Follett



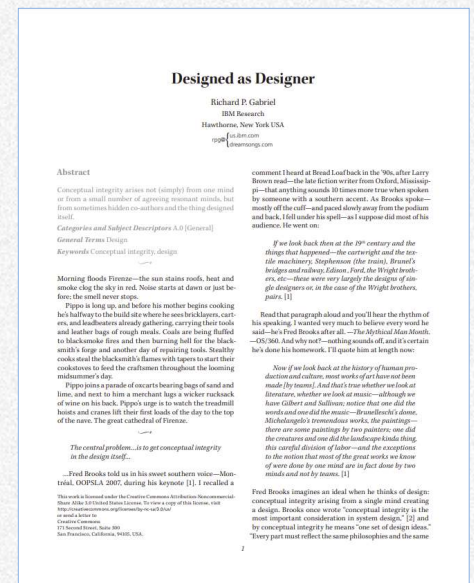
The “Law of the Situation”

Conceptual and design integrity includes the degree of fit – fit within the system, fit of the system to its context, and fit to purpose. That opens the question of the thing designed, as designer (at least, playing a suggestive, even formative, role in its own design). So we’re attending to what the system is and is becoming. Mary Parker Follett suggests that we explore and understand the situation, to understand and shape our response. (Which many of us would relate to Domain Driven Design.)

So design integrity brings with it fit or coherence, which begs the question: how do we build coherent systems? And how do we do this, with teams (of teams, even)?

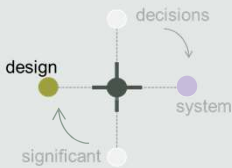
"understand the situation, must see it as a whole, must see the interrelation of all the parts [...] must do more than this. He [sic] must see the evolving situation, the developing situation. His wisdom, his judgment, is used, not on a situation that is stationary, but on one that is changing all the time."

— Mary Parker Follett, 'The Giving of Orders'



"That's always our problem, not how to get control of people, but how all together we can get control of a situation."

— Mary Parker Follett



- System?
- Properties of the whole
- Coherence
- System integrity
- Conceptual integrity



Conceptual Integrity

“[conceptual integrity]—another contribution from Brooks—is roughly the state of having a unified mental model of both the project and the user, shared among all members of the team.”

— Dorian Taylor



Conceptual Integrity

In 1975, Fred Brooks said: *I will contend that Conceptual Integrity is the most important consideration in system design. It is better to have a system omit certain anomalous features and improvements, but to reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas.*

In 1995, Brooks still hasn't changed his mind: *I am more convinced than ever. Conceptual Integrity is central to product quality. Having a system architect is the most important single step toward conceptual integrity...after teaching a software engineering laboratory more than 20 times, I came to insist that student teams as small as four people choose a manager, and a separate architect.*

See also: [ArchitectAsKeeperOfTheFlame, ChiefArchitect](#)

Discussion:

According to Fred Brooks, "Conceptual integrity in turn dictates that the design must proceed from **one mind**, or from a **very small number** of agreeing **resonant minds**". To me, a very small number would only mean the entire team only when that team is a very small number. In my opinion, Conceptual Integrity is a required ingredient for achieving the principle (I

Image Source: <https://wiki.c2.com/?ConceptualIntegrity>

"Having a system architect is the most important single step toward conceptual integrity." — Fred Brooks

"It is better to have a system omit certain anomalous features and improvements, but to reflect one set of design ideas, than to have one that contains many good but independent and uncoordinated ideas." — Fred Brooks

Whence Conceptual Integrity

Though Fred Brooks does not define conceptual integrity exactly, he wrote: "conceptual integrity is the most important consideration in system design," and "Every part must reflect the same philosophies and the same balancing of desiderata" (*Mythical Man Month*, 20th Aniv ed). Also, "Conceptual integrity in turn dictates that the design must proceed from one mind, or from a very small number of agreeing resonant minds."

Richard Gabriel**, in his critical engagement with Fred Brooks' OOPSLA 2007 keynote*, offers:

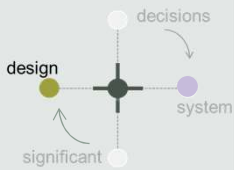
"The ingredients for conceptual integrity are these:

- the talent(s) of the human designer(s)—all of them;
- the thing designed;
- the luck that brought the designer(s) [...] to the right place(s)/[]time(s); the luck of the thing designed to have the right ingredients"

That is, Gabriel is differing from Brooks on the matter of a single architect-designer to achieve conceptual integrity.

* Fred Brooks, Collaboration and TeleCollaboration, OOPSLA 2007, http://www.oopsla.org/podcasts/Keynote_FrederickBrooks.mp3#t=535

** Richard Gabriel, "Designed as Designer," <https://www.dreamsongs.com/Files/DesignedAsDesignerExpanded.pdf>



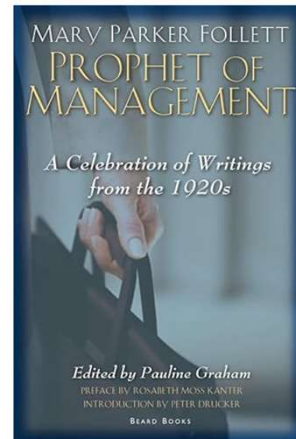
- System?
- Properties of the whole
- Coherence
- System integrity



Not Aggregation, but Integration

"It has been taken as self-evident, as a mere matter of arithmetic like 2 and 2 making 4, that if everyone does his best, then all will go well. But one of the most interesting things in the world is that this is not true, although on the face of it, it may seem indisputable. Collective responsibility is not something you get by adding up one by one all the different responsibilities. Collective responsibility is not a matter of adding but of interweaving, a matter of the reciprocal modification brought about by the interweaving. It is not a matter of aggregation but of integration."

— Mary Parker Follett (MPF)



Observations on Ackoff and Systems

Systems consist of constituent elements and relationships among them, but the process is not simply additive. This, from Trond Hjorteland's notes on the Russ Ackoff talk:

"We still haven't taken onboard the interconnectedness of the parts in a system. We still believe we can break things down and treat it in isolation. See it all the time, everywhere, both in design, but also team structure, projects, etc."

is underscored in Joonas Koivunen's point too:

"I guess my main question which comes out of the understandable/intuitive examples is, why is system thinking still such a niche/unpopular idea."

There are no easy answers, but systems concepts, and how we achieve system coherence and integrity over time, as the system, the organization building it, and the context, all evolve, are matters for important discussion and attention. Follett was early among those to advocate for collaborative, integrative work, to shape integrative responses.

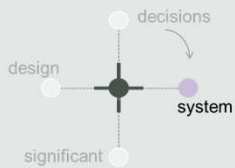
Coherence and integrity bring along concepts of fit. Fit together, fit to context, and fit to purpose. In order for work to fit, in these various senses, we need to provide enough context, including intent and understanding of what "fit" entails, in this context.

"They say that every organization has a form, a structure, and that what that organism does, its unified activity, depends not on the constituents alone, but on how these constituents are related to one another"

— Mary Parker Follett, 1926

"My solution is to depersonalize the giving of orders, to unite all concerned in a study of the situation, to discover the law of the situation, and obey that."

— Mary Parker Follett



- System?
- Properties of the whole
- Coherence
- System integrity
- Conceptual integrity
- Requisite variety



Ashby's Law: Requisite Variety

Address variety with variety

"If a system is to be stable, the number of states of its control mechanism [its variety] must be greater than or equal to the number of states in the system being controlled"

— Ross Ashby

Ashby's law of requisite variety—which is an interpretation of Shannon's Theorem 10 in Shannon and Weaver 1949—states that given the variety of disturbances, the only way to reduce the variety of outcomes is to increase the number of responses. Or, as he puts it, "[O]nly variety can destroy variety" — Gerald Flueckiger

Ashby's Law: Address Variety with Variety

Of course, our systems exist in complex contexts, with (generally) complex demands.

"In colloquial terms Ashby's Law has come to be understood as a simple proposition: if a system is to be able to deal successfully with the diversity of challenges that its environment produces, then it needs to have a repertoire of responses which is (at least) as nuanced as the problems thrown up by the environment. So a viable system is one that can handle the variability of its environment. Or, as Ashby put it, only variety can absorb variety." — John Naughton

Jabe Bloom: "The quickest way to explain Ashby's Law is as follows: If I am a fencer and I have 3 ways of thrusting at people, and everybody else has three ways of parrying those thrusts, it will be an even game. [...] I will be as in control as I can be. If someone else figures out another thrust, I will then be required to learn another parry otherwise I will always lose." Implication: The more different kinds of customers your business has, the more complexity you will need to absorb, in order to respond to that.

Brian Marick: 'In the 80's, Robert Glass analyzed bugs in fielded avionics software. Found faults of omission most important. I liked his characterization of them: "code not complex enough for the problem"'

Jabe Bloom: "Sounds like Ashby's Law."

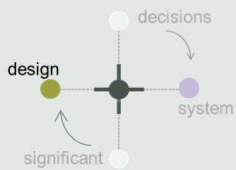
"Ashby's law dictates that complex environments (and wicked problems) require complex organizations."

— Jabe Bloom

"The Battle Royale: Ashby's Law vs Herbert Simon's Bounded Rationality"

— Jabe Bloom

*Coherence with too much convergence, reduces variety;
too little coherence and the system loses integrity*



Requisite Coherence

“And requisite coherence is the idea that if everyone is in a Tower of Babel we’re not able to speak or work together. So the balancing point here is common ground.”



— Jabe Bloom



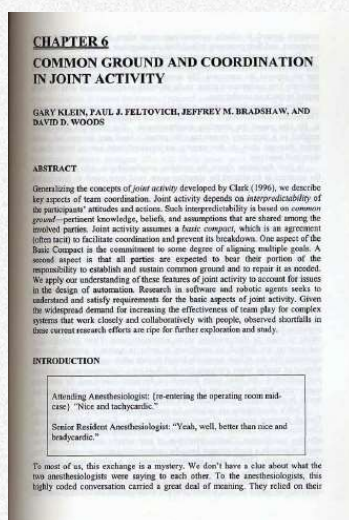
- System?
- Properties of the whole
- Coherence
- System integrity
- Conceptual integrity
- Requisite coherence

*Incoherence Penalty: :
"Whatever time the
team members spend
re-establishing a
common view of the
universe"*
— Michael Nygard

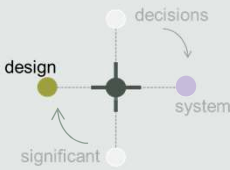
Common Ground

“So there’s two ideas: *requisite variety* meaning that a system that’s going to address a complex space needs to have complexity inside of it in order to react to the complexity outside of it; it’s like a balancing act; so there’s this idea that you should have lots of variety in the system. And the other side of it is *requisite coherence*. And requisite coherence is the idea that if everyone is in a Tower of Babel we’re not able to speak or work together. So the balancing point here is *common ground*. And it’s this idea that we need just enough common concepts to make progress — not maximally but minimally. In order preserve the scanning and perceptual abilities of multiple mental models.” — Jabe Bloom, VirtualDDD 1/16/20


“Joint activity depends on interpredictability of the participants’ attitudes and actions. Such interpredictability is based on common ground—pertinent knowledge, beliefs and assumptions that are shared among the involved parties. Joint activity assumes a basic compact, which is an agreement (often tacit) to facilitate coordination and prevent its breakdown. One aspect of the Basic Compact is the commitment to some degree of aligning multiple goals. A second aspect is that all parties are expected to bear their portion of the responsibility to establish and sustain common ground and to repair it as needed.” — Gary Klein et al.



If we treat alignment as something that is done to (we align the team), that's mechanistic and...



- System?
- Properties of the whole
- Coherence
- System integrity
- Conceptual integrity
- Requisite variety
- Requisite coherence



Coherence and Alignment

We need variety to respond to complexity in the environment

We need coherence to produce a system with integrity


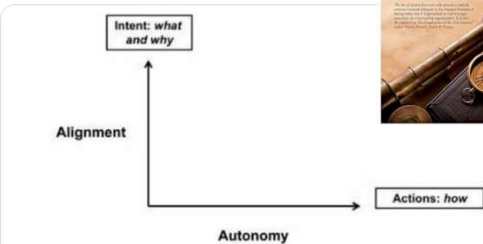



Figure 8 High alignment enables high autonomy

The insight is that alignment needs to be achieved around intent, and autonomy should be granted around actions. Intent is expressed in terms of what to achieve and why. Autonomy concerns the actions taken in order to realize the intent; in other words, about what to do and how. By requiring his commanders to distinguish between “what and why” on the

Highlights by Jabe Bloom; quote snip from Bungay's book

Alignment, Autonomy and Common Ground

As we evolve systems respond to complex demands (e.g., for system capabilities offered to users, and system capabilities for system health, from scalability to security to monitoring, defending the system against threats), they become more complex, with implications for expertise and organization capacity (teams, of teams even). And with that organizational complexity, there's this matter of requisite coherence. Requisite, because variety (different expertise, experience, perspectives) and independence (and autonomy) are also needed.

For discussion: how do we achieve and balance alignment and autonomy?

“Autonomy is the ability to choose which action to take

Agency is the ability to choose an action to take (to have intentions) and to be able to observe the results of those actions in the system one acts within”

— Jabe Bloom

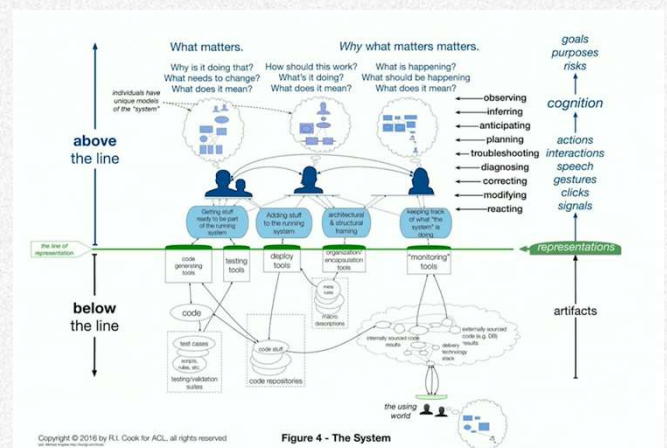
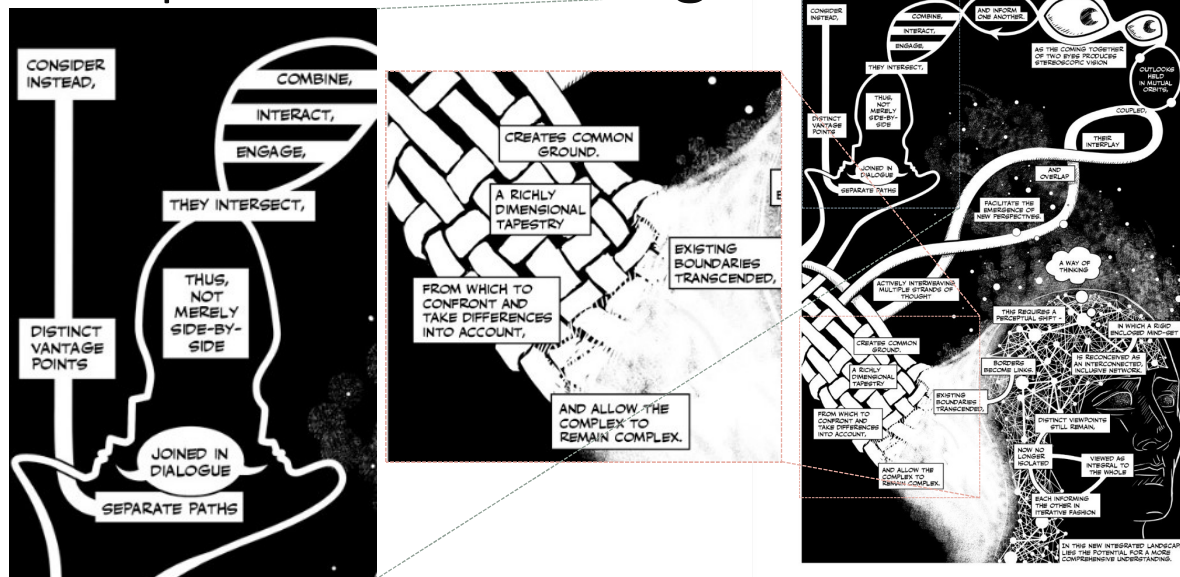


Figure 4 from the STELLA Report (figure by Richard Cook). Different actors interacting with the system, have different mental models of the system. These differences are not bad, for they are related to the different ways they interact with the system, and their different areas of expertise. These different areas of expertise create adaptive capacity, so these differences are important to resilience. (Allspaw via Bloom). Requisite coherence relates to what is necessary for there to be requisite variety (to respond to variety in the use, development, operations, etc. space) and yet still create a whole that has integrity.

Perspectives in Dialog

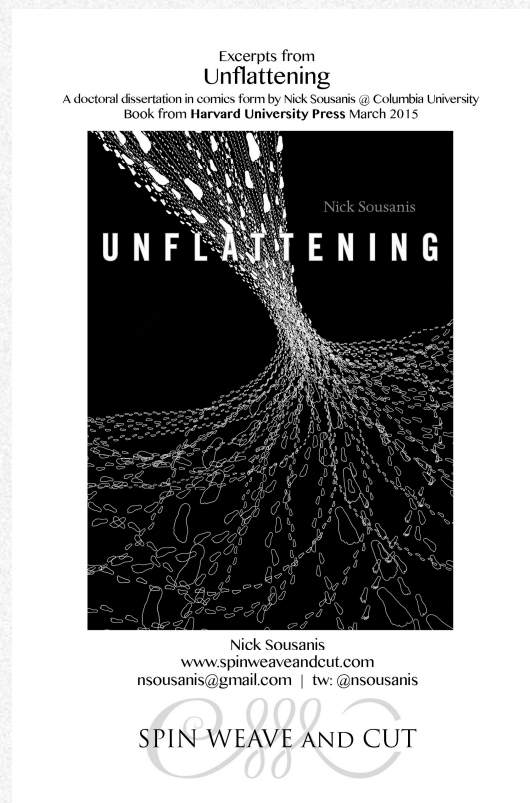


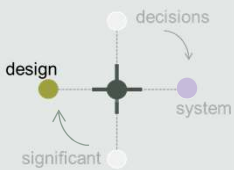
From: *Unflattening*, by Nick Sousanis

Integration of Perspectives

"Consider instead,
 distinct vantage points
 Separate paths
 Joined in dialog
 Thus not merely side-by-side
 They intersect,
 engage,
 interact,
 combine,
 and inform one another
 As the coming together of two eyes in stereoscopic vision
 Outlooks held in mutual orbits
 Coupled, their interplay and overlap, facilitate the emergence
 of new perspectives.
 Actively interweaving multiple strands of thought
 Creates common ground"

— Nick Sousanis (@Nsousanis), *Unflattening*, pg 37





Coherence and Alignment

"understanding of complex systems is distributed"
— Chris McDermott

- System?
- Properties of the whole
- Coherence
- System integrity
- Conceptual integrity
- Requisite variety
- Requisite coherence



Jessica Joy Kerr
@jessitron

...

When you build a Wardley Map, the artifact is less important than the conversation that generates it, and the ways we are collectively changed by that.

We create a common understanding, as each person's individual assumptions are challenged. @suksr @ExploreDDD

<https://twitter.com/jessitron/status/1768687764353228997>

Conversations and Common Ground

Conversations (and conversational forms, like Slack, RFCs, etc., even though async) draw in, and draw on, various perspectives. (We need to consider whose perspectives are not being drawn on, and in, and why, too.) Visual forms help create "I see what you mean" moments. A canvas, map, or diagram focuses collaborative exploration, while also drawing attention to areas the discussion might otherwise avoid or neglect. The "how" is collaborative, guided and yet open, integrative, ... There are times we need to think through something carefully, writing and interacting with our writing. However, participative exploration, sense-making, and decision making brings more perspectives to bear, and helps generate understanding and create integrative (rather than compromise) views.

See also:

Meetings **Are** the Work, by Elizabeth Ayer:
<https://medium.com/@ElizAyer/meetings-are-the-work-9e429dde6aa3>

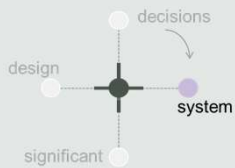
"Alignment and direction is so hard to get; clarity of what you're doing and how you fit into and contribute to a system is so hard to maintain. But it's so important that it should never be neglected.

*I see executives working on decision matrices, and engineers working on refactoring, and infra building platforms, but I don't see people **actually communicating together**"*

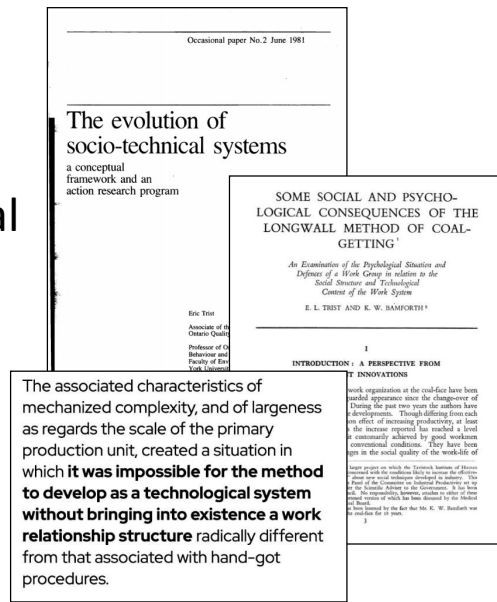
— Hazel Weakly

Sociotechnical Systems

Sociotechnical systems refers to systems that have social and technical elements, and there is mutual influence and interaction of technical and social elements



- System?
- Properties of the whole
- Coherence
- System integrity
- Conceptual integrity
- Requisite coherence
- Requisite variety



Sociotechnical Systems

Sociotechnical systems draws attention to this partnering of people and technology in complex systems, where people add capability to technical systems, and especially their adaptive capacity. Technical systems, in turn, extend capabilities of people involved in some way, but also impact how work is done, changing the “work relationship structure,” affecting interactions, groups and individuals (potentially lowering adaptive capacity, making work unsatisfying, etc.).

The term socio-technical systems was coined by Eric Trist, Ken Bamforth and Fred Emery, based on their World War II era work with workers in English coal mines, studying the impact of replacing the manual and team-intensive “hand got” method with the “longwall method” (using mechanical conveyors and coal-cutters). They pointed out that a technological system impacts the social system it interacts with:

“So close is the relationship between the various aspects that the social and the psychological can be understood only in terms of the detailed engineering facts and of the way the technological system as a whole behaves in the environment of the underground (mining) situation.”

— Eric Trist and Ken Bamforth, 1985

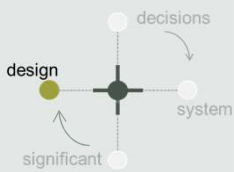
“the claim is that the technology and the sociology cannot be seen as independent parts, that the system as a whole can only be improved by joint optimization of those parts. Productivity and wellbeing are seen as emergent properties of the system”

— Trond Hjorteland

Trist, Eric. “The evolution of socio-technical systems.” Occasional paper 2 (1981): 1981.

Trist, Eric. “A concept of organizational ecology.” Australian journal of management 2.2 (1977): 161-175.

Elbanna, Amany, “Doing Sociomateriality Research in Information Systems,” 2016



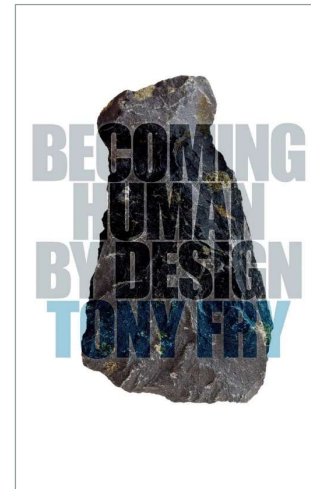
- Design!
- Which system?



Which System?

We aren't only designing the software intensive system we're design-evolving. Systems change their contexts.

"we design our world, while our world acts back on us and designs us" — Anne-Marie Willis



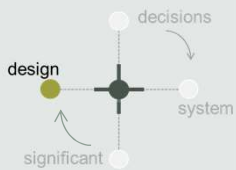
"the wish [or intention] confronts an environment as altered by the wish; the environment confronts a wish as altered by the environment"
— Mary Parker Follett,
Creative Experience, 1924

"we design, that is to say, we deliberate, plan and scheme in ways which prefigure our actions and makings — in turn, we are designed by our designing and by that which we have designed"
— Anne-Marie Willis
"Ontological designing" 2006

Design of the System in Context

We have focused thus far mostly on design within the system — parts and interconnections, and dynamic interactions that give rise to systems and their properties. But design has to do, also, with shaping intention: what system capabilities and properties are we directing design and engineering intention and experience at building? Yes, this is the purview of product design, or system design with an emphasis on the system in its context(s) of use. And here we are often guided to "identify user needs" but the essence of the design work here is more than understanding users and their challenges and what would be of value to them. It is also anticipating how the system we're design-evolving will change the systems they participate in (their workflows, social connections, and so on). Not perfectly. But enough to form theories about value and consequences, and our responses. So we're back to "system of concern" and where we draw boundaries.

"We can never understand the total situation without taking into account the evolving situation" — Mary Parker Follett



- Design!
- Which system?
- Reality construction

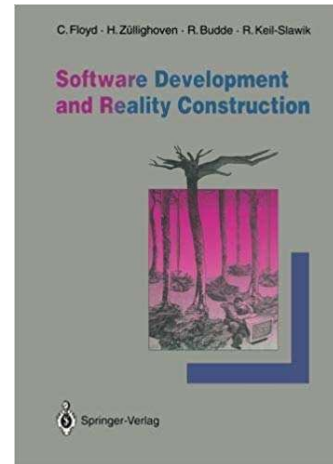


Reality Construction

“We do not analyze requirements;
we construct them”

“Their emergence is specific to the individual design process; it is not determined by the given problem. Instead the problem itself is grasped in the course of the design process.”

— Christiane Floyd



Co-Evolutionary Design

Meir Lehman (1980):

“The installation of the program together with its associated system [...] change the very nature of the problem to be solved. The program has become a part of the world it models, it is embedded in it. Analysis of the application to determine requirements, specification, design, implementation now all involve extrapolation and prediction of the consequences of system introduction and the resultant potential for application and system evolution. This prediction must inevitably involve opinion and judgment.”

Cameron Tonkinwise (2021):

“the ways in which designers design, the ways in which design is ontological, even at a human product scale, because it creates worlds, habits, dispositions. A designer is never [...] just designing a product: they are reinforcing particular models of the human”

Christiane Floyd:

“We do not analyze requirements; we construct them from our own perspective. This perspective is affected by our personal priorities and values, by the methods we use as orientation aids, and by our interaction with others”

“jointly creating computer-supported contexts of action with users”

Ref: Software Development as Reality Construction, by Christiane Floyd, 1992

*“All that you touch
You Change
All that you Change
Changes you”*

— Octavia Butler,
Parable of the Sower

“Design designs”

— Tony Fry

*“there is a feedback loop
here that says actually
designing things [...] changes what we will
design in the future, and
doesn't stop — it's a
loop.” — Jabe Bloom*

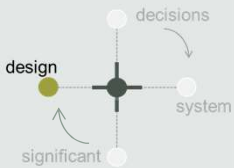
Kenny Baas-Schwiegler reposted



Black Tulip Techno @Technology · Aug 27 ...
Today alas I must once again tweet that software is not a complex adaptive system. If you've built something that suffers from unintended consequences, has non-linear responses to the same input, and is unpredictable in its operation you have badly designed software not a CAS.



Which Environment?



- Design!
- Which system?
- Which environment?

- Users and their task environment?
- Operations and their task environment?
- Developers and their task environment?

All of them! (managers too)

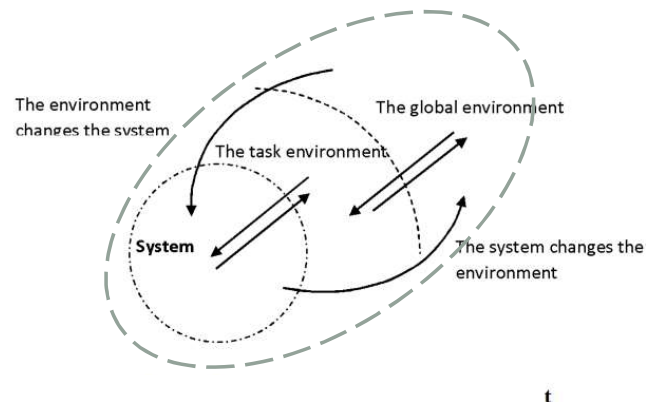


Image adapted from Merrelyn Emery, "Self management of the self managing organization: an update"

Various Contexts of Work and Ecosystems

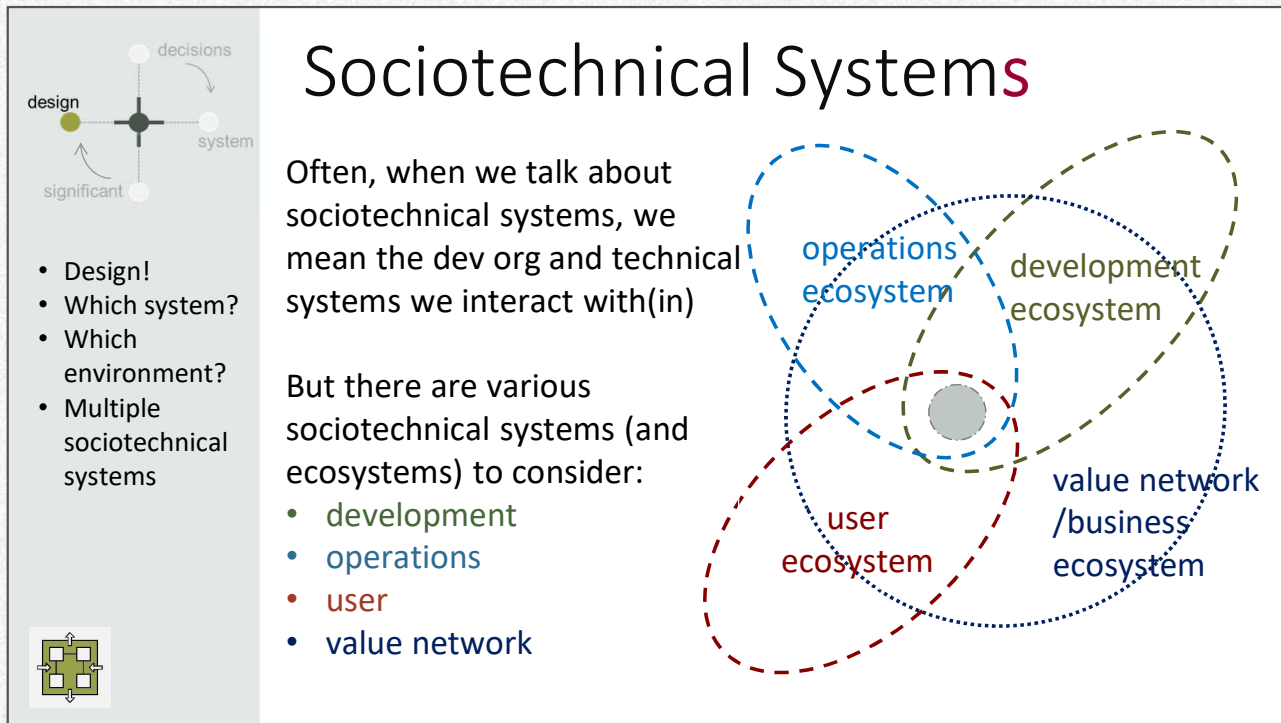
More! Our decisions as system designers impact, and hence need to be informed and influenced by, *various* contexts: the work or other (entertainment, etc.) contexts of users; our partners in the value stream and what they are contending with; our operations teams or, if we're designing embedded software for products, the maintenance and repair teams and their work environments; developers and others in our organization evolving the system; and so forth. So as we scan for what is relevant to the decision or decisions we're making, sure, we need to use discipline and scope our exploration. But we also need to use discipline in the sense that this adds complexity, and ignorance makes things easier in the short haul, but...

Sociotechnecological was coined by Jabe Bloom, to bring together sociotechnical, techne, and ecological. 'Techne is a term in philosophy that refers to making or doing, which in turn is derived from the Proto-Indo-European root "Teks-" meaning "to weave," also "to fabricate". As an activity, technē is concrete, variable, and context-dependent.' (<https://en.m.wikipedia.org/wiki/Techne>)

In sociotechnecological, Jabe Bloom (@cyetain) is extending sociotechnical (in sociotechnical systems) in important ways — he adds to techne's sense of skillful, underscoring it with skillful coping, and adds ecological.

"Software development then always creates sociotechnical, socio-economic, sociocultural systems."
— Christoph Becker





Sociotechnical Systems

Much of the emphasis on sociotechnical systems in our field has been on ourselves in the development context, and the impact of technology on our work — how we organize to build and evolve systems, the way we structure our code and how that impacts the organization and vice versa (Conway's Law/Mirroring Hypothesis), how our development environment and CI/CD platform impacts developer experience, and more. Or we focus on users and how the systems we're building impacts their work and direct and indirect experience. In either case, noting that we need to jointly design the system and how work is done, and factor the mutual impact of technology and people and organizations. And so on. The point here is simply to remind ourselves that this needs to happen in multiple dimensions, considering these various interacting spaces of sociotechnical systems (STS) — dev STS (code, development platform, team and organizational concerns like team responsibilities, and more); user STS (software in use, user workflows and their organizational contexts as relevant, and more); etc.

"The field sees its focus as the development of technical systems with clear boundaries and identifiable parts and connections, modules, and dependencies. But fifty years after the founding of software engineering as a field, the boundaries between software and its social and environmental contexts are rapidly dissolving. Software systems now have become part of our societies' fabrics and shape the relationships that constitute them"

— Christoph Becker

- Design!
- Which system?
- Which environment?
- Multiple sociotechnical systems
- Ecosystems

Ecosystems

ec·o·sys·tem
/ ˈɛkō.sɪstəm/

noun **ECOLOGY**

a biological community of interacting organisms and their physical environment.
"the marine ecosystem of the northern Gulf had suffered irreparable damage"

- (in general use) a complex network or interconnected system.
"Silicon Valley's entrepreneurial ecosystem"

Images: <https://www.exploringnature.org/>
<https://www.cgma.org/resources/reports/the-extended-value-chain.html>

Ecosystem

"A (biological) community of interacting organisms and their (physical) environment."

"Complex of living organisms, their physical environment, and all their interrelationships in a particular unit of space."

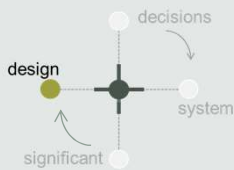
— Encyclopedia Britannica

Source: James F. Moore, *The Death of Competition: Leadership & Strategy in the Age of Business Ecosystems*. 1996.

Business Ecosystem

"An economic community supported by a foundation of interacting organizations and individuals—the organisms of the business world. The economic community produces goods and services of value to customers, who are themselves members of the ecosystem. The member organisms also include suppliers, lead producers, competitors, and other stakeholders. Over time, they coevolve their capabilities and roles, and tend to align themselves with the directions set by one or more central companies. Those companies holding leadership roles may change over time, but the function of ecosystem leader is valued by the community because it enables members to move toward shared visions to align their investments, and to find mutually supportive roles." — James F. Moore

An ecosystem is not only a system of innovation-driven change, but of weaving relationships that stabilize and repair. Adapting to change, coping with uncertainty, these are things we talk about in a VUCA (volatility, uncertainty, complexity, ambiguity) world. Ecosystem activities involve flows and transformations, using and creating value. As well as activities by which stability is maintained, including repair, and building what we learn back into our systems. Or at least, we should. Maintenance (reducing tech and environmental debt), should play a larger role in our organizations and communities.



- Design!
- System design is about system viability (thriving even)



What is systems design?

“What is systems design? It's the thing that will eventually kill your project if you do it wrong, but probably not right away. It's macroeconomics instead of microeconomics. [...] It's knowing when a distributed system is or isn't appropriate, not just knowing how to build one.”

— Avery Pennarun

2020-12-27 [u](#)

Systems design explains the world: volume 1

“Systems design” is a branch of study that tries to find universal architectural patterns that are valid across disciplines.

You might think that's not a possibility. Back in university, students used to tease the Systems Design Engineers, calling it “boxes and arrows” engineering. Not real engineering, you see, since it didn't touch anything tangible, like buildings, motors, hydrochloric acid, or, uh, electrons.

I don't think the Systems Design people took this criticism too seriously since everyone also knew that programme had the toughest admittance criteria in the whole university.

(A mechanical engineer told me they saw electrical/computer engineers the same way: waveforms on a screen instead of real physical things that you could touch, change, and fix.)

I don't think any of us really understood what boxes-and-arrows engineering really was back then, but luckily for you, now I'm old. Let me tell you some stories.

What is systems design?

I started thinking more clearly about systems design when I was at a big tech company and helped people refine their self-promotion employee review packets. Most of it was straightforward, helping them map their accomplishments to the next step up the engineering ladder.

<https://apenwarr.ca/log/20201227>

What is Systems Design?

“Most of all, systems design is invisible to people who don't know how to look for it. At least with code, you can measure output by the line or the bug, and you can hire more programmers to get more code. With systems design, the key insight might be a one-sentence explanation given at the right time to the right person, that affects the next 5 years of work, or is the difference between hypergrowth and steady growth. ” — Avery Pennarun (@apenwarr), “Systems design explains the world: volume 1”

Source: <https://apenwarr.ca/log/20201227>

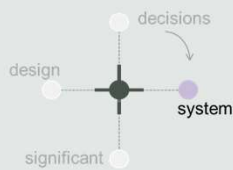
System design is about shaping decisions that impact the very viability of the system, over different horizons. That is, it is about identifying and overcoming strategic hurdles at different evolutionary points, and making decisions we have to live with over longer horizons — with discernment.

“They were doing some hard jobs — translating business problems into designs — with great expertise”

— Avery Pennarun

“Compasses are for learning how to use them.”

— kid (at 7)



- System?
- Wicked problems are wicked!



Wicked Problems

Reflecting on the nature of design problems, Rittel highlighted several of their characteristics:

1. As seen above, analysis and synthesis are not separable, "the problem can't be defined until the solution has been found". Or every formulation of the problem hints at a solution.
2. Every problem can be seen as a symptom of another and problems cannot be separated into disciplines.
3. Unlike in chess, there are no stopping rules that tell us when we are done
4. Solutions are neither true nor false; they are either good or bad. What is good or bad is a matter of values and judgments.
5. There are no immediate, or ultimate tests to design problems. A solution may work at first just as planned only to prove to have deleterious consequences later. DDT ex.
6. Every problem is essentially unique; there are no classes of problems. No matter how similar two problems may look there is always a chance that there is a difference that matters more than all the similarities.
7. Every problem is a "one-shot" operation. Every implemented plan has consequences that cannot be undone. If she doesn't like a wall that has been set up, the architect can always sign a change order, but that has consequences: labor cost, demolition materials to be disposed, etc.
8. There are no grand phases (i.e. analysis-synthesis-evaluation) nor are there any agreed upon, specific procedures.

From: Design Thinking: What is That? By Jean-Pierre Protzen

Wicked Problems are Wickedly Hard

"The problems that scientists and engineers have usually focused upon are mostly "tame" or "benign" ones. As an example, consider a problem of mathematics, such as solving an equation; or the task of an organic chemist in analyzing the structure of some unknown compound; or that of the chessplayer attempting to accomplish checkmate in five moves. For each the mission is clear. It is clear, in turn, whether or not the problems have been solved.

Wicked problems, in contrast, have neither of these clarifying traits; and they include nearly all public policy issues--whether the question concerns the location of a freeway, the adjustment of a tax rate, the modification of school curricula, or the confrontation of crime."

"1. There is no definitive formulation of a wicked problem: [...] The information needed to understand the problem depends upon one's idea for solving it. That is to say: in order to describe a

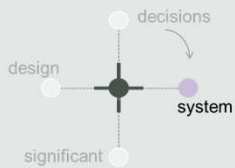
wicked-problem in sufficient detail, one has to develop an exhaustive inventory of all conceivable solutions ahead of time. The reason is that every question asking for additional information depends upon the understanding of the problem — and its resolution — at that time. Problem understanding and problem resolution are concomitant to each other.

2. Wicked problems have no stopping rule: The planner terminates work on a wicked problem, not for reasons inherent in the "logic" of the problem. He stops for considerations that are external to the problem: he runs out of time, or money, or patience. He finally says, "That's good enough," or "This is the best I can do within the limitations of the project," or "I like this solution," etc.

Source: Horst Rittel and Melvin Webber, Dilemmas in a General Theory of Planning, 1973

A great collection of references on "messes" and "wicked problems":

<https://github.com/lorin/messiness>



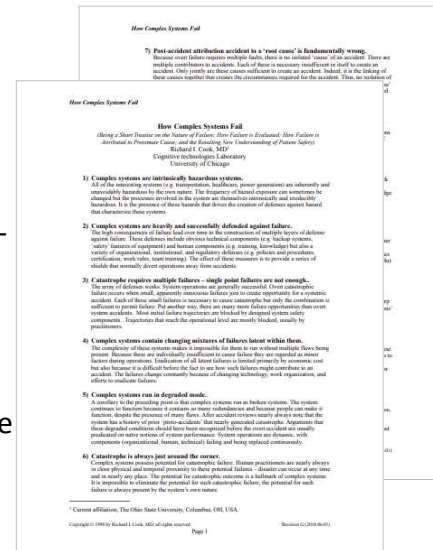
- System?
- Wicked problems
- How complex systems fail (and don't)



How Complex Systems Fail

1. Complex systems are intrinsically hazardous systems
2. Complex systems are heavily and successfully defended against failure
3. Catastrophe requires multiple failures – single point failures are not enough
4. Complex systems contain changing mixtures of failures latent within them
5. Complex systems run in degraded mode
6. Catastrophe is always just around the corner

-- Richard I. Cook



Complex Systems (Guard Against) Fail(ure)

From Adrian Colyer's notes on Richard Cook's classic paper:

- *Complex systems are intrinsically hazardous*, which drives over time the creation of defense mechanisms against those hazards. (Things can go wrong, and we build up mechanisms to try and prevent that from happening).
- *Complex systems are heavily and successfully defended against failure*, since the high consequences of failures lead to the build up of defenses against those failures over time.
- Because of this, a *catastrophe requires multiple failures* – single point failures are generally not sufficient to trigger catastrophe.

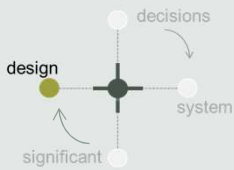
"The state of safety in any system is always dynamic; continuous systemic change insures that hazard and its management are constantly changing." – Richard I. Cook

- The complexity of complex systems makes it impossible for them to run without multiple flaws being present. Because these are individually insufficient to cause failure, they are regarded as a minor factor during operations.
- *Complex systems* therefore *run in degraded mode* as their normal mode of operation!
- *Changes introduce new forms of failure*
- *Safety is a characteristic of systems and not of their components*
- *People continuously create safety*
- *Failure free operations require experience with failure.*

Much of Richard Cook's and others work in resilience engineering and safety and human factors, is addressed at operations and the role of operators in the continuous creation of safety: "Recognizing hazard and successfully manipulating system operations to remain inside the tolerable performance boundaries requires intimate contact with failure." (Cook, 2000)

As system designers and architects, we're looking at implications for design.

Sources: "How Complex Systems Fail," by Adrian Colyer, Morning Paper Richard I. Cook, How Complex Systems Fail, <https://how.complexsystems.fail/>



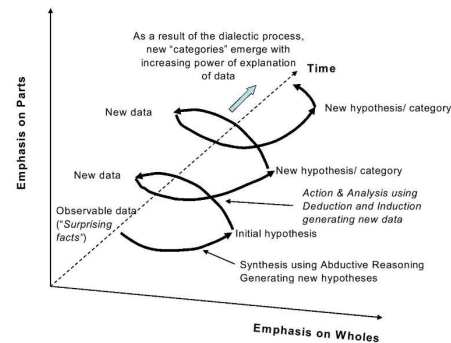
- Design!
- Evolving design



Evolutionary Design

"A complex system that works is invariably found to have evolved from a simple system that worked. A complex system designed from scratch never works and cannot be patched up to make it work. You have to start over, beginning with a working simple system."

— John Gall



From: Barton and Haslett

"complex systems will evolve from simple systems much more rapidly if there are stable intermediate forms than if there are not."

— Herbert Simon

"A complex system, such as a living organism or a growing economy, has to develop its structure and be able to adapt that structure in order to cope with changes in the environment."

— Paul Cilliers

Evolutionary Design

While "It's the thing that will eventually kill your project if you do it wrong" sounds intimidating, systems don't spring forth fully-formed in a moment.

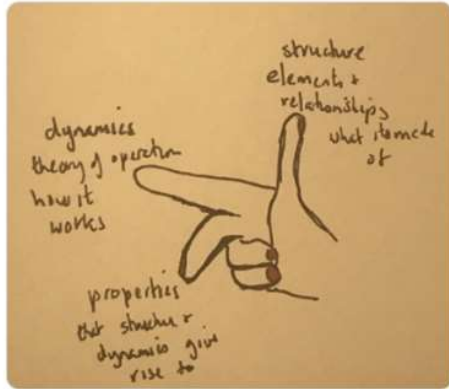
As designers, we're shaping the role the system plays in its contexts, (hopefully) contributing more value than it uses, captures, or extracts (in good senses or bad) to become and remain viable, and even thrive. And this is an active, dynamic learning process in which designers seek to understand the system environments and how they are reshaping and evolving (eg as new technologies enter the landscape, and new uses are found, etc), and adapt the system to respond to changes and discoveries. And feed back learning about system vulnerabilities, and how to make the system more resilient, into the design. It's a process of forming theories of value and of system design to fit the challenges faced and anticipated, and testing hypotheses.

Aside: The diagram (on the slide) is from a paper about evolution in science, but holds a nice image for us (in systems design/evolution), moving between synthesis and analysis and synthesis, whole and part and whole. In the large, and in smaller movements, continually.

Evolving Understanding and Design



systems understanding (and so design) involves understanding context, structure and dynamics, and the properties the interaction of structure and dynamics in context give rise to



12:44 AM · Feb 7, 2019 · Twitter for iPhone

(WHOLE) SYSTEMS UNDERSTANDING = {

UNDERSTANDING "BASIC CONTEXT" {

STRUCTURE X DYNAMICS

}

+

UNDERSTANDING OF THE "EMERGING CONTEXT" {

PROPERTIES THAT RISE FROM THE INTERACTIONS THE STRUCTURE AND

DYNAMICS IN THE CONTEXT

}

}

"Code" Image Source: Eduardo da Silva

"After all, if architecture is about a system's being, behaving, balancing, and becoming, we should be clear about "what is the system?" and "what isn't the system?"

— Charlie Alfred

*"Shops are for
Buying more stuff
for them" (kid at 7)*

Worlds create worlds. "Systems develop goals of their own the instant they come into being" (John Gall). Open systems put energy into self-renewal and self-repair, continuity or resilience.

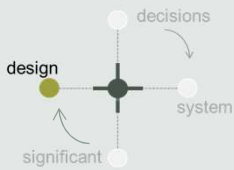
Systems Evolve and Emerge

"The image [above] [...] In a nutshell: we focus on understanding the structure and the dynamics of the system; and furthermore we also look at properties that emerge from the interactions of structure and dynamics (as in the right hand rule from physics) — and also with context." — Eduardo da Silva

[The image is a composite created by Eduardo da Silva (using my tweet and ... sketch, and his "code" for the insights) https://esilva.net/articles/evolve_tech_orgs_using_sociotech]

Due to this interaction of parts, wholes emerge and interact within contexts (or situations, or other systems in ecologies or ecosystems), and the context acts back and the system adapts or is adapted (or exapted, if the containing/using system is changing faster than the focal system). And so it goes.

The point, for design leaders, being that we're seeking to understand what the system is being and becoming, while balancing demands and forces. As we look across the seams and gaps and what falls between, we're not only considering the system we're building, but the organization that is reflected in the system (Conway's Law) and the situation it alters and is altered by.



- Design!
- Evolving design
- Starting problems



Co-Evolutionary Design

‘Expert design is more a matter of developing and refining both the formulation of a problem and ideas for a solution in concert, in a process called “co-evolution”

— Kees Dorst



And! We're co-evolving the situation, too

Co-Evolutionary Design

Donald Schön, *Reflective Practitioner*: Design is a "reflective conversation with the situation" and "a conversation with the materials of the situation" and "the situation 'talks back' and [the designer] responds to the situation's 'talk back'"

Fred Emery: "Such mutual determination can only be a result of a process of co-evolution. Our perceptual and affective systems have evolved so that we are, as a species adapted to living in the environment the world provides. [...] We have shaped that world with a view to it supporting the purposes we consistently pursue."

Winnograd and Flores: "The significance of a new invention lies in how it fits into and changes this network. Many innovations are minor—they simply improve some aspect of the network without altering its structure. The automatic transmission made automobiles easier to use, but did not change their role. Other inventions, such as the computer, are radical innovations that cannot be understood in terms of the previously existing network. The challenge for design is not simply to create tools that accurately reflect existing domains, but to provide for the creation of new domains. Design serves simultaneously to bring forth and to transform the objects, relations, and regularities of the world of our concerns"

"all systems are what emerges over its history of adaptation to stressors"

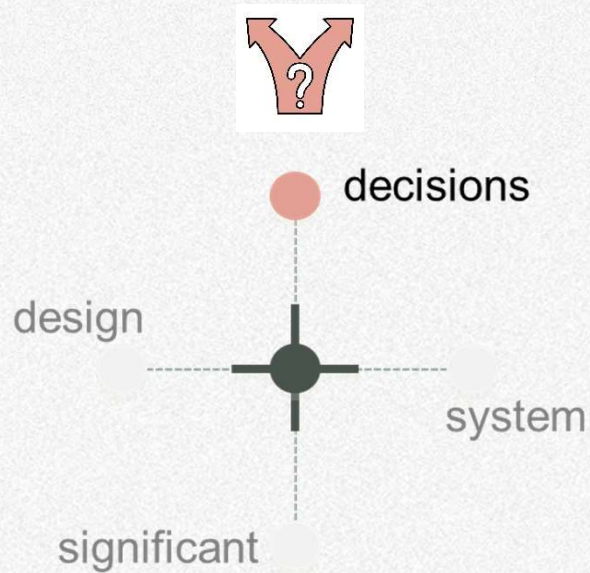
— David Woods

"expert design involves a period of exploration in which problem and solution spaces are unstable until (temporarily) fixed by an emergent bridge which identifies, or frames, a problem-solution pairing."

— Kees Dorst

Quote source: *Frame Innovation: Create New Thinking by Design*, Kees Dorst, 2015

Decisions

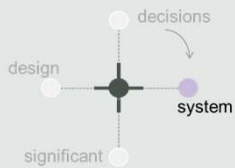


Architecture

Systems

Design

Decisions



- Complexity



Complexity

complexity (n.)

1721, "composite nature, quality or state of being composed of interconnected parts"

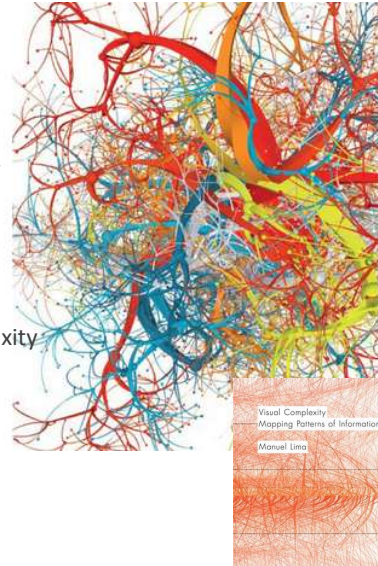
Etymology: <https://www.etymonline.com/word/complexity>

Complex: from the Latin *complexi*

Completi: from *com* ("together") and *plectere* ("to braid")

Miriam Webster

Image source: *Visual Complexity*, Manuel Lima



"yes, but"

Complexity: Parts and Dynamic Relationships

While complexity may be associated with many parts, a pile of sand, composed of many grains (parts), is not complex. Relationships, interconnection, gives rise to complexity. And yet, complexity as originally defined (in terms of composites of entwined or related parts), including notions of intricacy, could today be more associated with "complicated." A mechanical watch, for all its intricate, and intricately interconnected, parts, is complicated, not complex. Generally, when we talk about complexity and complex systems, we're addressing not just "not simple" or "not easily analyzed," but nondeterminism in system behavior, with interactions over time and changing contexts, influencing the system in non-deterministic ways.

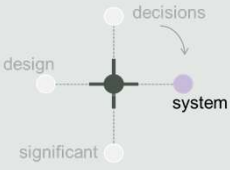
Mereology (from the Greek μέρος, 'part') is the study of system structure: of the relations of part to whole and the relations of part to part within a whole.

That's a very nice word you have there, but what's it good for? Well. It's like this. When (system and software) architecture isn't defined in terms of "the important stuff" or "the stuff that's hard to change" or the "stuff that makes you fail, if you get it wrong," it's defined in terms of structure. System structure; parts and relations of part to part and part to whole. But we're designing dynamic adaptive systems in dynamic, shifting, evolving contexts. And we can't merely ignore that. Or ought not to.


Quote source: "The Architecture of Complexity," Herbert Simon, 1962
Interesting read: "There's no such thing as a tree (phylogenetically)"

"Roughly, by a complex system I mean one made up of a large number of parts that interact in a non-simple way. In such systems, the whole is more than the sum of the parts, [...] in the important pragmatic sense that, given the properties of the parts and the laws of their interaction, it is not a trivial matter to infer the properties of the whole."

— Herbert Simon



- Complexity
- Complex systems

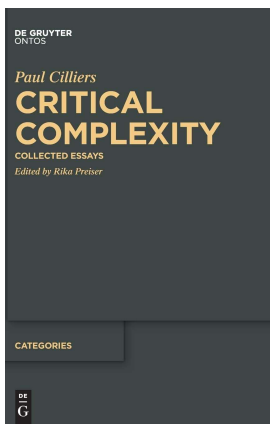


Characteristics of Complex Systems

1. Complex systems consist of a large number of elements that in themselves can be simple.
2. The elements interact dynamically by exchanging energy or information. These interactions are rich. Even if specific elements only interact with a few others, the effects of these interactions are propagated throughout the system. The interactions are nonlinear.
3. There are many direct and indirect feedback loops.

— Paul Cilliers

Source: "What can we learn from a theory of complexity?" by Paul Cilliers



"Since the nature of a complex organization is determined by the interaction between its members, relationships are fundamental. [...] The point is merely that things happen during interaction, not in isolation."

"Part of the vitality of a system lies in its ability to transform hierarchies."

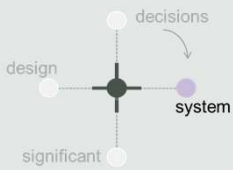
— Paul Cilliers

Paul Cilliers: What Characterizes Complex Systems

4. Complex systems are open systems—they exchange energy or information with their environment—and operate at conditions far from equilibrium.
5. Complex systems have memory, not located at a specific place, but distributed throughout the system. Any complex system thus has a history, and the history is of cardinal importance to the behavior of the system.
6. The behavior of the system is determined by the nature of the interactions, not by what is contained within the components. Since the interactions are rich, dynamic, fed back, and, above all, nonlinear, the behavior of the system as a whole cannot be predicted from an inspection of its components. The notion of "emergence" is used to describe this aspect. The presence of emergent properties does not provide an argument against causality, only against deterministic forms of prediction.
7. Complex systems are adaptive. They can (re)organize their internal structure without the intervention of an external agent.

Certain systems may display some of these characteristics more prominently than others. These characteristics are not offered as a definition of complexity, but rather as a general, low-level, qualitative description."

Source: "What can we learn from a theory of complexity?" by Paul Cilliers



- Complexity
- Complex systems
- Dynamic behavior



Dynamic Behavior

“a double pendulum is a pendulum with another pendulum attached to its end, and is a simple physical system that exhibits rich dynamic behavior”



Gif from https://en.wikipedia.org/wiki/Double_pendulum

“In a complex system, the interaction among constituents of the system and the interaction between the system and its environment, are of such a nature that the system as a whole cannot be fully understood simply by analysing its components.”

— Paul Cilliers

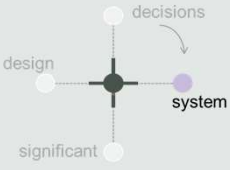
More Than Connections

“A double pendulum executes simple harmonic motion (two normal modes) when displacements from equilibrium are small. However, when large displacements are imposed, the non-linear system becomes dramatically chaotic in its motion and demonstrates that deterministic systems are not necessarily predictable.” (harvard.edu)


The human leg wouldn’t be much good if it was a simple double pendulum. The knee is a hinge joint with a limited range of motion (0, straight, to roughly 140 degrees).

“A complex system cannot be reduced to a collection of its basic constituents, not because the system is not constituted by them, but because too much of the relational information gets lost in the process.”

— Paul Cilliers



- Complexity
- Complex systems
- Dynamic behavior
- Prerequisites of complexity



Prerequisites of Complexity

“Collier identifies three prerequisites of complexity: a source of energy, gradients, and interactions that convert some of the energy influx made available by gradients


[..]

Collier’s first requirement is “a source of energy”

[..]

Harnessing energy is as important as the energy sources themselves; it allows energy to flow, and flow is necessary for order and structure to emerge. Collier’s first requirement, a “source of energy,” therefore implicates his second requirement, the presence of gradients”

— Alicia Juarrero



“Constraints are entities, processes, events, relations, or conditions that raise or lower barriers to energy flow without directly transferring kinetic energy. Constraints bring about effects by making available, structuring, channeling, facilitating, or impeding energy flow.”

— Alicia Juarrero

Complexity Formation

“Complexity formation therefore requires more than just a gradient; to evolve more complex dynamics, matter & energy must be coordinated and organized into coherent patterns

[..]

Coordination harnesses gradients by capturing energy and converting it into persistent structure and order

[..]

Collier’s third prerequisite, interactions, can also be subsumed under the general idea of constraint

[..]

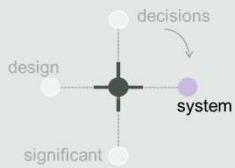
Constrained interactions leave a mark. They transform disparate many into coherent interdependent Ones”

— Alicia Juarrero

“Connectivity and interaction are necessary conditions for the emergence of complexity”

— Alicia Juarrero

Source: “Context Changes Everything: How Constraints Create Coherence,” Chapter 3, by Alicia Juarrero



- Complexity
- Complex systems
- Dynamic behavior
- Prerequisites of complexity
- Constraints constrain



Constraints.. Constrain

‘Limiting or closing off alternatives is the most common understanding of the term “constraint.”’

— Alicia Juarrero



Image from video posted by Will Evans, from LeanUX 2015

Constraints are limitations we need to be aware of. They restrict choices open to us.

“The notion of a constraint is not a negative one. It's not something which merely limits possibilities, constraints are also enabling.”

— Paul Cilliers

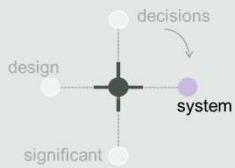
Illustration of Constraints that Limit

“The connection of the tibia and the peronei to the knee joint constrains the movement of the lower leg in such a way that it makes no sense to examine the tibia's physiology, for example, independently of the knee. The tibia's connection to the knee gives the former characteristics which it wouldn't have otherwise: it can move in some ways but not others. The constraints which the connections subject the lower leg to reduce the number of ways in which the leg can move: it can bend backwards but not forwards, for example. In this example a constraint is a reduction of the leg's state space. This is the most common understanding of the term “constraint” .”

— Alicia Juarrero, “Causality as Constraint”

Decisions Reduce the Options Space

Decisions constrain — they eliminate options. Alicia Juarrero observes that this is what we commonly mean by constraint — this limiting or closing off of alternatives; this altering of the probability distribution of available alternatives. But! In so doing, Alicia notes, they make the system “diverge from chance, from randomness.”



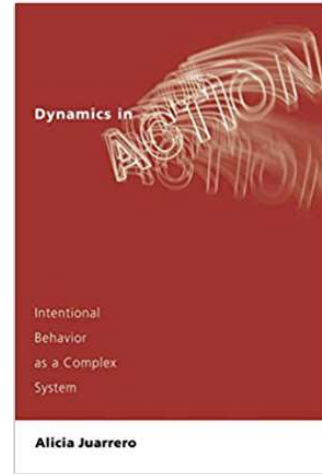
- Complexity
- Complex systems
- Dynamic behavior
- Prerequisites of complexity
- Constraints constrain
- But!



Yes, Constraints Restrict, But

“But if all constraints restricted a thing's degrees of freedom in this way, organisms (whether phylogenetically or developmentally) would progressively do less and less.”

— Alicia Juarrero



While True, ...

Constraints close off avenues, restrict the degrees of freedom, but if this was all they did, systems, including organisms, would just do less and less, as they became more constrained (Alicia Juarrero).

From Alicia Juarrero's talk (Deliberate Complexity Conference):

Constraints are conditions or factors that raise or lower barriers to energy, matter, and information flow – without themselves directly transferring energy. Example: an organisms vasculature does not impart energy directly; it channels and organizes energy flow.

Context dependent constraints enable complexity: some constraints link separate and independent elements and processes such that they become conditional on one another. They become inherently context-dependent. Enabling constraints facilitate the weaving together of interdependencies (among parts, and between parts and context). Examples: synchrony, entrainment, alignment. Enabling constraints self-organize interdependent, coherent, coordination dynamics (to create/enable new coherent dynamics). As a result, a complex system is embedded (not just plunked) in a context (temporal as well as spatial).

Source: Video of Alicia Juarrero's talk at the Deliberate Complexity online conference in 2022: Complexity is not Complication, <https://www.youtube.com/watch?v=WmtjQZCIsqY>

"Think of constraints not just as a restrictions, but as changes in probability of what's going on, changes in the likelihood of something"

— Alicia Juarrero

Rules of the road are constraints that enable high levels of agency as cyclists and drivers choose and navigate to individual destinations.

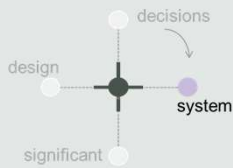
Also recommended: Constraints that Enable Innovation - Alicia Juarrero <https://vimeo.com/128934608>

Constraints Enable

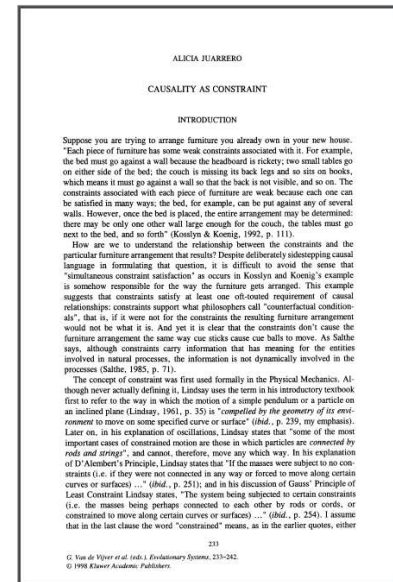
“constraints not only reduce the alternatives — they also create alternatives. Constraints, that is, can also create properties which a component exhibits in virtue of its embeddedness in a system, properties it would otherwise not have.”

— Alicia Juarrero

“Causality as Constraint”



- Complexity
- Complex systems
- Dynamic behavior
- Prerequisites of complexity
- Constraints constrain
- But! Constraints enable



Constraints Create Alternatives

“Constraints not only reduce alternatives—they also create alternatives.” If we take (Alicia Juarrero's example of) language, the constraints of syntax allow meaning to emerge.

Wholes arise from Constraints, and Wholes give rise to Constraints

“parts interact to produce novel, emergent wholes; in turn, these distributed wholes as wholes regulate and constrain the parts that make them up”

“The constraints that wholes impose on their parts are restrictive insofar as they reduce the number of ways in which the parts can be arranged, and conservative in the sense that they are in the service of the whole.”

— Alicia Juarrero, “Dynamics in Action: Intentional Behavior as a Complex System”

Juarrero (1999) distinguishes governing from enabling constraints: governing constraints regulate and restrict, while enabling constraints make a new level of complexity possible.

Context-sensitive constraints [..] synchronize and correlate previously independent parts into a systemic whole

— Alicia Juarrero

By curtailing the potential variation of component behavior, [..] context-dependent constraints paradoxically also create new freedoms for the overall system.

— Alicia Juarrero

Decisions Constrain and Enable

"In Context Changes Everything, Juarrero shows that coherence is induced by enabling constraints [..] and that the resulting coherence is then maintained by constitutive constraints."

— MIT Press

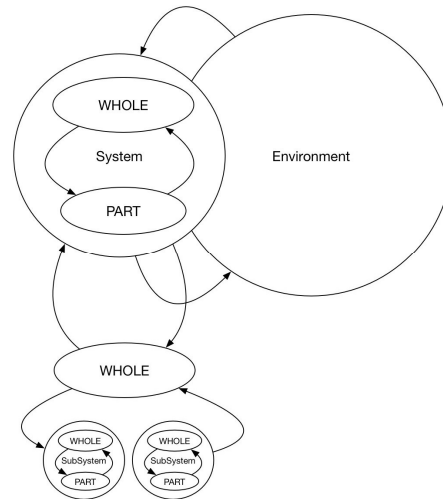
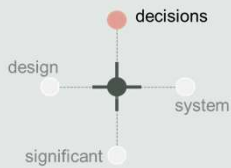


Image source: Jabe Bloom (on twitter)



- Decisions constrain

What are the implications?



Systems – but make it Wicked!

The systems we are designing interact within environments, that act back on the system. As the system begins to emerge, it also starts to act back on itself, placing constraints on its elements, to enable connections and flows, and so on. This "placing constraints on" may be more intentional and considered, or more accidental. The sociotechnical system is also placing constraints on itself, to foster coherence. Protocols, standards, decisions and other agreements. Creating common ground by collaboratively modeling, so that a shared language and shared understandings emerge.

Decisions

Decisions, making choices and constraining the subsequent design space, is both inevitable and necessary. Decisions about modularity and coupling, decisions about mechanisms to support capabilities, decisions about technology we will integrate and depend on (within the system, or our development and operations environments), all contribute to our ability to create and evolve a system that is sufficiently stable to exist, yet dynamic and evolving.

"The causal mechanism at work between levels of hierarchical organization can best be understood as the operations of constraint"

— Alicia Juarrero

"coherence is induced by enabling constraints, not forceful causes, and that the resulting coherence is then maintained by constitutive constraints. Constitutive constraints, in turn, become governing constraints that regulate and modulate the way coherent entities behave."

‘You need strategies that help rule things out. That's the opposite of saying, “This is what my gut is telling me; let me gather information to confirm it.”’

— Gary Klein

A Failure to Disagree

Daniel Kahneman
Gary Klein
Princeton University
Applied Research Associates

This article reports on an effort to explore the difference between two approaches to intuition and expertise that are often viewed as conflicting: *heuristics and biases* (HB) and *expertise* (E). The decision-making (DM) literature has long held the obvious fact that professional intuition is sometimes *more* relaxed and sometimes *flawed*, the authors attempt to state the boundary conditions that separate true intuitive skill from overconfident and biased impressions. They conclude that evaluating the likely quality of an intuitive judgment requires an assessment of the predictability of the environment in which the judgment is made and of the individual's opportunity to learn the regularities of that environment. Subjective experience is not a reliable indicator of judgment accuracy.

still separated in many ways: by divergent attitudes, preferences about facts, and feelings about fighting words such as "bias." If we are to understand the differences between our respective communities, such emotions must be taken into account.

We begin with a brief review of the origins and precursors of the NDM and HB approaches, followed by a discussion of the most prominent points of contrast between them (NDM: Kleis, Grassia, Calkinswood, & Zamboni, 1993; HB: Glöwicz, Griffin, & Kahneman, 2002; Tversky & Kahneman, 1974). Next we present some claims about the conditions under which skilled intuitions develop, followed by several suggestions for ways to improve the quality of judgments and choices.

The NDMP approach, which focuses on the successes of expert intuition, grew out of early research on man-machine interaction, conducted by the RAND (1970) and later by Chase and Simon (1973). DeGroot showed that chess grandmasters were generally able to identify the key features of a chess position, and that the number of features often did not even exceed the best moves. The chess grandmasters mainly differed from weaker players in their inability to appreciate the dynamics of complex chess positions. Chase and Simon (1973) described the performance of chess experts as a form of perceptual skill that is not based on rational calculation. They suggested that chess masters acquire a repertoire of 50,000 to 100,000 immediately recognizable patterns, and that this repertoire is used to identify and evaluate the most promising moves. To calculate all possible contingencies, Strong players need a decade of serious play to assemble a large collection of basic patterns, but of course they achieve impressive levels

515

515

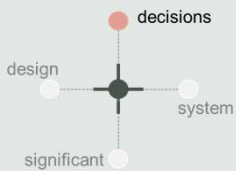
*"Rigor is not a substitute
for imagination."
—Gary Klein*

The rational in rationalize is a head-fake. And yet. We want to develop our reasons and reasoning. Make decisions with significant impact explicit, and probe and improve them.

"I worry about leaders in complex situations who don't have enough experience, who are just going with their intuition and not monitoring it, not thinking about it."

— Gary Klein

A Simple Model of a Decision



- Decisions



Decision



What we will do,
what approach we will take

Decisions

So we're talking about how we make better consequential (system architecture, organizational architecture, strategic, etc.) decisions. So let's start there. With a decision, which we'll model, as one does, with black box or abstraction.

How does a decision come into view? In the previous module (Sense/Make Sense), we explored situation or context awareness and orienting to the landscape, identifying where action and leadership is needed. It is helpful (as we explore and clarify, and also as we document, the decision) to briefly describe the situation prompting the decision.

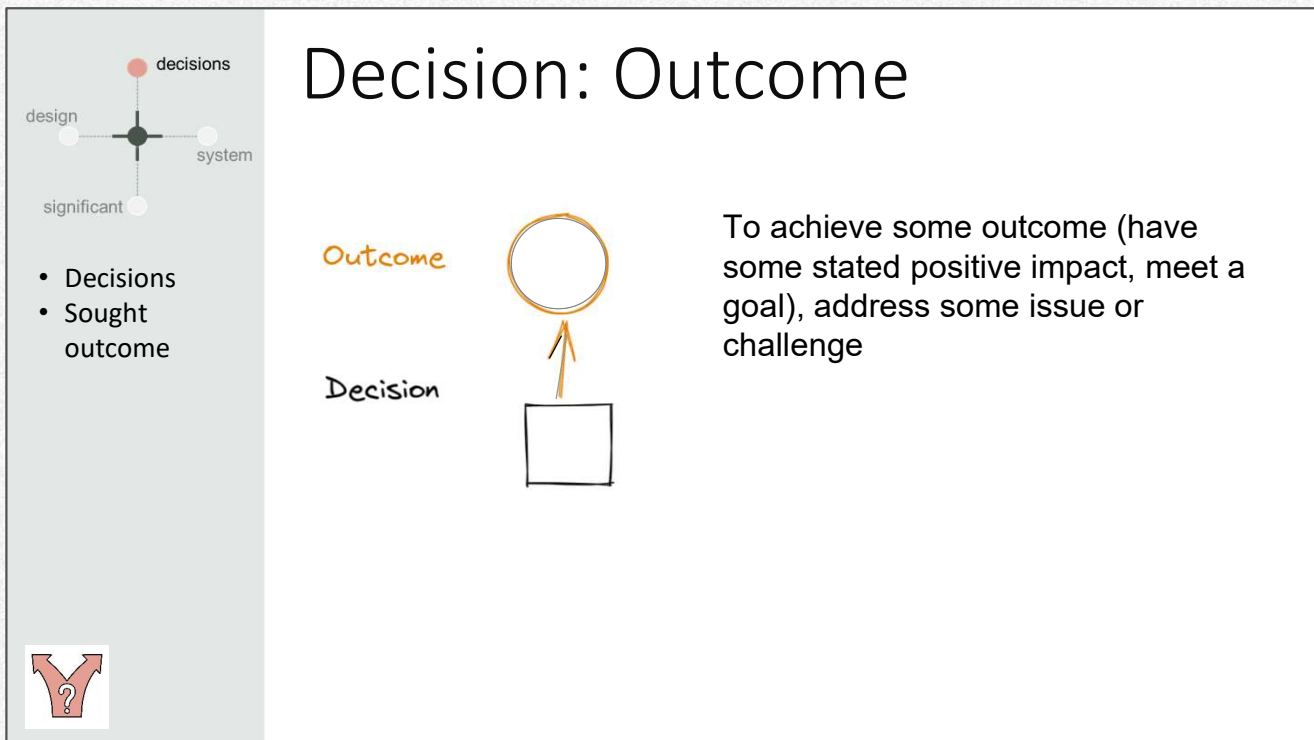
The Anatomy of a Decision

To understand something like a decision, and what factors in making a consequential decision, the structure (and diagram of structure) isn't enough — though how we structure our thinking about decisions, focuses attention and indicates what we seek to bring into view, for consideration. It helps to understand and frame the problem or situation, separately from identifying the solution or decision options, and determining relative fitness to the situation.

Of course, whether a decision "sinks or swims," depends on much, including the socio-political context, and how we influence and are influenced.



Image Source:
<https://en.wikipedia.org/wiki/Anatomy>



Outcomes

In the context of a technical decision, an outcome may be a capability we need to build for users or the business or for the system (logging, or co-ordination and consistency mechanism, etc.), or a system property (quality attribute) we want to improve (scaling or latency or some other aspect of availability as we improve as demand grows or grows in new regions, etc.). Or it may be some issue (or risk) we face in the dev or devOps organization, that we want to address for ourselves, and see benefit to others in the organization. (This is often enough the case, that some decision templates use "issue" rather than "outcome" and it may even be separated out.)

The outcome sought, frames the problem that the decision addresses. It identifies what we are concerning ourselves with (as we explore and make this decision), and why.

The framing of the outcome or problem is itself a (set of) judgment call(s), as it helps bound the consideration space or frame the situation that we are attending to. Because it bounds the consideration space, we want to hold the frame somewhat loosely, at least to begin with, as we explore options (and possible reframings that bring other options into view).

Speaking of judgment calls, can we stop here?

For intentional, considered decisions, what is our intended outcome, goal, or objective? What does the decision seek to make true in this context or situation?

Problem: What are we trying to solve specifically? I try to be as specific as possible here. Since people have read the context, they can now understand the questions much better.

Opportunity: Here, I describe why solving this problem is essential and how it improves things for the business or the user.



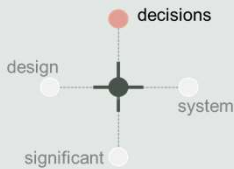
Question to decide on: Describe in a sentence what question you are trying to answer.

From Indu Alagarsamy, *Document your product and software architecture decisions*, <https://domainanalysis.io/p/document-your-product-and-software>

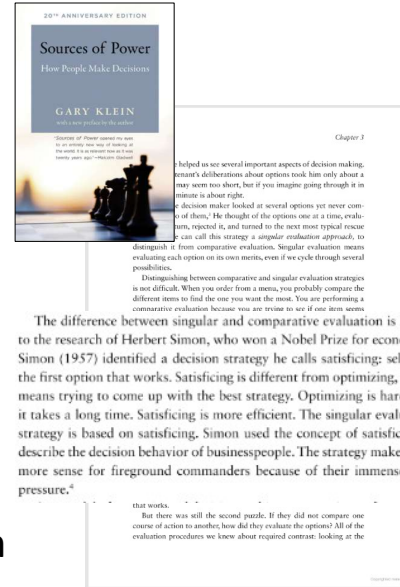
Experience

“Their experience let them identify a reasonable reaction as the first one they considered, so they did not bother thinking of others. They were not being perverse. They were being skillful.”

– Gary Klein



- Decisions



Expertise and Decisions

Gary Klein and colleagues have studied experts and the way they make decisions, coming to the conclusion that often experts make decisions not by extensive analysis, but based on experience recognizing situations, and reaching for a workable solution in that situation, and proceeding. And he points out this isn't being perverse, it's being skillful. So where we have seen something play out multiple times, and have learned a reliable response set, that may be enough. We make all manner of satisficing decisions in the course of doing things.

In an interview with McKinsey's *The Quarterly*:

Gary Klein: It depends on what you mean by "trust." If you mean, "My gut feeling is telling me this; therefore I can act on it and I don't have to worry," we say you should never trust your gut. You need to take your gut feeling as an important data point, but then you have to consciously and deliberately evaluate it, to see if it makes sense in this context. You need strategies that help rule things out. That's the opposite of saying, "This is what my gut is telling me; let me gather information to confirm it."

The Quarterly*: "Is intuition more reliable under certain conditions?"

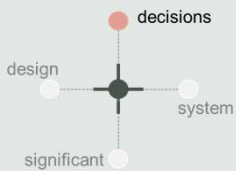
Gary Klein: "We identified two. First, there needs to be a certain structure to a situation, a certain predictability that allows you to have a basis for the intuition. If a situation is very, very turbulent, we say it has low validity, and there's no basis for intuition. [...] The second factor is whether decision makers have a chance to get feedback on their judgments, so that they can strengthen them and gain expertise. If those criteria aren't met, then intuitions aren't going to be trustworthy.

Most corporate decisions aren't going to meet the test of high validity. But they're going to be way above the low-validity situations that we worry about. Many business intuitions and expertise are going to be valuable; they are telling you something useful, and you want to take advantage of them."

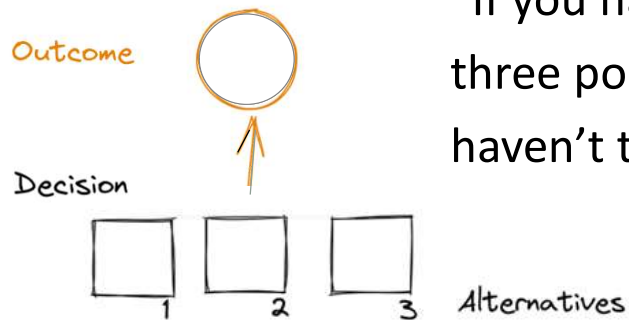
Daniel Kahneman: "One of the problems with expertise is that people have it in some domains and not in others. So experts don't know exactly where the boundaries of their expertise are."

Source*: <https://www.mckinsey.com/business-functions/strategy-and-corporate-finance/our-insights/strategic-decisions-when-can-you-trust-your-gut>

Decision: Alternatives



- Decisions
- Sought outcome
- Alternatives



“If you haven’t thought of three possibilities, you haven’t thought enough.”

— Jerry Weinberg

Architecturally Significant

However. For strategically or architecturally significant decisions, we want to explore what our options are.

“architecturally significant” decisions: those that affect the structure, non-functional characteristics, dependencies, interfaces, or construction techniques’

“One ADR describes one significant decision for a specific project. It should be something that has an effect on how the rest of the project will run.”

— Michael Nygard, Documenting Architecture Decisions

That is, if we’re making a technology choice that will shape other choices in an impactful way, or we’re coming up with, designing, an approach to building a system capability or mechanism, or addressing some critical issue or challenge, we want to be intentional about it, to bring consideration to bear, and also to be able to visit and revisit our reasoning. So we bring options or alternatives into view. Moreover, as pointed out by Wisen Tanasa, it’s helpful to consider whether a hybrid of what we’d thought of as alternatives, positions us better in the tradeoff space.

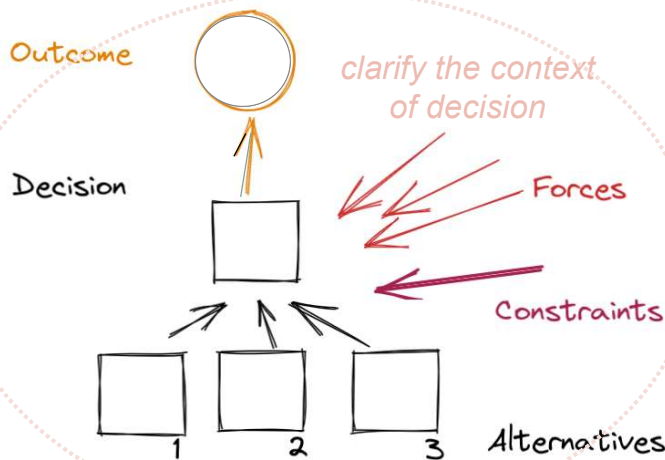
Each option considered, is described briefly, outlining trade-offs, and impact. Typically the option proposed/adopted comes first in this list. You may want to describe why the other alternatives were not chosen, as it is part of the reasoning/argumentation (later when looking back at the decision, others can see which objections were already taken into account). — source?

While we’re at it, think of 3 ways we might be wrong!

*“Eric Evans had recommended having at least 3 options in a proposal
1 option leads to evaluation of that option: yes/no
2 options lead to comparisons of A vs B
3 options suggest there are a set of possible solutions, of which there may be more.”*

— James Maier

Forces and Constraints: Decision



“A force [..] is [..] anything that has a potential non-trivial impact of any kind on an architect when making decisions.”

— Uwe van Heesch et al

Forces, Considerations, What Impinges

Whether we're weighing options or developing alternative approaches, the situation has a bearing — we need to identify and characterize the relevant forces, contributing factors, governing variables, complications, assumptions, constraints.

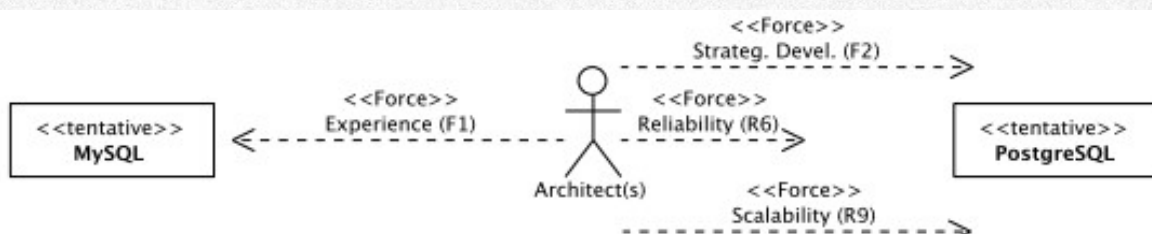
“A force [..] is [..] anything that has a potential non-trivial impact of any kind on an architect when making decisions.” We're using force to mean something *impactful*, impinging on an architectural problem. Forces arise in the system or its environment — the operational, development, business, organizational, political, economic, legal, regulatory, ecological, social, etc.) context or situation.

“Forces arise from many sources; most often from requirements, but also from constraints, architecture principles and other “intentions” imposed upon the system; including personal preferences or experience

of the architect(s) and the development team; and business goals such as quick-time-to market, low price, or strategic orientations towards specific technologies (see [9] for an empirical study on influence factors on software architecture).”

“The architect evaluates each architectural decision alternative in the context of the forces. As a result of the evaluation, a force can have a positive, negative, currently unknown, or neutral impact on the architect with respect to a decision; it either attracts the decision maker towards a specific decision alternative, or it repels the decision maker from an alternative, or it has no effect.”

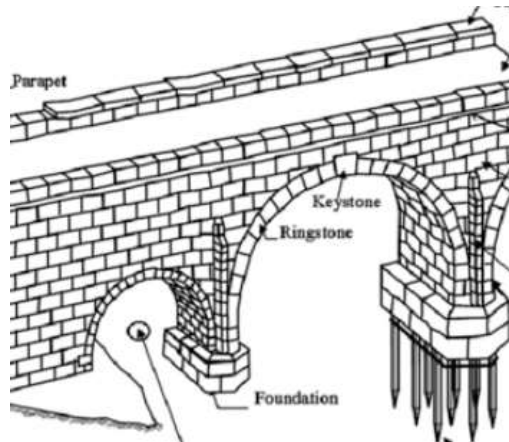
— Uwe van Heesch, Paris Avgeriou, Rich Hilliard
Forces on Architecture Decisions – A Viewpoint



Designing a Bridge: Forces

“Masonry is strong when you try to squeeze it and weak when you try to stretch it. In the jargon, it’s strong in compression and weak in tension. That has consequences.”

— Brian Marick



What forces are relevant, and how does our design behave under those forces?

Forces (in bridges and buildings)

“Suppose you’re required to build a bridge, meaning a horizontal surface over some empty space. The simple solution would be a series of walls to hold up the floor of the bridge. OK, but now consider a horizontal floor span going from one wall to its neighbor. The span is supported on its ends, but unsupported in the middle. Gravity pulls down on the middle, creating tension. Since masonry is weak in tension, you’d have to have short spans and a lot of walls, which would be expensive, plus awkward if you want any traffic to go under the bridge – like, say, boats going down a river that it spans.

The arch is a clever solution to this problem. Consider an arch made out of bricks. Each brick mostly presses down on the brick next to and below it, meaning that all the bricks are in compression. The full weight of the structure supported by the arch is delivered to the feet of the arch. Some of the force is vertical, which is opposed because the arch is sitting on the ground. Some of the force is horizontal, which can be opposed if there’s the leg of another arch of the same weight pushing against it - like in a bridge with multiple arches. Or, for the end two arches of the bridge, by anchoring them to a strong enough foundation. Essentially the forces transferred down the arch to the ground are balanced by forces *from* the ground, and it’s all compression, all the time.”

Source: Brian Marick, “Christopher Alexander’s forces”

Forces push or pull, attract (gravity) or repel, inhibit (friction or drag, resistance) or propel (applied, spring), can be used to hold in place (tension, compression, ..)

“a flying buttress [...] uses the power of downward compression to balance an outward force. (Or something like that - I'm not an architect.)”

— Brian Marick

Potential Forces

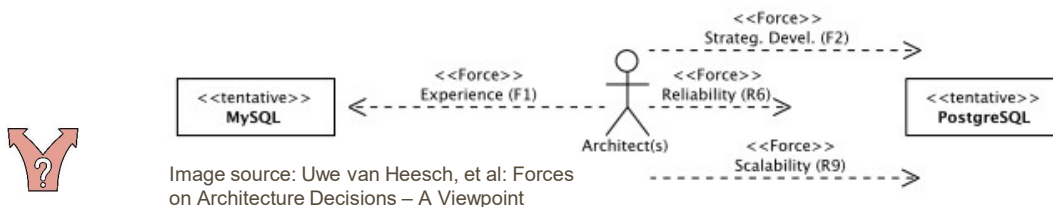
What shapes this decision space?

- user or business need and criticality
- experience/capabilities
- system properties (availability, reliability, observability, auditability, ..)
- Costs (cost to build, license costs, etc.)
- Time: how long will this last? (short term impact, or something users/engineering will have to live with for long time)
- Time: engineering effort

What pushes or pulls, distorts or organizes, resists or attracts, ...

- Time: time to value; feedback loops and learning cycles
- complexity, technical challenges
- team autonomy, independence, co-ordination costs
- consistency (UX, devX, OpX)

What attracts or repels, inhibits or induces, creates friction, drag and inertia or flow, prevents or fosters, impacting the outcome in good or bad ways?



Forces, Considerations, What Impinges

As we’re making a decision, and then as part of conveying it, we want to understand (and convey) what has substantive bearing on the decision. This means characterizing the situation in terms that are relevant to the decision.

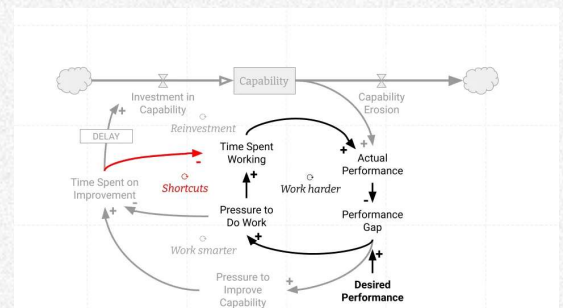
Whether we call them forces (or “forces” as an analogy) or factors or criteria, we’re exploring what matters (in the use, development, operations, or broader context or situation), and how much it matters. And how that interacts. And what doesn’t matter, that we thought might, and why.

What concerns do stakeholders have, that we need to take into account and address with this decision? Now, and as various stakeholders have to “live with” it. What makes a difference to the outcome and attributes of the solution, and how do the various alternatives we’re weighing impact these concerns and goals (and objections)?

We want to identify what is consequential or significant to this decision, and get this out where we can see it, and reason about it and do so together, and bring others in to the process of identifying what matters and what interacts, and how we can best resolve the forces and tradeoffs (due to interacting and even conflicting goals and constraints).

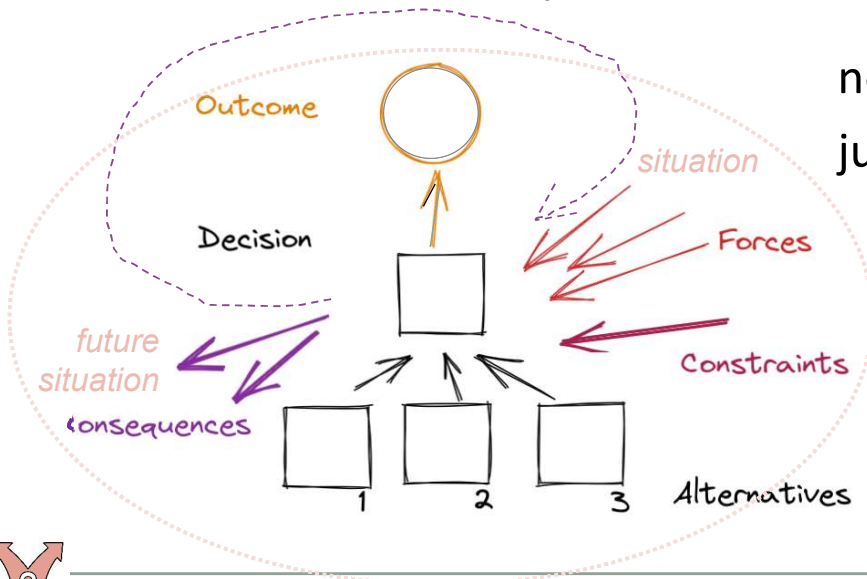
Note about the diagram on slide: F2 is development of strategically important capability — it will become critical to the business, given the evolution of the system (increasingly large datasets, complicated queries, ...).

What matters to our situation? To our stakeholders, now and over time?



Causal loop diagrams can be used to explore effects (what is impacted, and how). Image source: Xavier Briand, What is technical debt? And how to talk about it?

Decision: Consequences



nothing you do has just one effect

Consequences and Second Order Effects

Further, in addition to the outcome or positive impact we're directing our attention to achieve by making this decision, and the forces and demands impinging on it, we also need to take into account, and weigh, the effects or consequences of the decision and arguably, the consequences of the consequences or second order effects.

"Second-order thinking is the practice of not just considering the consequences of our decisions but also the consequences of those consequences. Everyone can manage first-order thinking, which is just considering the immediate anticipated result of an action. It's simple and quick, usually requiring little effort. By comparison, second-order thinking is more complex and time-consuming. The fact that it is difficult and unusual is what makes the ability to do it such a powerful advantage." — fs.blog

Consequence Scanning is an important approach to discovering the wider impacts of our technical products and choices. Ask:

- What are the intended and unintended consequences of this product or feature?
- What are the positive consequences we want to focus on?
- What are the consequences we want to mitigate?

More at <https://doteveryone.org.uk/project/consequence-scanning/>

*"If you give a mouse a cookie,"
"he's going to ask for a glass of milk."*

"When you give him the milk,"

"he'll probably ask for a straw"

"When he's finished, he'll ask for a napkin."

"Then he'll want to look in the mirror

To make sure he doesn't have a milk mustache."

— Laura Numerof

Consequences: What Changes?

“a consequence is just a
statement about how the future
will differ from the past”

— Michael Nygard



“As you make this list of consequences, try to avoid coloring your thoughts about the consequences by what your intentions are. [...] once the change is made your intentions are irrelevant. Only the resulting system state matters.”

— Michael Nygard*

* Michael Nygard, Consequences are not Pros or Cons,
<https://www.michaelnygard.com/blog/2020/06/consequences-are-not-pros-or-cons/>

How Does the Decision Change Things?

In a blog post* that is a great companion to his post describing how he recommends documenting architecture decisions, Michael Nygard observes that we tend to focus on pros and cons, and can lean into justifying the choice we have or want to make. He notes:

“Instead, I suggest we first describe simply consequences, not benefits or problems. That’s because a consequence is just a statement about how the future will differ from the past. [...]”

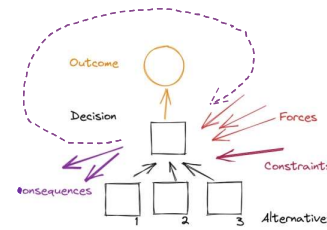
Whether you judge that consequence to be a “pro” or “con” depends entirely on your relationship to the change. If you perceive the change as an improvement to status quo then you call it a “pro”. If you don’t like the version of the future which includes that consequence, then you call it a “con”. That means labelling a consequence as a benefit is subjective. It describes the relationship of you and the change.

What about the changes that you don’t particularly like or dislike? The ones that are neither “pro” nor “con”? Most of the time those don’t get written down at all!

I recommend that you begin by listing the consequences. Find all the ways that the future will be unlike the past, if we choose that path. Look for second-order effects — the consequences of the consequences.”

Tip: Document Decisions

← aka thinking it through



Title: short noun phrase

Context: desired outcomes and the forces at play
(probably in tension)

← Alternatives

Decision: describes our response to these forces

Status: proposed, accepted, deprecated or superseded

Consequences: describes the resulting context, after
applying the decision

From: Michael Nygard, Documenting Architecture Decisions, Nov 2011

Architecture Decision Records

ADRs are a way to share decision reasoning. Examples of ADRs at:

<https://web.archive.org/web/20210506014629/https://upmo.com/dev/decisions/0010-som-synthetic-monitoring.html>

Michael Nygard's Template:

Title These documents have names that are short noun phrases. For example, "ADR 1: Deployment on Ruby on Rails 3.0.10" or "ADR 9: LDAP for Multitenant Integration"

Context This section describes the forces at play, including technological, political, social, and project local. These forces are probably in tension, and should be called out as such. The language in this section is value-neutral. It is simply describing facts.

Decision This section describes our response to these forces. It is stated in full sentences, with active voice. "We will ..." Justify the decision.

Consequences This section describes the resulting context, after applying the decision. All consequences should be listed here, not just the "positive" ones. A particular decision may have positive, negative, and neutral consequences, but all of them affect the team and project in the future.

The consequences of one ADR are very likely to become the context for subsequent ADRs. This is also similar to Alexander's idea of a pattern language: the large-scale responses create spaces for the smaller scale to fit into.

"In practice, our projects almost all live in GitHub private repositories, so we can exchange links to the latest version in master. Since GitHub does markdown processing automatically, it looks just as friendly as any wiki page would."

— Michael Nygard

"Writing about your decision forces you to explain your thinking." — fs.blog, Creating a Decision Journal

Decisions Are Trade-offs

“For me, “engineer” means knowing that all decisions are tradeoffs. It means considering both upsides & downsides of each technical choice, and doing so with explicit consideration of the larger system context.”

– Sarah Mei



TECHNICAL
LEADERSHIP



“When you build a bridge, you don’t build it as a perfect structure that will never collapse. Instead you build it to withstand 500 year winds, 200 year floods, 300% expected maximum load, etc. If you didn’t make these design trade-offs, every bridge would be solid concrete [...] Engineering is all about making these compromises”

Pragprog.com/articles/the-art-of-tradeoffs

Decisions Entail Tradeoffs and Tradeoffs Don’t Stay Their Lane _(ツ)_/

As a manager in IT or product development, our decisions don’t just impact teams but the systems they create. We see this in Conway’s Law:

“The basic thesis [...] is that organizations which design systems [...] are constrained to produce designs which are copies of the communication structures of these organizations.”

-- Melvin Conway, How Do Committees Invent?, 1968

Likewise, as an architect, the choices we’re making are technical, but the impacts don’t remain neatly in the technical space. The tradeoff space isn’t just about qualities that impact developer experience, or security properties or operational complexity, but user experience and partner experience through properties of the system in use. And more. So we investigate the upsides and downsides of our technical decisions, in these various contexts.

We want to surface the trade-offs inherent in our decisions, both to better understand the decision space, and because we may be able, or need, to contend with the downsides of these decisions explicitly, to offset them.

An Example

Read (next slide) and identify

- the Decision
- the Outcome(s)
- Forces (identified, and not)
- Consequences (identified, and not)

Spotify

This organization structure, combined with the global-ish nature of JavaScript in the browser, has made us build the desktop client UI out of many small, self-contained web apps called *Spotlets*. They all run inside Chromium Embedded Framework, each app living within their own little iframe, which gives squads the ability to work with whatever frameworks they need, without the need to coordinate tooling and dependencies with other squads. While this approach has the disadvantage that we have many duplicate instances of different versions of libraries, increasing the size of the app, but it offers the *massive* advantage that introducing a library is a discussion between a few people instead of decision that involves ~100 people and their various needs. Not only would such a big discussion extremely time-consuming and hard, it would also force us to use a least-common-denominator approach to picking libraries, instead of picking the ones specifically tailored to the problem domain of each squad. Considering the size of a single song compared to the size of a JavaScript library, this trade-off is a no-brainer for us. [Mattias Petter Johansson, on Quora \(2017\)](#)



Weigh tradeoffs

To make better decisions, we need to weigh and resolve the inherent tradeoffs — the upsides and downsides of the choice or approach.

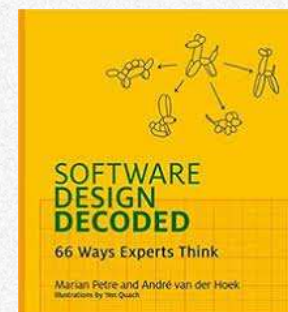
That is, to make tradeoffs intentionally, we need to identify and characterize the tradeoff space. What is relevant to the decision is a (set) of judgment calls. How we balance and resolve the tradeoffs is again a set of judgment calls (though of course there may be precedent in the industry, or in our experience, that gives us more to go on).

	DESIGN ALTERNATIVE 1	DESIGN ALTERNATIVE 2	DESIGN ALTERNATIVE 3
DEVELOPMENT TIME	****	**	*****
COST OF ACQUIRING COTS COMPONENTS	*****	***	****
REUSE OF OUR EXISTING CODEBASE	*****	*****	****
COMPATIBILITY	*****	****	*****
PERFORMANCE	*****	*****	*****
SECURITY	***	*****	****

"strive for the least worst combination of trade-offs"

— Neal Ford et al

Design Alternatives image from:



We're going to consider an example (next slide)

A remote presentation like this has some advantages in terms of screen distance, but for those who can't read the screen we will recap some of the main points in just a bit. Now, though, we will take a moment to allow a chance to read the text on the next slide, and identify the decision, the outcome, the forces impinging on this situation (those identified in the description, and those your experience is prompting) and consequences or effects of this decision.

Spotify

This organization structure, combined with the global-ish nature of JavaScript in the browser, has made us build the desktop client UI out of many small, self-contained web apps called *Spotlets*. They all run inside Chromium Embedded Framework, each app living within their own little iframe, which gives squads the ability to work with whatever frameworks they need, without the need to coordinate tooling and dependencies with other squads. While this approach has the disadvantage that we have many duplicate instances of different versions of libraries, increasing the size of the app, but it offers the *massive* advantage that introducing a library is a discussion between a few people instead of decision that involves ~100 people and their various needs. Not only would such a big discussion extremely time-consuming and hard, it would also force us to use a least-common-denominator approach to picking libraries, instead of picking the ones specifically tailored to the problem domain of each squad. Considering the size of a single song compared to the size of a JavaScript library, this trade-off is a no-brainer for us.



Mattias Petter Johansson, on Quora (2017)

Example

Let's spend a moment and read the discussion (see slide above) from Mattias Peter Johansson on Quora, about Spotify (written in 2017).

Ref: <https://www.quora.com/How-is-JavaScript-used-within-the-Spotify-desktop-application-Is-it-packaged-up-and-run-locally-only-retrieving-the-assets-as-and-when-needed-What-JavaScript-VM-is-used>

One thing to note, is that this was written 5 years ago, about the past; things changed.

Spotify

Decision

Divergence

—

—

+

Team autonomy



This organization structure, combined with the global-ish nature of JavaScript in the browser, has made us build the desktop client UI out of many small, self-contained web apps called *Spotlets*. They all run inside Chromium Embedded Framework, each app living within their own little iframe, which gives squads the ability to work with whatever frameworks they need, without the need to coordinate tooling and dependencies with other squads. While this approach has the disadvantage that we have many duplicate instances of different versions of libraries, increasing the size of the app, but it offers the massive advantage that introducing a library is a discussion between a few people instead of decision that involves ~100 people and their various needs. Not only would such a big discussion extremely time-consuming and hard, it would also force us to use a least-common-denominator approach to picking libraries, instead of picking the ones specifically tailored to the problem domain of each squad. Considering the size of a single song compared to the size of a JavaScript library, this trade-off is a no-brainer for us.

Mattias Petter Johansson, on Quora (2017)

What do we Notice?

Our point here isn't to criticize Spotify's choices in that timeframe and point of the evolution (in the market and of the technology and organization), but to appreciate how, even in this narrative format, so much of the decision and considerations are being conveyed, and to explore the decision.

The decision: to use Spotlets, or small, self-contained apps within their own iframe

The outcome: increased team independence or autonomy

Positive effects (or forces): reduced cross-team coordination; speed of movement (so speed of learning)

Negative effect (or force): duplicate instances of different versions of libraries

Negative consequence: reduced cross-team communication; divergence among teams as a result

(These social costs and consequences are not just as a result of this decision, but the decision is part of a reinforcing loop.)

Negative consequence (not surfaced; potentially future): multiple versions of licenses and purchasing and security headaches (knowing what patches to roll out where)

Tradeoff? size of songs so dominates size of app, that they could make this decision to support team autonomy without perceived cost to user.

We see that allowing duplicate instances of different versions of various libraries enabled Spotify squads (teams) considerable independence, removing the need to coordinate with other squads on libraries and versions. Because song size so dominates considerations that it generally falls beneath the threshold of sensitivity for the user, the tradeoff of team freedom for app size is easily (in their view) within the design acceptance envelope.

So in this case, a technical decision is being made for organizational gain (lowering team coordination costs and increasing team's degrees of freedom) at the expense of app size, which works as long as it's below the app user's tolerance threshold for resource consumption.

We're building econo-sociotechnical systems, within econo-sociotechnical systems, and we need to factor this in, as we scan for forces, constraints and consequences (that we factor in as forces).

Impact of the Decision

Who gains? Who feels the pain? When (e.g., gain now versus pain in a year)?

... and impact on other humans/creatures/planet

Experience of others, including security and operations, and cost to business

UX and cost to user (/customer)

devX and cost of change



Different Impact in Different Areas

What this example highlights, is “what’s going on” in terms of what is being paid attention to in the decision, what the forces and tradeoffs are and what has not been drawn into explicit consideration, possibly because it isn’t yet a significant noticed force. And in particular, this important point: impacts (positive outcomes, as well as other positive and negative effects) and consequences (including downstream and future consequences) are borne by different sets of “stakeholders” – not just different persons or internal groups, but users (downloading the app and listening to songs), customers (paying bills), these people in different regions of the world, with different bandwidth and cost constraints. As well as different stakeholders within the organization, and not limited to developers.

But we would draw on experience to point out what to be watching for, as the situation evolves.

“Good engineering is less about finding the “perfect” solution and more about understanding the tradeoffs and being able to explain them.”

— Jaana B. Dogan

[Reflecting on the Ackoff video] “The systemic cultural and societal impacts of the software we build: I feel that especially in venture capital backed startups, the software industry is prone to not thinking in systems when it comes to the impact of what their products are doing — as opposed to the return on investment they have. From the harms of social media on mental health, to discriminatory bias in AI, I see many parallels with the notion of “doing the wrong thing right.” — Mike Stallard

Decision Space and Pareto Front

“Tradeoffs only occur when you reach [a] Pareto frontier.”

— Donald Reinertsen



Pareto efficiency

From Wikipedia, the free encyclopedia



Pareto efficiency or **Pareto optimality** is a situation where no individual or preference criterion can be made better off without making at least one individual or preference criterion worse off. The concept is named after [Vilfredo Pareto](#) (1848–1923), Italian civil engineer and economist, who used the concept in his studies of [economic efficiency](#) and [income distribution](#). The following three concepts are closely related:

- Given an initial situation, a **Pareto improvement** is a new situation where some agents will gain, and no agents will lose.
- A situation is called **Pareto dominated** if there exists a possible Pareto improvement.
- A situation is called **Pareto optimal** or **Pareto efficient** if no change could lead to improved satisfaction for some agent without some other agent losing or, equivalently, if there is no scope for further Pareto improvement.

Pareto Front

What we’re seeing in this example, is that, with respect to team degrees of freedom and app size on the one hand, and song size and by implication user experience and space and cost concerns, a Pareto Front has not been reached. These things aren’t being traded off for one another. We can improve team independence without decreasing user experience in an appreciable way.

Experts and Seeing Tradeoffs

From John Cutler*:

‘Ask an everyday driver about driving tradeoffs, and you’ll likely hear something like, “When you go around a corner, you need to trade off speed and control.” The mental model is “slow down just enough to keep control around the corner.”

A professional driver will think differently. Their mental model revolves around tire grip and temperature, the optimal racing line, throttle control, suspension, aero settings, brake balance, tuning the car for the track, and weight transfer management. They might point out that the amateur isn’t exactly wrong, but they might say, “At the end of the day, it boils down to tire friction, aerodynamic limits, mechanical limits, and human limits.”

Amateurs aren’t entirely wrong; they’re recognizing some inherent limitations. What differentiates professionals is their ability to approach these limits more closely and consistently without exceeding them. They have a deeper understanding of where the boundaries are and how to navigate them.’

“A threshold effect exists when there is a critical level of effort necessary to affect the system. Levels of effort below this threshold have little payoff.”

— Richard Rumelt

* John Cutler, “Dear Executive...” 2023,
<https://cutlefish.substack.com/p/tbm-250-dear-executive>

Trade-offs

Space-Time Trade-Off



□ Spotify Example:

↑ Size of app, to

↓ Co-ordination overhead between teams



TECHNICAL
DECISIONS

"A trade-off (or tradeoff) is a situational decision that involves diminishing or losing one quality, quantity or property of a set or design in return for gains in other aspects. In simple terms, a tradeoff is where one thing increases and another must decrease."

— wikipedia

Space-Time or Time-Memory Trade-Off

"Usually, a TMTO is developed to improve the speed of an algorithm by utilizing one-time work, which results in increased storage (memory) requirements when the resulting algorithm is executed. Of course, it is also possible to work in the opposite direction by reducing the one-time work at the expense of more work each time the algorithm is executed. The goal is to balance the one-time work (memory) requirement with the speed of the algorithm (time)."

— Mark Stamp, Once Upon a Time-Memory Tradeoff

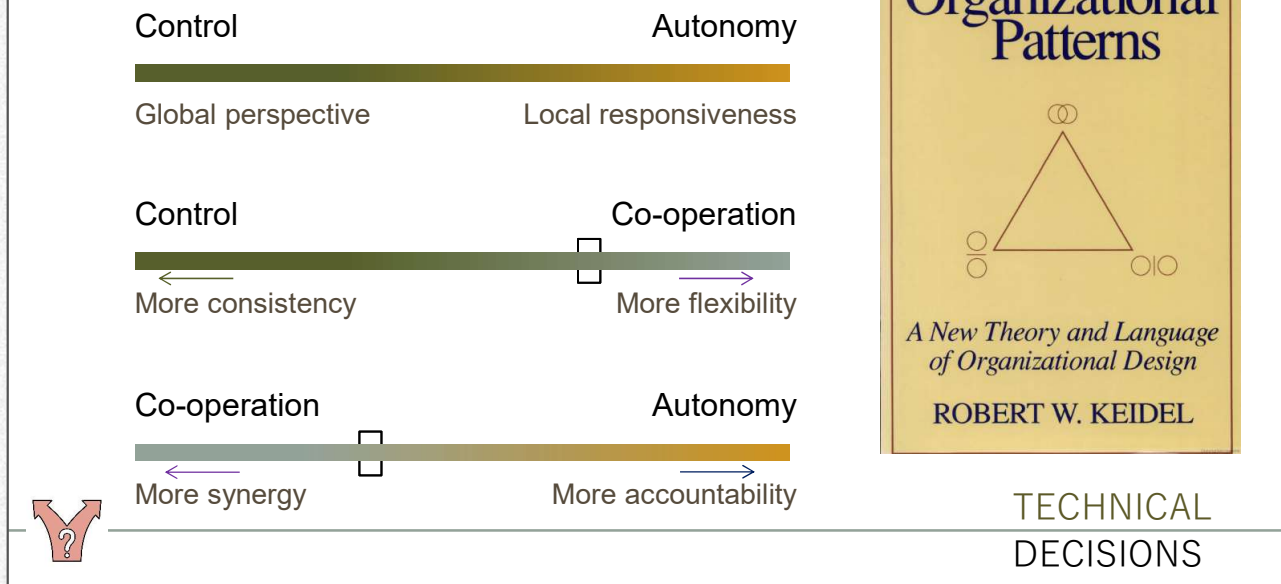
A classic illustration of the trade-off entails using a lookup table (uses upfront work and a lot of space to enable a fast lookup when the result is needed) versus calculating on demand (uses little space, but can take a long time at the point of demand, depending on the calculation).

Another space-time trade-off arises in data storage. If data is stored uncompressed, it takes more space but less time than if the data were stored compressed.

We're talking about this as a space-time trade-off, but it translates into a cost-performance (i.e., user experience) trade-off.

What are we giving up and what are we gaining? Do different groups gain and feel pain? Over different time horizons?

Trade-offs: Dyads



"For example, continuous evolution pulls against product stability[..]. Low-level decisions pull against strict process control"

— Eberhardt Rechtin and Mark Maier

Trade-Off Dyads (Picturing the Dilemma)

We have a trade-off when design variations improve one dimension (something we value, like a performance metric), but diminish another. Factor in multiple of these trade-off dimensions, and there is no unique optimal design; the choice lies in what is valued in that context.

By drawing the trade-offs out — making them visible — we can make judgments, and subject them to discourse to better assess impact and value.

Many trade-offs can usefully be thought of in terms of dyads: performance and cost (another way to frame the space-time trade-off); data confidentiality or security (via encryption) and performance; safety and cost; structural mass (for physical structures) and safety; usability or convenience and security; etc.

In *Seeing Organizational Patterns*, Robert Keidel considers organizational structures and interaction dynamics, and pivotal trade-offs underlying organizing choices.

These could be presented as the dyads shown (slide above).

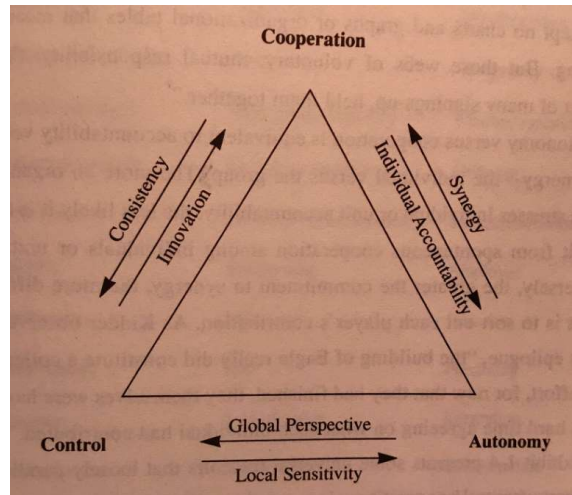
While considering pair-wise trade-offs can help understand the design space, it can obscure the tensions when multiple variables are simultaneously in play. Keidel points out that "every organization must blend autonomy, control, and cooperation." The trade-off space (the design options), is more usefully visualized as a triad, or triangle.

The multiple library versions example earlier, is missing impacts (eg security implications).

Trade-offs: Triads

“most organizational issues are a balance of three variables: individual autonomy, hierarchical control, and spontaneous cooperation. By learning to frame issues as trade-offs among these design variables, one can see underlying patterns”

— Robert Keidel



TECHNICAL
DECISIONS



A Trilemma of Trade-offs

According to Keidel, any particular organization will focus on at most two of autonomy, control, and co-ordination. (Attempting all three is an unstable form.) These are the organizational forms he identifies:

Organizations that are autonomy-based have as their distinctive competence adding value through solo performers; they are truly star systems. Example: any first-rate university.

Control-based organizations compete on the basis of their ability to reduce costs and/or complexity through global coordination. Authority, information, and initiative reside chiefly at the top levels.

A cooperation-based organization builds synergy across teams. The distinctive organizational competence is innovation through cooperation.

Probably the most familiar example of an autonomy/control hybrid is the divisionalized corporation.

A control/cooperation hybrid may be described as a "humanistic hierarchy." Top-down control remains essential but every effort is made to meld it with voluntary, lateral processes among individuals, functions, and units.

The autonomy/cooperation has the oldest roots. This combination goes back to the craft organizations of the late 18th century, which featured a blend of individual initiative and informal cooperation.

"Equally dangerous is an overemphasis on a single variable to the point that the other two are neglected. Autonomy becomes problematic when a relatively freestanding part-individual or organizational unit-overdoes its own thing."

— Robert Keidel

Trade-off Space

small self-contained web apps

single web app

- team independence, autonomy; devX
- speed (to market)
- lower inter-team communication costs

- system integrity (common/consistent UX; consistency and coherence)
- simplifies some things
- more inter-team communication (potential for shared understanding, ...)



More of this means less of this
(ceteris paribus)

Choices Among Options

When we are deciding among alternatives, we're deciding among the clusters of effects and consequences of those alternatives (like modular monolith or microservices; small self-contained web apps or single web app; etc.).

While the concept of "to decide" holds within it the notion of what we're deciding *not* to do, along with what we are deciding *to* do, part of (what we factor in) the trade-off space may include what it takes to mitigate the negative effects or downsides of the approach we go with.

Examples

We might seek to minimize downtime with rapid failure detection and recovery, but this incurs the overhead of continuous monitoring and detection. Additionally, automated detection and recovery mechanisms may be triggered by false positives (for example, a node acting as if it has failed, when it's just running slowly for a moment) or introduce performance degradation during failover. Balancing the trade-offs involves optimizing detection sensitivity and response times while minimizing false alarms and impact on performance.

"Two key questions I always advise people to reflect on [..]:

1. What happens if this succeeds? Does it make the [..] world better?

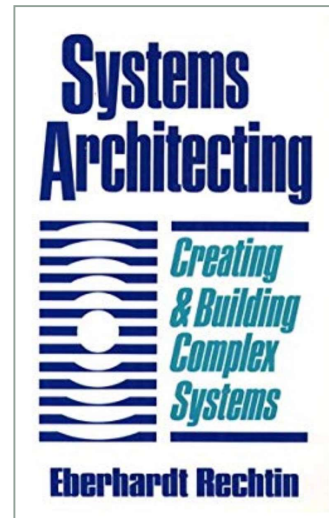
2. Who is harmed by the changes this causes? What would you choose to do if you loved them?

Every single choice gets easier if you know those answers.'

— Anil Dash

"A central tenant of the ecosystem approach is that the path to sustainability is one of tradeoffs. Science can illuminate the tradeoffs but a resolution, that is, the choice of path, is a political decision" — Michelle Boyle, et al

Design Force Field



TECHNICAL
DECISIONS

Tensions

Design has to balance tensions caused by different imperatives, needs, and perceptions.

"Some of competing technical factors are shown in [the figure in the slide above]. This figure was drawn such that directly opposing factors pull in exactly opposite directions on the chart. For example, continuous evolution pulls against product stability; a typical balance is that of an architecturally stable, evolving product line. Low-level decisions pull against strict process control, which can often be relieved by systems architectural partitioning, aggregation, and monitoring. Most of these tradeoffs can be expressed in analytic terms, which certainly helps, but some cannot"

Eberhardt Rechtin and Mark Maier

"design is the [...] structure or behavior of a system whose presence resolves or contributes to the resolution of a force or forces on that system. A design thus represents one point in a potential decision space."

— Grady Booch

"We're trying to find habitable zones in a large multidimensional space, in which we're forced to make regrettable, but necessary, tradeoffs."

— Robert Smallshire

Sources of Forces

“we build systems out of pure thought, in order to balance the static and dynamic forces of cost, schedule, functionality, performance, reliability, usability, and ethical implications”

— Grady Booch



Image source: Grady Booch

TECHNICAL
DECISIONS

Sources of Forces

"We do not analyze requirements; we construct them from our own perspective. This perspective is affected by our personal priorities and values, by the methods we use as orientation aids, and by our interaction with others" — Christiane Floyd

'The word "requirements" represents a fundamental misunderstanding of software. They're theories, at best.' — Sarah Mei

Design Envelopes

In engineering, we contemplate, weigh, and experiment to find the boundaries of the design envelope.

"Hard" requirements tend to be areas where our design envelope has less "give", so other parts of the requirements design have to flex.

"The better you understand the problem, the closer you can design to tolerances." — Dana Bredemeyer

We innovate by pushing the design envelope — extending the range of possible, into the adjacent possible.

[with reference to the slide:] "Of course they are categories: each describing a class of forces. For example, compatibility encompasses pressures that arise from legacy, frameworks, and standards" — Grady Booch

"Architecture is the set of design decisions that provide a reasonably satisfying resolution to the static and dynamic forces on the system." — Grady Booch

There is a multidimensional decision *space*. We want to surface not just options, but assumptions about forces in play.

"the force field of a software project starts with Requirements. Requirements are often categorized in some way, like "functional" and "nonfunctional", or "user requirements" and "system requirements. However, requirements of any kind [...] contribute to shape the overall field."

— Carlo Pescio

Forces in Dynamic Tension

Rasmussen's dynamic safety model describes the feasible operating space for a sociotechnical system.

Adapted here to explore interaction of code habitability and software habitability

Software* not habitable

privacy or security breaches, scalability failures, not accessible, poor fit to user purpose or need

* in operation and use

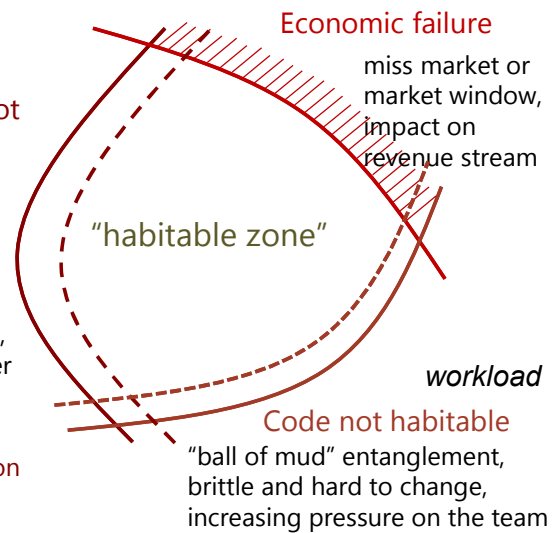


Image: Adapted from the Dynamic Safety Model presented in Cook and Rasmussen, 2005

Habitable Zone

The dynamic safety model was developed by Jens Rasmussen; adapted by Cook, Rasmussen, and others. It is described by Richard Cook in his presentation titled "Resilience In Complex Adaptive Systems" (Velocity 2013). This talk is available to watch on Youtube (under 19 minutes), and highly recommended.

We can combine the notion of habitable code and habitable software, adapting Rasmussen's dynamic safety model to design, to illustrate Rob Smallshire's point that "We're trying to find habitable zones in a large multidimensional space, in which we're forced to make regrettable, but necessary, tradeoffs." I'm not sure of the origin of the notion of code habitability, but it was Rebecca Wirfs-Brock who drew my attention to it. And in his keynote at OOPSLA in 1995, Christopher Alexander pleaded with our field to pay attention to the habitability of the software we create.

The idea that is being illustrated here is that if we push too hard to get features to market to stay away from the economic failure boundary, we may defer investments in code habitability and repair and in so doing increase developer decision fatigue and stress because of an overload of conceptual and decision burden with entangled code and hard to predict consequences of changing the code.

But some of the things we do to keep the code habitable may also keep us away from failures on the boundary of operation and use.

"Most software architects do not think of themselves accounting for social issues, but that is one of the characteristics of good architecture. Accounting for social issues gives designers an easier life, which gives the software a longer life."

— Alistair Cockburn

Over Time

'I've used 100% stacked area graphs to visualize tradeoffs or strategic allocation of "fixed" resources, where the allocation changes over time.'

— Juno Suárez

'I tend to use "graphic equalizer" with scaler x-axis so that you can overlay to compare and contrast snapshot values for variables for trade off.'

— Dawn Ahukanna

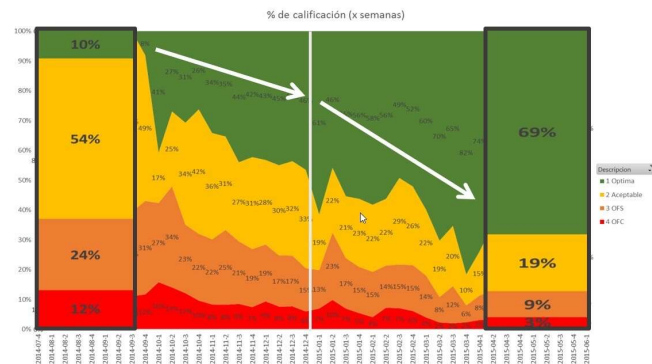


Image source: Juno Suárez, <https://hachyderm.io/@juno/110945173941162351>

Where the Forces Change over Time

I asked folk on mastodon what other visual forms they use to bring tradeoffs into view.

Dawn Ahukanna pointed out that many of our representations tend to be at a point in time. Dawn suggested: "'instance in time" snapshot metric for contrast and comparison with other snapshots. It's like taking a time-lapsed set of photographs/ sampling of a specific spot and turning it into an motion interaction where you can "pan through time."

Peter Gassner pointed to a neat prototype they developed for visualizing project constraints and dependencies:



<https://lab.interactivethings.com/confluence-diagram/#/>

And James Fairbairn: "I ask people more and more these days about their theory of change — like, understanding the complexity of this space, and how everything is a chain of causes and conditions, let's walk through how we think "X leads to Y" *actually* works..."

"Eg: on a platform team driving an enterprise technology migration, focusing time between focus areas like maintenance, new development, and support/training/customer success. Conditions and opportunities change over the migration lifecycle (adoption curve), and capturing these requires tradeoffs of team attention."

— Juno Suárez

Smart Decisions

The Smart Decisions Game highlights the tradeoffs inherent in each decision and across decisions



Images from: Smart Decisions Game site: <https://smartdecisionsgame.com/>

Bigtable
IOT / PATTERN / DATABASE
Massively scalable NoSQL wide-column database suitable for low-latency and high-throughput workloads. Integrates easily with Hadoop and Spark, it supports open-source, industry-standard HBase API.

Rechargeable with Energy Harvesting
IOT / PATTERN / POWER SOURCE
By harvesting energy from the environment, IoT devices can operate without need for manual charging. The source of energy will depend on the application.

CHARACTERISTICS:

- ★ Capacity — must be matched to power source so it doesn't run out too soon or take too long to charge
- ★★ Maintenance — energy harvesting device may require cleaning or upkeep, but no manual charging
- ★ Cost economy — cost of rechargeable battery plus energy harvesting device drive up the upfront cost, but lower maintenance cost from charging or replacing batteries
- ★★ Mobility — limited to similar environmental conditions where energy can be harvested
- ★ Ruggedness — exposing the energy harvester to the environment can cause failure points in the device's housing

SAMPLE USE-CASES:
Solar weather station, Piezo vibration monitor, Windmill on sailboat

The Smart Decisions Game

"Smart Decisions is a game that simulates the design process of software systems and promotes learning about it in a fun way." -- from the Smart Decisions Game website; but having played the game at SATURN, I agree. The game can be downloaded, and used in a team learning activity.

It's a good way to highlight for the team that each technology and related decision has its strengths and weaknesses, and architectures are not just about individual decisions, but weighing across the decisions for a fit to the context and purpose of the system. Further, there will not always be agreement on the approach to take, because the nature of tradeoffs is that they entail judgment about the strengths/weakness as well as the value of the outcomes, and the

degree to which the consequences (in other areas of the system, or its containing systems) need to be taken into account.

The SEI team has done important work in the system qualities and trade-offs space, including developing the Architecture Tradeoff Analysis Method:

"ATAM gets its name because it not only reveals how well an architecture satisfies particular quality goals, but it also provides insight into how those quality goals interact—how they trade off against each other"

Judgment Factors

We may notice where we're being constrained (that's where we've hit a point of tension in the tradeoff space). But discerning tradeoffs is very much a matter of experience and judgment.

"Because the situation is ill-structured, the goal cannot be optimization. The architect seeks satisfactory and feasible problem-solution pairs."

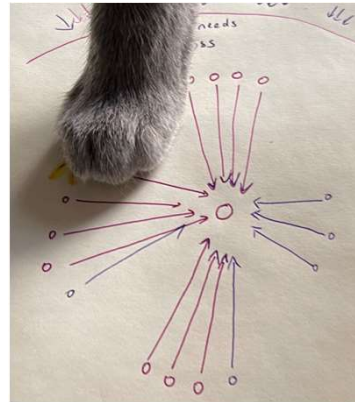
— Mark Maier and
Eb Rechtin

Real Talk

What (really) shapes this decision space?

- What are we avoiding (talking about)?
- What consequences are in "don't go there" zones?
- What forces feel too career-dangerous to write down?

Besides, we're addressing future impacts that are uncertain



... pawlitical forces...



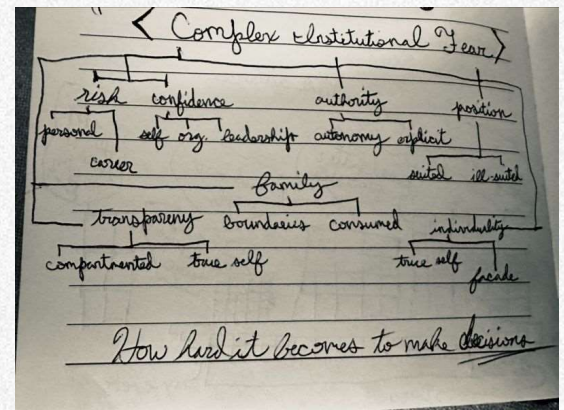
When Consequences have Consequences

Significant decisions impact the paths or options we have, but also change the possibility space of some areas of system context or environment. They create paths, and close off paths, for ourselves, for users, for others impacted. Decisions and situations have reciprocal effects on one another; design acts back as consequences, that we (may) take into account in making the decision. When this has to do with action now, and possible future consequences, it may cause indecision, or be costly, in organizational terms, to probe and discuss openly. There are no pat answers here. The culture of the organization overall, and the part of the organization involved, plays a role. We can point to the importance of psychological (or social) safety in creating a learning environment where implications can be probed, and responded to together. And we're weighing positive outcomes (intended direct effects, and as side effects or positive externalities) along with negative. In the context of uncertainty. Sometimes avoiding real talk may be about uncertainty/ambiguity or conflict avoidance, but restricting the consideration space may be due to decisions made elsewhere... Similar to learning from incidents, we need to be able to seek even conflicting perspectives, and explore options and impacts, and feed that learning back into the decision. While being pragmatic about uncertainty and the need to make decisions.

Part of what makes leadership and experience important here, is the willingness and ability to discern and take on these kinds of organizational challenges, and navigating them. (Caveats apply; alternately put: there is more to say, or nuance to add.)

*decision (n.) from decidere
"to decide, determine,"
literally "to cut off," from
de "off" (see de-) + caedere
"to cut" (from PIE root
kae-id- "to strike")

etymonline.com



https://www.linkedin.com/posts/christoph-erwilliams2018_complexsystems-journals-systemsexploring-activity-7159612234229301248-l4ip/

That's ... a Lot ... so

How do we clarify the situation and identify forces?

— modeling, canvases and structured conversations


How do we design and compare alternatives?

— modeling, canvases and structured conversations

How do we reach a decision?

— modeling, canvases and structured conversations

How do we build understanding and buy-in?

 — modeling, canvases and structured conversations

j k
ust idding
(but also not)

How We Work is Part of the Work

We've covered a fair bit of conceptual ground. The "what" of the Architecture (or other strategically significant) Decision Record indicates areas of work that are separable but intertwined. There's exploring the context or situation (sometimes this goes by "the problem"), with an emphasis on forces (or criteria) so that when we evaluate alternative approaches ("solutions") we can identify tradeoffs (identifying pains or costs we incur for what gains) and consider approaches against the desiderata we've established. But how?

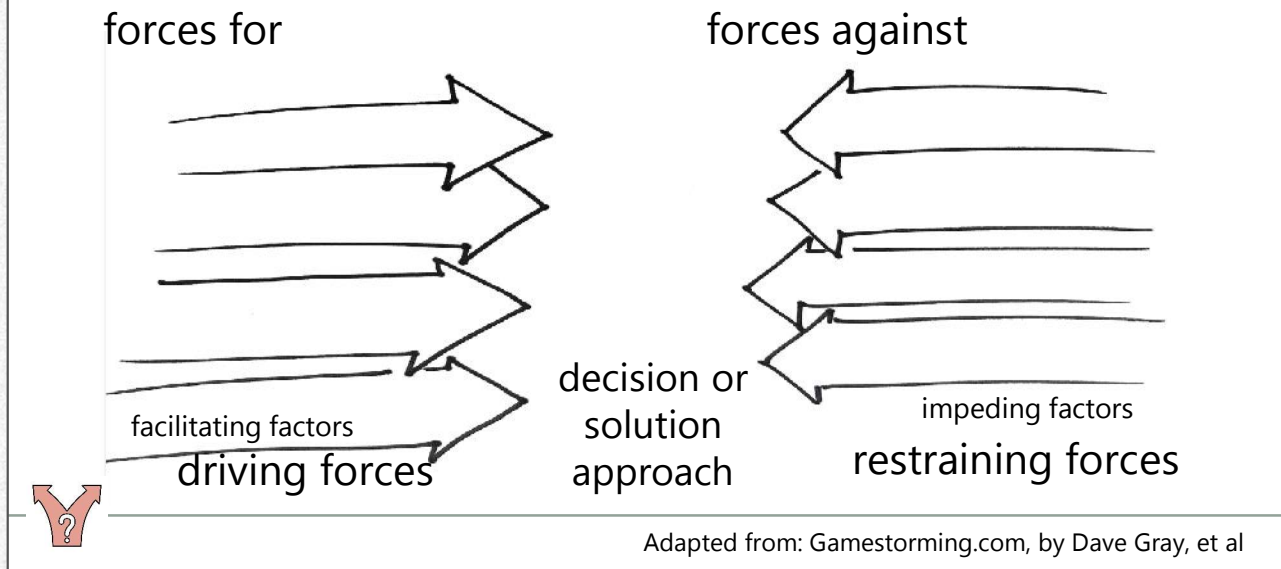
Ideally, we do this in a collaborative way, together with those who have perspectives and experience that inform our understanding of the situation, its challenges, and options. An informal session at a white board is generative, but canvases (such as the Force Field canvas from *Gamestorming* on the next page) and diagrams focus the discussion, while also drawing attention to areas the discussion might otherwise avoid or neglect. It's good to have them in the mix. It also builds a deeper understanding of the decision than one made in a "hub-and-spoke" way, where one person is the main center of thinking about the decision and puts ideas out for response. How we work, can get a good part of the larger work done, if we're strengthening the decision and building understanding and "buy-in" organically.

"You cannot coordinate purpose without developing purpose, it is part of the same process." — Mary Parker Follett

"I get it. Meeting culture sucks. It's too easy for people to thoughtlessly take each others' time, occupy standing slots, show off with performative teamwork, and generally suck your energy. Meetings feel like dead time. Meetings are time spent with people yet strangely devoid of social gratification. Meetings typically bore most participants — the greatest sin in knowledge work — and when they're over, nothing has changed except us all being that much closer to retirement. [...]"

*But what if, hear me out, what if the *only* work that matters in a knowledge economy happens when we are together?."*
— Elizabeth Ayer, *Meetings *are* the work*

Force Field Analysis



Force Field Analysis

Kurt Lewin did pioneering work in group dynamics, Action Research, and organizational development.

Of particular interest to us here, is Force Field Analysis, using Force Field Diagrams, developed by Kurt Lewin. Lewin was interested in group and organizational change or adaptation, and forces holding the organization in quasi-equilibrium. Force field analysis is useful in the context of organizational change, but can also help visualize forces that any decision balances or compromises across.

'According to Kurt Lewin "An issue is held in balance by the interaction of two opposing sets of forces - those seeking to promote change (driving forces) and those attempting to maintain the status quo (restraining forces)." Lewin viewed organizations as systems in which the present situation was not a static pattern, but a dynamic balance ("equilibrium") of forces working in opposite directions. In order for any change to occur, the driving forces must exceed the restraining forces, thus shifting the equilibrium.

The Force Field Diagram is a model built on this idea that forces - persons, habits, customs, attitudes - both drive and restrain change.'

http://www.valuebasedmanagement.net/methods_lewin_force_field_analysis.html

"If you want truly to understand something, try to change it."

— Kurt Lewin*

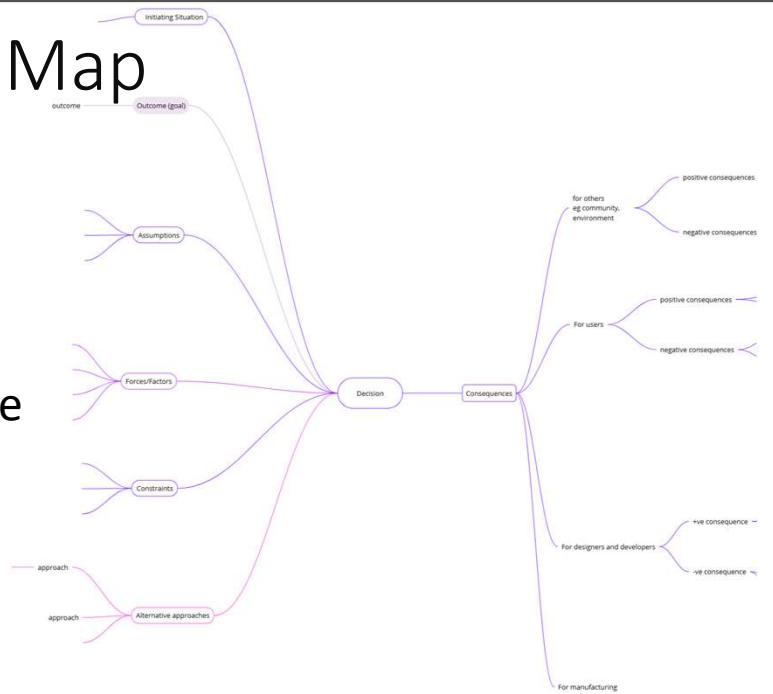
* this quote is attributed to Kurt Lewin by Charles Tolman in *Problems of Theoretical Psychology*,

"Any given change may be a positive for some people and a negative for others. Who benefits from the way things are now? Who will benefit from a change? Who will experience the negative space, and what will that negative be?"

— Esther Derby

Decision Mind Map

The decision template is itself a great structure for conversation. Here, the template is in “mind map” like form



Decision Template as Mind Map

Using the decision template in textual form, or mind map* form, encourages attention to the different facets of the decision. Starting with the context (or situation) and sought outcome, and identifying forces, constraints, assumptions, before turning to alternatives and describing options or design ideas. (We will return to explore this more fully later in this module). The idea with the mind map, is to tease out — adding tendrils and following threads, exploring down a path. By having the emerging picture on a whiteboard or (miro, etc.) frame, we’re encouraged to add relevant detail to other areas of the map, whenever such a detail emerges in the course of the conversation. For example, if we notice we’re making some assumptions while we explore forces or alternatives, we add those in. It is just as well to notice that as we explore the decision space, the outcome may come into clearer focus (and even shift, as we understand the problem better). As we explore consequences, we might find ourselves revisiting alternatives and exploring trade-offs and consequences further. The “how” is non-linear. We document the decision so that it reads in a way that conveys clarity. But getting to clarity means some holding space for ambiguity that uncertainty and complex interactions kicks up.

"Our job [...] how to devise methods by which we can best discover the order integral to a particular situation."

— Mary Parker Follett

* Mind Maps were popularized by Tony Buzan. Simon Wardley protests such a use of the word “map.” Perhaps we can call it a Decision Root Ball (haha).

That's... a Lot...Which Decisions?

"The value of every decision we make depends on the context in which we make it. In *The Lord of the Rings*, Frodo's journey to destroy the ring is meaningful inside the context of Middle Earth. Otherwise, he's a short, hairy guy with apocalyptic hallucinations."

— Diana Montalion



Decisions are central, and it is a great template, but you can just hear the captain in the cockpit yelling "pull up, pull up" — we'll run into a veritable forest of decision trees if we speed too far too fast down that runway just now. Which decisions?

*'wisdom is the ability to know what "it depends" on' **

Which Decisions?

When I first read Martin Fowler's "Who needs an Architect?" column, I playfully summarized it as:

Which decisions does the architect make?

Architecturally significant decisions!

What is architecturally significant?

The architect decides!

Yes, it's a tautology. Yes, I repeat myself. To playfully indicate that architects discern what is architecturally significant, to determine where to focus architectural (*system* design) attention.

As systems become more complex, we mean "the architects [plural!] decide", and note that decisions that prove to be architecturally significant aren't all known or knowable upfront. The system evolves. Many people are involved, making decisions that shape the system. Still, architects bring experience, context awareness and system understanding — and a *system* design purview — to judgments about what is architecturally significant.

Judgment. Can we say more?

* <https://x.com/vladikk/status/1134124637925892096>

Which Decisions?

“Software architecture is the set of design decisions which, if made incorrectly, may cause your project to be cancelled.”

— Eoin Woods



SOFTWARE
ARCHITECTURE

Which Decisions? Those that Contribute to Significant Outcomes

Further, architecture decisions are those that we need to make, to ensure the integrity of the system being built. Where integrity includes:

- *structural integrity*: design to make the system hold up under anticipated forces (staying within the operating safety boundary), and limit, and limit the consequence of, failures (so matters like security and discovering and limiting breaches; etc); matters of robustness (detecting component failure and fast restart with limited impact; etc) and resilience (supporting adaptive capacity),
- *design integrity*: conceptual integrity and requisite coherence (for example, developing sufficient common ground and understanding to work within more independent teams yet build a coherent system), and
- *organizational integrity*: making decisions consonant with our shared values even as we learn more what we value, and what impact we have, and want to have.

Architecture interprets the identity of the system in technical terms, and sets direction for and makes key system design decisions to enable that identity. It informs and is informed by design for users (determining capabilities the system offers users and other systems), and fit to context and to purpose. And it sets context for further (technical) design decisions. Since these decisions shape — shape system identity, shape teams, shape context for further decisions — they become harder to reverse. So it's important to notice which decisions are of this nature (yes, judgment factors).

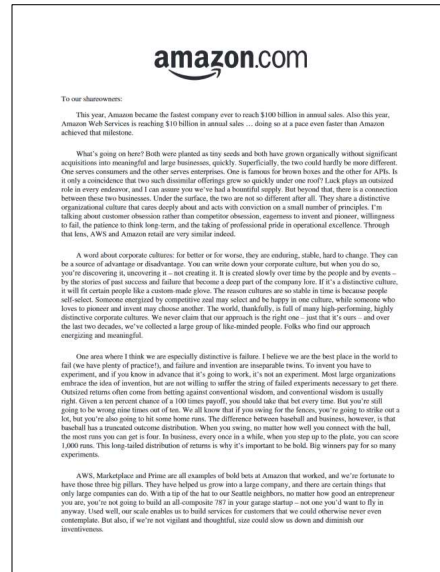
“Engineering is the design and making of - systems- with integrity.”

— Alan Kay

Irreversible Decisions

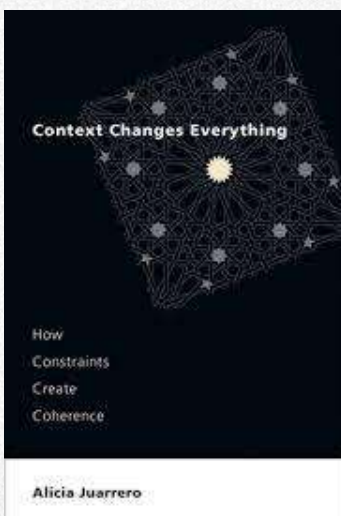
“Some decisions are consequential and irreversible or nearly irreversible [...] and these decisions must be made methodically, carefully, slowly, with great deliberation and consultation.”

— Jeff Bezos



SOFTWARE
ARCHITECTURE

“One common pitfall for large organizations – one that hurts speed and inventiveness – is “one-size-fits-all” decision making.”
— Jeff Bezos



No “One Time to Rule Them All” Decision Making

So, strategy and architecture are about scope and impact, and not something that is simply determined by being done upfront — that is, by timing. Rather, the other way round. If it’s strategically or structurally significant, we want timing to factor in decision making. Is this something we need to pay attention to now? Why?

We’re using judgment to decide on the timing of decisions. And one way to inform this judgment, as pointed out by Sidharth Masaldan, is to consider risk. What is highest risk and needs our (scarce!) expertise, perspective, attention and time now? And what do we need to enable (by deciding and building)? Yes, in the sense of enabling constraints.

No “One-Size Fits All” Decision Making Either

In his 2015 letter to Amazon shareholders, Jeff Bezos made this important distinction between irreversible and reversible decisions, emphasizing that consequential irreversible decisions need to be made with great deliberation and consultation.

Source: <https://www.sec.gov/Archives> **Not all decisions are equal. What differences make a difference?**

Irreversible Decisions

"If you walk through and don't like what you see on the other side, you can't get back to where you were before."

— Jeff Bezos



SOFTWARE
ARCHITECTURE



Attending to Irreversible, Consequential Decisions

Shane Parrish collected together a useful series of decision making heuristics in a twitter thread. Here are several (the numbers are Parrish's) that we've selected for their bearing in the case of more consequential decisions [and we've added a few notes]:

17. Put things on a reversibility/consequence grid — irreversible and high consequence decisions likely require more time. The rest of the time you can usually go fast.

Source:

<https://twitter.com/farnamstreet/status/1026105498372845571>

10. The rule of 5. Think about what the decision looks like 5 days, 5 weeks, 5 months, 5 years, 5 decades.

11. Let other people's hindsight become your foresight. [Do the research; draw on expertise.]

13. Ask what information would cause you to change your mind. If you don't have that information, find it. If you do, track [it] religiously.

We need to make those decisions deliberately, attentively

22. Walk around the decision from the perspective of everyone implicated (shareholders, employees, regulators, customers, partners, etc.)

26. Ask yourself "and then what?" [and "what if?" and "what else?"]

Source: Shane Parrish (@farnamstreet), on twitter, 5 Aug, 2018

'Legacy code is often defined as "code that makes more design decisions than the team working on it".'

— Ángel Siendones Sillero

Reversible Decisions

“But most decisions aren’t like that – they are changeable, reversible – they’re two-way doors.”

— Jeff Bezos



SOFTWARE
ARCHITECTURE



Reversible Decisions

It’s worth highlighting two takeaways from Bezos’s insights here:

- where we can, make decisions reversible — reduce the cost of change.
- pay particular attention to consequential irreversible decisions — attend to those that have high cost of change

“If you’re good at course correcting, being wrong may be less costly than you think” — Jeff Bezos

Reversibility Approaches

In *Taming Complexity with Reversibility*, Kent Beck outlines several approaches used at Facebook for making changes smaller, and getting feedback more rapidly, so decisions can be tried out and assessed, and reversed if they don’t pan out well (enough), before they become entangled in other decisions, expectations and habits. These include:

- *Development servers.* Each engineer has their own copy of the entire site. Engineers can make a change, see the consequences, and reverse the change in seconds without affecting anyone else.
- *Code review.* Engineers can propose a change, get feedback, and improve or abandon it in minutes or hours, all before affecting any people using Facebook.
- *Internal usage.* Engineers can make a change, get feedback from thousands of employees using the change, and roll it back in an hour.

Source: Kent Beck, *Taming Complexity with Reversibility*

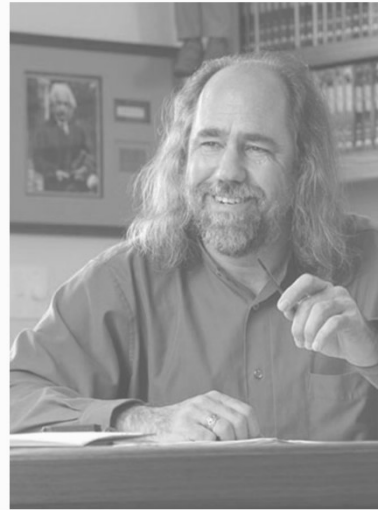
In part, these satisfy the second of Palchinsky’s Principles:

“when trying something new, do it on a scale where failure is survivable” — Peter Palchinsky

(Ir)reversible Decisions

“Architecture represents the significant design decisions that shape a system, where significant is measured by **cost of change**.”

— Grady Booch



(ir)reversibility of decisions
↑
high cost of change
↑
low(er(ed)) cost of change

SOFTWARE
ARCHITECTURE

The Point?

When we talk about cost of change in the context of architecture, we're typically thinking about the cost to make changes to the system. But cost of change plays in, in different ways. As the system takes shape, other systems develop expectations of our system – some implicit and some explicit, some critical and constraining, others not so much. As our system becomes embedded in expectations and commitments and reliance on its role in the broader ecosystem, it becomes hard to “reverse” or back out of shaping decisions. We might want to relate this to “sheering layers,” but it's at least good to recognize that we seek stabilities, even as we seek to adapt and evolve.

We have this interplay between decisions made early or next, to bring the benefit of those decisions forward, and decisions deferred to retain options. Even decisions about where we start, have consequences. We canalize pretty quickly. That is, we reduce the space of designs that are reachable. We gain an identity, internally and in the market. That shapes in ways we notice and don't — we make assumptions about value to customers and users, about system capabilities we're creating and so need to build in our teams, and so on. Sure, we (or the market) test(s) our theory of value — in so doing, shoring up the assumptions we proceed on. As users integrate our system into their workflows and systems, they build up expectations or assumptions about how things work, and ought to. As do we. There are a myriad ways our systems become coupled and resist change. Pretty soon, we call them “legacy systems” in that wry sense of a legacy we both value (or at least depend on) and regret.

“A good architecture reduces disruptive change. For example, if a on-the-wire protocol has version support you can do this. If it was forgotten in the architecture then the change is more disruptive or very inefficient.”

— Martin Thompson

“When reversibility is important to you, that's part of your context. The decision section should state what you're doing in light of that context. (Pilot project, wrapping interface, abstraction layer, etc.)

— Michael Nygard

(Ir)reversible Decisions

“Realized that the word “context” is shorthand for the cumulative effect of all the past decisions that we cannot change now. Decisions about what business we're in, which clients we serve, what compromises we made, where we've invested time and effort, and where we didn't. All of it adds up.

And here and now we are deciding things that will become tomorrow's context.”

— Elisabeth Hendrickson



Elisabeth Hendrickson
@testobsessed@ruby.social

Been thinking a lot about context lately. As in:

Q: What's the right way to...?

A: It depends

Q: Depends on...?

A: Context

Realized that the word “context” is shorthand for the cumulative effect of all the past decisions that we cannot change now. Decisions about what business we're in, which clients we serve, what compromises we made, where we've invested time and effort, and where we didn't. All of it adds up.

And here and now we are deciding things that will become tomorrow's context.

Dec 07, 2023, 12:40 · 45 ★ 53

Adaptive Capacity ... and Entanglements

Software is highly mutable. Humans lend adaptive capacity to our sociotechnical systems, allowing us to evolve them into astonishingly complex, and useful systems. One characterization of legacy systems:

“Legacy systems are valuable because they continue to exist; they wouldn't continue to exist if they weren't valuable.” — Kevlin Henney

Nonetheless, our systems tend to canalize – internal structures are adapted to fit shifts in context, but that fit comes at a cost, including becoming embedded in other systems that rely on them, and resist change. The 737 MAX story is illustrative of forces in tension...

“So when Boeing designed the 737 MAX, they were trying to balance two conflicting requirements. [Accounts differ: bigger engines for fuel efficiency or for range.] The other was to keep the design sufficiently similar to the existing 737 aircraft that pilots wouldn't need a new type rating [which aircraft pilots are allowed to fly]. But it turns out those new engines on the 737 MAX were actually so big they wouldn't quite fit under the wings. They couldn't redesign the airframe to make the wings higher, otherwise it wouldn't have been a 737 any more, so instead they mounted those new engines a little further forward and a little higher than the old ones. And this is where it gets complicated. That new engine placement introduced handling problems – it meant that when you open the throttle, the aircraft had a tendency to stick its nose up in the air. And that's bad, because if the nose goes up too high the plane is going to stall. And so the solution to this was software. Specifically, a software system called the MCAS – the Maneuvering Characteristics Augmentation System.” — Dylan Beattie, *The Cost of Code*, 2019

Continually adapting, but the possibility envelope is shaped by prior decisions.

“The law of stretched systems:

every system is stretched to operate at its capacity; as soon as there is some improvement, for example in the form of new technology, it will be exploited to achieve a new intensity and tempo of activity.

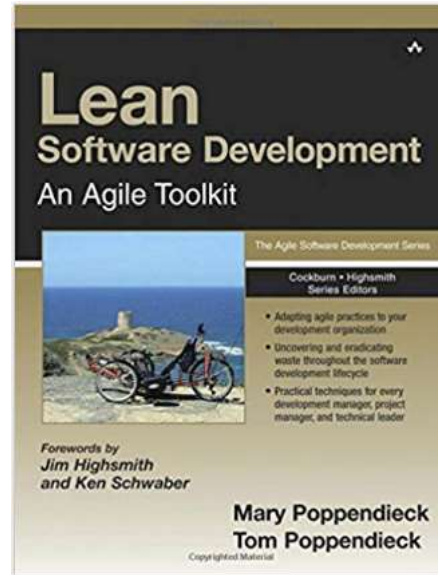
— David Woods

David Woods, *Laws that Govern Cognitive Work*, 2002

When? Last Responsible Moment

“the last responsible moment
[:] the moment at which failing
to make a decision eliminates
an important alternative.”

— Mary and Tom Poppendieck



Last Responsible Moment

Jeremy Miller on delaying decisions until the last responsible moment:

“The key is to make decisions as late as you can responsibly wait because that is the point at which you have the most information on which to base the decision.”

And Jeff Atwood:

“Deciding too late is dangerous, but deciding too early in the rapidly changing world of software development is arguably even more dangerous. Let the principle of Last Responsible Moment be your guide.”

Source: <https://blog.codinghorror.com/the-last-responsible-moment/>

Eb Rechtin and Mark Maier:

“Build in and maintain options as long as possible in the design and implementation of complex systems. You will need them.”

Some wry? @nonspecialist@aus.social got you:

- “any decision you make now will be wrong
- you have to make a decision now, or things will be worse
- if you don’t make a decision, it will be made for you and you’ll have to live with the consequences”

*“delay commitment until the **last responsible moment**, that is, the moment at which failing to make a decision eliminates an important alternative. If commitments are delayed beyond the last responsible moment, then decisions are made by default, which is generally not a good approach to making decisions.”*

— Mary and Tom Poppendieck

YouArentGonnaNeedIt (often abbreviated **YAGNI**, or YagNi on this wiki) is an ExtremeProgramming practice which states:

“Always implement things when you *actually* need them, never when you just *foresee* that you need them.”

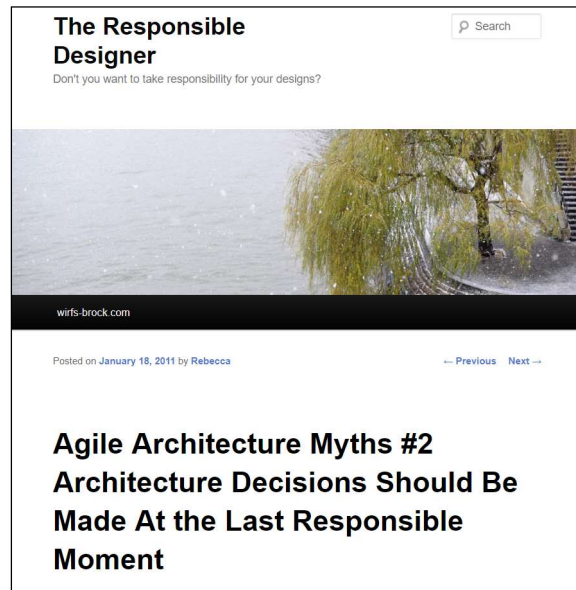
<http://c2.com/xp/YouArentGonnaNeedIt.html>

When? Earliest Responsible Moment

"I prefer to make decisions when they have positive impacts. Making decisions early that are going to have huge implications isn't bad or always wasteful. Just be sure they are vetted and revisited if need be."

— Rebecca Wirfs-Brock

"I prefer calling that opportune moment of when it is reasonable to decide, the Most Responsible moment ... as it is based on your judgment of the context, the situation, the risks, and everyone impacted by that decision." — Rebecca Wirfs-Brock



Creating Ground Under the Feet

Some decisions, like strategy and architecture decisions, create context for further decisions, establishing relationships, and reducing the decision space. This is good. It reduces the overload of overwhelming ambiguity and uncertainty, by narrowing the space and putting stakes in the ground. Now we can probe and test, to see how we're doing. We make certain key decisions early, to "put ground under our feet." Huh? Ground? Metaphorically speaking, but to be able to move forward, we have to start to shape the space, gain traction. More metaphors.

We need to decide what we are going to do (next, and at all, and if we want to be proactive about cohesive and concerted action, where we are headed), and how.

"I believe that you can and should look ahead. And that most developers, given half a chance, are pretty good at incorporating past experiences and making anticipatory design choices."

— Rebecca Wirfs-Brock

We may make ad hoc decisions implicitly on the fly without considered reflection, but some of our decisions are going to cleave the design space, ruling some opportunities out. This will be true whether they are implicit or explicit, considered, reasoned and probed, or made on the fly on guesses or without even knowing there were other choices we could have made. Better, if we anticipate they'll be highly consequential, if well considered.

You know the adage: "What's the best time to plant a tree? 20 years

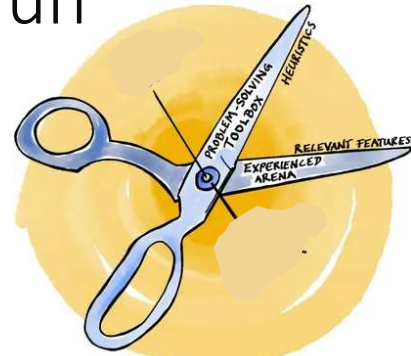
ago. What's the second best time? Now." Well, that's true, unless we don't need a tree. And there isn't something more critical to do now. But the point is important too — trees can't be moved so they constrain and set context for other landscaping decisions and they take a long time to grow, so to have the benefit of a bigger tree, we need to start as soon as we can.

As Mayoor Salva pointed out, opportunity cost is a useful concept to draw on here.

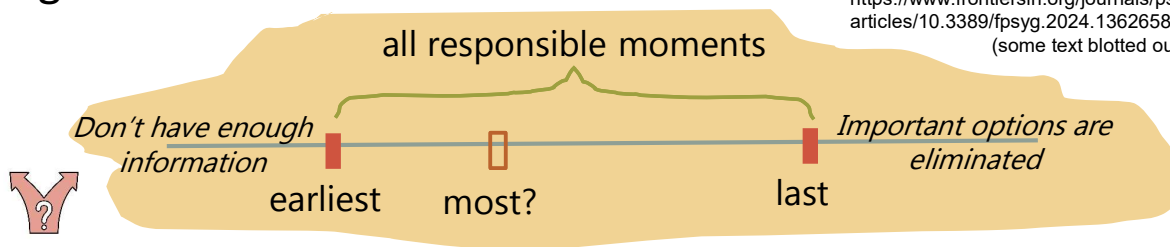
When? Judgment, again huh

The point isn't that we *know*
what is earliest, last, and most

It's that we explore what we
gain and risk



Simon's scissors Image source: Jaeger et al
<https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2024.1362658/full>
(some text blotted out)



When? Think About It

If there is some time frame in which we can "responsibly" make (significant) design decisions, there is some "earliest" and some "last" "responsible moment" — conceptually, anyway. We don't *know* where those points are, but the point is more about (for significant decisions) exploring (just enough) the tradeoffs... of earlier benefit from the decision (being put into play) versus knowing more later, and retaining degrees of freedom longer. What depends on the decision, and is held up? What is risky to defer, or to move forward on without learning more?

"What skills would we need in order to *not* have to make this decision until later?" (Kent Beck)

And! What should we bring forward, and for what reason? And some of those reasons are engineering reasons and some are market/user facing reasons. So what skills do we need to develop, to think strategically about the difference that makes a difference here? (Where "strategically" is relative to the scope at which we are designing.)

"To make sense of such an ill-defined and open-ended world — in order to survive, thrive, and evolve — the organism must first realize what is relevant in its environment. It needs to solve the problem of relevance." — Johannes Jaeger et al, 2024

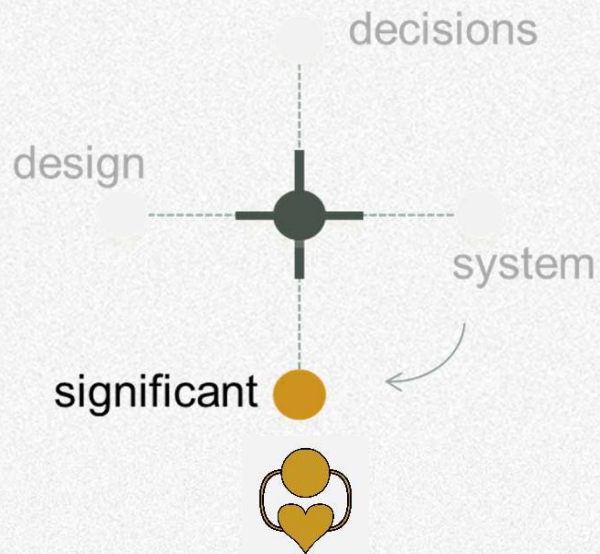
(<https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2024.1362658/full>)

What do we pull forward because it underpins, and what do we push out, because we need to learn more, etc....

"Oh yeah, this is a golden year for least responsible decisions."

— Einar W. Høst

Significant



Architecture

Systems

Design

Decisions

Decisions Across Boundaries

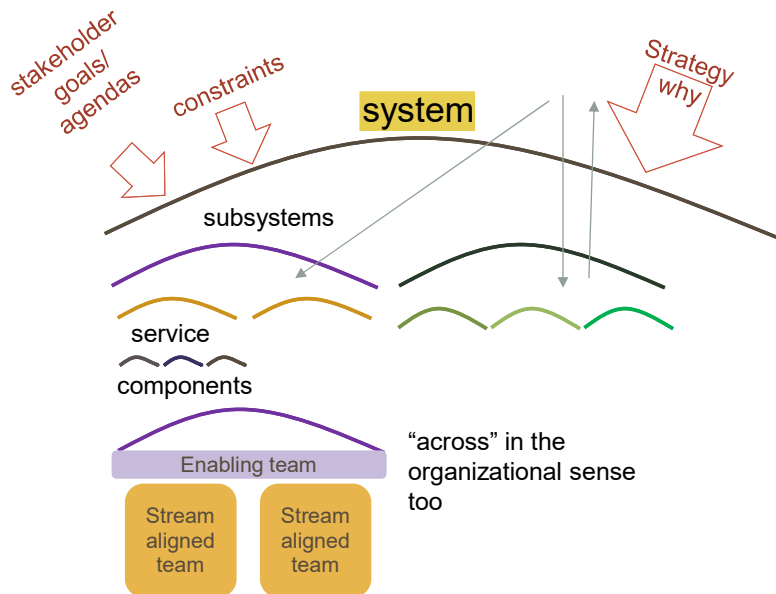
Software architecture is:

“significant design decisions that **shape a system**”

– Grady Booch

“the decisions that need to be made from the perspective of the system, to achieve sought system outcomes”

~ Dana Bredemeyer



Systems: Cohesion, Integrity and Leadership

Recap: Our systems, and organizations, are complex, or grow to be (Lehman’s Laws). Organization design, like other system design, entails a set of tradeoffs to weigh and balances to strike. Organizations have sub-entities because we organize to focus, to build and leverage capabilities, to get work done.

Communication costs – in terms of time but also in terms of focus of attention. Diverse perspectives are important to innovation; too many perspectives diffuses attention, increasing cognitive burden and demands on relationship fostering.

Interdependencies cost in terms of potential for delays as well as interactions and relationships which need to be established and maintained.

So we seek to identify responsibility boundaries so teams can be more independent. And yet we want to create systems with structural and design integrity – that is, conceptual integrity, as well as robustness where it matters, and resilience or adaptive capacity. And organizational integrity (matters of ethics, and social and environmental responsibility).

Decision making in strategy and architecture is about setting direction and context, so that decision making and work at more narrow scope, produces something coherent at broader (system or system-of-systems) scope. Without these decisions, we have piecemeal contributions which fail to add up to a system with integrity. These decisions have impact across boundaries (and their associated arenas of responsibility), and it takes organizational will (determination, because they are hard and other things compete for attention), and a commitment to understanding the decision and its ramifications, to follow through.

Participation in decision making helps build understanding and a sense of priorities, but broad participation in every decision doesn’t scale. So “higher level” decisions (decisions that impact across boundaries) need to be attended to and made in a smaller decision setting (a few people), but advocated for and shared in a way that brings others along, so that impacted work is consonant with these decisions. That is, decisions that impact others’ work across boundaries, entail leadership across boundaries.

Decisions Across Time

“One (of many) ways to think of product [and systems] work is to imagine a series of interlocking and related sense and respond orbits....it is all happening NOW, but the orbits range in terms of length...”

— John Cutler

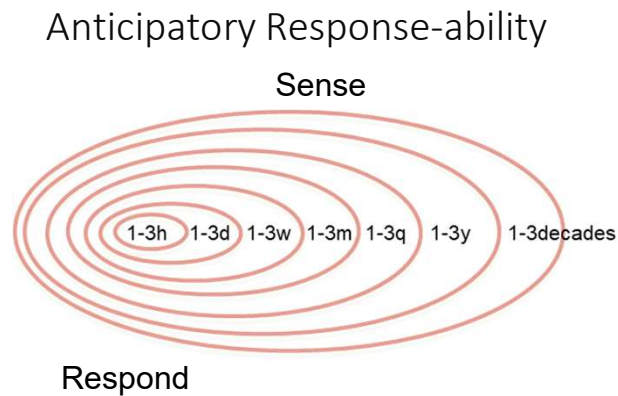


Image and quote source: <https://twitter.com/johncutlefish/status/1571582435598675970>

Implications for Horizons of Concern

One way to think about strategic significance, has to do with what shapes the ecosystem and system possibility space. What decisions lay down more or less binding “tracks” – create constraints, and relationships and value flows. What decisions are long horizon “bets,” that set us up for years of value creation and transformation, and enable viability and establish identity, but also bind us into expectations and ecosystem (legacy) relationships that are harder to vacate without damaging market relationships. And what are local decisions we can adjust to and away from quite readily.

A decision with broad impact across the organization, that underpins a myriad subsequent decisions and hence shapes the outcome possibility space over time, has many social and technical implications.

We introduce these concepts of scope and timespan, to offer some language and distinctions around scopes of influence and impact. A leader in a small group setting is working with qualitatively different challenges (in terms of complexity, uncertainty, feedback loops) than a leader working across groups within an organization, or across organizations.



“the cost of change from an executive, is completely different from the cost of change from a development team”

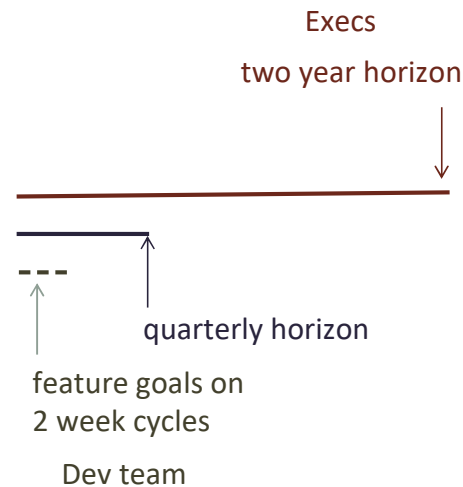
— Jabe Bloom

Image Source: Pavel A Samsonov,
<https://twitter.com/PavelASamsonov/status/1296818042928861184>

While, in general, we're seeking to shorten feedback loops, some roles are expected to make decisions with longer horizons

Time Span of Discretion

A person's time span of discretion is about the (time and complexity related) span of the work they have discretion (decision power) over.



Reference: *Requisite Organization*, by Elliot Jaques

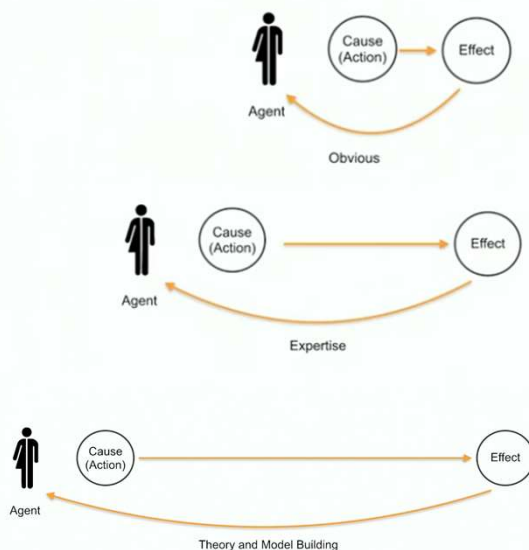
Leadership and Time/Scope

Seniority generally comes with increasing scope of concern (across systems, of systems, and, with more seniority, more impact on ecosystems). Increases in scope mean we're with dealing with greater complexity, and need expertise and experience that is rooted in the technical but is increasingly strategic and organizational. And we're dealing with longer time horizons, so more uncertainty.

Elliott Jaques' concept of time span of discretion/span of complexity provides a way to talk about roles and decision span. Those with shorter time span of discretion (and more narrowly scoped decision frames) are making decisions with more immediate impact and conscribed decision autonomy (e.g., the time horizon for completion of work made visible to others on the team or management, may be days or weeks). More senior roles are paying attention to longer term outcomes, across more of the organization. More hinges on what decisions are made, and not. All of these different scopes of concern take attention and cognitive bandwidth, and demand experience and expertise, but the focus shifts from more immediate observable effects, to making judgment calls under greater uncertainty and complexity. (That said, the essentialism aspect of Jaques work is... hard pass.)

While "time span of discretion" flavors the concept with what decisions we have discretion over or power to effect, Yvonne Lam draws attention to what timespan infuses our work and so draws/shapes our focus: "different entities (orgs, roles, etc.) have a span of time in which they can effect change, so that's the span of time to which they tend to pay attention." What I'm attending to, shapes what I perceive and attend to.

"thought about it as time travel: the higher up you go, the more you live in the future. As a senior eng you live 1-2 sprints out. A manager, 1-3 months. A director 3-9 months and so on. " — Danielle Leong



Source: Jabe Bloom, *Whole Work: Sociotechnicity & DevOps*

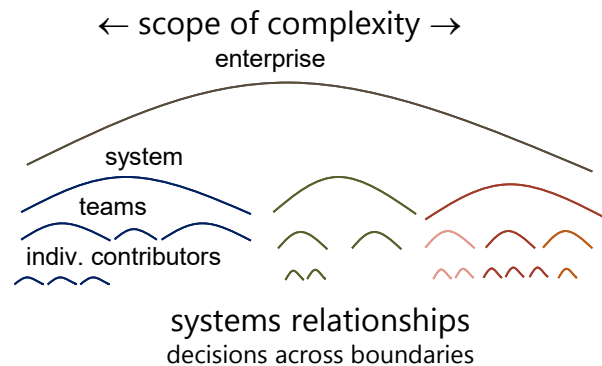
"As a punk-ass programmer, I'd grumble about 'management.' Well, they have a job to do, and it's a really difficult job."

— Kent Beck

Scope or Span of Complexity

Scope of complexity is about the span of responsibility (taken on, or inherent to) and focus of attention of a role

Scope within a team evolving a (part of a) system is less than scope at the business unit level, for example.



Who is thinking about the system (across internal and system boundaries)?

Leadership Across (Scopes of Complexity)

The management hierarchy is an accountability hierarchy in the contractual, fiduciary and financial sense. It manages people but also resources, like getting funding early on, and allocating budgets across priorities, including new business creation, later on. Obvious, and yet we can overlook both the importance, work and attention required, and the stresses involved, in being responsible for keeping salaries paid, investors and boards satisfied, and making choices where outcomes may only be fully visible years ahead.

It is also a part of the (broader) communication and co-ordination network. Influence networks, or informal relationships, facilitate communication, creating alternate pathways in the organization, and can help to get cross-boundary things done with less bureaucracy. They may be largely invisible (the kind of thing where it would take many interviews to map the influence network out, and still miss much) until they kick into higher gear to effect or impede change.

There's also the network of relationships in place to get work done. We're going to focus on complex systems built, evolved and operated by several, or even many, teams. Some of the system spanning work is reflected in the management hierarchy;

some in technical roles that span, like architect roles; some is (ad hoc) "glue" work. As the span of responsibility increases in scope, from responsibility for some local part of a system, to responsibility across subsystems and systems, the compass (span) of complexity in technical and organizational terms increases, and the demands on mastery shifts.

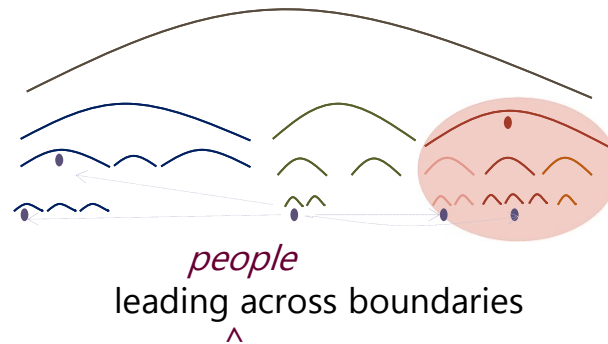
Back to hierarchy for a moment, and a couple of points from the Jo Freeman classic ("The Tyranny of Structurelessness"): "Contrary to what we would like to believe, there is no such thing as a 'structureless' group." and "The structure may be flexible, it may vary over time. It may evenly or unevenly distribute tasks, power and resources over the members of the group." An explicit hierarchy is visible, and hence can be worked on, to make it more inclusive and more about leadership (power-with rather than dominance and power-over).

"the scales of information, people, time horizons and information all changes. As a result so does the impact." — Nivia Henry

Unique System Expertise

Each system, and its intertwining in other systems, is unique and (co-)evolving.

And we have a unique responsibility and opportunity to understand, to recognize, to anticipate, to draw out.



Unique Perspective

Leading, whether informal/ad hoc or a demand of a role, happens across – across responsibilities and organizational space. And for whatever scope we're leading across, we have commitments that relate to the systems, subsystems, or initiatives we're responsible for and to, at the scope. This may be formally associated with our role, or informal, if it's an initiative we see a need for, and have stepped up to lead on. Those commitments are to outcomes, and to those we lead.

To re-iterate for emphasis: we design, and we lead, to make things more the way they ought to be. We lead, to make it more a matter of *we*. And to discover, together, how things are, and ought to be. Still, we have a unique vantage point. Unique because of what we bring, but also because the organization gives us, or we take on, a unique *across* perspective.

This uniqueness of commitment and perspective means that we have a unique opportunity, and need, to develop our expertise in the very unique systems space we have taken on leadership responsibility for. It's not often that I write "unique" three times in a sentence, but I want to explain why this "observe" section is so important. We're noticing, to respond skillfully. And we're perceiving and building a point of view and expertise that no-one else is in a position to build.

"Reality is sedimented out of the process of making the world intelligible through certain practices and not others. Therefore, we are not only responsible for the knowledge that we seek but, in part, for what exists."

— Karen Barad

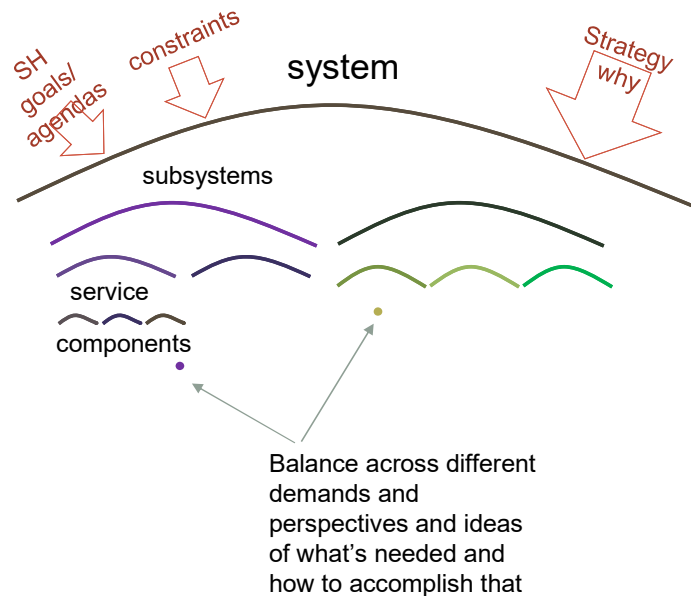
"Listen to the wisdom of the system."

— Donella Meadows

Decisions Across Boundaries

System Design: Those decisions that must be made:

- From the perspective of the whole (eg impacts system outcomes, capabilities and properties)
- First/early/before/./if not last year, then now! (considering what decisions are highly consequential to other decisions, become irreversible, etc.)



Minimalist Architecture Principle

From a Moose cartoon:

"Thank you for introducing me to minimalism"

"It's the least I could do"

The Minimalist Architecture Principle (Dana Bredemeyer): only make a decision part of the architecture, if it is *necessary* to achieving architecturally significant system outcomes. That is, if it is necessary to make because it has impact across the system and needs system perspective; it is make or break significant to business and/or technical outcomes; etc.

SocioTechnical

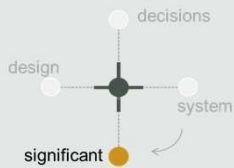
Decisions made from a system perspective strive to be "and" decisions that achieve better outcomes for all. But that may be over a longer horizon than teams are being evaluated on. Or may require taking a more broadly scoped view than the focus of a single team, to see beyond any immediate negative impact (on time or skills, etc.). Or may be about consequential second order effects, that take courage and "social capital" to even lead conversations about. This can put us in a place where navigating the social and organizational implications is challenging (the soft skills are the hard skills, kinds of things). In part, we navigate these challenges with participative design (of various styles), so various teams are represented in the sense-making and deciding.

"There are three main ways of dealing with conflict: domination, compromise and integration."

[..]

"The first rule, then, for obtaining integration is to put your cards on the table, face the real issue, uncover the conflict, bring the whole thing into the open."

— Mary Parker Follett,
Constructive Conflict, Ch1 in Dynamic Administration, first presented in January, 1925



- Significant decisions!
- Significant?

Software Architecture

Significance is indicated by (aot):

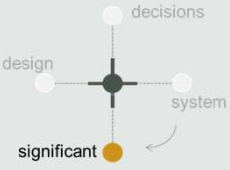
- impact across system boundaries
- implications for business and engineering success over short and long term




SOFTWARE
ARCHITECTURE

Pulling Ideas Together

Architecture is system design, which means design of the system in its various contexts (of use and operation and development) as these contexts and the system evolve and co-evolve, paying attention to failure boundaries (economic, workload, operational) and habitability (the software in use and operation, the work with and around the code, more) and to offering value that keeps the system viable.



- Significant decisions!
- Significant?



Software Architecture: recap

“Architecture represents the significant design decisions that shape a system, where significant is measured by cost of change.”

— Grady Booch

where significant is a matter of judgment, but includes system integrity and sustainability.

SOFTWARE
ARCHITECTURE

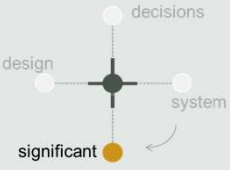
Where are we at with: What is Software Architecture?

The focus on change as a driving force in architecting, gets us a long way: some decisions are significant for they are hard to reverse, and bear high cost of change, so we need to pay close attention to them and act while still reversible; others are significant because, drawing on experience, anticipatory judgment and technical leadership, we reduce the cost and impact or ramifications of changes that would otherwise destabilize the system or slow its evolution. It's important for systems that endure, to be evolvable, as their contexts shift. And with agile and CI/CD, it's all the more important to have the capacity to extend the system and to respond to change (in our understanding of what the system needs to be and become, and in the context itself). Indeed, this is a reason we put so much emphasis on architecting for change resilience: change is a future consequence and cost, and it takes anticipatory awareness and organizational leadership to invest attention, expertise and wisdom in being proactive about change (early, *and* as we evolve the system) in balance with current pressures.


But the most important thing, is that we are designing a *system* — yes, we said that! Still, if, given that architecture is design, but not all of

design, and we're teasing out which decisions are architecturally significant, then our foremost answer is: those that we need to make to ensure desired system outcomes. That is, not only may architectural significance be determined by cost of change, but by strategic impact. What is make or break? For developers? For operations? For users? For the business team and other stakeholders? For the broader social good? In order for the system to be the kind of system it is, and uniquely so, what capabilities and properties does it need to have? What does this mean in terms of technical priorities? And architectural mechanisms we need to design and provision?

What about architects? If we want systems that hew toward integrity and sustainability and fit to purpose and to context (balancing the various tensions), we need designers who pay attention to system design — the design of *this* system (in its contexts of use, operation, development, management, supply chains, and more) and of *systems*. It's a learning journey. And it's a significant set of responsibilities. These should be shared, though it is useful to have a locus of system design attention and responsibility. Hence the role. It does not mean others don't play a role in architecture decisions! We partner with product and across engineering teams.



- Significant decisions!
- Significant?



Software Architecture

Significance is indicated by, aot:

- cost of change
- impacts system integrity
- competitive impact/impacts viability

Make or break decisions with structural, market, or organizational impact. High cost of being wrong

SOFTWARE
ARCHITECTURE

What Does this Mean for Architecting?

We design to get more the outcomes stakeholders want (Herbert Simon). We undertake to apply conscious purpose, or intentionality, along with experience, knowledge and insight, and design techniques that aid in surfacing and resolving design demands and tensions. In competitive situations that characterize business systems and products or services, we're designing systems that tend to be complex in that they forge new frontiers, innovating to provide differentiating value. That is, they are not only composed of non-trivial components and interactions, but there is considerable uncertainty as the design envelope is explored and pushed, and contexts of operation and use continually shift and evolve. Hence, we want to architect to support agility, integrity and sustainability.

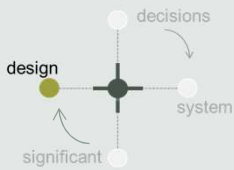
Agility has two essential components — sensing and responding. Sensing opportunity to create and define value. Sensing threat (uncertainty, change, unmet needs, escalation or drift into failure, ...), and sensing when and how to turn threat into opportunity. And responding adaptively. Which itself has various facets: what is done and how. There's an element of how quickly (under threat or to take advantage of windows of opportunity), but also how we respond can severely undermine our ability to respond adaptively in the future.

We explored integrity earlier. By sustainability we mean sustainability (not just in the short term, but well into the future) in all its senses, including:

- *economic*: business sustainability through (net) value creation and delivery
- *technical*: scalable, extensible, adaptable and evolvable, resilience...
- *social*: organizationally viable, as the social system(s) face challenges like scaling and adapting as the ecosystem evolves; provides social context for its people to thrive and find joy in work
- *environmental*: putting more value into the ecosystem than we take out, including taking care to environmental

"Architecture is "the art of the frame" [...] "the art of framing possibilities for purpose"

— Ann Pendleton-Julian and John Seely Brown



- Design!
- Evolving design
- Starting problems



Strategic Challenges

"Chicken-egg problems appear all the time when building software or launching products. Which came first, HTML5 web browsers or HTML5 web content? Neither, of course. They evolved in loose synchronization, tracing back to the first HTML experiments and way before HTML itself, growing slowly and then quickly in popularity along the way."

— Avery Pennarun

Jessica Joy Kerr @jessitron · Mar 25, 2017

Which came first, the **chicken** or the egg?

Egg. **chickens** come from eggs.

Where'd the egg come from?

...Dinosaurs.

Some Common System Design Problems

In "Systems Design Explains the World," Avery Pennarun outlines some of the common problems system designers address (<https://apenwarr.ca/log/20201227>). These include:

Chicken and Egg Problems: "The defining characteristic of a chicken-egg technology or product is that it's not useful to you unless other people use it. Since adopting new technology isn't free (in dollars, or time, or both), people aren't likely to adopt it unless they can see some value, but until they do, the value isn't there, so they don't. A conundrum." (Avery Pennarun)

With respect to control structures, Pennarun observes:

"In systems design, there is rarely a single right answer that applies everywhere. But with centralized vs distributed systems, my rule of thumb is to do exactly what Jo Freeman suggested: at least make sure the control structure is explicit. When it's explicit, you can debug it."

This reflects the underlying principle: formulate responses given an understanding of how the system relates to its environment (in use, and in operation and development), and what this means for the viability and sustainability (economic, technical, organizational, and environmental) of the system. And understand how this is impacted by its current point of evolution, and evolutionary trajectory.

"3. A statement of system principle (mission or goals) is a short-hand way of referring to the special forms of interdependence that exist between the system and its environment.

4. Thus, 'a system can only be properly characterized if we also characterize its environment' and, conversely, an environment can only be characterizing the kinds of systems it provides support to."

— Fred Emery, *On Defining Systems*

Strategic Coherence

Through system design and architecture, we bring strategic and structural context to subsequent decision making

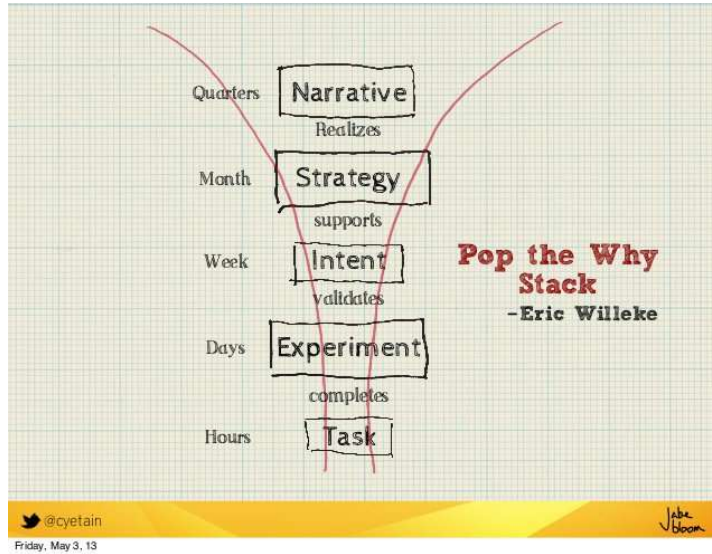


Image source: Jabe Bloom <https://x.com/cyetain/status/1005115673973059584>

What Does this Mean for Architecting?

Across (boundaries) work* can fall through the cracks when it has no "place" in the org. So part of our work is convening those system level conversations and decisions. And part of it is bringing strategic and context understanding to more narrowly focused discussions and decision making.

Yes. I am once again suggesting that systems need attention. Understanding that takes time and attention and collaboration to build. Proactively as well as responsively.

Organizational relationship building. Fostering dialog and investigative discovery. Emphasis on fostering, because there is much that competes for organizational will (willingness and determination).



Lorin Hochstein ✓

@norootcause@hachyderm.io

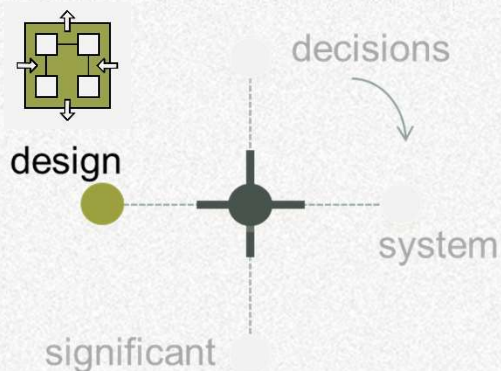
I think we vastly underestimate the difficulty of human coordination work. We attribute bad outcomes as pathologies specific to our organizations, where these are fundamentally really hard problems.

Jun 30, 2024 at 11:39 AM · 🌐

"Some problems are so complex that you have to be [...] well informed just to be undecided about them."

— Laurence J Peter (*Peter's Almanac*, entry for 24 September 1982)

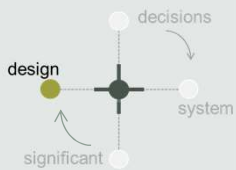
System Design



*System Design:
Surfaces and Essences*

*Product Design
Overview and Concerns*

*Conceptual Architecture
Overview and Concerns*



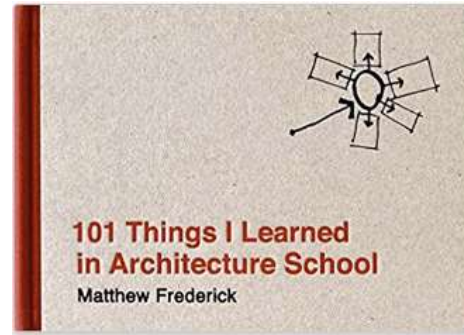
- Design!
- In next larger context



Design in Context

“Always design a thing by considering it in its next larger context.”

— Eliel Saarinen



Always Consider

“101 Things” is a book written for building architects, but has translatable lessons for software architects. In it, Eliel Saarinen* is quoted: “Always design a thing by considering it in its next larger context — a chair in a room, a room in a house, a house in an environment, environment in a city plan.” We could amend Saarinen’s point to “always decide a thing by considering its context.” Decisions, any decisions, must take context into account too. From desired outcome(s) to forces that impinge, to side-effects, interactions and consequences, context factors. [* Also, related by Eero Saarinen in *Time Magazine*, “The Maturing Modern,” 7/2/56, pp-50-57]

Christiane Floyd pointed out: “Design consists of a web of design decisions which, taken together, make up a proposed solution.”

This is true of any design – including organizational design, or UI design, and system design. As well as the design or formulation of initiatives and strategies.

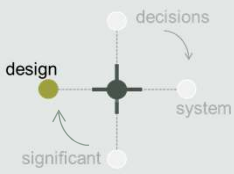
Back to Christiane Floyd: ‘By design I understand the creative process in the course of which the problem as a whole is grasped, and an appropriate solution worked out and fitted into human contexts of meaning. In [Peter] Naur’s words: “Software development is an activity of overall design with an experimental attitude”.’

“Software Development as Reality Construction” (by Christiane Floyd, 1992) is an exciting work, for it articulates software development as a co-evolving, dialogic process where we are learning what the design needs to be, even as we adapt both the system and its

context. That is, it exhibits what Nora Bateson termed “symmathesy” (learning together).

“The problem is connected to a larger system, and it’s not solved by the quick fix.”

— Mary Catherine Bateson



- Design!
- In next larger context
- Context matters



Context Matters

“Design quality is not a property of the code. It's a joint property of the code and the context in which it exists.”

— Sarah Mei



Image source: @sarahmei

"The value of every decision we make depends on the context in which we make it. In The Lord of the Rings, Frodo's journey to destroy the ring is meaningful inside the context of Middle Earth. Otherwise, he's a short, hairy guy with apocalyptic hallucinations."

— Diana Montalion

Context Factors

"[system design] strives for fit, balance and compromise among the tensions of [stakeholder] needs and resources, technology, and multiple stakeholder interests" (Rechtin and Maier) There is no perfect solution. Eb Rechtin put it this way: "The essence of architecture is structuring, simplification, compromise, and balance."

We joke about the two word answer to any question, that distinguishes the architect: "It depends." But a good architect tells you what it depends on.

At a recent conference, Diana Montalion shared her definition of wisdom:

Wisdom = knowledge + experience + good judgment

According to this definition, wisdom is the ability to know what "it depends" on.

'better expression than "common sense" is contextual sense — a knowledge of what is reasonable within a context'

— Eb Rechtin

C4: Context, Containers, Component

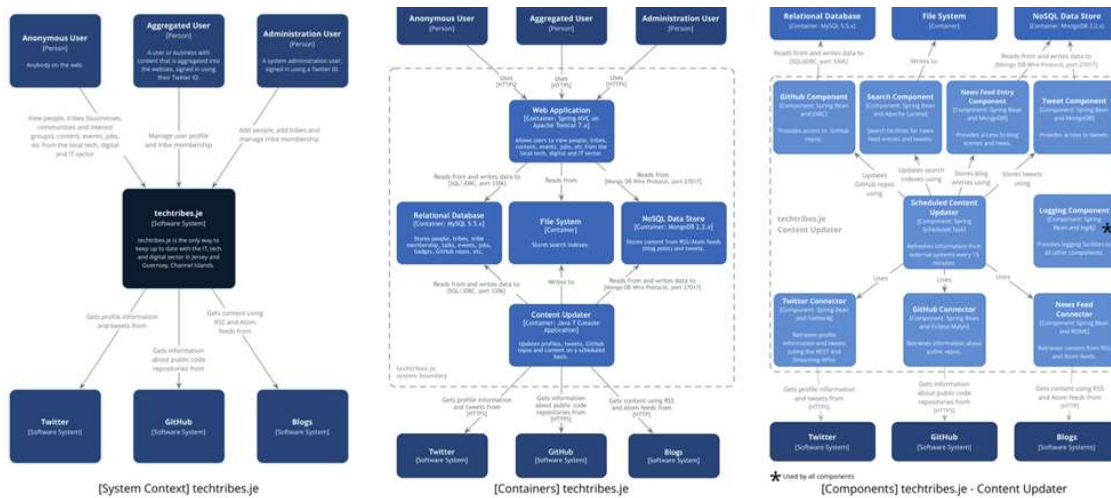


Image:
Simon Brown's C4 Model <https://c4model.com/>

Design in next larger Context: Context, Containers, Component

From Simon Brown's C4 Model overview:

"A **System Context diagram** is a good starting point for diagramming and documenting a software system, allowing you to step back and see the big picture. Draw a diagram showing your system as a box in the centre, surrounded by its users and the other systems that it interacts with.

Detail isn't important here as this is your zoomed out view showing a big picture of the system landscape. The focus should be on people (actors, roles, personas, etc) and software systems rather than technologies, protocols and other low-level details. It's the sort of diagram that you could show to non-technical people.

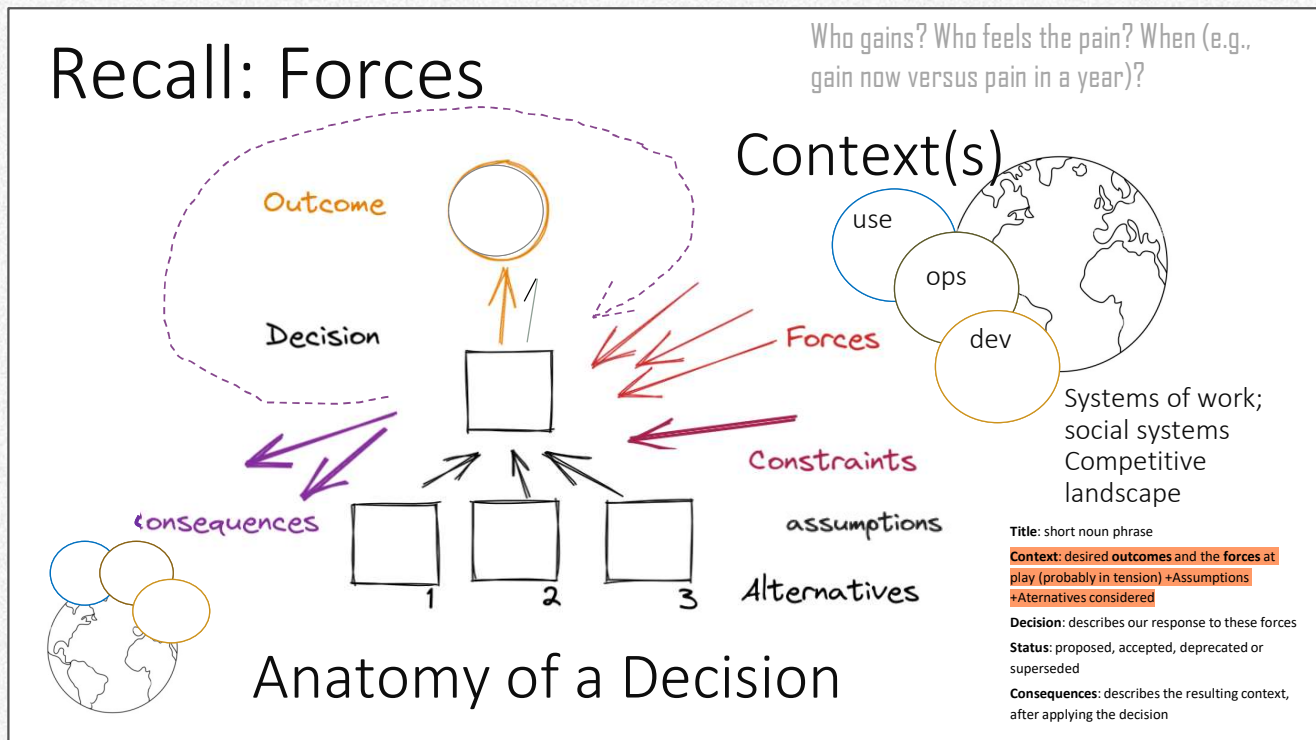
Container Diagram: Once you understand how your system fits in to the overall IT environment, a really useful next step is to zoom-in to the system boundary with a Container diagram. A "container" is something like a server-side web application, single-page application, desktop application, mobile app, database schema, file system, etc. Essentially, a container is a separately runnable/deployable unit (e.g. a separate process space) that executes code or stores data.

The Container diagram shows the high-level shape of the software architecture and how responsibilities are distributed across it. It also shows the major technology choices and how the containers communicate with one another. It's a simple, high-level technology focused diagram that is useful for software developers and support/operations staff alike.

Component diagram: Next you can zoom in and decompose each container further to identify the major structural building blocks and their interactions.

The Component diagram shows how a container is made up of a number of "components", what each of those components are, their responsibilities and the technology/implementation details."

Source: Simon Brown, <https://c4model.com/>



Architecture Decisions

In the opening module, we considered architecture decisions, and we return to that topic here, because it is so central.

Context: Why Though?

One (set of) reason(s) to take context into account, is to gain more understanding of what complexity we need to take on because it is essential (to support users in the domain, or our engineering teams, or our strategy,...). Decisions we make accidentally in the course of things, (tend to) create complexity: "Like most tech debt, we didn't make this decision, we just did not *not* make this decision." (Jack Lindamood*)

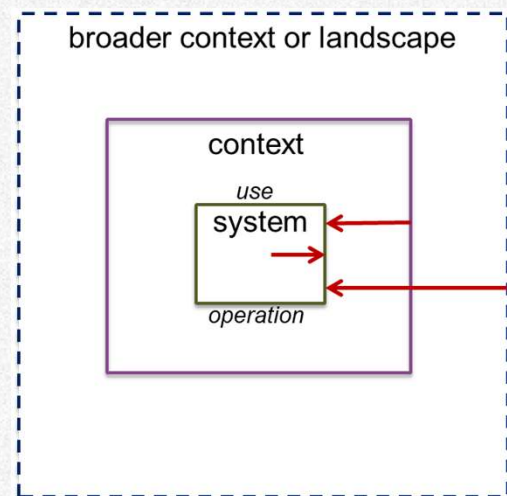
A system that has ill fit to its context, will struggle to survive and needs to adapt. And by fit, we don't mean over-fit. In dynamic, evolving, changing contexts, a system needs adaptive capacity to evolve to fit shifting contexts.

Quote source: Jack Lindamood, 2024

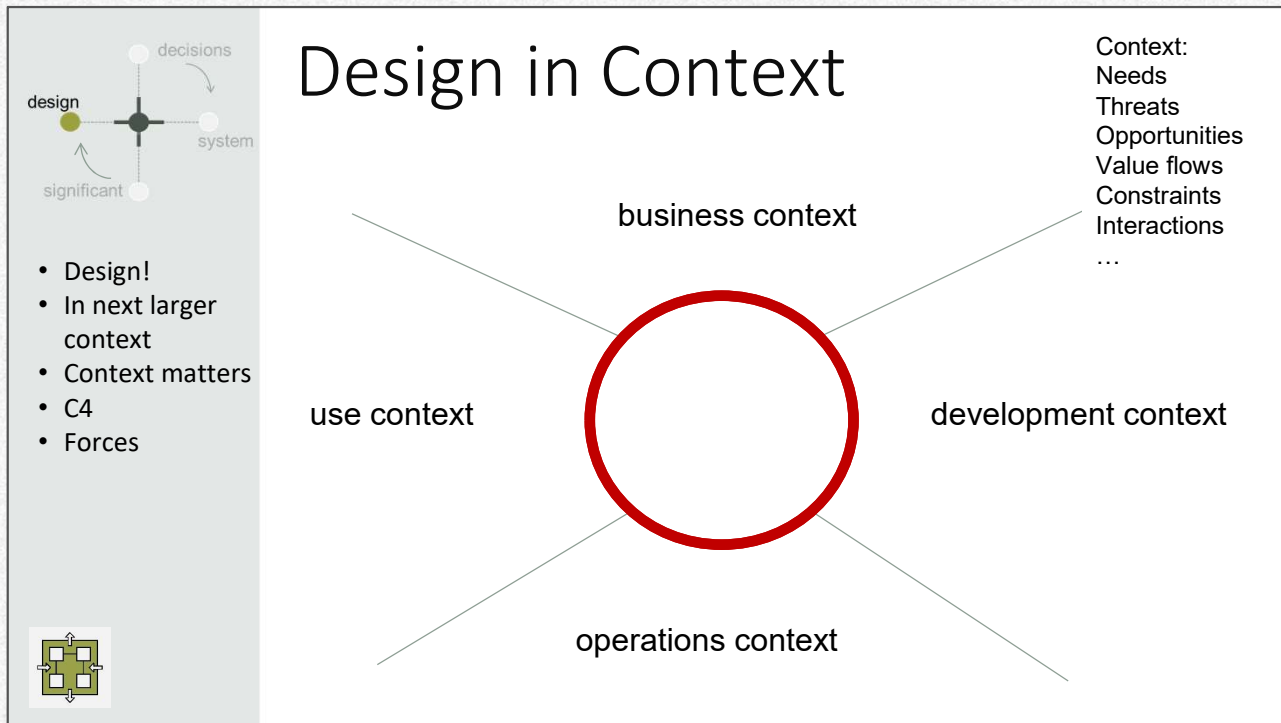
<https://cep.dev/posts/every-infrastructure-decision-i-endorse-or-regret-after-4-years-running-infrastructure-at-a-startup/>

Donald Schon, Design lecture, 1989

<https://hiredthought.com/2021/02/24/donald-a-schon-at-iowa-state-university-talk-transcript/>



"In a situation of uncertainty, the problem that you face is the problem of constructing a problem because you don't know what the problem is. And the problem of constructing a problem is not a technical problem. In fact, the opposite is true, you have to construct the problem before you can carry out any technical activity." — Donald Schön



Design is Making Trade-offs

The forces on the system arise from various contexts, and interact. We're designing a system that meets user needs (has customers and users who integrate it into other systems, including systems of work and other facets of individual and social life; it is sustainable in the ecosystems that it delivers value to), and is internally viable and sustainable (it addresses the challenges of delivering the capabilities users need, across users, and as use scales across more users in different contexts, etc.).

And this "fit" is not over-fit, or too closely fit. We evolve the system for various reasons: we can't do everything at once, so there is an aspect of sequencing and incremental development; we don't know everything at once, so there is an aspect of learning what is valued and learning how users adapt the system capabilities into their larger contexts and systems of work and play and life; and things are changing including as a result of what we are adding to the ecosystems (of use, of engineering, of production and supply chain relationships, etc.). So we adapt the system to more closely fit the challenges and needs it faces, while retaining (attempting to retain) adaptive capacity.

"Design is rooted in concerns"

"Design is, then, always multiperspective, even where pursued by individuals."

"Perspectivity necessarily entails blindness. I cannot see what I cannot see from my perspective."

—Christiane Floyd

"Conflict situations are situations where you face conflicting values and where you don't [yet] have a technical problem to solve, because you must make the values consistent before you can solve such a problem." — Donald Schön

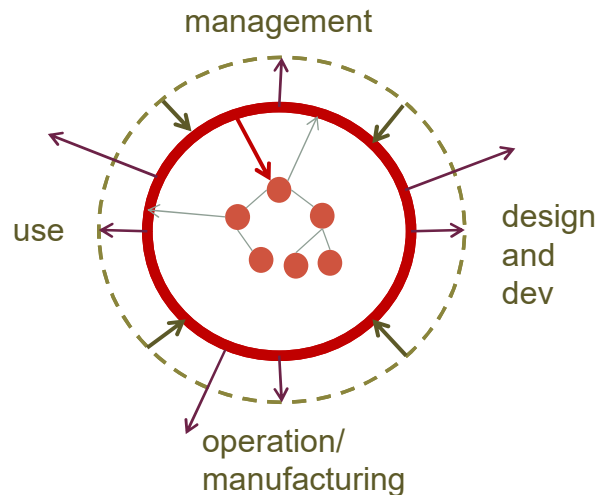
System Design: in Context



Design in context(s)

- Contexts of use, of design and development, of manufacturing and operation, of management
- Social, political, economic, technical contexts

→ move inwards (zoom in), move outwards (zoom out); pan around and scan; surface forces and constraints and consequences



System in Context and System Design

I switched from rectangles to circles for this slide to draw on the notion that we might think the “target” of our (system/engineering) design is internal to the system, and we’re “given” requirements to work to. That is more or less true, and more true in some organizations and system contexts than others.

As system designers, however, it is important to identify and understand the forces that impinge on and shape our design (option) space. Some of those forces are from within the system – systems place constraints their inner environments (some are intentional design choices; some emerge from interactions among parts, etc.). There are matters of fit among parts. But also, of parts and contexts, and system and context, ... A technology choice for a part, that acts back and constrains other choices beyond the part. (For example, a licensing agreement, that then shapes the cost profile and ties our product pricing increases to those of the vendor. Whoops.)

If we’re designing a part, because it needs our specialist knowledge, we might notice that we have an arena of knowledge that no-one else has. Knowledge about our design but also about what is happening in the technologies associated with that space. We bring that knowledge to the *system* design table.

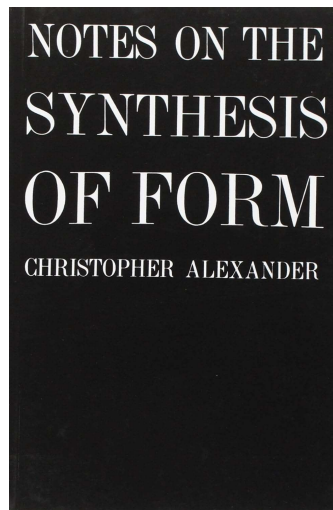
Knowledge of complex systems is distributed, and knowledge of the contexts of the parts and the system is distributed. Systems design is about fostering knowledge not just of the design, but the spaces that produce forces on the design, and exploration and understanding of the consequences of design choices (and feeding that back into our design responses).

But this is a lot, and we need to navigate the space of concerns, building, evolving and repairing our understanding, but also making design commitments and designs and moving the design (and implementation) forward.

Form and Context

“Every design problem begins with an effort to achieve fitness between two entities: the form and its context. The form is the solution to the problem; the context defines the problem.”

— Christopher Alexander, Notes on the Synthesis of Form, 1964.



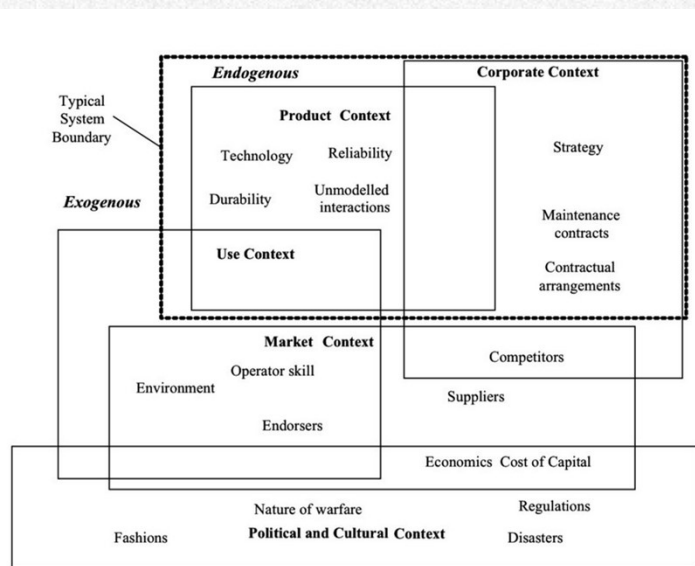
Building our Understanding of the Context

Christopher Alexander and other building architects, emphasize the fit of architecture to context.

DeWeck et al (image below), in their classification of sources of uncertainty, delineate various contexts of (potential) relevance to us. Our contexts are sources of change, uncertainty and ambiguity, forces (gradients, pushes and pulls,..).

“The form is a part of the world over which we have control, and which we decide to shape while leaving the rest of the world as it is. The context is that part of the world which puts demands on this form; anything in the world that makes demands of the form is context. Fitness is a relation of mutual acceptability between these two. In a problem of design we want to satisfy the mutual demands which the two make on one another.”

— Christopher Alexander, Notes on the Synthesis of Form, 1964



source: DeWeck et al, A Classification of Uncertainty

Theory Building

Peter Naur, Programming as Theory Building • 227

PETER NAUR, PROGRAMMING AS THEORY BUILDING

Peter Naur, widely known as one of the authors of the programming language syntax notation "Backus-Naur Form" (BNF), wrote "Programming as Theory Building" in 1985. It was reprinted in his collection of works, *Computing: A Human Activity* (Naur 1992).

This article is, to my mind, the most accurate account of what goes on in designing and coding a program. I refer to it regularly when discussing how much documentation to create, how to pass along tacit knowledge, and the value of the XP's metaphor-setting exercise. It also provides a way to examine a methodology's economic structure.

In the article, which follows, note that the quality of the designing programmer's work is related to the quality of the match between his theory of the problem and his theory of the solution. Note that the quality of a later programmer's work is related to the match between his theories and the previous programmer's theories.

"PROGRAMMING AS THEORY BUILDING"

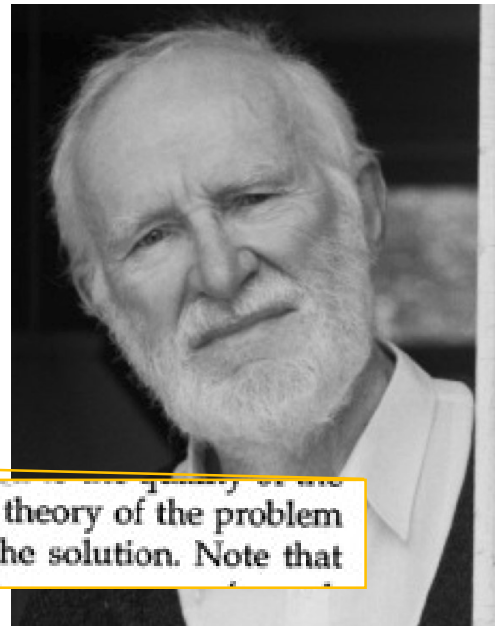
Introduction

The present discussion is a contribution to the understanding of what programming is. It suggests that programming properly should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand. This suggestion is in contrast to what appears to be a more common notion, that programming should be regarded as a production of a program and certain other texts.

Some of the background of the views presented here is to be found in certain observations of what actually happens to programs and the teams of programmers dealing with them, particularly in situa-

tions
happens
reaction
education
accomplish

match between his theory of the problem and his theory of the solution. Note that



Building our Theory of the Problem

Our field contends with complex software-intensive systems and their evolution, and one of the classics (1980) is "Programs, Life Cycles, and Laws of Software Evolution." In it, Meir Lehman observed:

"The installation of the program together with its associated system [...] change the very nature of the problem to be solved. The program has become a part of the world it models, it is embedded in it. Analysis of the application to determine requirements, specification, design, implementation now all involve extrapolation and prediction of the consequences of system introduction and the resultant potential for application and system evolution. This prediction must inevitably involve opinion and judgment."

Peter Naur, in "Programming As Theory Building" (1985), argues

"programming properly should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand."

A theory, that is, of the problem* being solved, and how the code relates to and addresses this problem.

Returning to Lehman:

"any program is a model of a model within a theory of a model of an abstraction of some portion of the world or of some universe of discourse"

* Where the "problem" is the opportunity we're creating, the need we're addressing, etc, with the capability we're building.

Between Lehman, Floyd, and Naur, we have an important set of ideas for software, or any systems, really. We're building a theory, that informs our (design) decisions. We need to anticipate the impact of our decisions, in making them. And probe, to assess/amend our theory.

"what has to be built by the programmer is a theory of how certain affairs of the world will be handled by, or supported by, a computer program."

— Peter Naur

These classics advanced ideas about design that are important today

Programming as Theory Building

Peter Naur, Programming as Theory Building • 227

PETER NAUR, PROGRAMMING AS THEORY BUILDING

Peter Naur, widely known as one of the authors of the programming language syntax notation "Backus-Naur Form" (BNF), wrote "Programming as Theory Building" in 1985. It was reprinted in his collection of works, *Computing: A Human Activity* (Naur 1992).

This article is, to my mind, the most accurate account of what goes on in designing and coding a program. I refer to it regularly when discussing how much documentation to create, how to pass along tacit knowledge, and the value of the XP's metaphor-setting exercise. It also provides a way to examine a methodology's economic structure.

In the article, which follows, note that the quality of the designing programmer's work is related to the quality of the match between his theory of the problem and his theory of the solution. Note that the quality of a later programmer's work is related to the match between his theories and the previous programmer's theories.

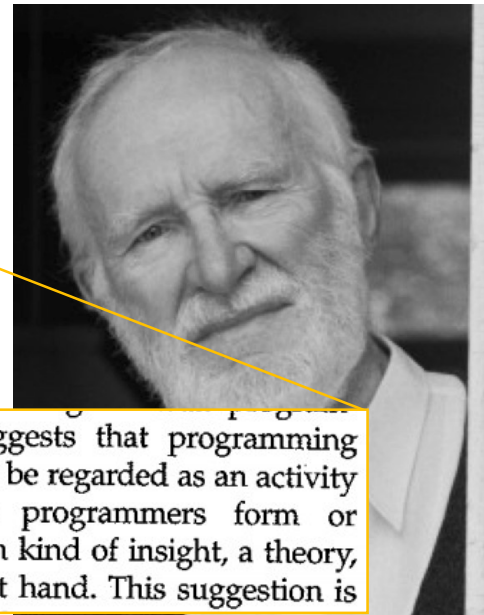
"PROGRAMMING AS THEORY BUILDING"

Introduction

The present discussion is a contribution to the understanding of what programming is. It suggests that programming properly should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand. This suggestion is in contrast to what appears to be a more common notion, that programming should be regarded as a production of a program and certain other texts.

Some of the background of the views presented here is to observations of what programs and the tea dealing with them, p tions arising from u haps erroneous prog reactions, and on the cations of programs accommodating such

ming is. It suggests that programming properly should be regarded as an activity by which the programmers form or achieve a certain kind of insight, a theory, of the matters at hand. This suggestion is



Theory Building

Peter Naur draws on Ryle's notion of theory and its role in intellectual (we might, today, rather emphasize cognitive) activity:

"where theory is understood as the knowledge a person must have in order not only to do certain things intelligently but also to explain them, to answer queries about them, to argue about them, and so forth. A person who has a theory is prepared to enter into such activities; while building the theory the person is trying to get it."

Peter Naur is articulating a theory of the difficulty of developing shared theories, and the importance of direct conversations in conjunction with the code to communicate our webs of understandings (or theories, or interrelated, interwoven mental models).

I would add that our theory (or system of theories) supports coherence and design integrity, and we're ever working towards requisite coherence among our (various folk design-build-evolving the system) individual mental models and perspectives. Seeking to understand the system, and its context(s) (or situation), and each other more, and evolve our theories in that direction, anyway.

*"people got their opinions
where do they come from?
each day seems like a natural fact
and what we think changes how we
act."*

— Why Theory? Gang of Four lyrics

*"[A social] system always contains at
least three elements or dimensions which
are locked into one another: a social
structure—which is a set of related roles
and authority relationships—a
technology and a theory. And by a theory
I don't mean an academic or sociological
theory about the system: I mean what it
is that's believed that causes people in
the system to do what they do."*

— Donald Schon, Reith Lecture 2

Programming as Theory Building

Peter Naur, Programming as Theory Building • 231

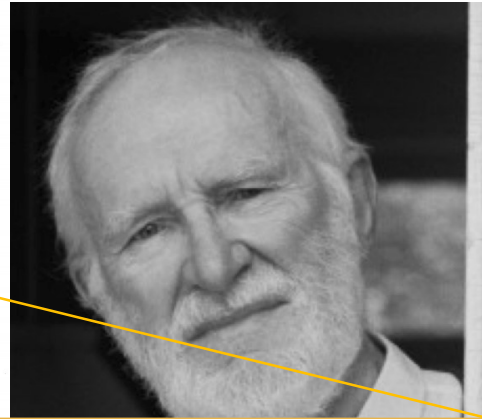
Very briefly, a person who has or possesses a theory in this sense knows how to do certain things and in addition can support the actual doing with explanations, justifications, and answers to queries, about the activity of concern.

The Theory To Be Built by the Programmer

In terms of Ryle's notion of theory, what has to be built by the programmer is a theory of how certain affairs of the world will be handled by, or supported by, a computer program. On the Theory Building view of programming the theory built by the programmers has primacy over such other products as program texts, user documentation, and additional documentation such as specifications. In arguing for the Theory Building View, the basic issue is to show how the knowledge possessed by the programmer

on a grasp between a situation and the world gives rise to a theory held by the programmer. This theory is not a set of rules, but a set of criteria. It is a theory of many things, as human beings can be thus in which the affairs of the world, both in their overall characteristics and their details, are, in some sense, mapped into the program text and into any additional documentation. Thus the programmer must be able to explain, for each part of the program text and for each of its overall structural characteristics, what aspect or activity of the world is matched by it. Conversely, for any aspect or activity of the world the programmer is able to state its manner of mapping into the program text. By far the largest part of the world aspects and activities will of course lie outside the scope of the program text, being irrelevant in the context. However, the decision that a part of the world is relevant can only be made by someone who understands the whole world. This understanding must be contributed by the programmer.

2) The programmer having the theory of the program can explain *why* each part of the program is what it is, in other words is able to support the actual program text with a justification of some sort. The final basis of the justification is and must always remain the programmer's direct, intuitive knowledge or estimate of where the justification reasoning, perhaps with design rules, questions with all the point being principles and rules are relevant. This must be in the matter of the program or having the ability to respond to a demand for a model.



what has to be built by the programmer is a theory of how certain affairs of the world will be handled by, or supported by, a computer program. On the Theory

2) The programmer having the theory of the program can explain *why* each part of the program is what it is, in other words is able to support the actual program text with a justification of some sort.

Building our Theories of the Problem and Solution

Our systems change the world, based on our understandings and assumptions, so these are, well, important.

Alberto Brandolini (@ziobrando on twitter): "It is not the domain experts knowledge that goes into production, it is the assumption of the developers that goes to production" (via Krisztina Hirth)

Theory is a way to talk about not just our mental models, and our reasoning-explaining and coherence making (attempts), but also a discipline of probing and testing our theories. Theories! Because we're not only developing theories of how our system works, or how it is structured to meet needs like adaptability and understandability, but theories of why the system matters and what matters (to users) and how.

And to the extent that we can build up shared, or at least sufficiently overlapping, theories, we're working on common ground (Klein et al) and requisite cohesion (Jabe Bloom). And we can probe and test elements of our theories, with thought experiments and walking through our reasoning "out loud" and "where we can see it," or user studies, or prototypes, as well as experiments in the A/B testing and market "bets" sense).

"It's developer's (mis)understanding, not expert knowledge that gets released in production"

— Alberto Brandolini

Brandolini's Law: "The amount of energy needed to refute bullshit is an order of magnitude bigger than that needed to produce it"

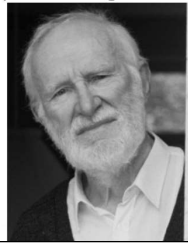
Design as Theory Building

Programming as Theory Building

Very briefly, a person who has or possesses a theory in this sense knows how to do certain things and in addition can support the actual doing, with explanations, justifications, and answers to queries, about the activity of concern.

The Theory Is In Us

The Theory Is In Us



*"It's developer's
(mis)understanding, not [domain]
expert knowledge that gets
released in production"*
— Alberto Brandolini

System-in-Context (use, dev, ops)

Developing
our **theory of
the problem**

Product Design
Design of system
capabilities/properties

System

Developing
our **theory of
the solution**

Architecture
Structure and
mechanisms

Design of What the System Is and Is Becoming

We use design in two senses: the system (as currently built) has a design (noun; what the system is), and we design (verb) to make the system more the way *we* want it to be (we design to shape, or bring intent to what the system is becoming). *We*, there, is a complex! We, users. We, designers. We, business leaders. And more. And "we" all have different ideas and experiences of what the system is, and ought to become.

In the STELLA report, David Woods draws on Richard Cooks' diagram (figure 4) of the various people (roles) interacting with a system (developers, architects, operations, users), where below the line there are the artefacts (code that may be internally and externally sourced, tools like monitoring, deploy and testing tools, etc.) and above the green line "of representation" people are interacting with the system to do things they need to do, via their (unique to each) mental models.
(<https://snafucatchers.github.io/>)

Design works to create a theory of not just what the system is and is becoming, but how that matters. And to build sufficient shared understanding.

"When you analyse a problem you see what kind of problem it is, and identify the concerns and difficulties you have to deal with to solve it."

— Michael Jackson

"In analytical thinking the thing to be explained is treated as a whole to be taken apart. In synthetic thinking the thing to be explained is treated as part of a containing whole. The former reduces the focus of the investigator; the latter expands it."

— Russell Ackoff

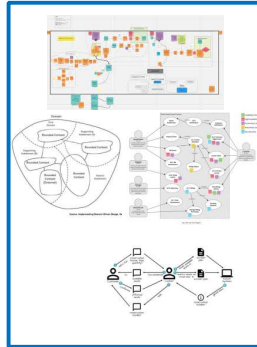
Design: System in Context

- What is the system used for (purpose and identity)?
- Which capabilities are we going to move across the system boundary?
- What new capabilities are we going to bring into existence?
- How is the system being adapted (and exapted) to new uses?

System behaviors and properties

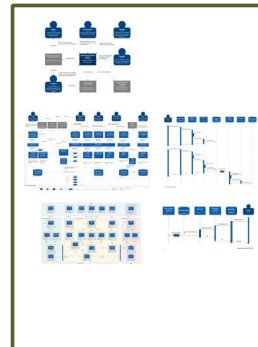
- impact (users, partners, operations) experience

System-in-Context
(use, dev, ops)



Product Design
Design of system
capabilities/properties

System



Architecture
Structure and
mechanisms

Design of What the System Is and Is Becoming

Some design tools we use:

- Event storming
- Domain story telling
- Rich pictures
- Use cases
- User story maps (and user stories)
- User journey maps
- Impact maps

ex-ap·ta·tion (ĕg'zăp-tă'shən)

n. Biology

The utilization of a structure or feature for a function other than that for which it was developed through natural selection.



Ruth Malan
@RuthMalan

Dec 23, 2023

"understanding of complex systems is distributed"
-- Chris McDermott

← 3



Ruth Malan
@RuthMalan

Dec 23, 2023

When we put that insight together with

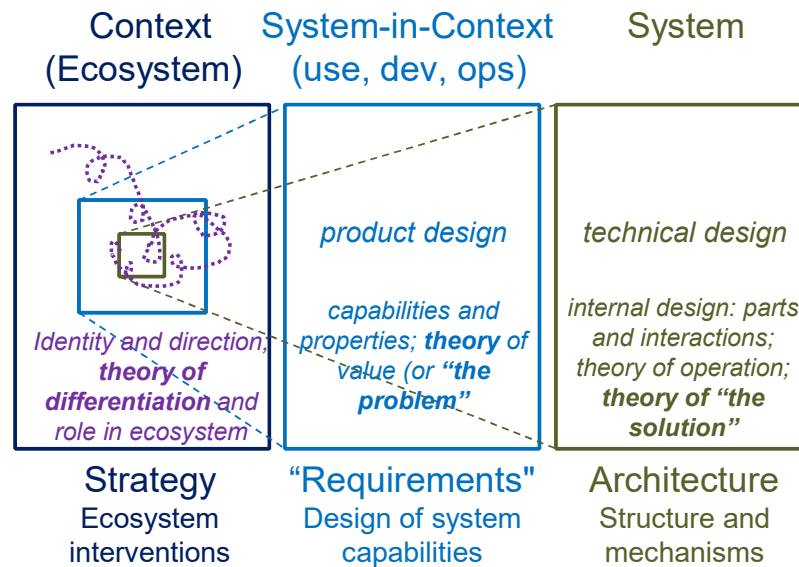
"It's developer's [...] understanding [...] that gets released in production" —
Alberto Brandolini

We might notice that there isn't any "one triewe way" to approach the mess
(technical term i'm borrowing from Ackoff) ...

But conversations sure are a big part of it.

Design Across Boundaries

System design is contextual design — it is inherently about boundaries (what's in, what's out, what spans, what moves between), and about tradeoffs. It reshapes what is outside, just as it shapes what is inside.



System Design is Contextual Design

Recognizing that a system changes its contexts, means recognizing we're designing the system-in-context — not just the system, but the socio-technical system or system-in-context (of use) too. While we have limited degrees of design freedom with respect to the context, everything the system takes on, impacts its (various) context(s), so we are redesigning at least some aspects of the containing socio-technical systems and broader context.

Alternately put, to develop our "theory of the problem," or to "load" the context into our mental models, so that we can uncover this multidimensional decision options and tradeoff space, we need to ask (not just) "what do users need?" but also "what do developers and testing need?" and "what do our operations and security teams need?" and "what do others in the value network need?"

"We need to ask: what does the code need?" — Michael Feathers

We architect across — across boundaries: across not just the code and the teams involved, but across the internal system design (architecture and code/tests) and design of the system-in-use or system-of-systems design (what our industry has tended to call "requirements"); across the different languages and concerns of these different spaces, the technical language of code and test and integration, deployment and operation, and the languages of the domains where the system is used; across the turfs and sense of ownership and decision responsibility; across views and perspectives; etc.

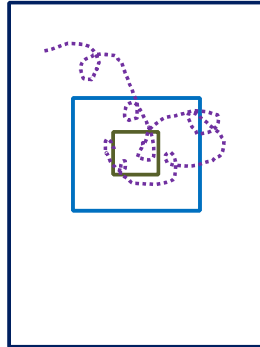
But of course, we can't attend to everything, at least not all at the same time, in detail. We "zoom out," as it were, to scan the ecosystem or value landscape, to identify opportunities and challenges that do warrant closer attention. To set framing for the problem, to understand the trends and forces that shape and constrain it. To get a bearing on the ecosystems that are or will be impacted.

"The greatest complexities arise exactly at boundaries"

— Donella Meadows

Design: Nonlinear

System design is contextual design — it is inherently about boundaries (what's in, what's out, what spans, what moves between), and about tradeoffs. It **reshapes** what is outside, just as it shapes what is inside.



“all models are wrong, but some are useful”
– George Box

Nonlinear Thinking / By Diana Montalion

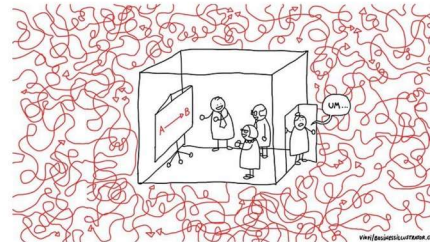


Image source: virpi/businessillustrator.com

Nonlinear Exploration

In system design and architecture we are moving fluidly between wholes in contexts, parts and relationships, insides and outsides, forces impinging from the context and forces emerging from the system, etc. We're thinking holistically sometimes, but drilling into, to further understand, zooming back out, and backtracking if we need to.

“This instinct to subdivide complex systems dominated scientific inquiry, and it advanced a framework that attempted to reorganize nature into deceptively simple components. This was the logic of reductionism, a way of thinking that can be traced at least as far back as Aristotle. Reductionism analyzed complicated things (bicycles, cities, humans) by breaking them down into distinct parts (wheels and gears, streets and people, organs and cells). In theory, everything is the sum of its parts. So if you understand those parts, you understand the whole.” — Mark Bittman

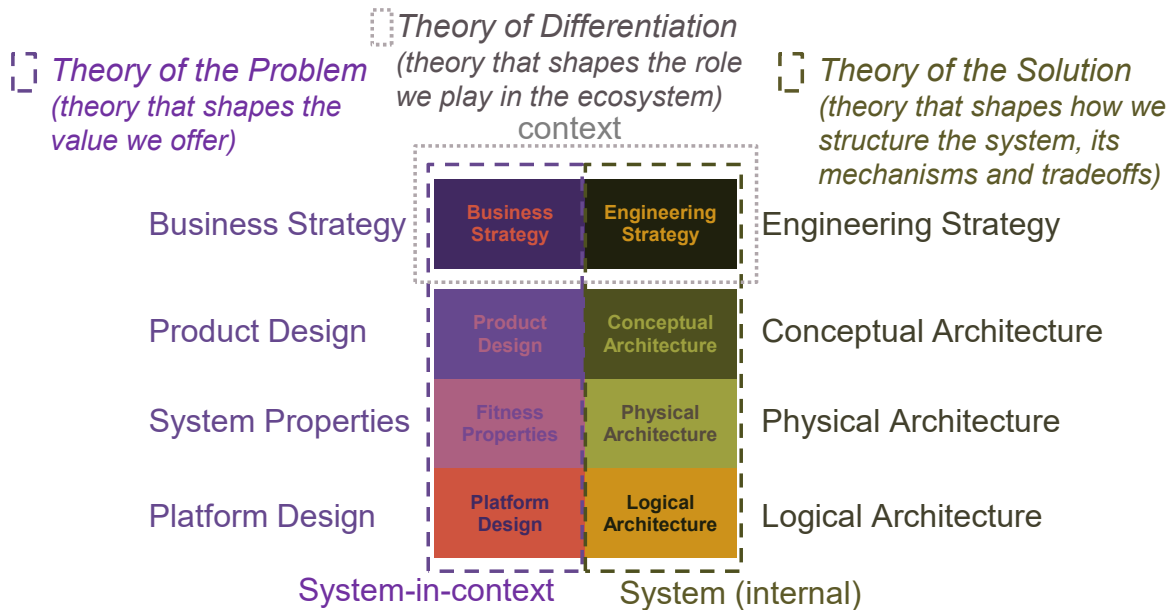
We need to think in terms of parts and relationships (to cope, as our systems become more complex) and in terms of what emerges from the dynamic interactions not just among parts, but systems in (broader) contexts.

One point that I hope emerges in the workshop experience, not just the "theory": system design and architecture is where we pay attention to the system — parts and interactions that give rise to wholes with more the capabilities and properties we want, which means understanding the whole(s) in its environments (technology, social/organizational, use, economic/business viability, ..).

This means partnering and collaborating across boundaries, and... that can be tricky to navigate and make real, and more so in some orgs and parts of orgs... "Understanding of complex systems is distributed" and we need conversations that scaffold bringing some of that understanding together. All of our design arenas are intense — demand expertise and attention/focus and work, and have their practices and communities. And yet influence needs to flow across the boundaries and not just one way.

"The menu is not the meal"
— Alan Watts

Frames and Practices



Organizing Structure

The framework on the slide outlines the organizing structure for this workshop, and also serves as a conceptual model for system design.

"A map is not the territory it represents, but, if correct, it has a similar structure to the territory, which accounts for its usefulness."
— Alfred Korzybski

Attribution

The format for these notes is adapted from a template from Nancy Duarte and team.

For more:

<https://www.duarte.com/slidedocs/>



TECHNICAL
LEADERSHIP

Duarte Slidedocs®

We recommend the Duarte material on slidedocs® in addition to the template; much that is valuable there.

Quotes and Photos

We have consciously brought various pioneers and contemporaries visibly into our materials for two reasons:

i. to acknowledge and celebrate the extent to which we are because of others (Abeba Birhane). It is a small way to bring into the room, so to speak, with us people whose insights and work has influenced us, and integrated with our experiences, other reading and conversations, and more, to build what we understand and can share.

ii. to recommend to you wonderful work you may want follow up on, and also to draw in our contemporaries who are sharing insights that you too may find useful, and want to follow them on twitter, etc.

*"Act always so as to
increase the number
of choices."*
— *Heinz von Foerster*

Stay in Touch

Ruth Malan:

Bluesky: @ruthmalan.bsky.social

LinkedIn: Ruth Malan

Web: ruthmalan.com

Masterclasses and Workshops

- **System Design and Architecture**, Feb 24-26 and Mar 3-5, 2025
- **Technical Leadership**, Dec 4 and 11, 2024

“What we care about is the productive life, and the first test of the productive power of the collective life is its nourishment of the individual. The second test is whether the contributions of individuals can be fruitfully united”
— Mary Parker Follett

Attribution — Please give appropriate credit if you quote from this book. You may do so in any reasonable manner, to a reasonable extent, respecting the work it takes to create something like this.