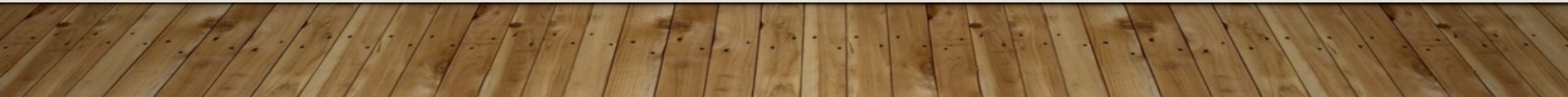
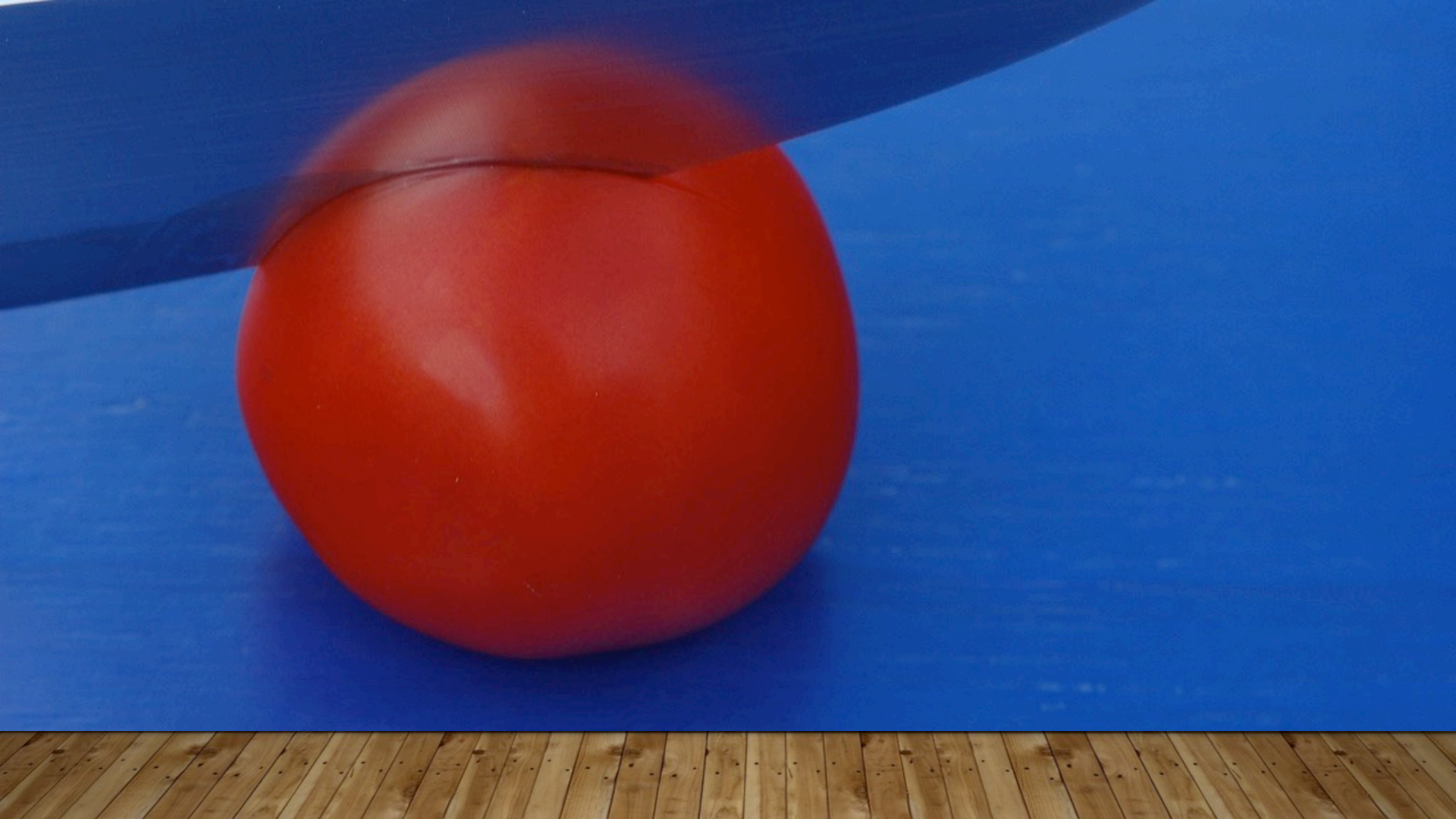
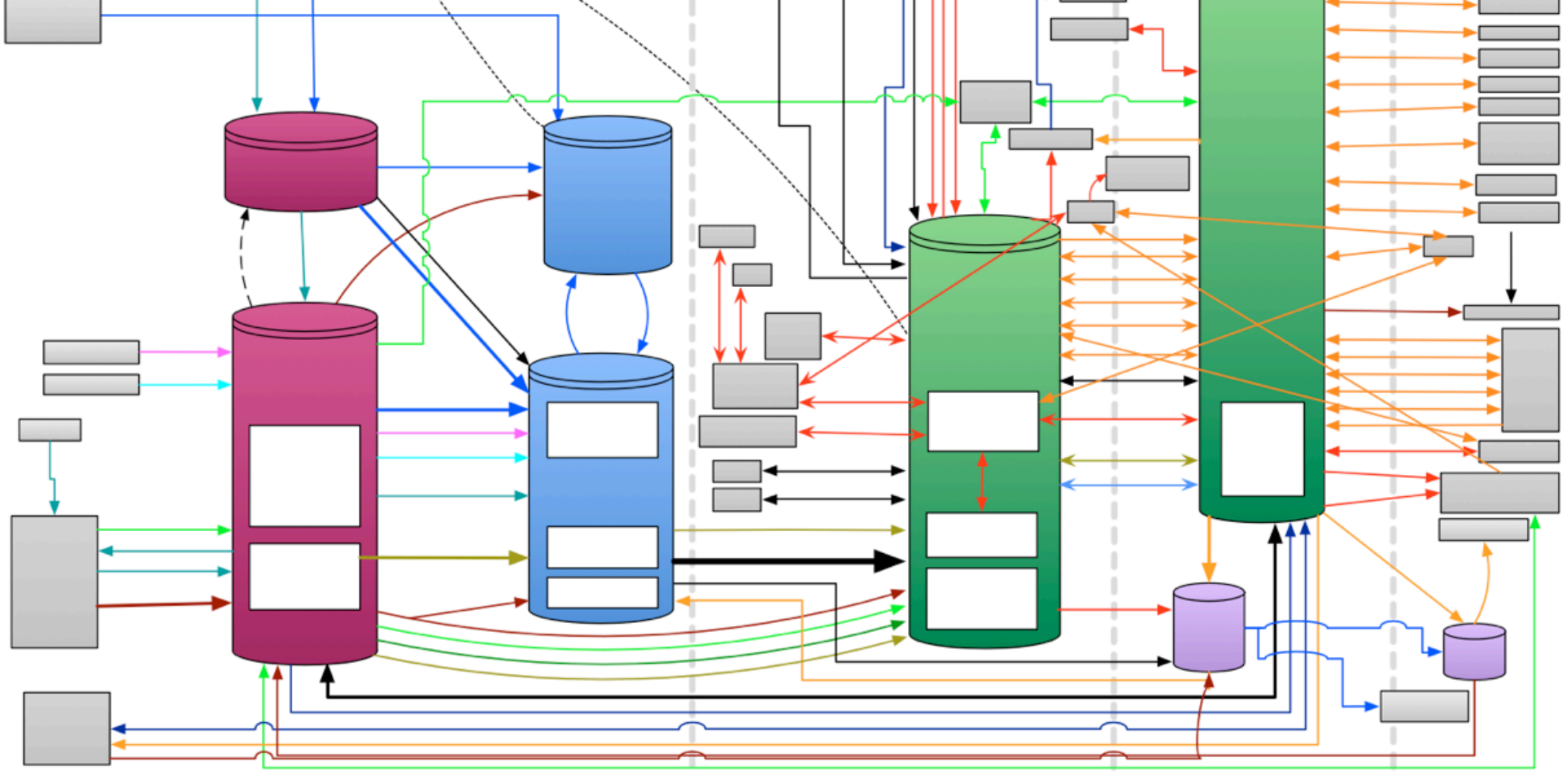
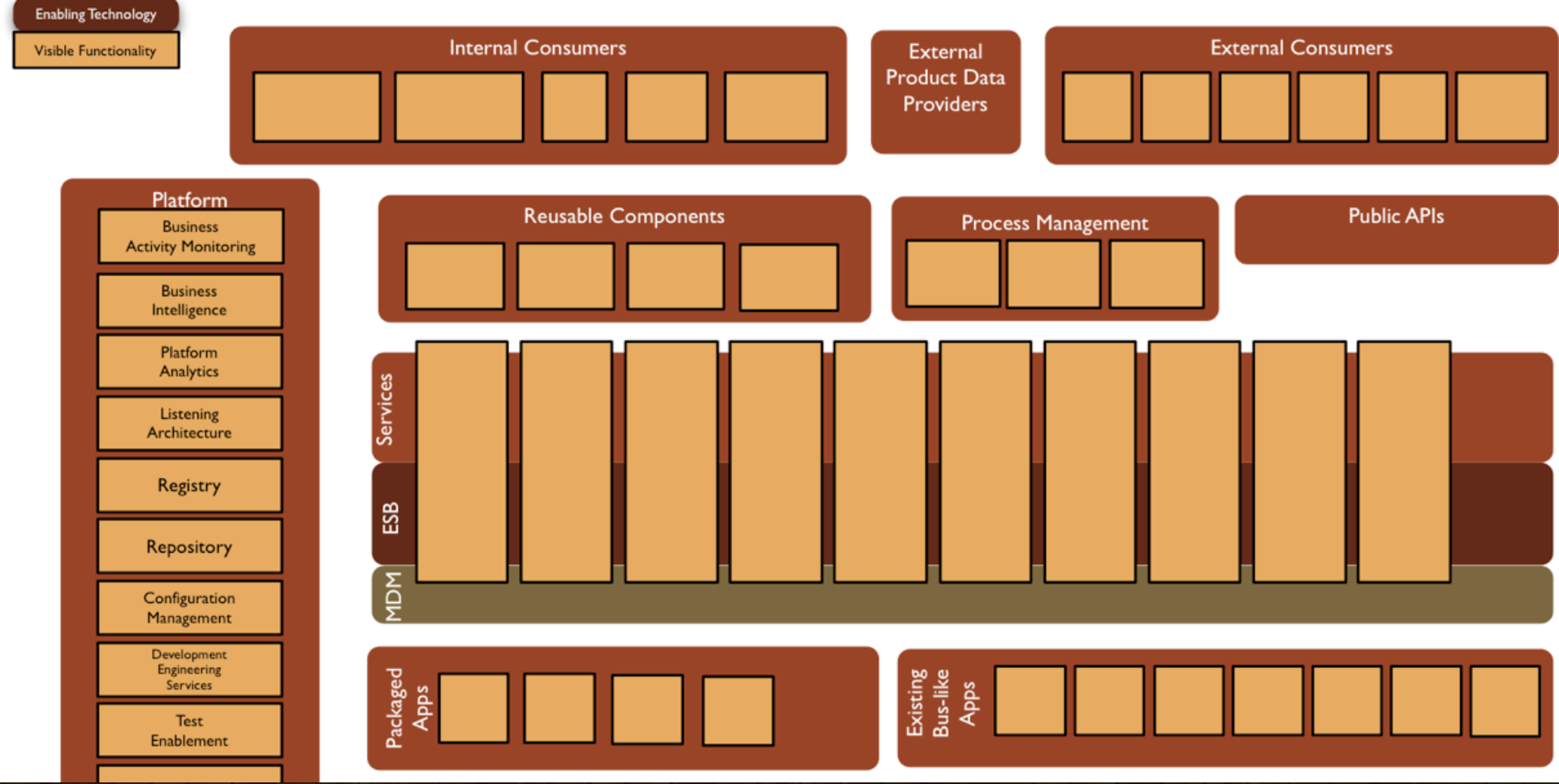
ARCHITECTURE WITHOUT AN END STATE

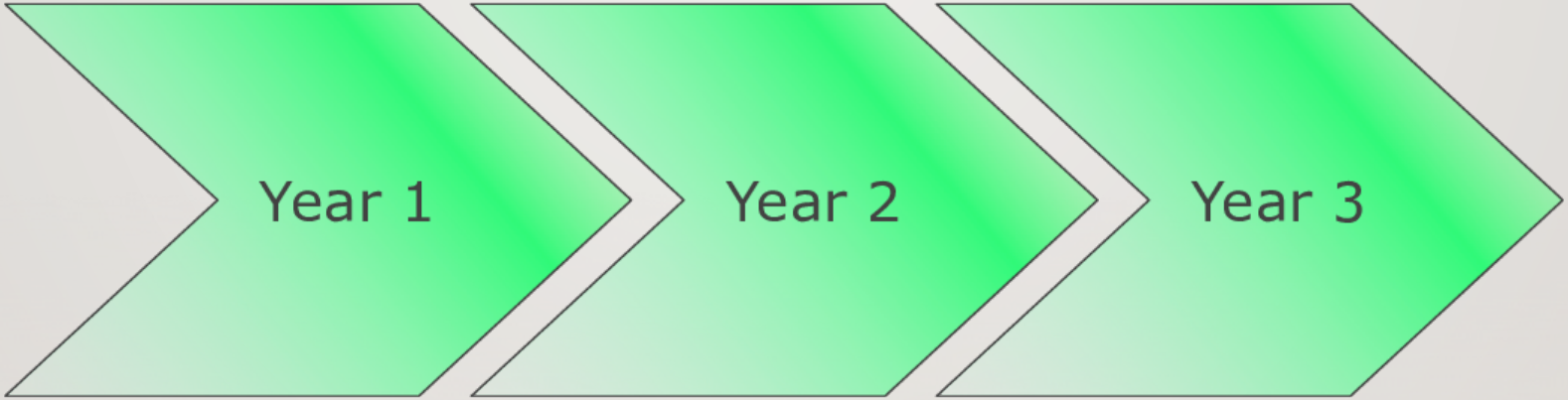
MICHAEL T. NYGARD







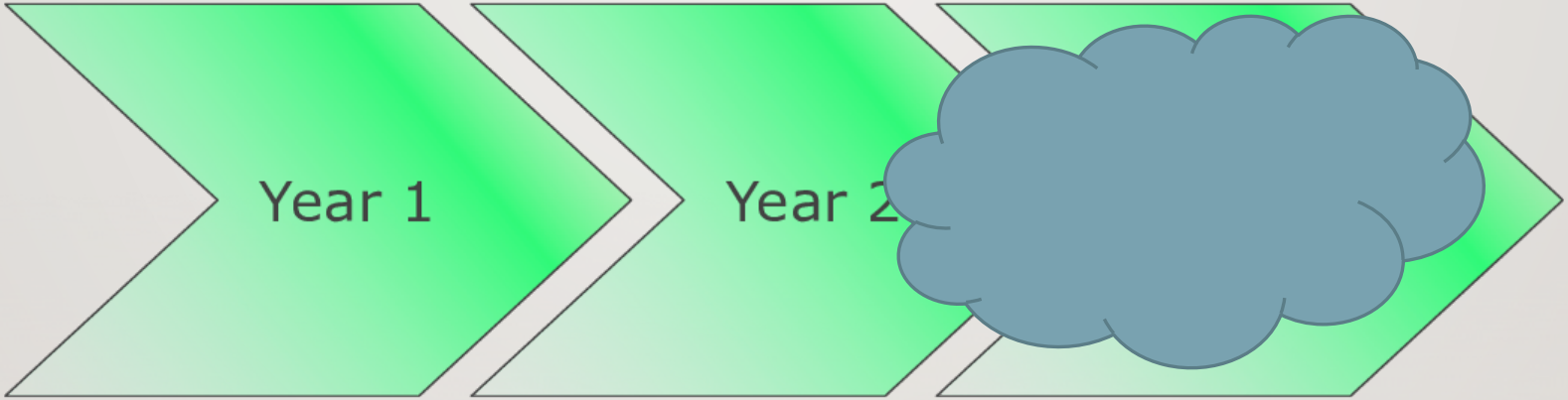

Enabling Technology
Visible Functionality
Internal Consumers
External Product Data Providers
External Consumers
Platform
Business Activity Monitoring
Business Intelligence
Platform Analytics
Listening Architecture
Registry
Repository
Configuration Management
Development Engineering Services
Test Enablement
Reusable Components
Process Management
Public APIs
Services
ESB
MDM
Packaged Apps
Existing Bus-like Apps



Vendor Selection
Proof of Concept

Rebuild the World

Experience Nirvana



Vendor Selection
Proof of Concept

Merger
Acquisition
Market change
Technology churn
Executive turnover

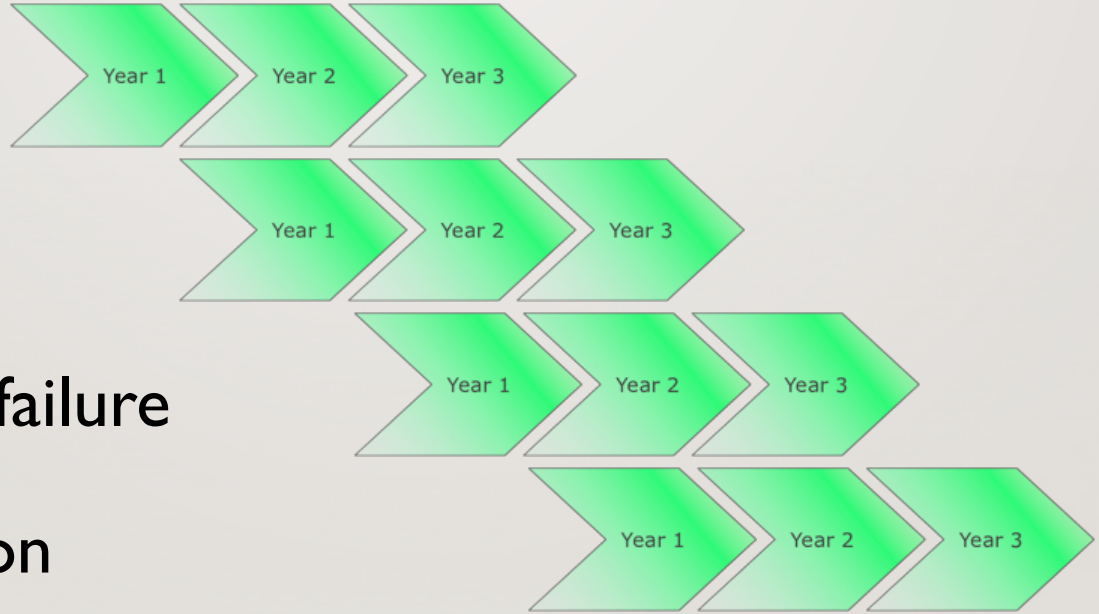
Limp to finish
Declare victory
Never finish migrations
Move on to new initiative

Legacy crap

Legacy crap

Previous CIO's failure

New CIO's vision



ACTIVITY: INTRODUCE YOURSELF

In the chat, enter the number (or numbers) that describes your situation:

1. My job title includes the word "architect."
2. I am working toward an architecture job.
3. My company treats architecture as a role, not a title.
4. My company treats architecture as a dirty word.

COURSE INTRODUCTION AND GOALS

YOUR INSTRUCTOR

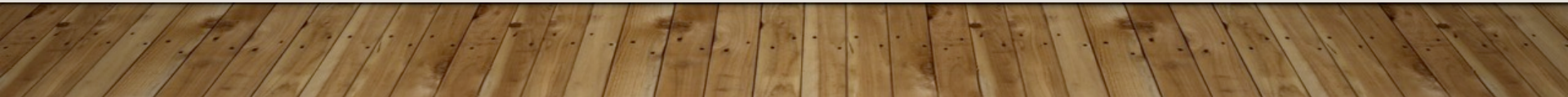
Michael Nygard

- Developer
- Architect
- Operations
- “Systems” person
- Author

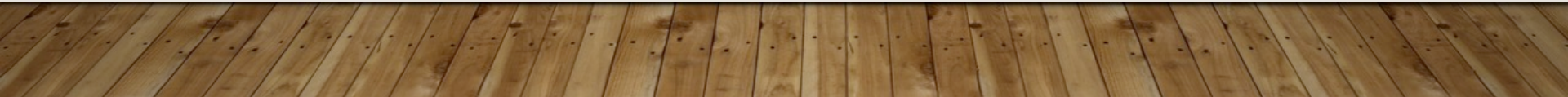
Build software to survive production, not just to pass QA

E-book available on Safari

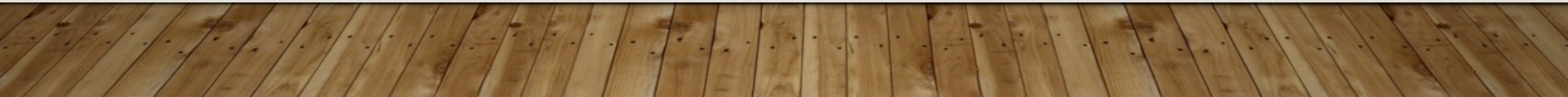
Learn how to design systems that evolve over time in the face of technological and business change.



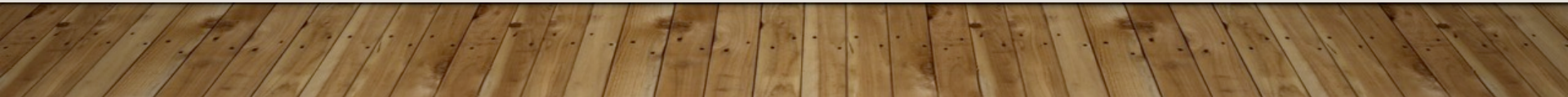
Learn how to **design systems** that evolve over time in the face of technological and business change.



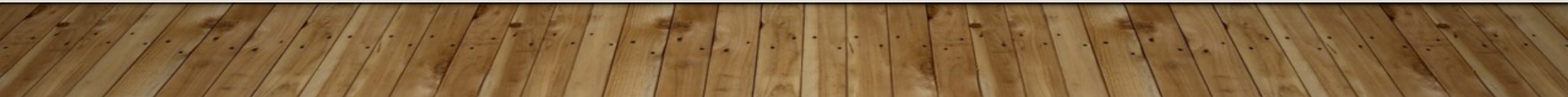
Learn how to design systems that **evolve over time** in the face of technological and business change.



Learn how to design systems that evolve over time in the face of technological and business change.



Learn how to design systems that evolve over time in the face of technological and business change.



YOU WILL BE ABLE TO

- Define internal and external boundaries
- Identify interfaces
- Partition and assign responsibilities
- Separate concerns for independent change
- Isolate information
- Build systems in simpler pieces

ARCHITECTURE AND TRADE-OFFS

WHAT IS ARCHITECTURE

1. Structure of a system

WHAT IS ARCHITECTURE

1. Structure of a system
2. Components and their relationships

WHAT IS ARCHITECTURE

1. Structure of a system
2. Components and their relationships
3. Mechanisms for cross-cutting concerns

PURPOSE OF SOFTWARE ARCHITECTURE

1. Choose and create desirable system-wide properties

PURPOSE OF SOFTWARE ARCHITECTURE

1. Choose and create desirable system-wide properties
2. Make trade-offs with deliberation and understanding

PURPOSE OF SOFTWARE ARCHITECTURE

1. Choose and create desirable system-wide properties
2. Make trade-offs with deliberation and understanding
3. Allow orderly construction of a system or systems

PURPOSE OF SOFTWARE ARCHITECTURE

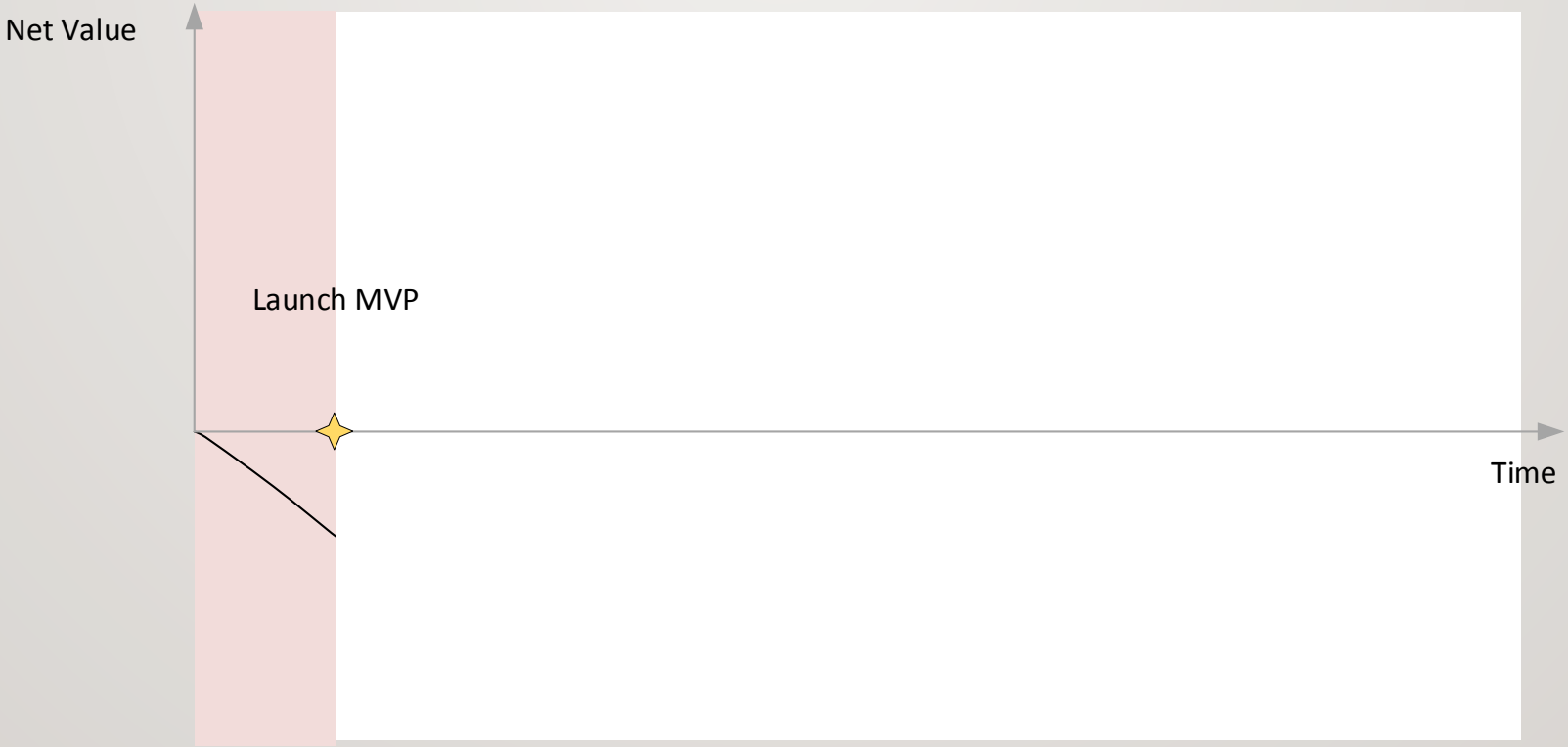
1. Choose and create desirable system-wide properties
2. Make trade-offs with deliberation and understanding
3. Allow orderly construction of a system or systems
4. Divide responsibilities among team members

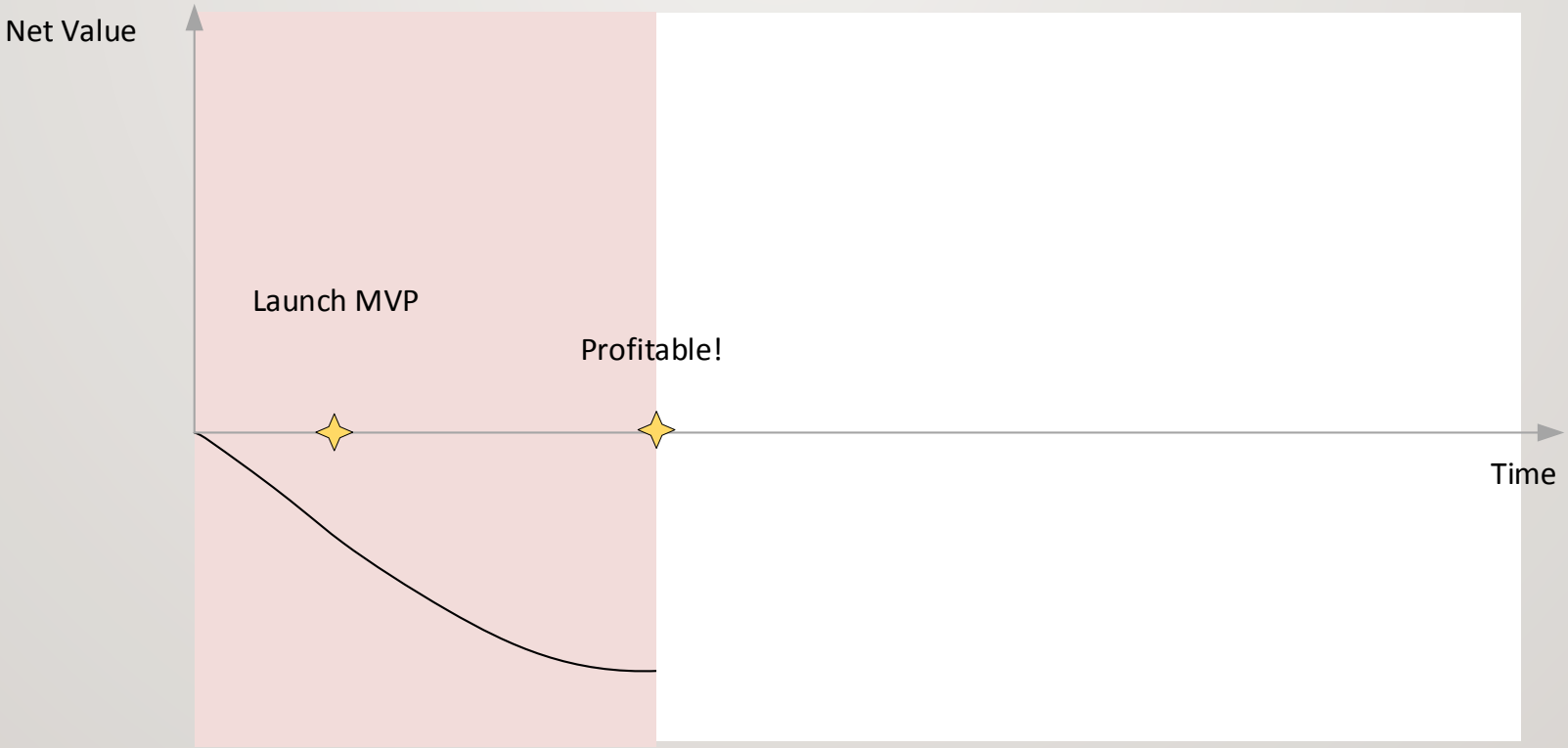
PURPOSE OF SOFTWARE ARCHITECTURE

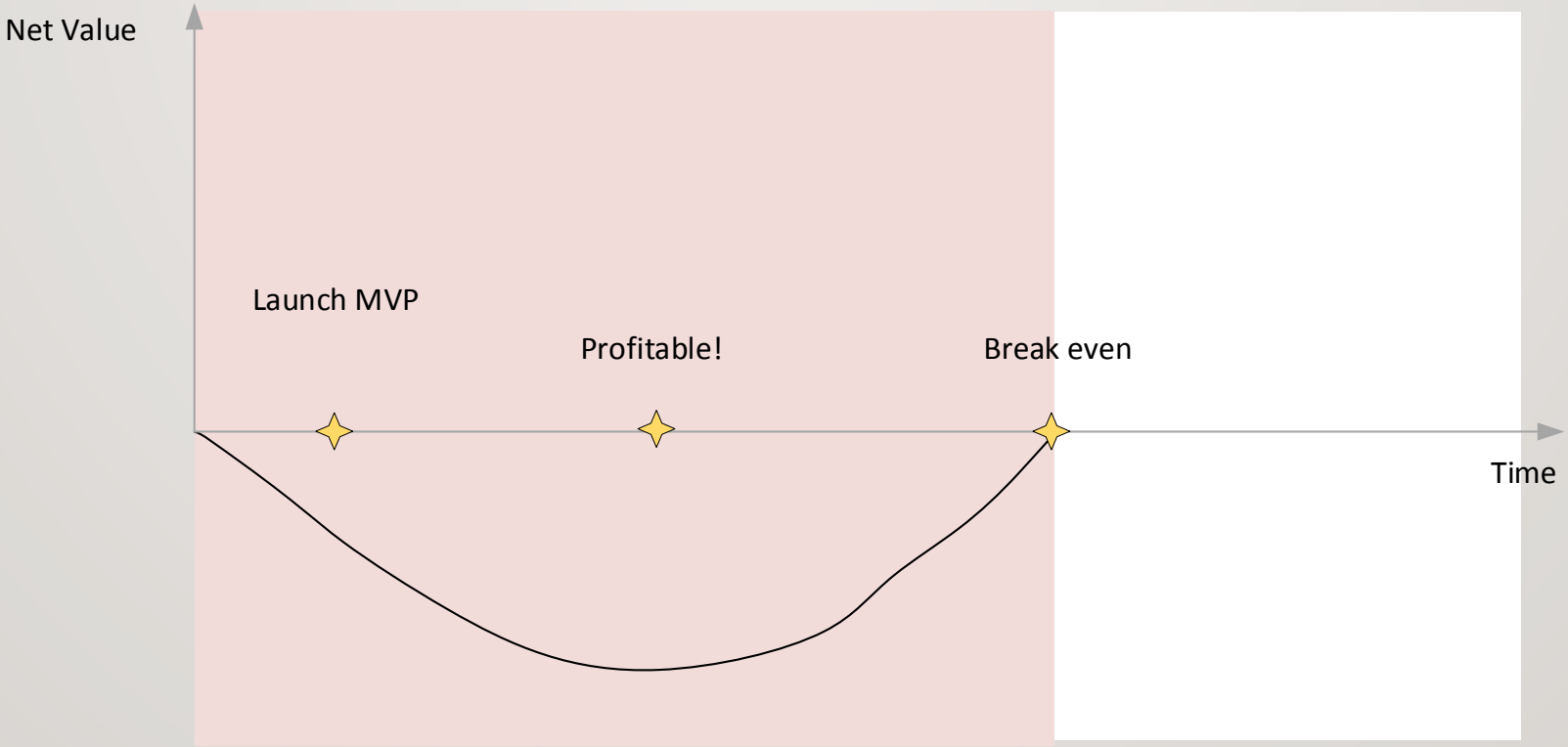
1. Choose and create desirable system-wide properties
2. Make trade-offs with deliberation and understanding
3. Allow orderly construction of a system or systems
4. Divide responsibilities among team members
5. Provide common vocabulary

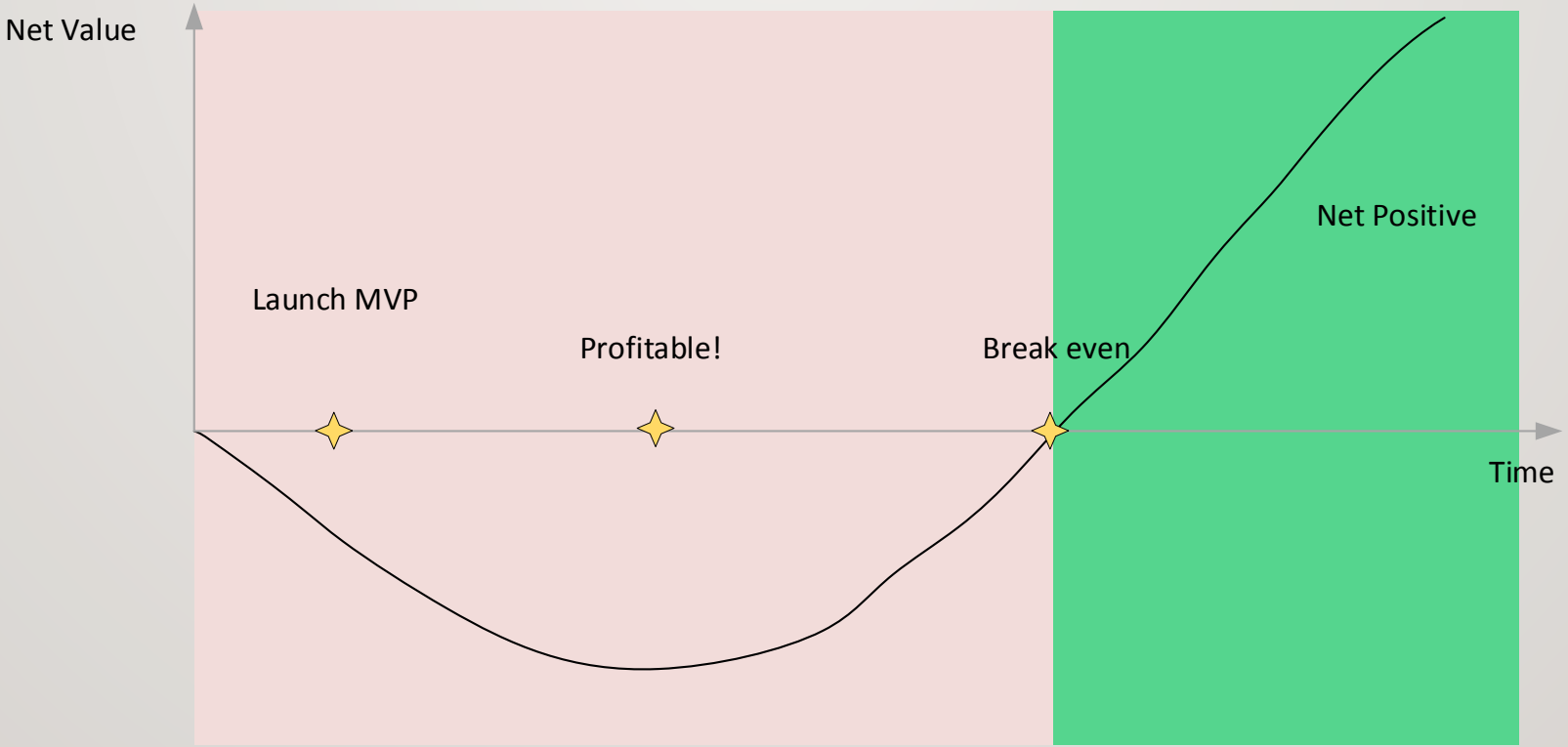
PURPOSE OF SOFTWARE ARCHITECTURE

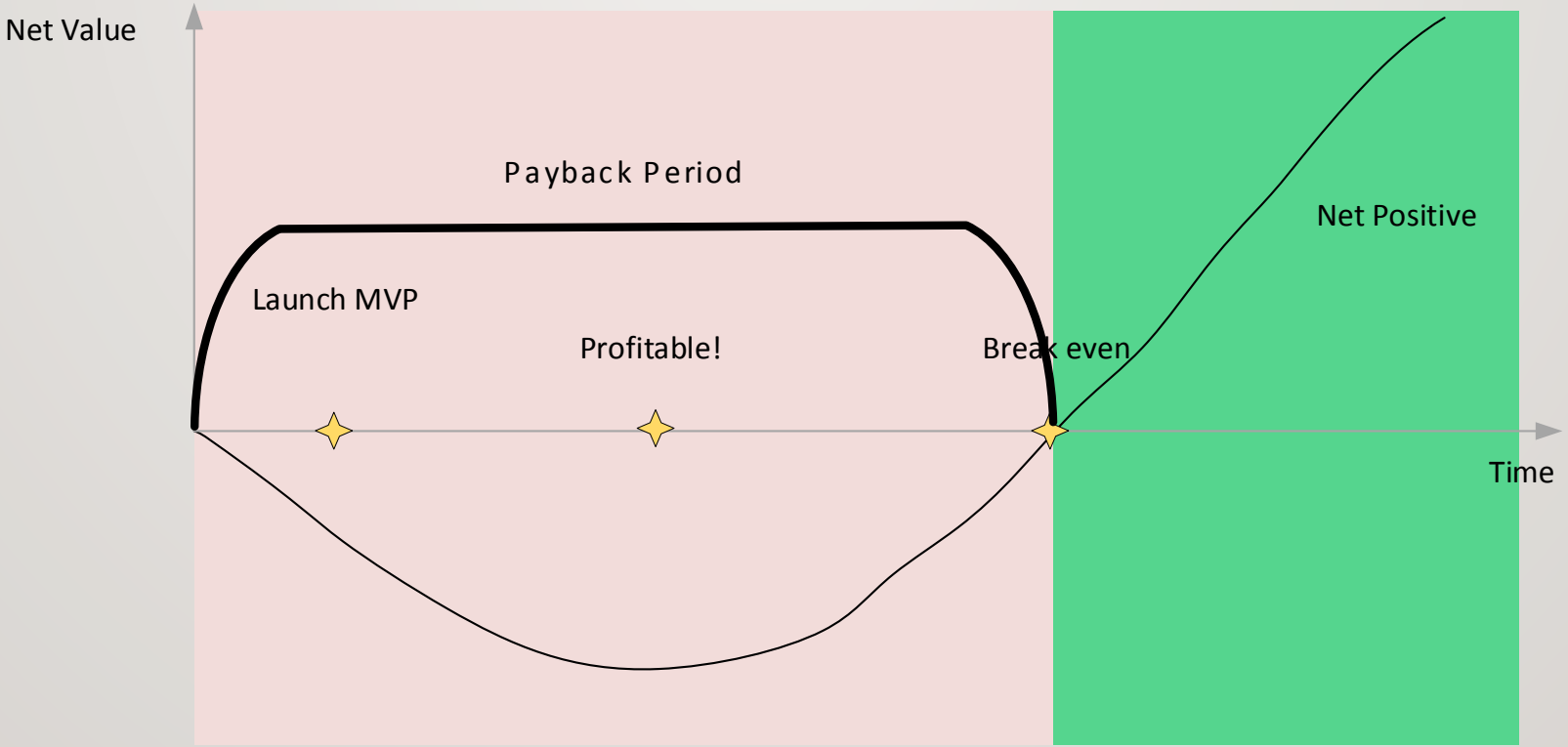
1. Choose and create desirable system-wide properties
2. Make trade-offs with deliberation and understanding
3. Allow orderly construction of a system or systems
4. Divide responsibilities among team members
5. Provide common vocabulary
6. Manage cost


Net Value
Launch MVP
Time


Net Value
Launch MVP
Profitable!
Time


Net Value
Launch MVP
Profitable!
Break even
Time

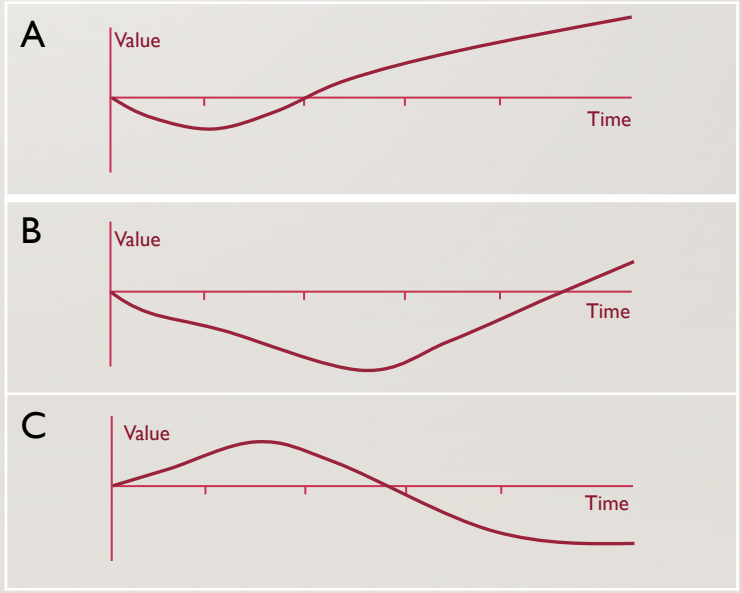

Net Value
Launch MVP
Profitable!
Break even
Net Positive
Time


Net Value
Payback Period
Launch MVP
Profitable!
Break even
Net Positive
Time

ACTIVITY: MATCH VALUE CURVE TO PLAN

In the chat, enter the "ENT" and the letter for the value curve that matches this project

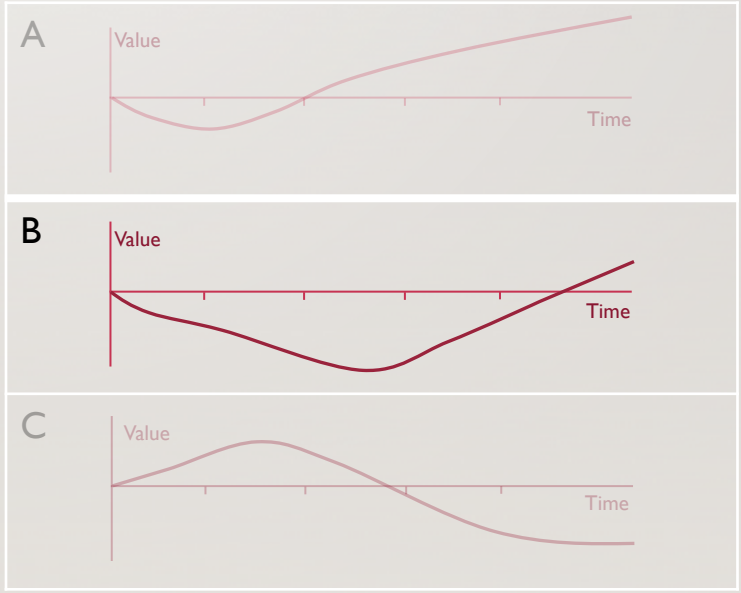
- Enterprise integration effort.
  Year 1: PoC, Vendor bake-off
  Year 2: Implementation team
  Year 3: Migrate applications



ACTIVITY: MATCH VALUE CURVE TO PLAN

In the chat, enter the "ENT" and the letter for the value curve that matches this project

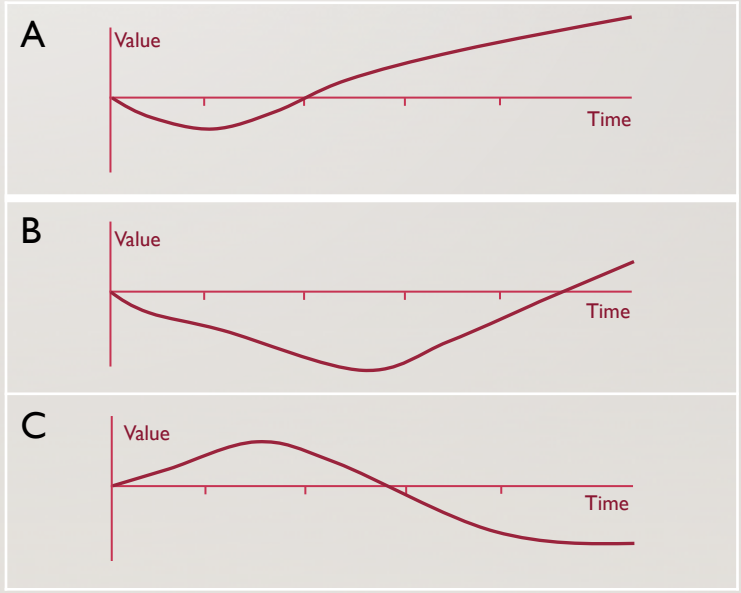
- Enterprise integration effort.
  Year 1: PoC, Vendor bake-off
  Year 2: Implementation team
  Year 3: Migrate applications



ACTIVITY: MATCH VALUE CURVE TO PLAN 2

In the chat, enter the "ACQ" and the letter for the value curve that matches this project

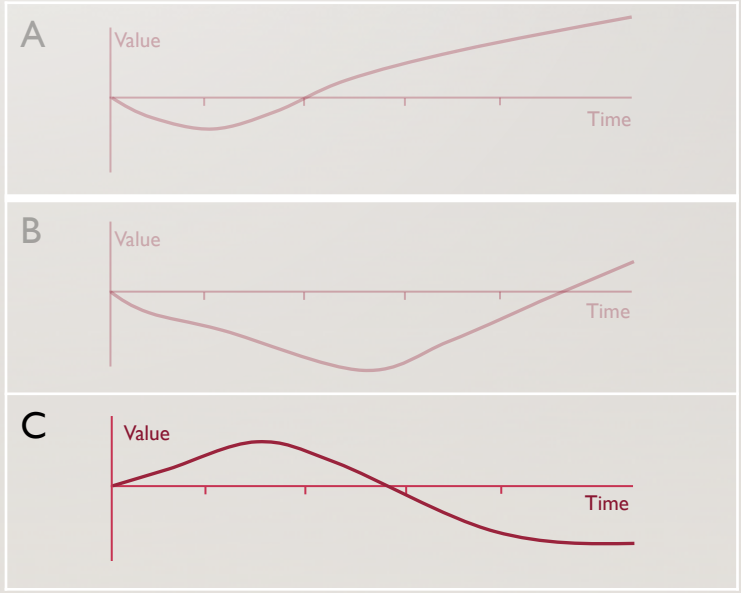
- Product Acquisition
  Buy a successful startup, reap profits
  Architectural debt slows releases, requires more developers.



ACTIVITY: MATCH VALUE CURVE TO PLAN 2

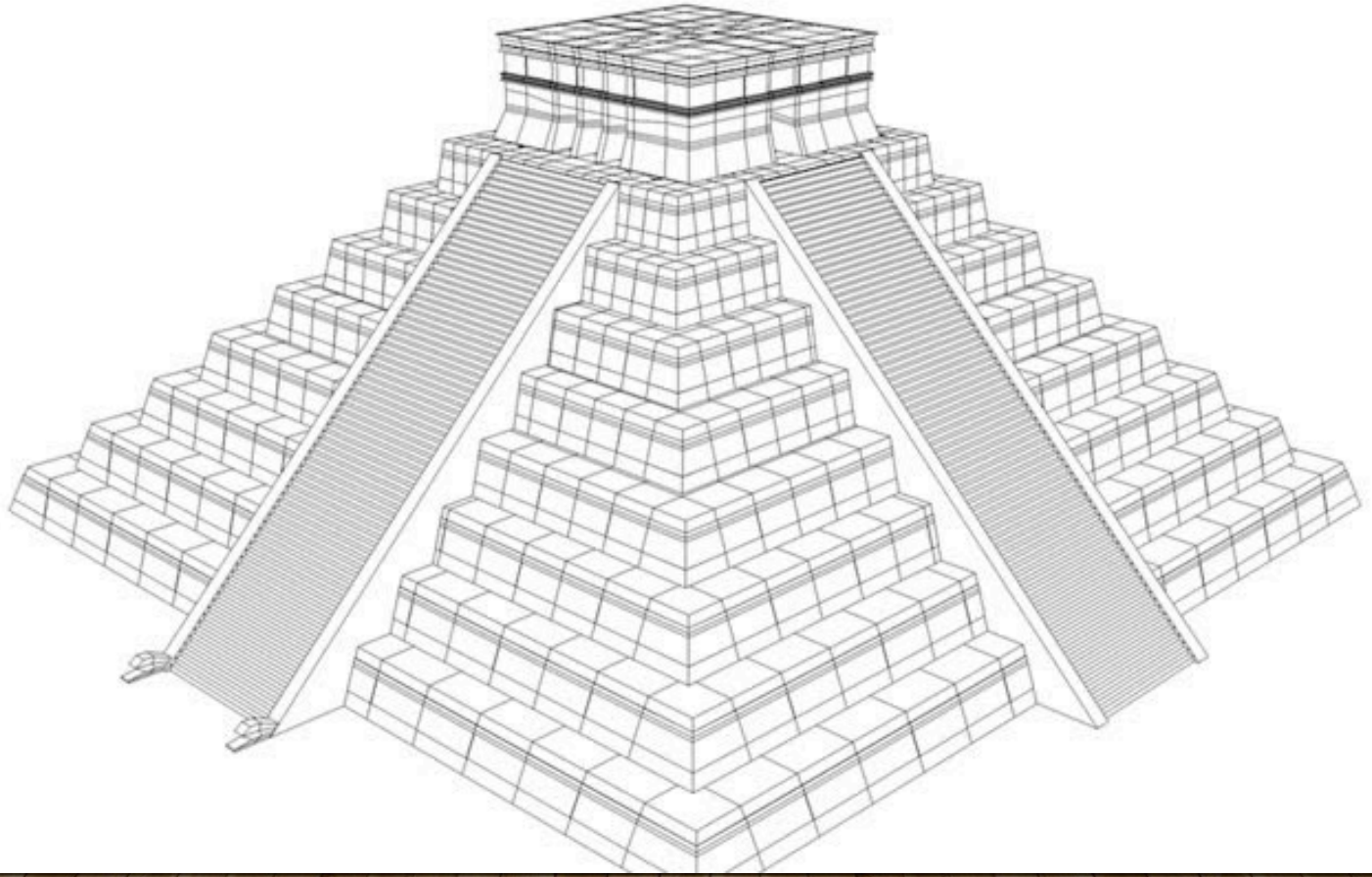
In the chat, enter the "ACQ" and the letter for the value curve that matches this project

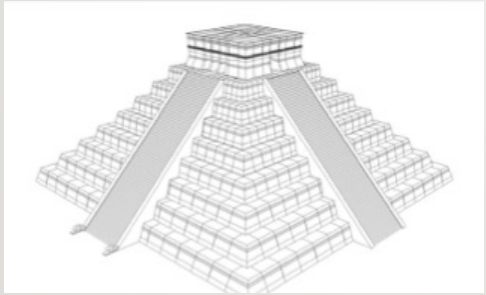
- Product Acquisition
  Buy a successful startup, reap profits
  Architectural debt slows releases, requires more developers.



ARCHITECTURE AND THE VALUE CURVE

- Design for early returns
- Systems should support themselves from the beginning
- But don't sacrifice the future





Build pyramids not arches

FUNDAMENTALS OF ARCHITECTURE

THE SAMPLE SYSTEM

CONTEXT DIAGRAM

ARROWS ACROSS THE BOUNDARY

ARCHITECTURE QUALITIES

ARCHITECTURALLY SIGNIFICANT REQUIREMENTS

ASRs

CONSTRAINTS

CHECKPOINT

Where are we now?

DECOMPOSING THE SYSTEM

VIEWS YOU CAN USE

C4 – Context, Containers, Components, Classes

THE SAMPLE SYSTEM

PURPOSE OF THE SAMPLE SYSTEM

- Allows me to illustrate with a real-ish system
- Allows us to compare solutions
- Relatively simple domain
- Enough complexity to be interesting

- Apologies in advance if this is your actual business plan

THE SYSTEM

- “Subscriptions as a Service” startup
- Supporting local businesses: dog walkers, dry cleaning
- “White label” sites
- We handle the money: bill customers, pay providers
- We handle customer service calls

PROVIDERS

- Set up their goods & services
- Define their service area
- Decide how often they can deliver (weekly, monthly, quarterly)
- But not inventory. We do not manage inventory.

SUBSCRIBERS

- Think they're dealing with the provider
- Choose a tier of service or package of goods
- Decide how often they want service
- Provide a payment method
- Decide whether to auto-renew

YOU

- Play the role of the technical co-founder
- It's your money on the line
- Seed round investors are interested but wary
- They want to see you get to $1,000,000 ARR (annual recurring revenue)

Clearing the Question Queue

I'll take some time to answer questions about the sample system.

ACTIVITY: WHERE DO WE START?

In the chat, pick the activity you think we should start with. Give one sentence why:

1. Pick a JavaScript framework
2. Decide the team structure
3. Shape the fundamental architecture
4. Explore the problem space
5. Make a decision matrix
6. Sign up for AWS and define your microservice APIs
7. Something else

ACTIVITY: WHERE DO WE START?

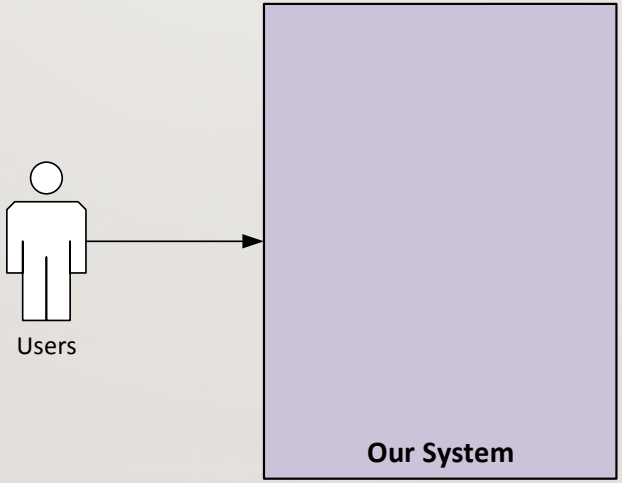
In the chat, pick the activity you think we should start with. Give one sentence why:

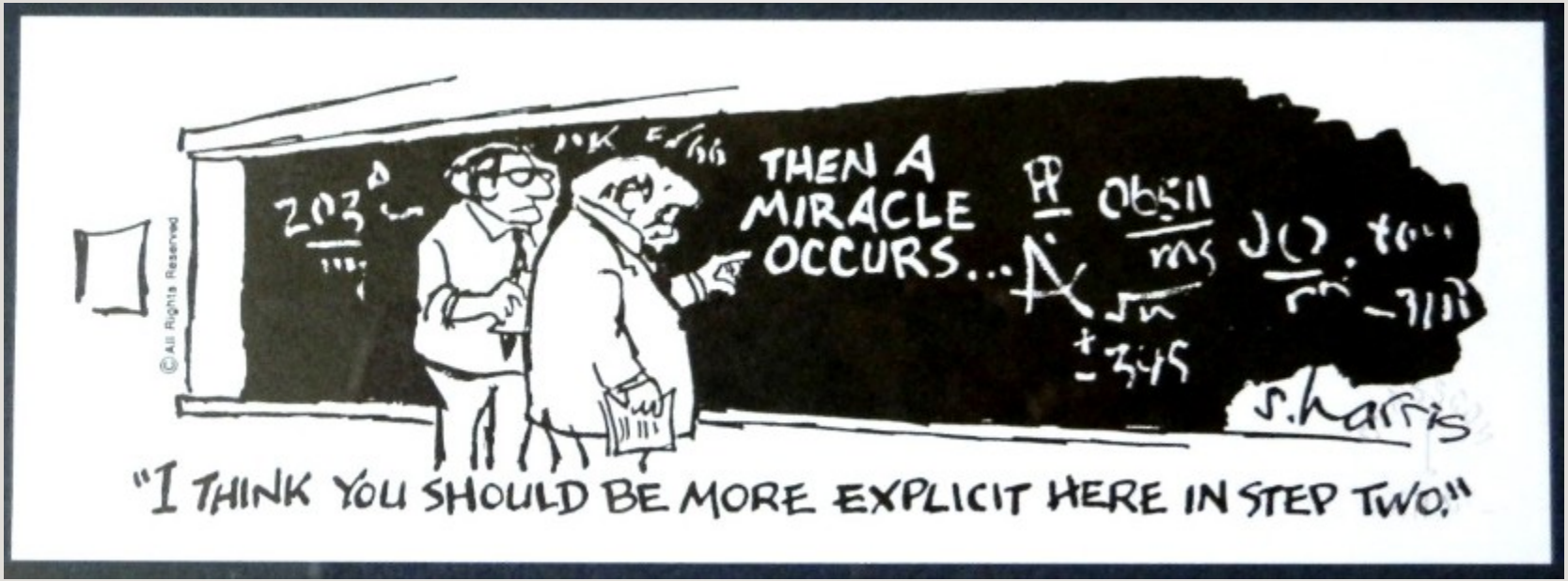
1. Pick a JavaScript framework
2. Decide the team structure
3. Shape the fundamental architecture
4. Explore the problem space
5. Make a decision matrix
6. Sign up for AWS and define your microservice APIs
7. Something else

CONTEXT DIAGRAM

SYSTEM CONTEXT

- Every system exists in a context, defined by:
  - Users
  - Other systems


Users
Our System



LOOK FOR ALL THE CONSTITUENTS

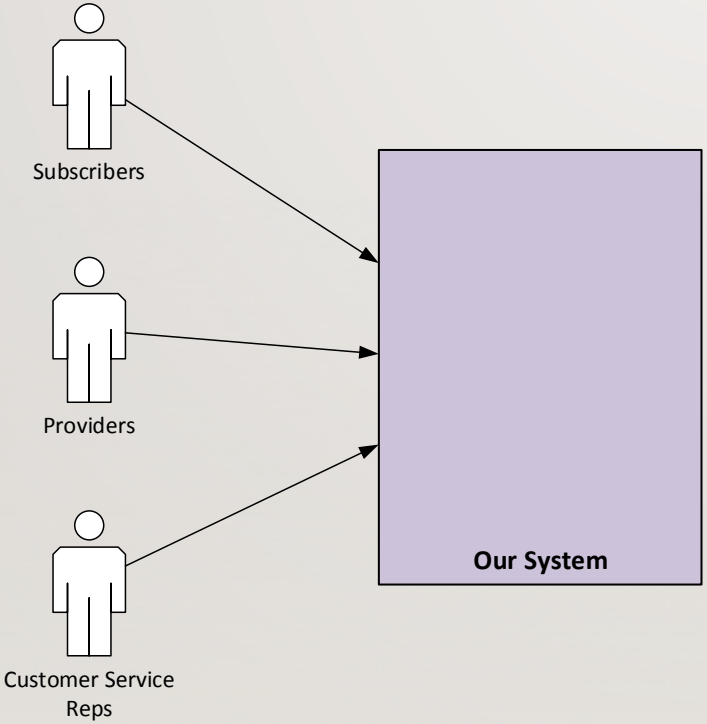
- Customers are easy to spot
- Look for users that facilitate business:
  - Customer service reps & managers
  - Finance staff
  - Sales
- Look for users that run the system itself:
  - Technicians
  - Operators & administrators

ACTIVITY: USER GROUPS FOR OUR SYSTEM

In the chat, name the user groups that you can identify.

Think about:

1. People who pay the bills
2. People who run the business
3. People who run the system



DECISION TIME!

How do providers get set up?

You choose:

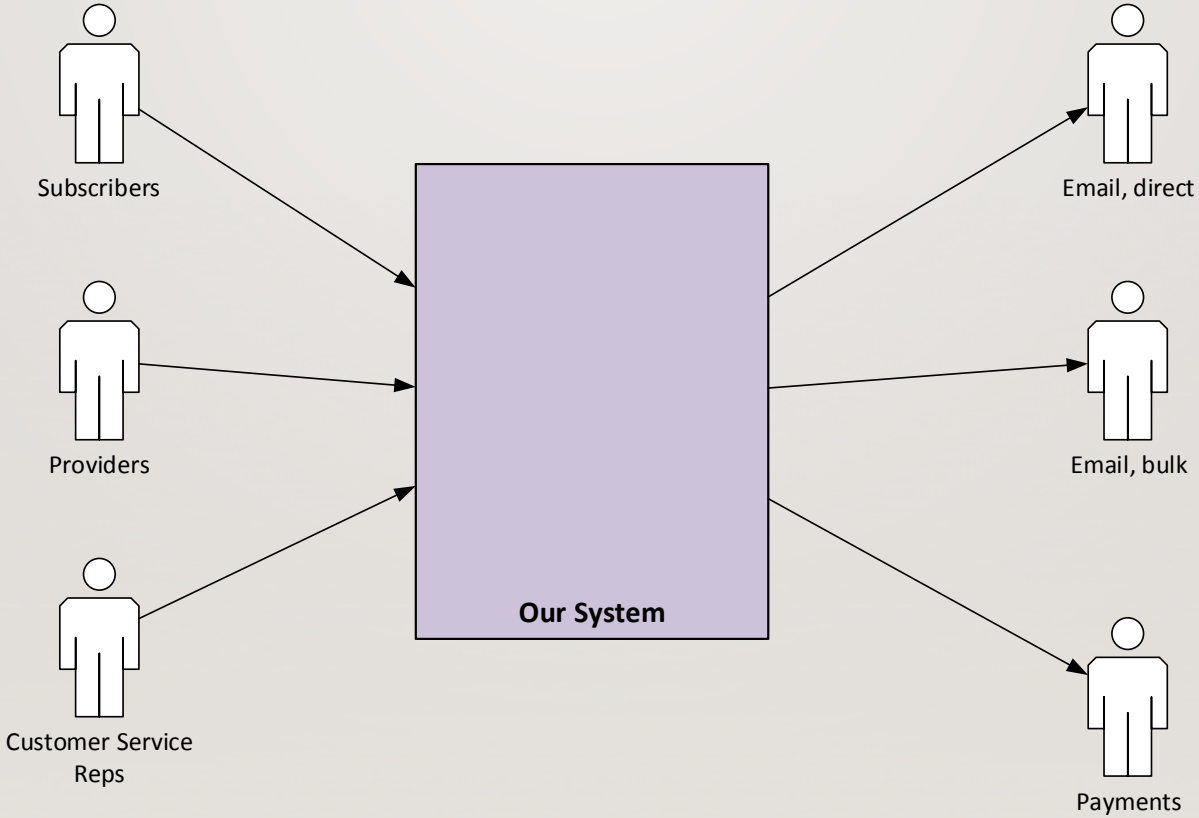
A. Self-service.

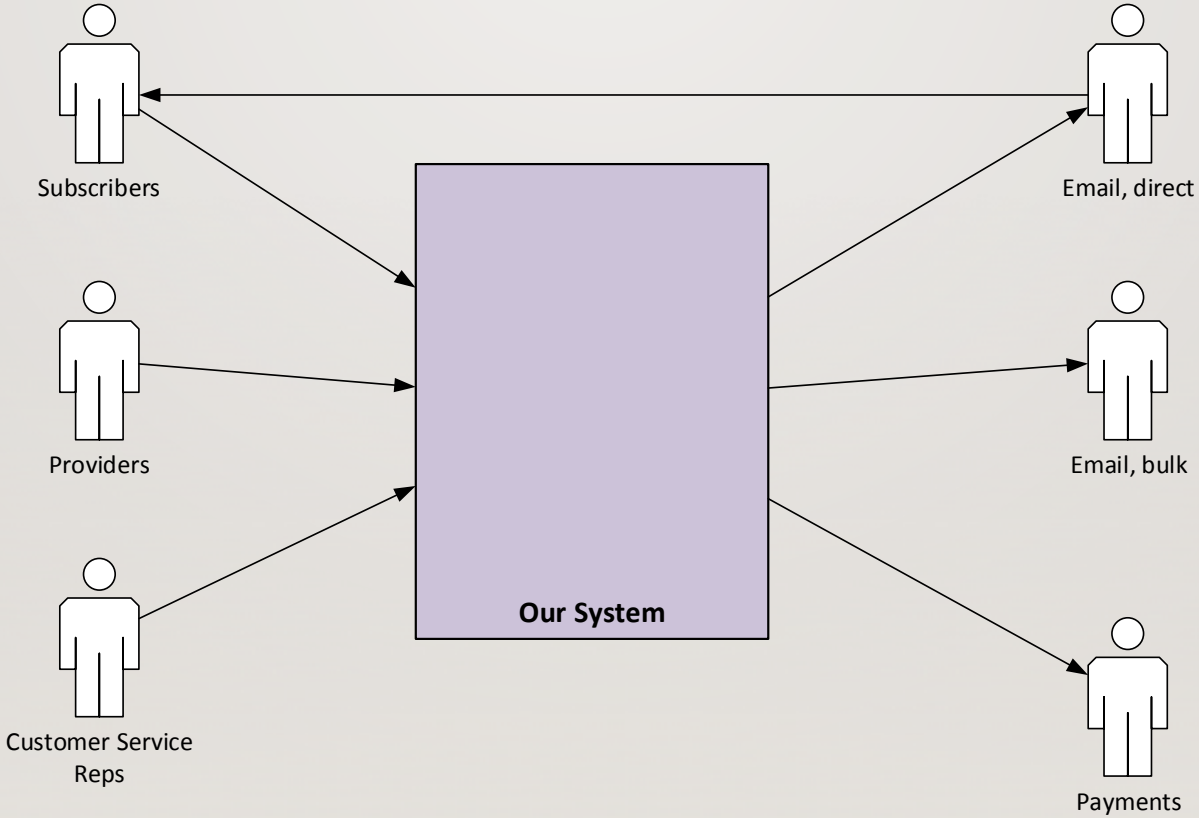
B. Sales reps on the ground.

If you chose B, then the sales reps are a new user group.

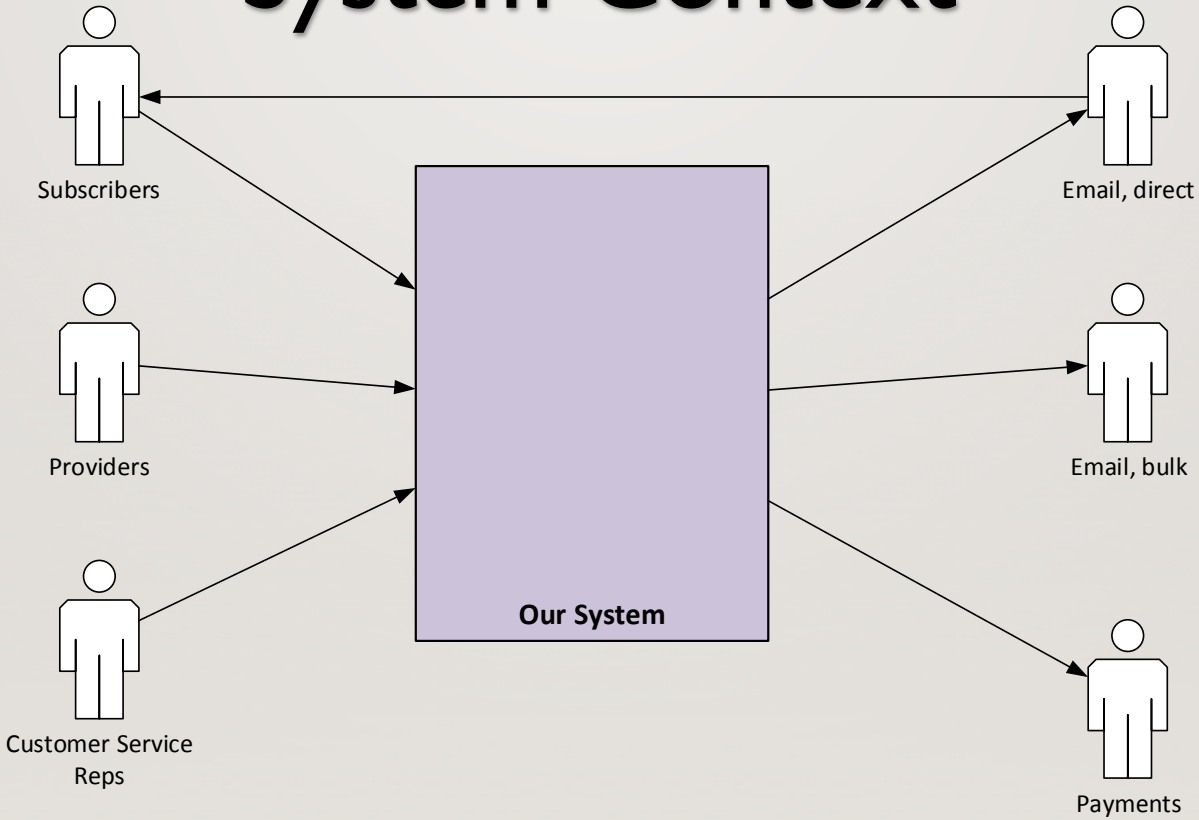
OTHER SYSTEMS: WHAT SHALL WE OUTSOURCE?

- Email, bulk and direct
- Payments
- Calendar?
- Currency exchange?
- Authentication?
- Authorization?
- SSO?
- Monitoring?
- Alerting?
- Fraud detection?


Subscribers
Providers
Customer Service Reps
Our System
Email, direct
Email, bulk
Payments


Subscribers
Email, direct
Providers
Email, bulk
Our System
Customer Service Reps
Payments

System Context



SYSTEM INTERIOR

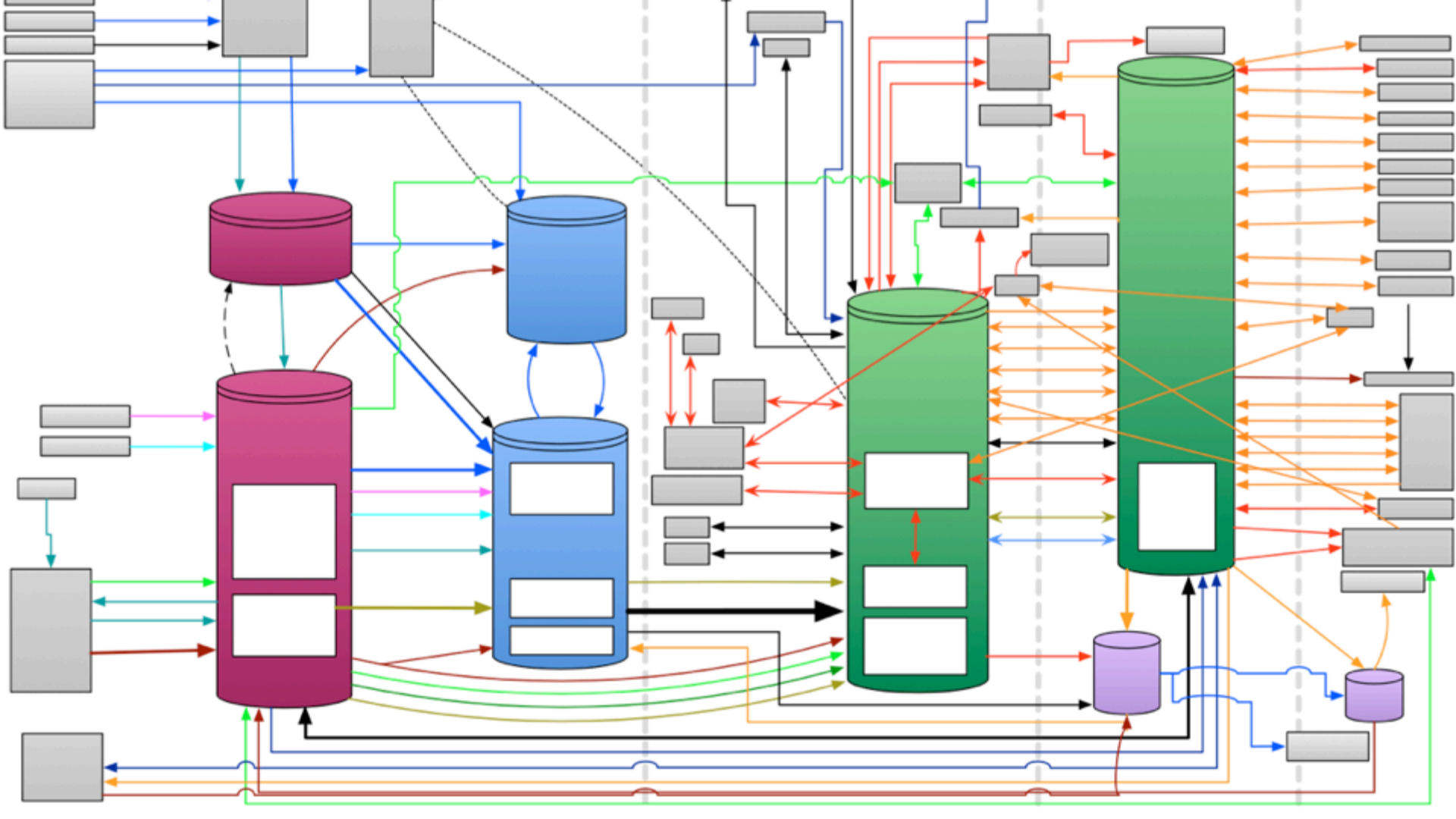
- Left blank on purpose
- For now, focus on the *interfaces*

ABOUT DIAGRAMS

- Diagrams, and documents in general, get a bad rap
- We do them to help discover and communicate, **not** because the process tell us to
- Keep them lightweight
- Provide a legend

VIEWS AND VIEWPOINTS

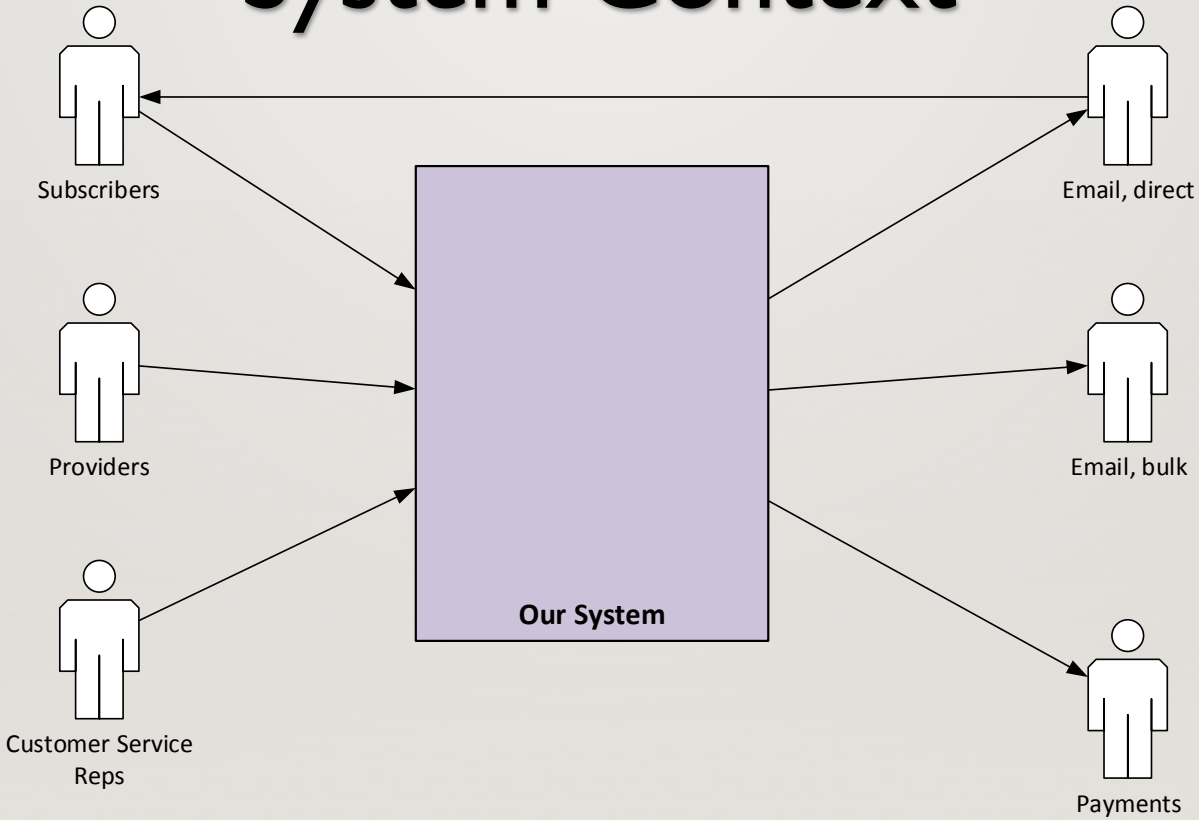
- Different team members have different needs
- One diagram **cannot** serve them all.



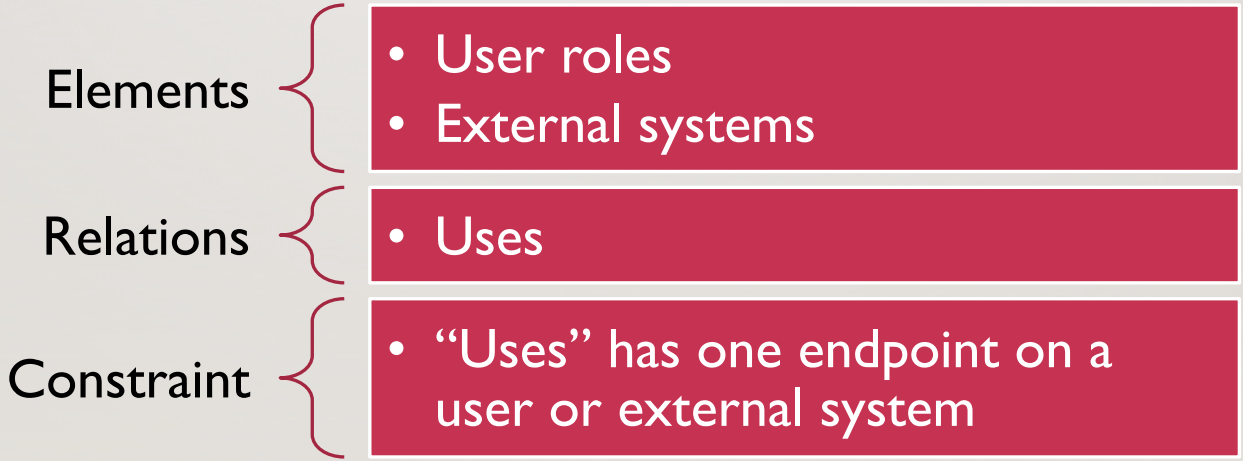
VIEWS AND VIEWPOINTS

- Different team members have different needs
- One diagram **cannot** serve them all.
- A *view* has a notation:
  - Elements
  - Relations
  - Constraints
- And a usage

System Context



SYSTEM CONTEXT VIEW



ACTIVITY: CONTEXT DIAGRAM

Take a moment to create your own context diagram. Use any tool at hand:

- Diagramming tools: Visio, OmniGraffle, Gliffy
- Text-to-diagram: PlantUML
- Whiteboard + phone camera
- Paper napkin

Feel free to share it in chat.

Have you found more user groups?

Are you integrating with more systems?

ARROWS ACROSS THE BOUNDARY

interface (n.)

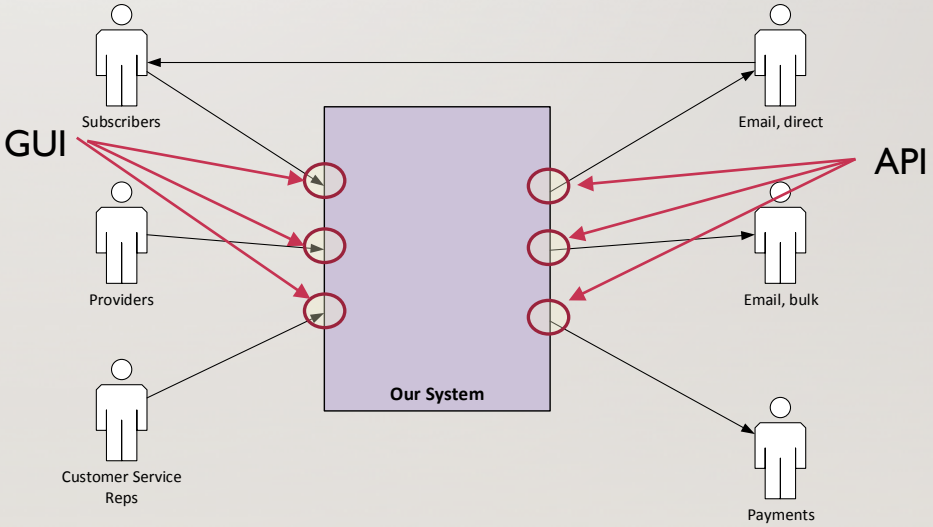
"a plane surface regarded as the common boundary of two bodies," 1874

facade (n.)

"front of a building," 1650s, from French *façade*

ARROWS ARE INTERFACES

- We pay too much attention to the boxes, not enough to the arrows.
- Every place an arrow crosses a boundary, we have an interface



FOR EACH ROLE

- Identify the constituents
- Find their needs, create use cases
- Determine how their needs get prioritized

FOR EACH API

- Discover throughput & latency requirements
- Decide on synchronicity
- Decide transport, framing, semantics
- Define test harness needs

ID	Name	Initiation	Sync?	Frequency	Size per	Transport	Framing	Encoding	Semantics
IF1	F5 Commands	Outbound	Sync	1 / min	5 KB	HTTP	XML	UTF-8	F5 Big IP
IF2	Healthcheck	Outbound	Sync	12 / min	1 KB	HTTP	JSON	UTF-8	Doc
IF3	Log aggregation	Inbound	Sync	12 / min	100 KB	SSH	Syslog	ASCII	Doc

ACTIVITY: INTERFACE TABLE

Create an interface table for our sample system:

- Outbound payment processor
- Outbound email, direct
- Outbound email, bulk

Estimate volumes like a Fermi problem:

- How many subscribers do we expect?
- How many emails do we send them, over what time period?

https://en.wikipedia.org/wiki/Fermi_problem

Clearing the Question Queue

I'll take some time to answer questions about the sample system.

ARCHITECTURE QUALITIES

ARCHITECTURE QUALITIES

A.k.a. "the –ilities"

Sometimes called "non-functional requirements"

QUALITIES OBSERVED AT RUN-TIME

- Performance
- Security
- Availability
- Usability

From “Software Architecture in Practice”, Bass, Clements, Kazman

QUALITIES **NOT** OBSERVED AT RUN-TIME

- Scalability
- Modifiability
- Portability
- Integrability
- Reusability
- Testability

From “Software Architecture in Practice”, Bass, Clements, Kazman

There can be only one

top priority.

RANK THE QUALITIES

- Important to guide trade-offs later
- Very important that this isn't just the tech team ranking these
- Hard discussions uncover assumptions

ACTIVITY: RANK THE QUALITIES

For our sample system, what are the most important qualities?

Write your top three in chat as one post, with the most important one first.

You are making a decision here, not taking a quiz.

But be sure you can make the case for your choice!

MVP, OFR, & TECH DEBT

Sometimes, "tech debt" just means architecture priorities have changed.

During	Emphasize
Before MVP	Modifiability
Finding Product/Market Fit	Scalability, Modifiability
Ramping Up	Security, Availability, Scalability

ARCHITECTURALLY SIGNIFICANT REQUIREMENTS

ASRs

https://xkcd.com/1425/
Sept, 2014

Sept, 2014

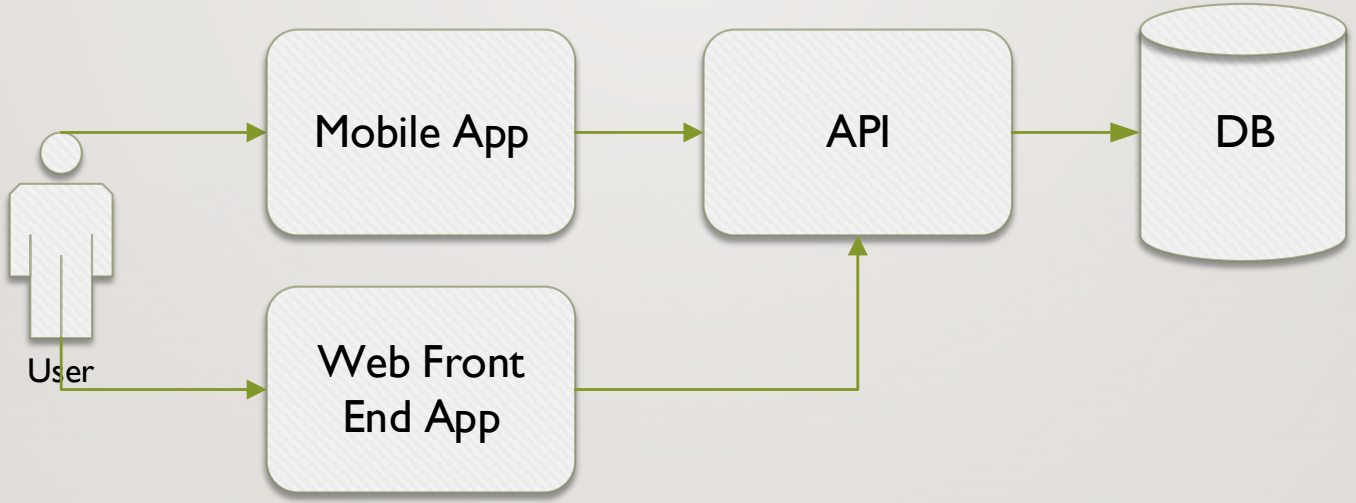
Oct, 2014

Feb, 2004

Sept, 2014

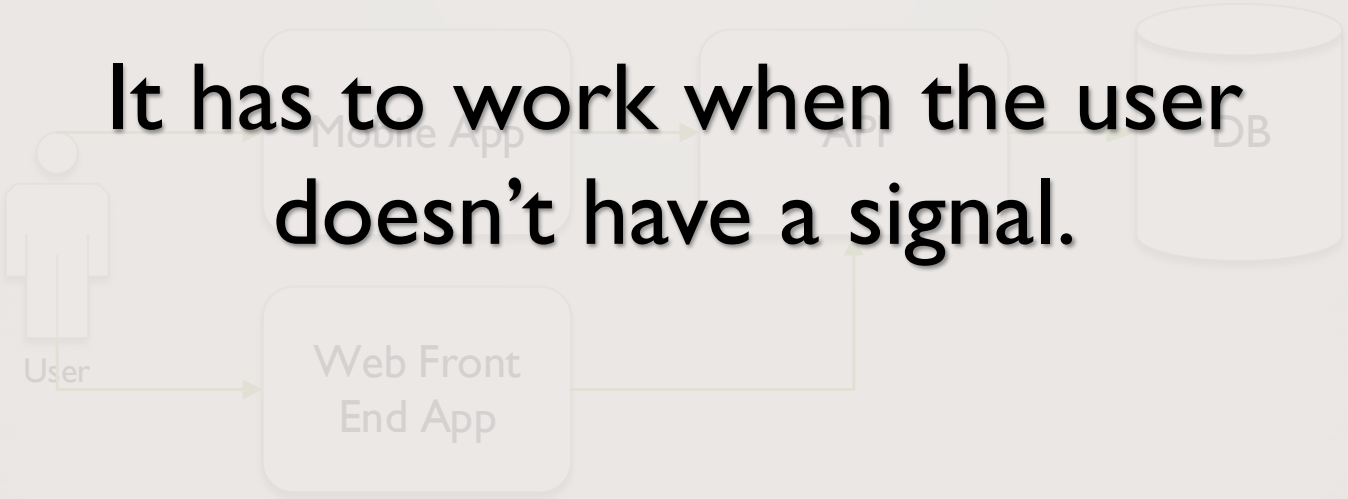
Oct, 2014

flickr



EXAMPLE: MOBILE + WEB APP

It has to work when the user doesn't have a signal.



EXAMPLE: MOBILE + WEB APP

ASR

- Dramatically alters the architecture
- Causes addition of new "moving parts"
- Without it, the system will fail

"Architecture Killers"

HUNTING FOR ASRS

- Multiple languages, currencies, timezones
- Compliance & reporting
- Disconnected operation
- Any kind of database synchronization
- Active/active deployments
- Intelligence & semi-automated processes
- Customer service needs
- Human overrides
- High volume, low latency
- Strict serialization
- Interface with physical objects that move (especially when momentum is involved)
- Toxic, hazardous, radioactive, or remote environments

ACTIVITY: ASR OR NOT?

ID	Requirement	ASR?
R1	A vendor can set up a new service any time of day or night.	No
R2	A subscriber can upgrade a subscription any time, but can only downgrade at subscription renewal.	No
R3	A subscriber can pay with paper checks.	Yes – new role & GUI required
R4	A vendor can do their own customer service using our system.	Yes – new role, GUI, & new authn/authz
R5	A subscriber can call us to report a problem with service.	
R6	A vendor can set up items in their own catalog service and they will appear on our site.	No Yes – Data integration

CONSTRAINTS

CONSTRAINTS ARE MORE THAN REQUIREMENTS

- It maybe costly to break a requirement, but a constraint is absolute.
- If a constraint is broken, then the system **must not** go live.
- Constraints are imposed from external forces:
  - Law
  - Industry regulation
  - Stakeholders

ID	Headline	Originator	Local Expert	Brief Description	More Detail
C1	GDPR	EU Law	Niles Summerbottom	Limits on collection & storage of PII. “Right to be forgotten”	Doc
C2	CAN SPAM	US 16 CFR Part 316	Ricky Bobby	Restrictions and requirements on email	Doc
C3	PCI DSS	Payment Card Industry	Scrooge McDuck	Restrictions on handling credit card data	Doc
C4	HIPAA	…			
C5	Peppa	…			
…					

CONSTRAINTS ARE LESS UNIQUE THAN REQUIREMENTS

- We will find similar constraints when we look at systems in the same:
  - Domain
  - Market
  - Company

- Much of the guidance can be shared from one system to the next.
- But it's usually left as tacit knowledge.

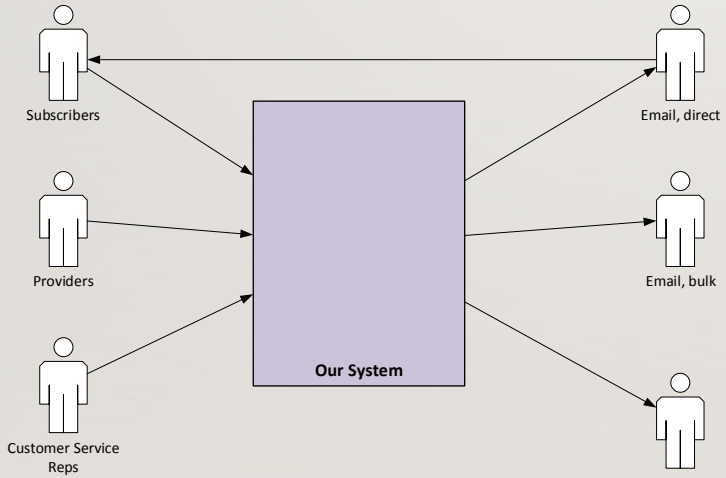
ACTIVITY: CONSTRAINTS IN YOUR LOCALE?

- In the chat, enter some constraints that our sample system would face in your market.

CHECKPOINT

Where are we now?

THE STORY SO FAR



Context Diagram

- ASR
  - White-label with custom domains
- Constraints
  - CAN SPAM, PCI DSS, GDPR
- Architecture qualities
  1. Availability
  2. Security
  3. Scalability

Time to open the box and go inside

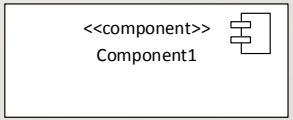
DECOMPOSING THE SYSTEM

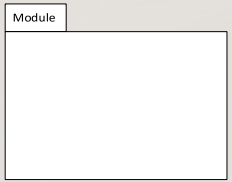
INTERNAL BOUNDARIES

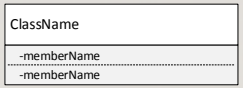
- Our decomposition *must*:
  - Terminate every external interface (human or system) at a component.
  - Deliver the features
- Our decomposition *should*:
  - Support the system-wide qualities
  - Allow independent development
  - Separate concerns into mechanisms
  - Isolate the effects of changes

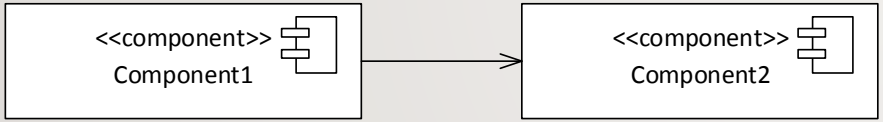
TERMINOLOGY & NOTATION

- Component
  - Part of the dynamic behavior
  - A runtime entity
  - Interacts with other components
- Module
  - Part of the static structure
  - Development/compile time entity

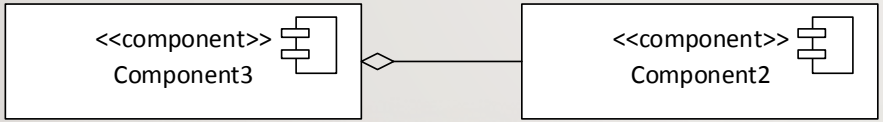




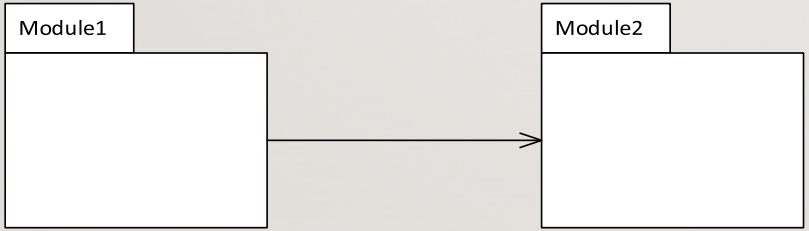




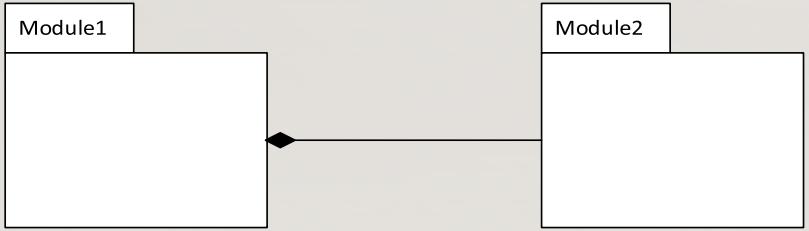
Statement about runtime behavior.
Component1 calls Component2



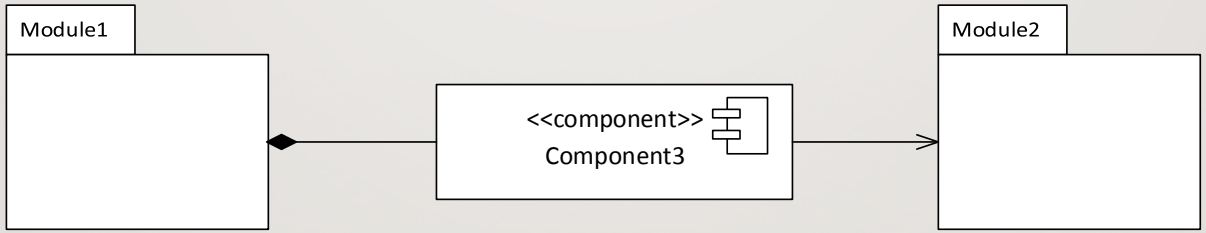
Statement about runtime behavior.
Component1 contains Component2



Statement about source code structure.
Module1 uses Module2.



Statement about source code structure.
Module1 encloses Module2.

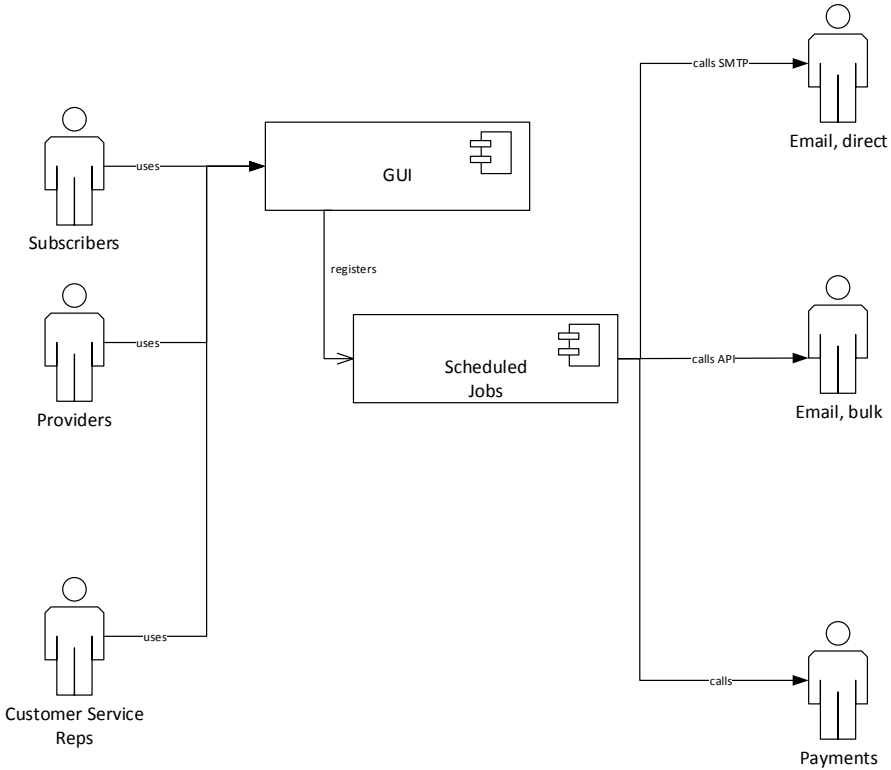


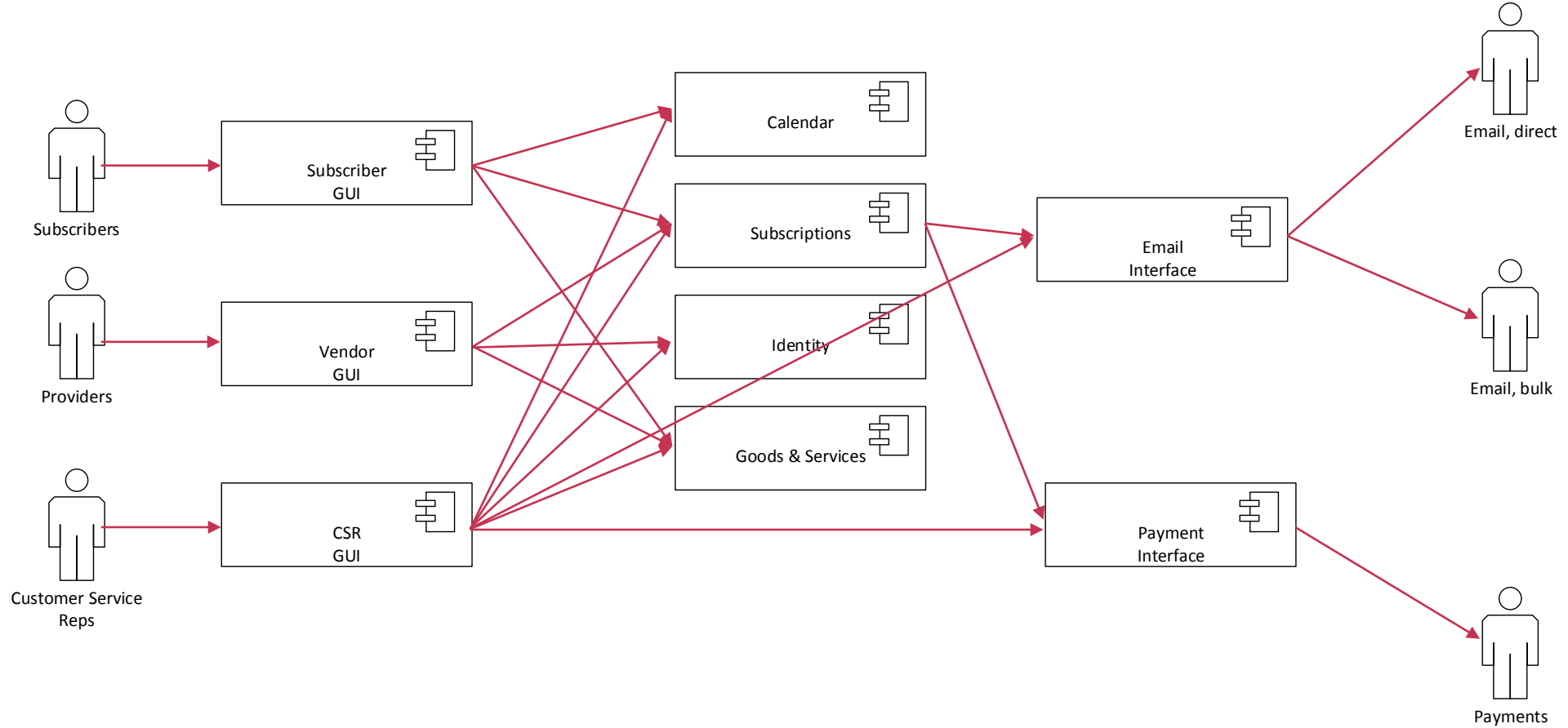
Doesn’t look like anything to me.

I have no idea what this is supposed to mean.

MAXIMS

- Use both static and dynamic structures
  - Can this be a library?
  - Does this need to be a service?
- Look for common mechanisms you can factor out.
- Ask "what knowledge must be shared across this boundary?"


Subscribers
uses
GUI
registers
Scheduled Jobs
Providers
uses
Customer Service Reps
uses
calls SMTP
Email, direct
calls API
Email, bulk
calls
Payments


Subscribers
Providers
Customer Service Reps
Subscriber GUI
Vendor GUI
CSR GUI
Calendar
Subscriptions
Identity
Goods & Services
Email Interface
Payment Interface
Email, direct
Email, bulk
Payments

EVALUATE YOUR DECOMPOSITION

- "Test" your design by simulation.
- Assign people to play each component
- Pick a use case
- Pass a ball or token around to simulate control flow
- Walk through the use case and pass the token
- Score it like golf

INFORMATION HIDING

- David Parnas, "On the Criteria To Be Used in Decomposing Systems into Modules", Communications of the ACM, 1972
- KWIC Index – a permuted index
- Input – ordered lines, made up of ordered words, made up of ordered characters.
- Output – listing of all "circular shifts" of all lines, in alphabetic order

A KWIC EXAMPLE

- Input

  Software comprises an endless supply of structures.

- Output

  an endless supply of structures. Software comprises
  comprises an endless supply of structures. Software
  endless supply of structures. Software comprises an
  of structures. Software comprises an endless supply
  Software comprises an endless supply of structures.
  structures. Software comprises an endless supply of
  supply of structures. Software comprises an endless

MODULARIZATION 1

1. **Input**
   Read EBCDIC characters, store them in core. 6-bit characters packed 4 per word. EOL is a special character.
2. **Circular shifter**
   Prepare index; pair of addr of first char of shift, original index of line in input array
3. **Alphabetizer**
   Take arrays from 1 & 2, produce new array of pairs like in 2, but in alphabetical order.
4. **Output**
   Using arrays from 1 & 3, format output
5. **Control**
   Allocate memory, call operations in1 - 4, report errors.

ACTIVITY: EFFECT OF CHANGES

For each change case listed here, count how many modules have to be changed:

1. Read and print ASCII instead of EBCDIC.
2. Stop using packed characters, store one character per word.
3. Write index for circular shifts to offline storage instead of core to support larger input documents.
4. Read and write Unicode and UTF-8. Collate properly for Unicode in locale.

Compute the "change efficiency" of this design as $E = 20 - N/20$. Enter your result in chat.

MODULARIZATION II

1. **Line Storage**
   Offers functional interface: SETCH, GETCH, GETW, DELW, DELLINE
2. **Input**
   Reads EBCDIC chars, calls line storage to put them into lines.
3. **Circular Shifter**
   Offers same interface as line storage. Makes it appear to have all shifts of all lines.
4. **Alphabetizer**
   Offers sort function INIT, and access function iTH that gets a line.
5. **Output**
   Repeatedly call iTH on alphabetizer, printing the line.
6. **Control**
   Similar to first approach, call each module in sequence.

ACTIVITY: EFFECT OF CHANGES

Let's evaluate the second modularization against the same change cases:

1. Read and print ASCII instead of EBCDIC.
2. Stop using packed characters, store one character per word.
3. Write index for circular shifts to offline storage instead of core to support larger input documents.
4. Read and write Unicode and UTF-8. Collate properly for Unicode in locale.

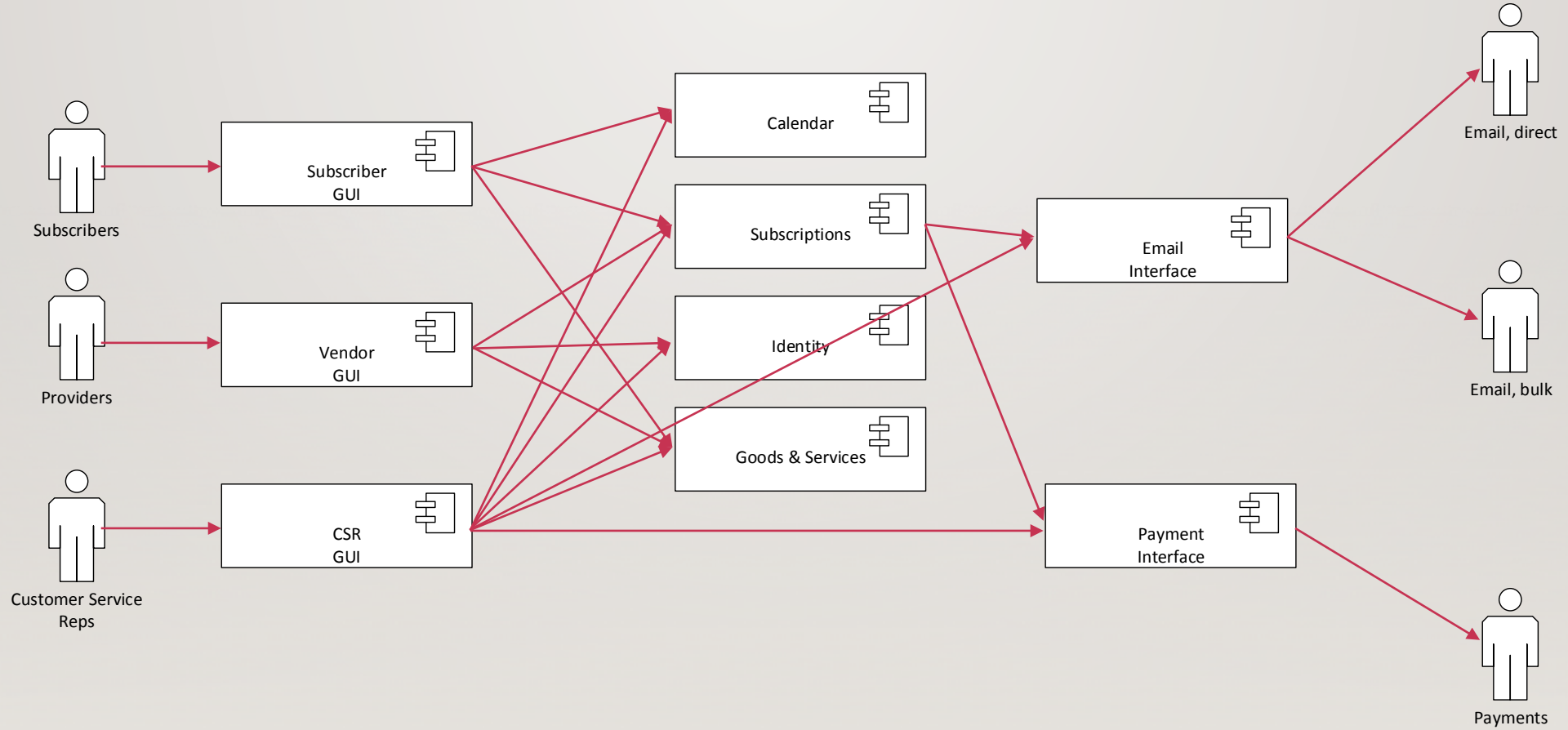
Compute the "change efficiency" of this design as $E = 24 - N/24$. Enter your result in chat.

WHY IS THE SECOND ONE BETTER?

- It hides decisions inside modules.
- Functional interfaces provide an abstract representation of the underlying data.
- Information hiding

IN OUR SAMPLE SYSTEM

- We need to decompose the interior
  - Allow multiple people to work
  - Work toward our system-wide priorities


Subscribers
Providers
Customer Service Reps
Subscriber GUI
Vendor GUI
CSR GUI
Calendar
Subscriptions
Identity
Goods & Services
Email Interface
Payment Interface
Email, direct
Email, bulk
Payments

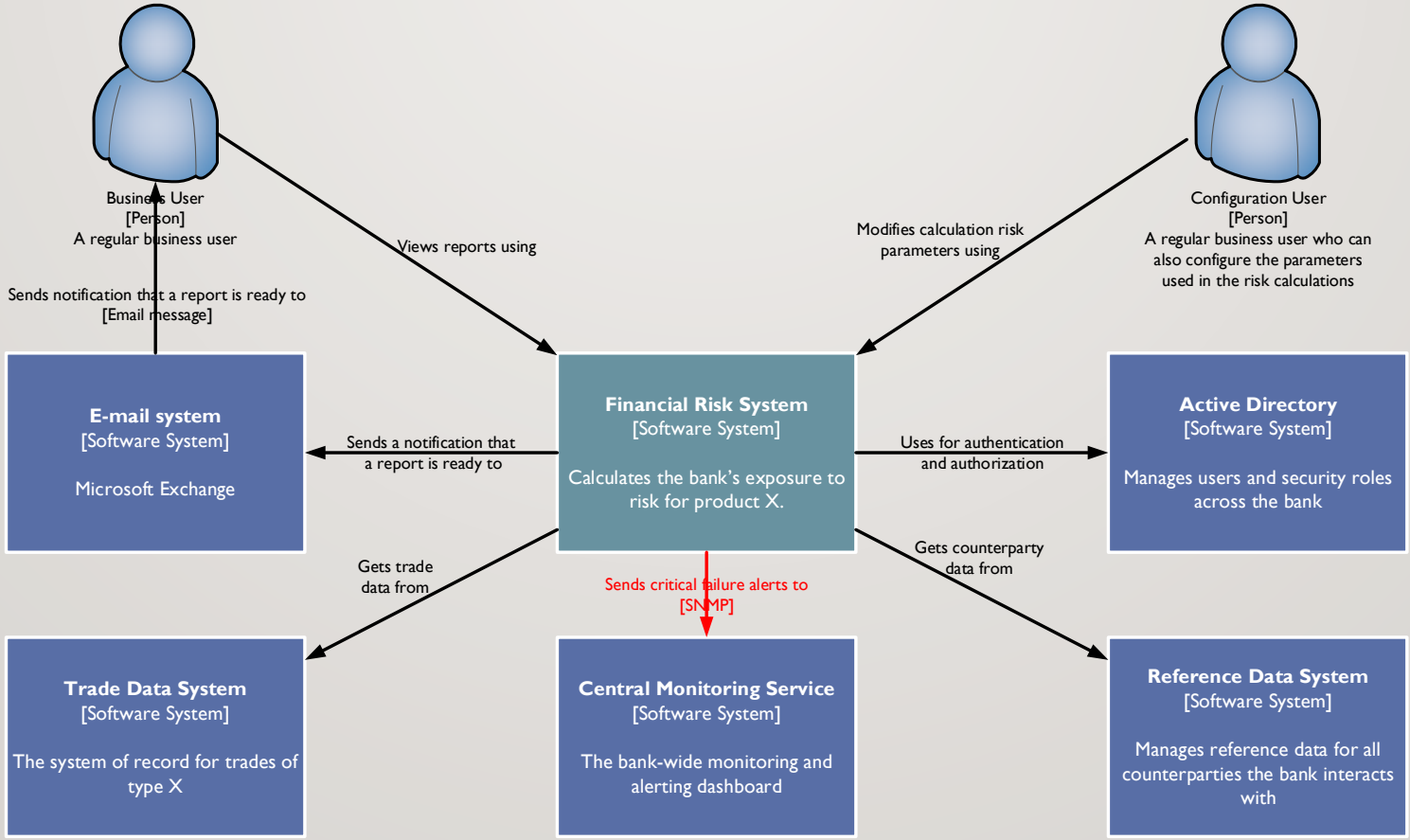
HOMEWORK

- Find a better decomposition for our subscription system.
- Take advantage of components & modules.
  - Think about libraries that would be needed in more than one place.
  - Think about components that would be useful *on their own*
- Component ≠ microservice
- This decomposition should be independent of deployment decisions.
- **Hint**: If you can solve the "CSR problem" you're way ahead.
- **"Extra Credit"**: Compute the Modularization Quality (MQ) for different structures.

VIEWS YOU CAN USE

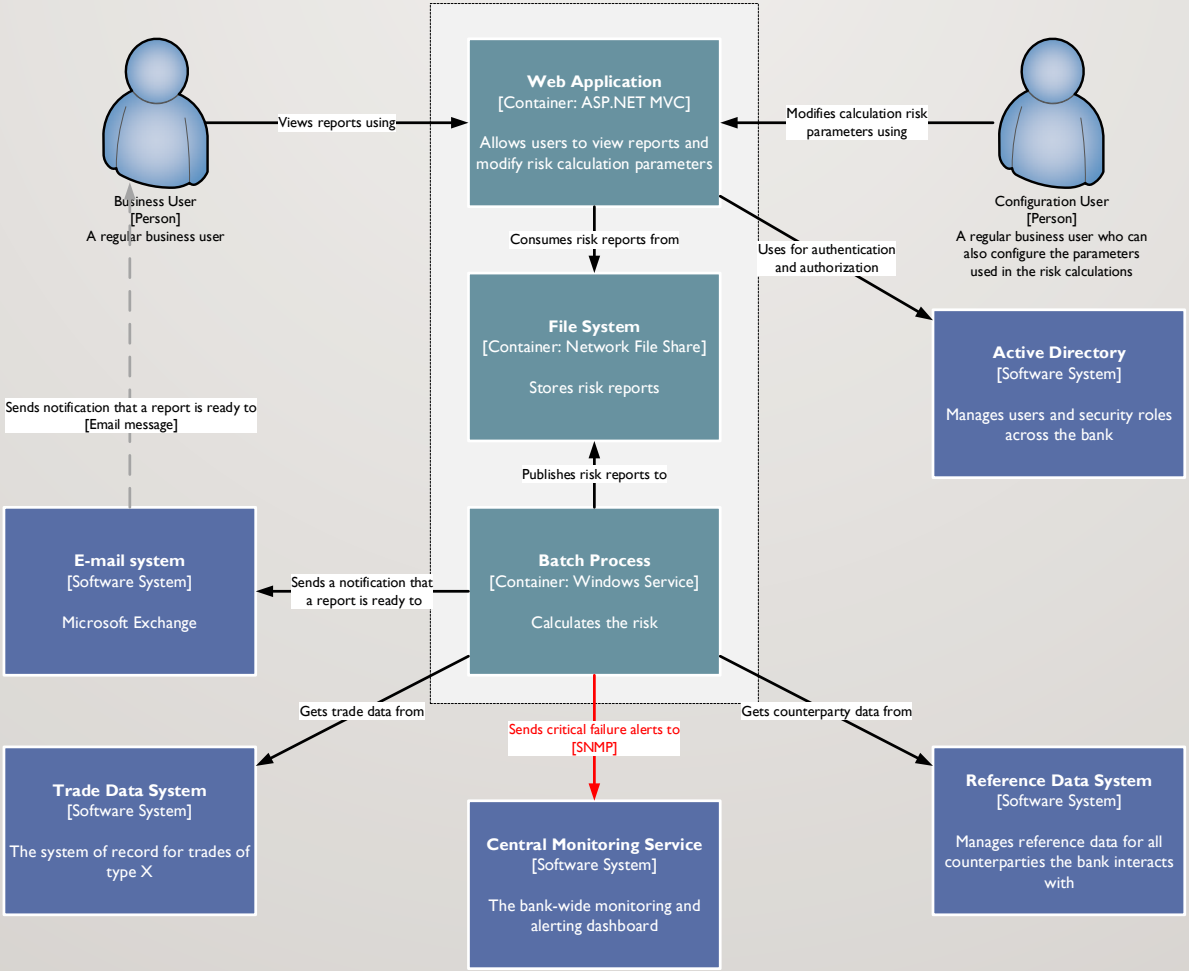
C4 – Context, Containers, Components, Classes

CONTEXT DIAGRAM



http://c4model.com/

CONTAINER DIAGRAM

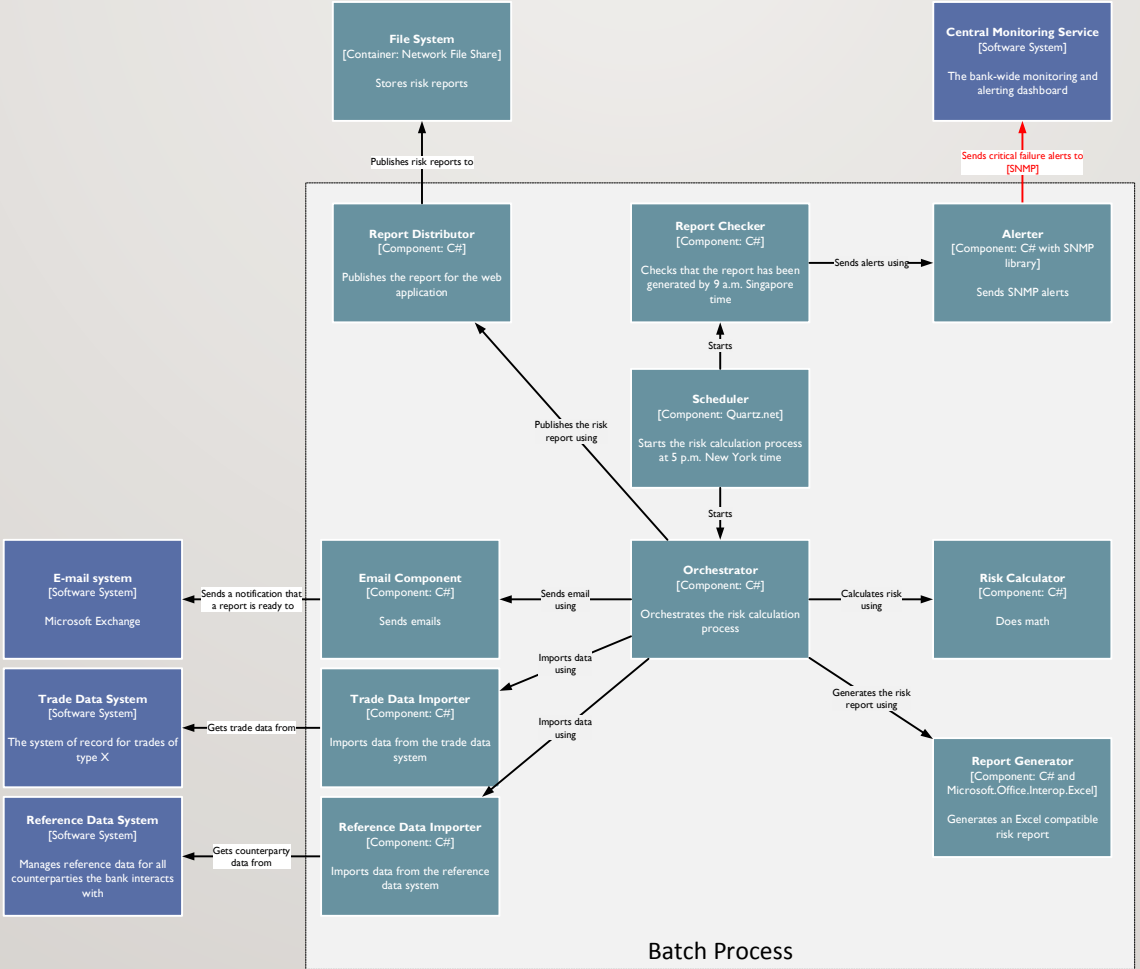


http://c4model.com/

ACTIVITY: MEMORY CHECK

- What was the difference between a “Component” and a “Module”?

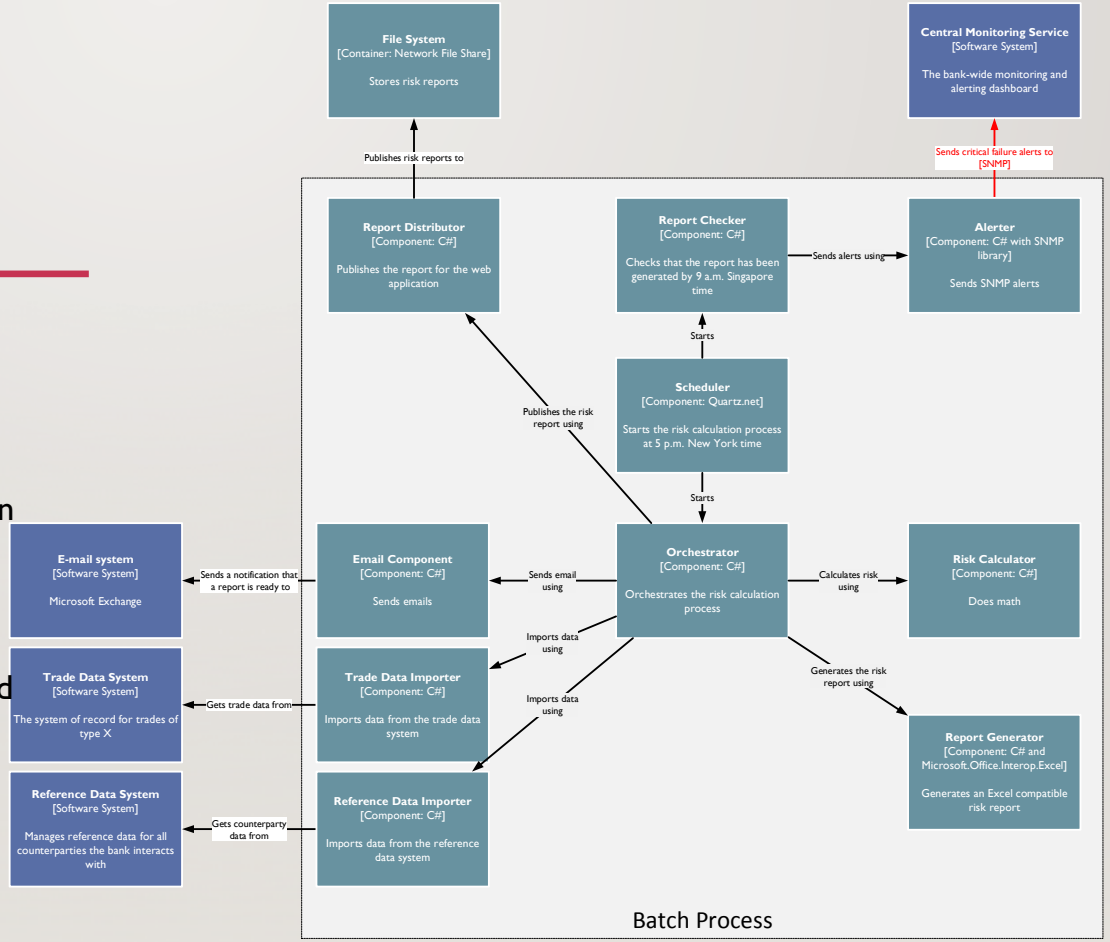
COMPONENT DIAGRAM



http://c4model.com/

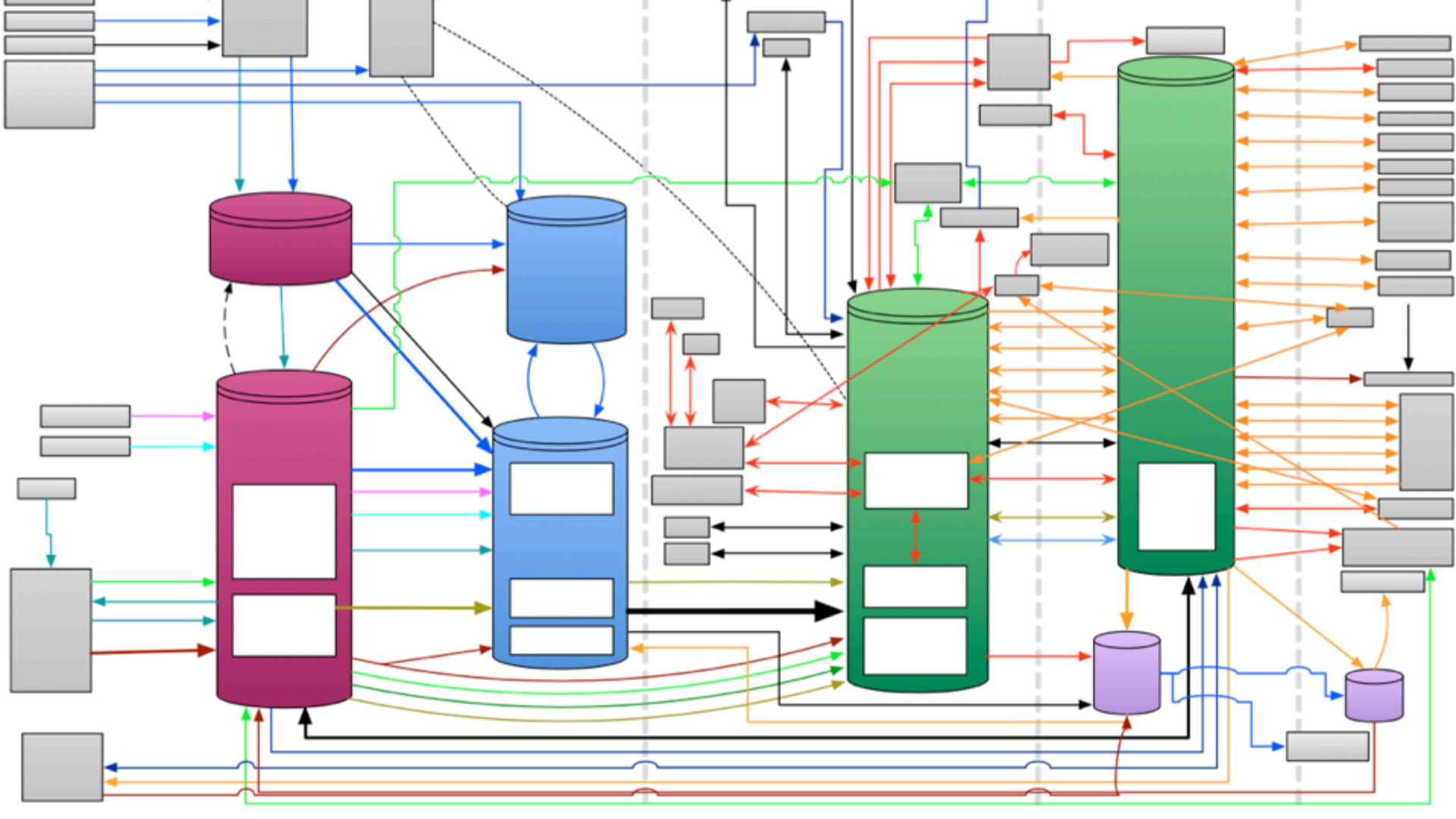
COMPONENT DIAGRAM

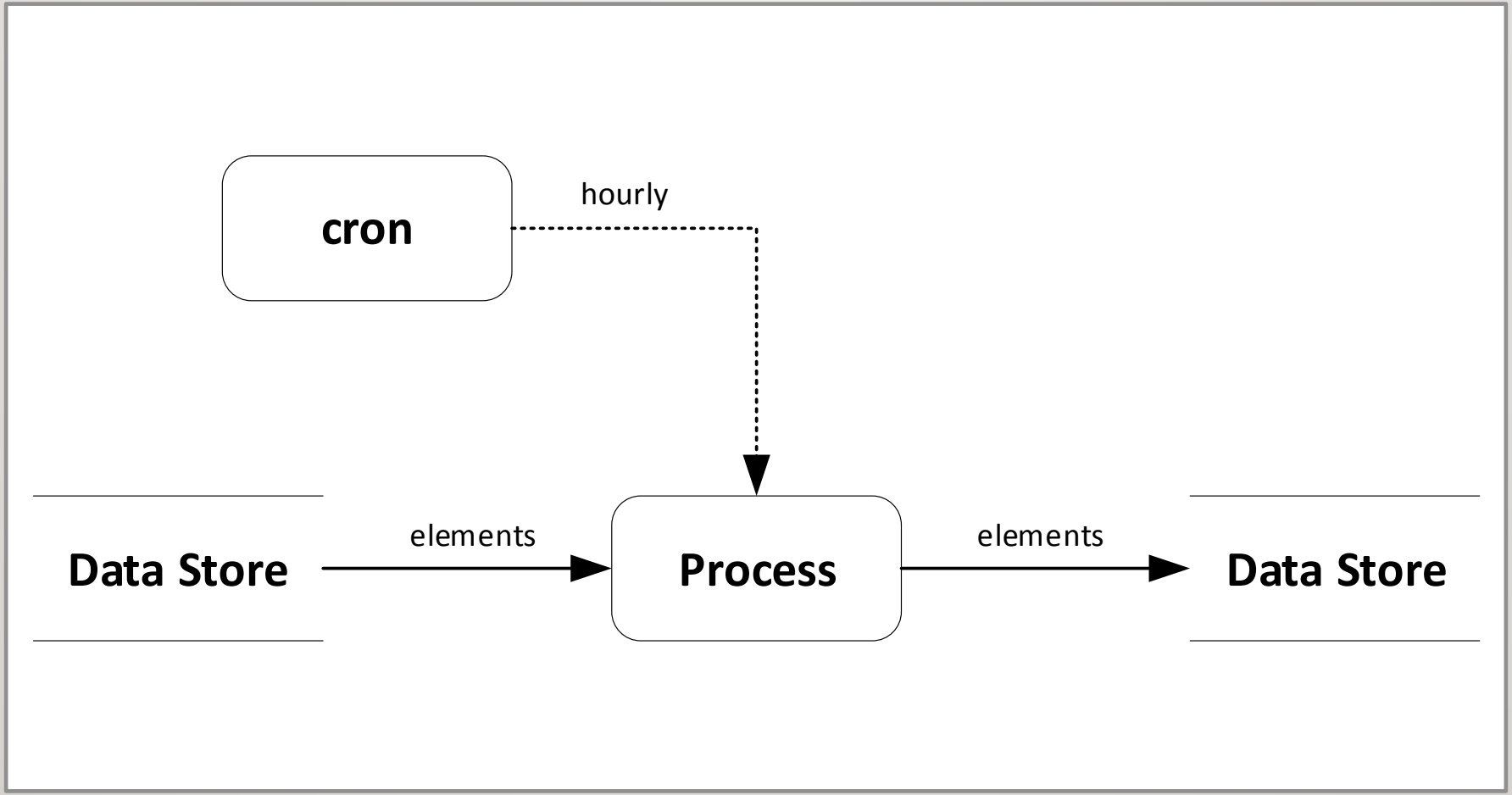
- Remember the principle of decision hiding.
- How do you think this decomposition does at that?
- What decisions appear to be exposed across boundaries here?
- Answers in chat, please.



VIEWS YOU CAN USE

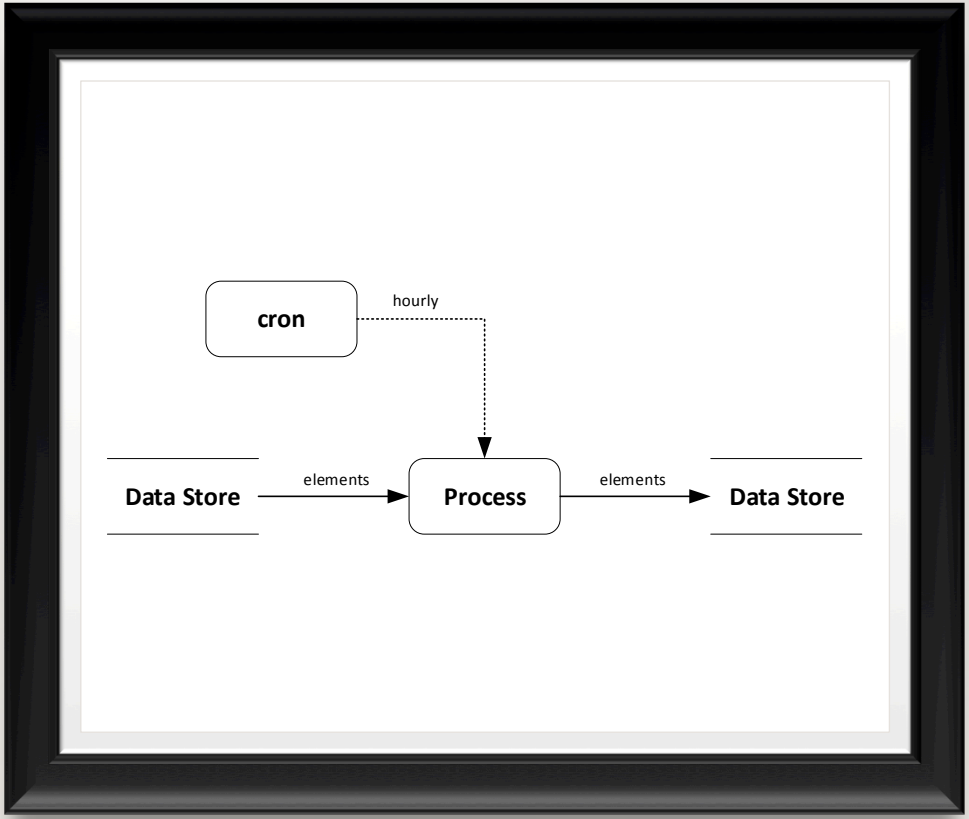
Data Flow Diagram




cron
hourly
Data Store
elements
Process
elements
Data Store

RULES FOR DFDs

1. Data never flows directly from one store to another.
2. Processes receive, transform, and transmit data.
3. Processes are triggered by a control flow.
4. Arrows show the direction data goes, not the kind of call.

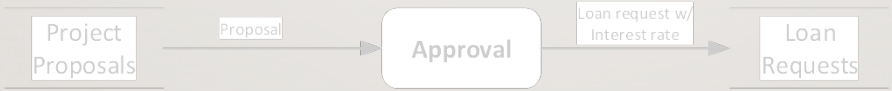


WHAT'S WRONG WITH EACH OF THESE?

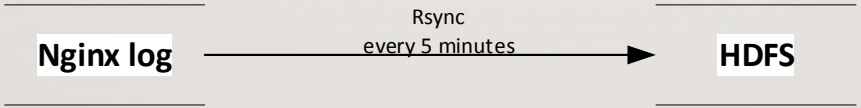
A

WHAT'S WRONG WITH EACH OF THESE?

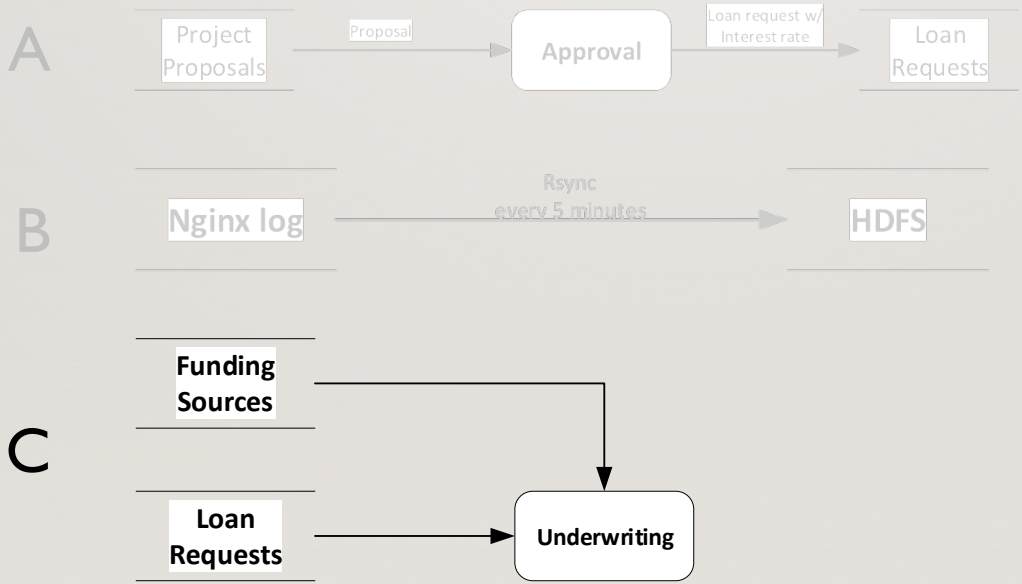
A



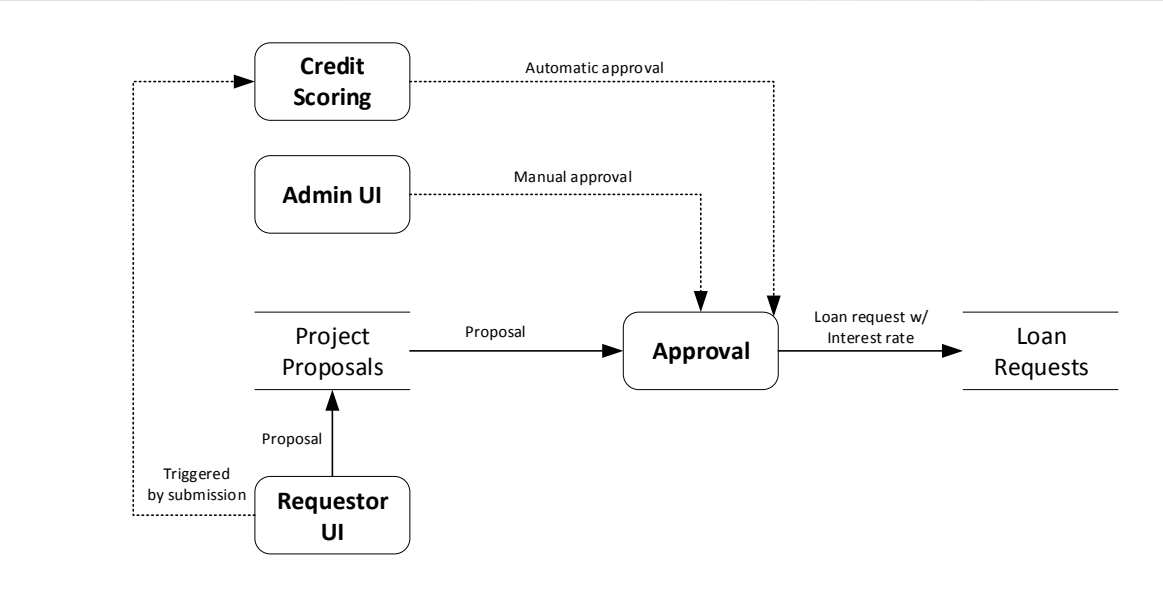
B



WHAT'S WRONG WITH EACH OF THESE?



EXAMPLE FROM A PROJECT



NEXT UP: INCREMENTAL ARCHITECTURE

INCREMENTAL ARCHITECTURE

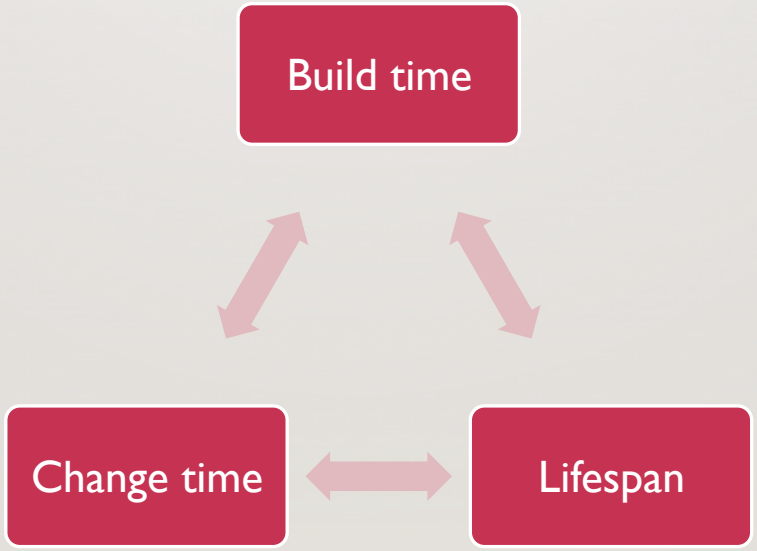
The architecture is only done when you unplug the last server.

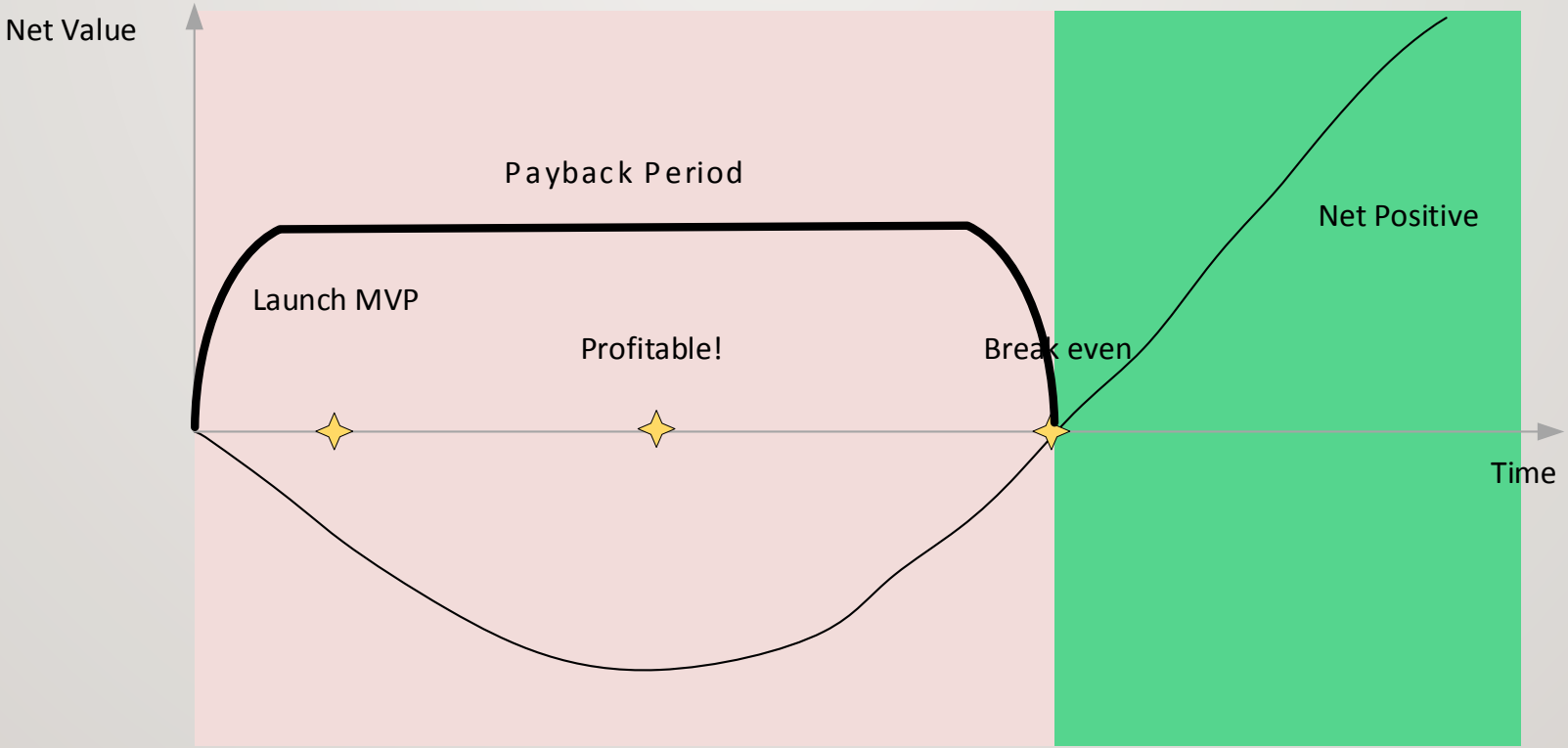
WHEN? HOW MUCH?

TIME IS FUNDAMENTAL

- Build time
- System lifespan
- Change time after launch

CONFLICTS IN TIME




Net Value
Payback Period
Launch MVP
Profitable!
Break even
Net Positive
Time

What limits system lifespan?

THE WORK

- The work of architecture must support timely delivery of a viable system.

EARLY WORK

- Business Goals
- Constraints
- Architecture Quality Ranking
- Architecturally Significant Requirements

ONGOING WORK

- Interface Table
- Architecture Quality Scenarios
- Internal Boundaries (and dispute resolution)
- Information Architecture
- Evaluations of the Architecture
- Verification the Implementations

Wherever possible, leave options open for the future.

HOW MUCH?

- Depends on project risk
- Risky projects get more attention
- Low risk gets less.

RISK FACTORS: TECHNOLOGY

- New language
- New deployment platform (new cloud, new OS, new mobile device)
- "Exotic" search, storage, security, or analytics tech
- Tech stack with highly general applicability. (No out-of-the-box architecture.)
- Long edit-compile-test cycle
- Integration with hardware that isn't abstracted away
- Components that cannot be replicated in development
- High barrier to deployment
- Creating a protocol that must be supported for multiple releases

RISK FACTORS: COMPLEXITY

- Architecture style other than centralized application or streaming data (e.g., mesh networked, peer-to-peer)
- High # integrations with external systems
- High throughput or availability requirements
- Soft or hard real-time requirements
- Active-active deployment to multiple locations
- Multiparty stateful interactions
- Tight resource constraints or high resource needs
- Presence of any of the following: ESB, SOA, SOAP, SAML, OAuth, IBM MQSeries.
- Presence of PCI Level 1 or 2, HIPAA, FDA, J/SOX, other compliance regimes
- Supporting a vertical industry standard

ACTIVITY: RISK SCORE FOR OUR SAMPLE SYSTEM

Which risk factors are present in our sample system?

ACTIVITY: RISK SCORE FOR OUR SAMPLE SYSTEM

Which risk factors are present in our sample system?

Add up the risk score and enter your total in chat.

Hold up…

We haven't picked a deployment platform or tech stack.

ACTIVITY: RISK SCORE FOR OUR SAMPLE SYSTEM

- Which risk factors are present in our sample system?
- Assume AWS deployment with front end app in JS + React, back end in Go.
- Add up the risk score and enter your total in chat.

THINK ABOUT "SET-BASED CONCURRENT ENGINEERING"

"One of the ideas in lean product development is the notion of set-based concurrent engineering: considering a solution as the intersection of a number of feasible parts, rather than iterating on a bunch of individual "point-based" solutions. This lets several groups work at the same time, as they converge on a solution."

—Bill Wake, xp123.com

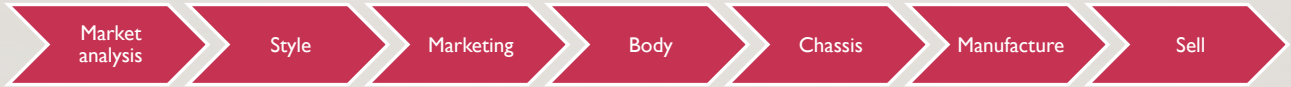
https://xp123.com/articles/set-based-concurrent-engineering/

See also https://xp123.com/articles/resources-on-set-based-design/

THINK ABOUT "SET-BASED CONCURRENT ENGINEERING"

Example from "Toyota's Principles of Set-Based Concurrent Engineering," Sobek, Ward, Liker, Sloan Management Review, 1999

Traditional "point-based" serial engineering



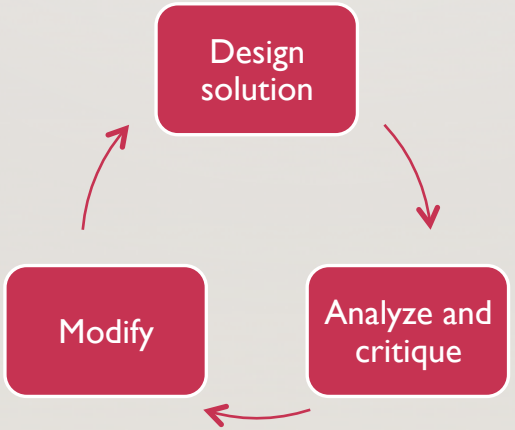
We would call this "waterfall" development

THINK ABOUT "SET-BASED CONCURRENT ENGINEERING"

Example from "Toyota's Principles of Set-Based Concurrent Engineering," Sobek, Ward, Liker, Sloan Management Review, 1999

"Point-based" concurrent engineering

Cross-functional team:
Marketing
Body
Chassis
Manufacturing



"Agile" or "iterative" development

THINK ABOUT "SET-BASED CONCURRENT ENGINEERING"

Example from "Toyota's Principles of Set-Based Concurrent Engineering," Sobek, Ward, Liker, Sloan Management Review, 1999

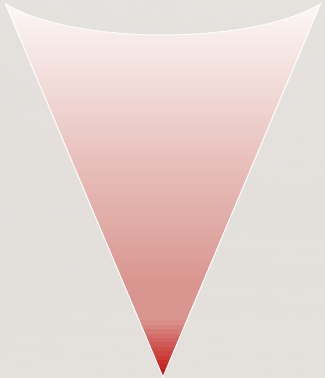
Design Eng.

Several designs meet our requirements

Working within those limits, these are the possible designs

This is nearly final, please review.

Final design



Manufacturing Eng.

Our capabilities are best suited to these designs

We can order tool steel and start planning the line layout

We'd like these changes to optimize mfg cost, then we can order castings

We'll make the final tools and start pilot

EXAMPLE IN OUR SPACE

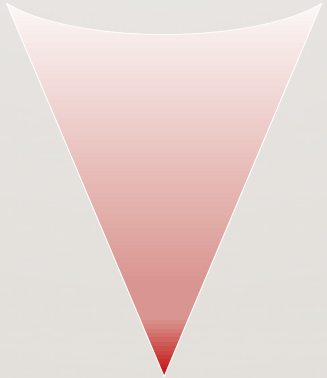
Architecture

We think a message bus is needed to decouple the components

Our latency and uptime requirements look like this. Therefore we're down to A & B

We think A will work. Here's the topology, queues, and volume.

Final design



Operations

We have support agreements with X, Y, and Z that might fit your needs.

We will install a monitoring backend that can support either one.

We agree that will work, but need some tweaks to topology to support our network.

We'll install the infrastructure and plug in monitoring.

A SERIES OF INCREMENTAL DECISIONS

1. “We need messaging.”
2. “We need at least once delivery with allowed downtime of an hour on the subscriber.”
3. Rabbit MQ vs Active MQ vs Hornet vs IBM MQ Series, etc.
4. Topic names, hierarchy, message format, encoding, relays

FEATURES OF SET-BASED CONCURRENT ENGINEERING

- More models, prototypes, and proofs-of-concept.
- Common to have multiple full-scale models at the same time.
- Slow progression from rough design to detailed design.
- Avoid premature commitment.
- Desire to avoid backtracking on commitments.

“THE LAST RESPONSIBLE MOMENT”

Leave decisions open until:

Cost of decision + switching cost > cost of working around lack of decision

Clearing the Question Queue

MATCH DEVELOPMENT METHOD TO ENGINEERING STYLE

Method	Engineering Style
Classic waterfall	Point-based serial
Iterative	Point-based concurrent
XP	Point-based concurrent
Scrum	Point-based concurrent
Agile	Point-based concurrent
RUP	Set-based concurrent
Kanban	N/A – Kanban is a workload management tool, not a development method!

MARKETABLE FEATURES & ARCHITECTURAL ELEMENTS

We don’t need to “do” all the architecture up front.

MINIMUM MARKETABLE FEATURE (MMF)

- Distinct and deliverable feature of the system.
- Observable to the user.
- Provides significant value to the customer.
- "Self-contained," can be delivered without other features.
- Smallest possible realization of that feature.

From "Software by Numbers", Denne & Cleland-Huang

OUR SYSTEM: MMF OR NOT?

Deliverable	MMF?
Comprehensive automated test suite	No – users don't care about tests
Class library for integrating with 3rd party cards	No – not user visible
Redis-backed job queue	Definitely not. Might be anti-feature if adds delay
Message bus	No
Renew early & save program	Yes!
CSR screen for rebilling an account	Yes!
Loyalty card integration	Maybe – is it visible? Is it minimal?

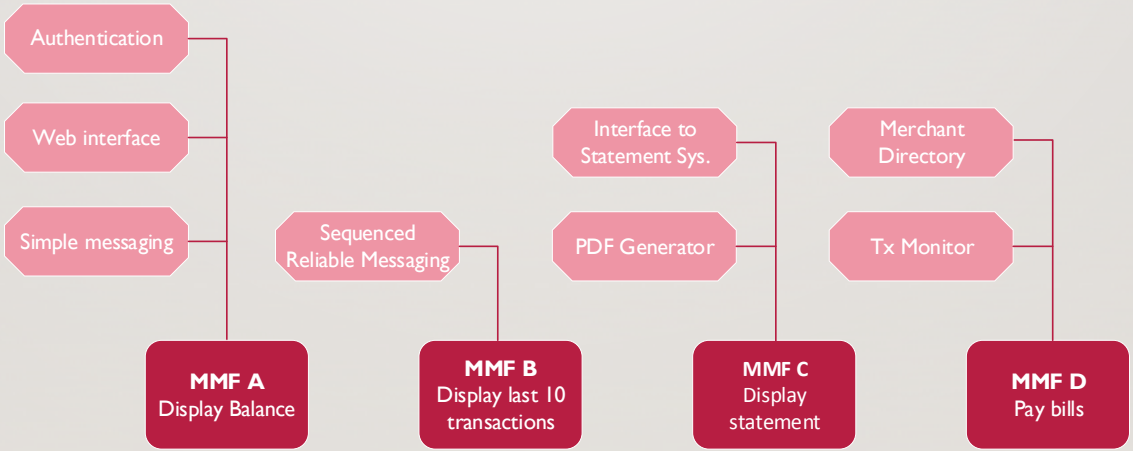
DECOMPOSE SYSTEMS BY MMFs

- Enable iterative development and rollout
- Realize benefits sooner
- Learn from production sooner
- Match investment to benefit at fine grain
- Avoid "Second System" and "Ivory Tower" syndromes

ARCHITECTURE ELEMENT (AE)

- Underlying support
- Involves hardware, software, or network components
- Allows delivery of one or more MMFs
- Purely cost elements. Do not deliver value on their own.

MMFs PULL AEs



BENEFITS OF RECOGNIZING AEs

- Makes architecture visible to stakeholders

BENEFITS OF RECOGNIZING AEs

- Makes architecture visible to stakeholders
- Clearly shows dependencies

BENEFITS OF RECOGNIZING AEs

- Makes architecture visible to stakeholders
- Clearly shows dependencies
- Allows incremental architecture

ACTIVITY: AEs NEEDED

- Some MMFs for our sample system:
  - MMF P: Vendor can list a single service
  - MMF T: Customer can autorenew
  - MMF V: We notify customers if their credit card is near expiration

In chat, name some architecture elements we need for each MMF.

Example
Suppose we had MMF A "Delivery occurs with nanosecond precision", you might say
A: atomic clock, ballistic missile guidance system, GPS-enabled terriers

SEQUENCING FEATURES

WHAT ORDER SHOULD WE DO THESE JOBS IN?

Job	Estimate
A	10 weeks
B	5 weeks
C	2 weeks

WHAT ORDER SHOULD WE DO THESE JOBS IN?

Sequence	All jobs finished in
A, B, C	17 weeks
A, C, B	17 weeks
B, C, A	17 weeks
B, A, C	17 weeks
C, A, B	17 weeks
C, B, A	17 weeks

WHAT ORDER SHOULD WE DO THESE JOBS IN?

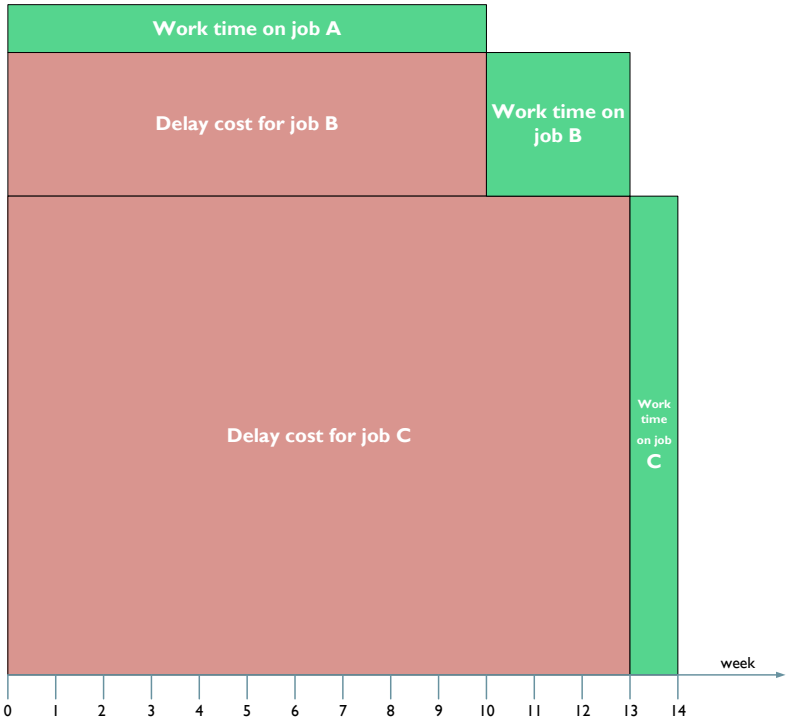
Sequence	All jobs finished in	Average Wait Time
A, B, C	17 weeks	14 weeks
A, C, B	17 weeks	13 weeks
B, C, A	17 weeks	9.6 weeks
B, A, C	17 weeks	12.3 weeks
C, A, B	17 weeks	9.3 weeks
C, B, A	17 weeks	8.6 weeks

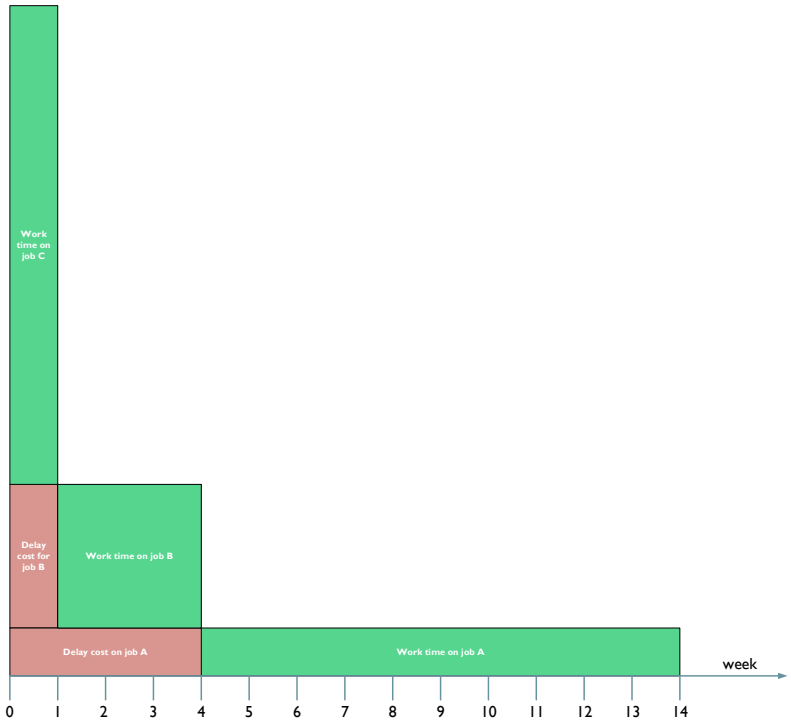
WHAT ORDER SHOULD WE DO THESE JOBS IN?

Sequence	All jobs finished in	Average Wait Time
A, B, C	17 weeks	14 weeks
A, C, B	17 weeks	13 weeks
B, C, A	17 weeks	9.6 weeks
B, A, C	17 weeks	12.3 weeks
C, A, B	17 weeks	9.3 weeks
C, B, A	**17 weeks**	**8.6 weeks**

WHAT ORDER SHOULD WE DO THESE JOBS IN?

Job	Estimate	Cost of Delay
A	10 weeks	1
B	3 weeks	3
C	2 weeks	10


Work time on job A
Delay cost for job B
Work time on job B
Delay cost for job C
Work time on job C
week
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14


Work time on job C
Delay cost for job B
Work time on job B
Delay cost on job A
Work time on job A
week
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

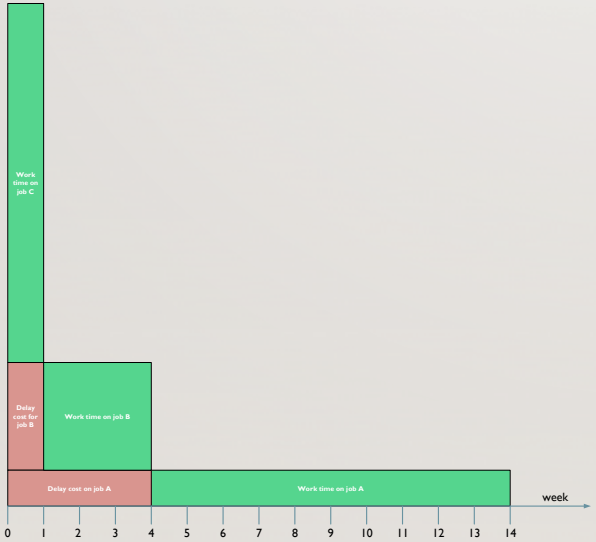
WEIGHTED SHORTEST JOB FIRST

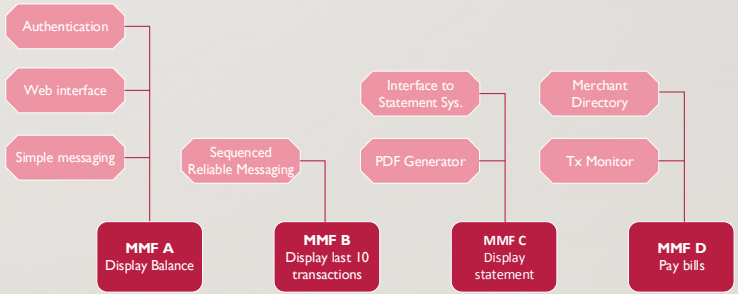
Weight = Cost / Duration

Job	Estimate	Cost of Delay	Weight
A	10 weeks	1	0.1
B	3 weeks	3	1
C	2 weeks	10	10

Pick the highest weight next

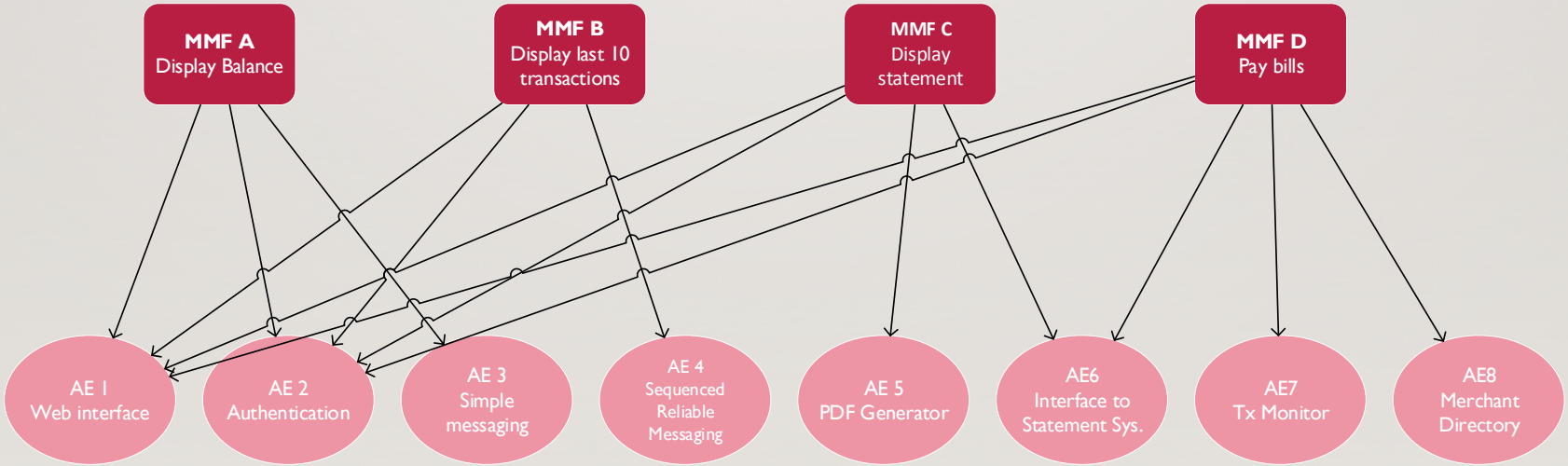
PUTTING IT TOGETHER: WSJF + MMF/AE





PUTTING IT TOGETHER: WSJF + MMF/AE

- Both duration and cost of delay depend on the sequence



PUTTING IT TOGETHER: WSJF + MMF/AE

- Both duration and cost of delay depend on the sequence

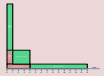
1st	2nd	3rd	4th
MMF A AE 1, 2, 3	MMF B AE 4	MMF C AE 5, 6	MMF D AE 7, 8

PUTTING IT TOGETHER: WSJF + MMF/AE

- Both duration and cost of delay depend on the sequence

1st	2nd	3rd	4th
MMF A AE 1, 2, 3	MMF B AE 4	MMF C AE 5, 6	MMF D AE 7, 8
MMF B AE 1, 2, 4	MMF A AE 3	MMF C AE 5, 6	MMF D AE 7, 8
MMF C AE 1, 2, 5, 6	MMF D AE 7, 8	MMF B AE 4	MMF A AE 3

APPLY WSJF TO EACH POSSIBLE SEQUENCE

1st	2nd	3rd	4th	WSJF
MMF A AE 1, 2, 3	MMF B AE 4	MMF C AE 5, 6	MMF D AE 7, 8	
MMF B AE 1, 2, 4	MMF A AE 3	MMF C AE 5, 6	MMF D AE 7, 8	
MMF C AE 1, 2, 5, 6	MMF D AE 7, 8	MMF B AE 4	MMF A AE 3	

Clearing the Question Queue

YAGNI OR NOT?

COMMUNICATION

THE ARCHITECT ELEVATOR

"This is what I call the architect elevator: architects ride the elevator up and down to move between the board room and the engine room of a large enterprise."

— Gregor Hohpe

http://www.enterpriseintegrationpatterns.com/ramblings/79_elevator.html

ARCHITECT AS TRANSLATOR

- Technology team
- Program/project management
- Stakeholders
- And posterity

ARCHITECT AS SHAMAN

Developer: "This is stupid! Why is this here?"

Architect: "Ah, come friend. Sit by the fire and let me tell you the story of our system."

— Dan North

ON DOCUMENTS

ON DOCUMENTS

The document is not the work.

It is a record of the work.

“Done with the document” doesn’t mean “done with the work”

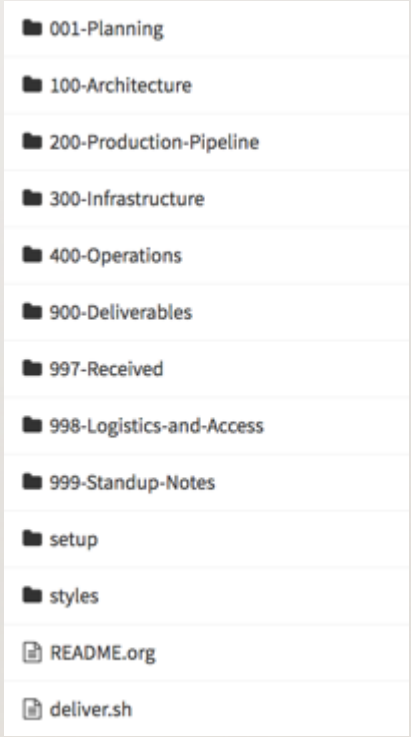
DOCUMENTS CAN

- Capture information
- Present your thinking
- Identify areas to explore

COMMUNICATION TOOLS

ARCHITECTS FOLDER

- Reminders to myself about areas to consider
- Useful scaffolding, but adapted for every project and system
- Mostly text



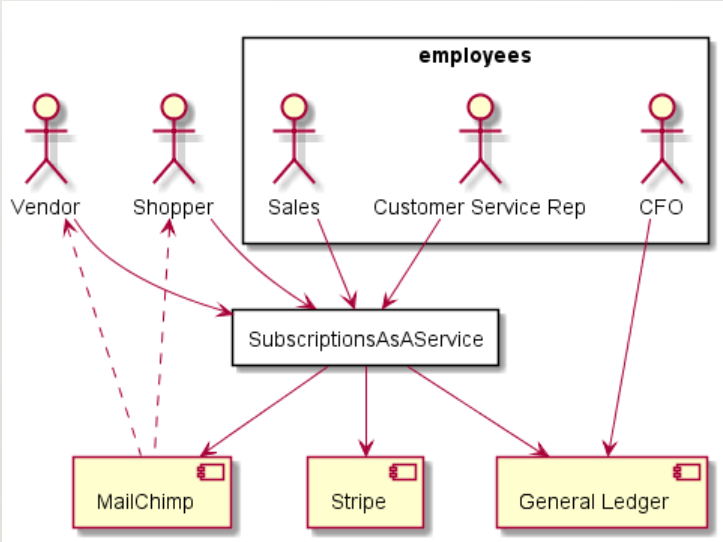
https://gitlab.com/mtnygard/arch-folder-template

DIAGRAMS FROM TEXT

- Graphics diff badly
- Specialized diagramming (Visio, Omnigraffle, SparxSystems) exclude people who don't have licenses or don't know how to use it.
- Try plain-text sources and text-to-image tools

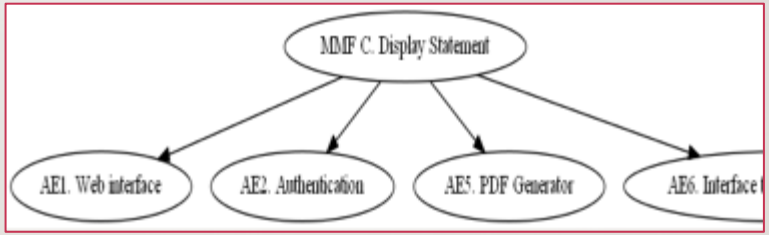
PLANTUML

```
"Shopper" as shopper
"Vendor" as vendor

rectangle "employees" {
  "Customer Service Rep" as csr
  "Sales" as sales
  "CFO" as cfo
}

[MailChimp] as mailer
[Stripe] as payments
[General Ledger] as gl

rectangle "SubscriptionsAsAService" as
saas {
```



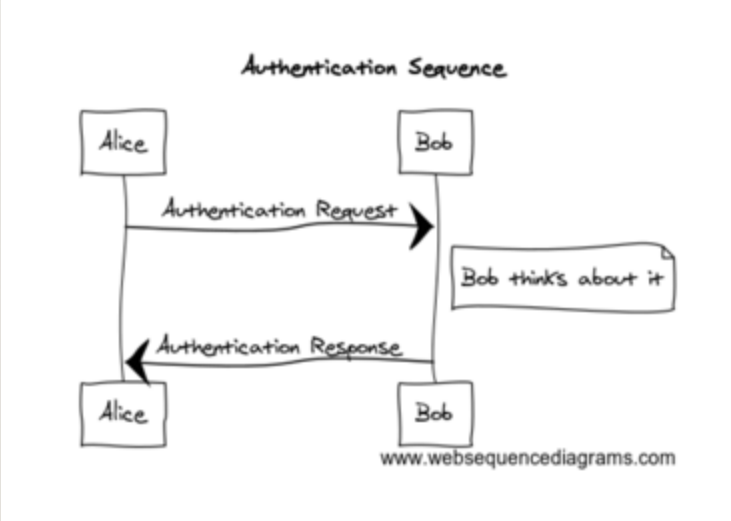
GRAPHVIZ

```
AE7 [label="AE7. Transaction Monitor"];
AE8 [label="AE8. Merchant Directory"];
MMF A [label="MMF A. Display Balance"];
MMF B [label="MMF B. Display Last 10
Transactions"];
MMF C [label="MMF C. Display Statement"];
MMF D [label="MMF D. Pay Bills"];
MMF C
MMF C  ->  AE1
MMF C  ->  AE2
MMF C  ->  AE5
MMF C  ->  AE6
MMF B
MMF B  ->  AE4
MMF A
MMF A  ->  AE3
```



WEB SEQUENCE DIAGRAMS

```
title Authentication Sequence

Alice->Bob: Authentication Request
note right of Bob: Bob thinks about it
Bob->Alice: Authentication Response
```



https://www.websequencediagrams.com/

COMMON THEME

- Spend little to no time pushing polygons
- Share thoughts
- Communicate to people
- Favor communication over completeness

SENDING MESSAGES TO THE FUTURE

THREE STAGES OF MEMORY LOSS

- “When can we remove this horrible hack?”

THREE STAGES OF MEMORY LOSS

- "When can we remove this horrible hack?"
- "Does anyone know why we do this?" "Go ask Alice."

THREE STAGES OF MEMORY LOSS

- "When can we remove this horrible hack?"
- "Does anyone know why we do this?" "Go ask Alice."
- "Nobody knows why we do that, so I'm afraid to remove it."

ARCHITECTURE DECISION RECORDS

- Point in time
- Rationale for a decision
- Immutable record
- But may be superseded later

Blog

NOV 15 2011

Documenting Architecture Decisions

Michael Nygard

Tags: agility and architecture

Context

Architecture for agile projects has to be described and defined differently. Not all decisions will be made at once, nor will all of them be done when the project begins.

Agile methods are not opposed to documentation, only to valueless documentation. Documents that assist the team itself can have value, but only if they are kept up to date. Large documents are never kept up to date. Small, modular documents have at least a chance at being updated.

KEYS TO A SUCCESSFUL ADR

- Think hard about the context
  - Include social, team, and skills factors
  - Include business situation, especially w.r.t. timelines
- Think hard about the consequences
  - Not "pros" and "cons"
  - Just this we need to do differently

Architecture Decision Record: Simplification of Chimera Model

Note: this ADR supersedes some aspects of ADR-15 and ADR-16.

Context

The Chimera data model (as described in ADR-15 and ADR-16) includes the concepts of *entity types* in the domain data model: a defined entity type may have supertypes, and inherits all the attributes of a given supertype

This is quite expressive, and is a good fit for certain types of data stores (such as Datomic, graph databases, and some object stores.) It makes it possible to compose types, and re-use attributes effectively.

However, it leads to a number of conceptual problems, as well as implementation complexities. These issues include but are not limited to:

- There is a desire for some types to be "abstract", in that they exist purely to be extended and are not intented to be reified in the target database (e.g, as a table.) In the current model it is ambiguous whether this is the case or not.
- A singe `extend-type` migration operation may need to create multiple columns in multiple tables, which some databases do not support transactionally.
- When doing a lookup by attribute that exists in multiple types, it is ambiguous which type is intended.
- In a SQL database, how to best model an extended type becomes ambiguous: copying the column leads to "denormalization", which might not be desired. On the other hand, creating a separate table for the shared columns leads to more complex queries with more joins.

All of these issues can be resolved or worked around. But they add a variable amount of complexity cost to every Chimera adapter, and create a domain with large amounts of ambigous behavior that must be resolved (and which might not be discovered until writing a particular adapter.)

All of these issues can be resolved or worked around. But they add a variable amount of complexity cost to every Chimera adapter, and create a domain with large amounts of ambigous behavior that must be resolved (and which might not be discovered until writing a particular adapter.)

Decision

The concept of type extension and attribute inheritance does not provide benefits proportional to the cost.

We will remove all concept of supertypes, subtypes and attribute inheritance from Chimera's data model.

Chimera's data model will remain "flat". In order to achieve attribute reuse for data stores for which that is idiomatic (such as Datomic), multiple Chimera attributes can be mapped to a single DB-level attribute in the adapter mapping metadata.

Status

PROPOSED

Consequences

- Adapters will be significantly easier to implement.
- An attribute will need to be repeated if it is present on different domain entity types, even if it is semantically similar.
- Users may need to explicitly map multiple Chimera attributes back to the same underlying DB attr/column if they want to maintain an idiomatic data model for their database.

https://github.com/arachne-framework/architecture/blob/master/adr-017-simplify-chimera-model.md

SAMPLE ADRs

https://github.com/arachne-framework/architecture

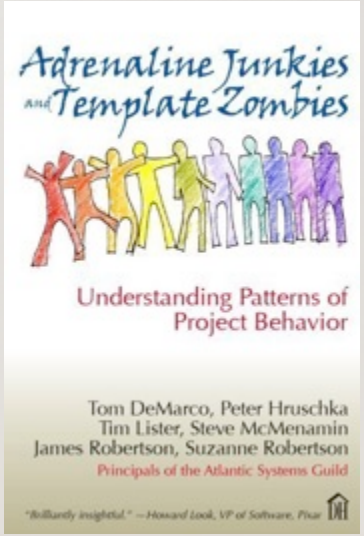
https://github.com/npryce/adr-tools

ACTIVITY: DISCUSSION

- Why is it important that an ADR isn't trying to "sell" a decision?
- Your thoughts in chat, please.

REMINDERS

- We document to:
  1. Communicate ideas
  2. Capture information
  3. Aid our memory
  4. Share our thinking with the future
- Use the right views to serve different viewpoints
- Never be a template zombie.



NEXT UP: RESPONSIBILITY ALLOCATION