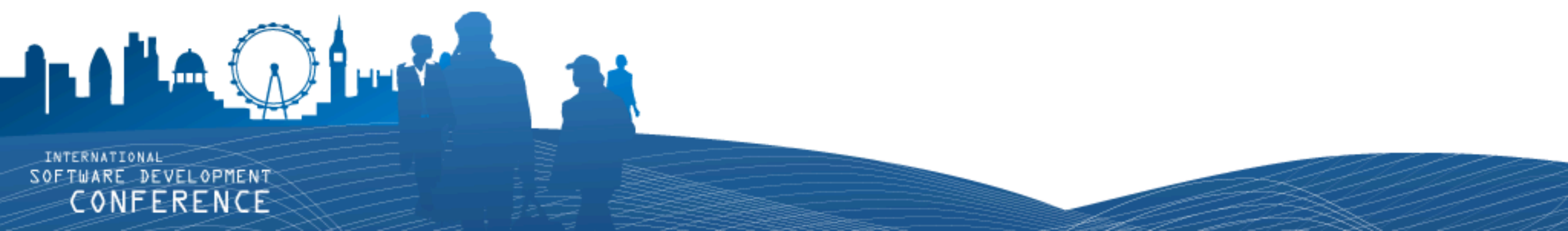
QCon

Please evaluate my talk via the mobile app!



Dev "Programming" Ops for DevOps Success

Damon Edwards

@damonedwards

dev2ops.org

DevOps Cafe

Disclosure: DevOps (to me)

Disclosure: DevOps (to me)

DevOps is <u>**not**</u>

- a specific methodology or prescriptive steps
- only achievable by “one true way”
- a job title or group name

Disclosure: DevOps (to me)

DevOps is <u>**not**</u>

- a specific methodology or prescriptive steps
- only achievable by “one true way”
- a job title or group name

DevOps is

- a way of seeing your problems
- a way of evaluating solutions
- a way of communicating these things
- always evolving

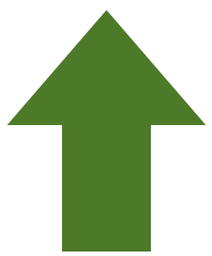
Damon Edwards

DevOps Consulting → Automation Design

#SimplifyOps

Open Source → Tools

What every business wants



Time-to-market

Quality

What every business wants

- ↑ Time-to-market
- ↑ Quality
- ↑ Effectiveness

What every business wants

- ↑ **Time-to-market**
- ↑ **Quality**
- ↑ **Effectiveness** (*Do more, but don't spend any more!*)

What’s stopping them?

Photo credit: Doc Searls on Flickr

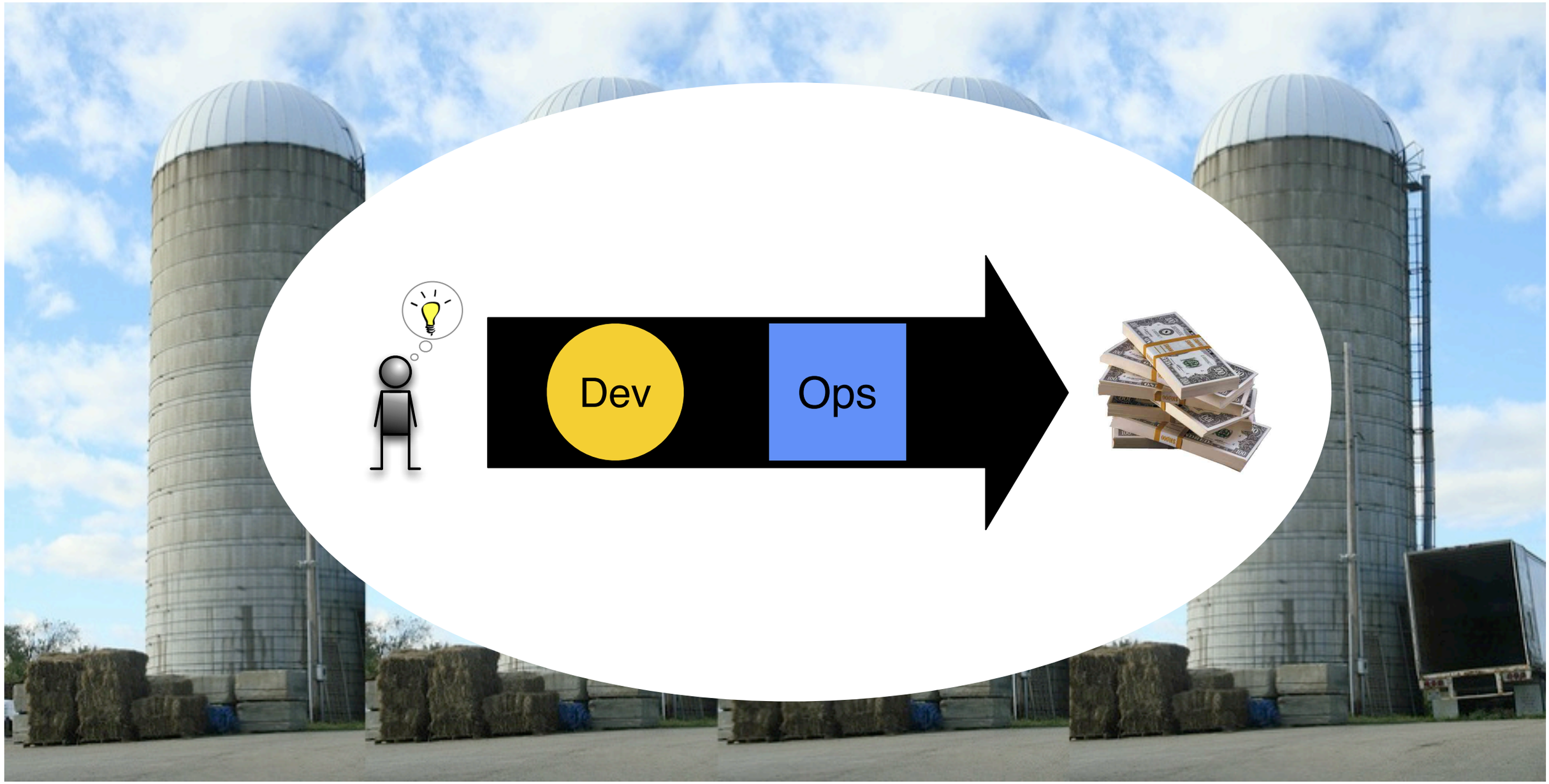


Photo credit: Doc Searls on Flickr

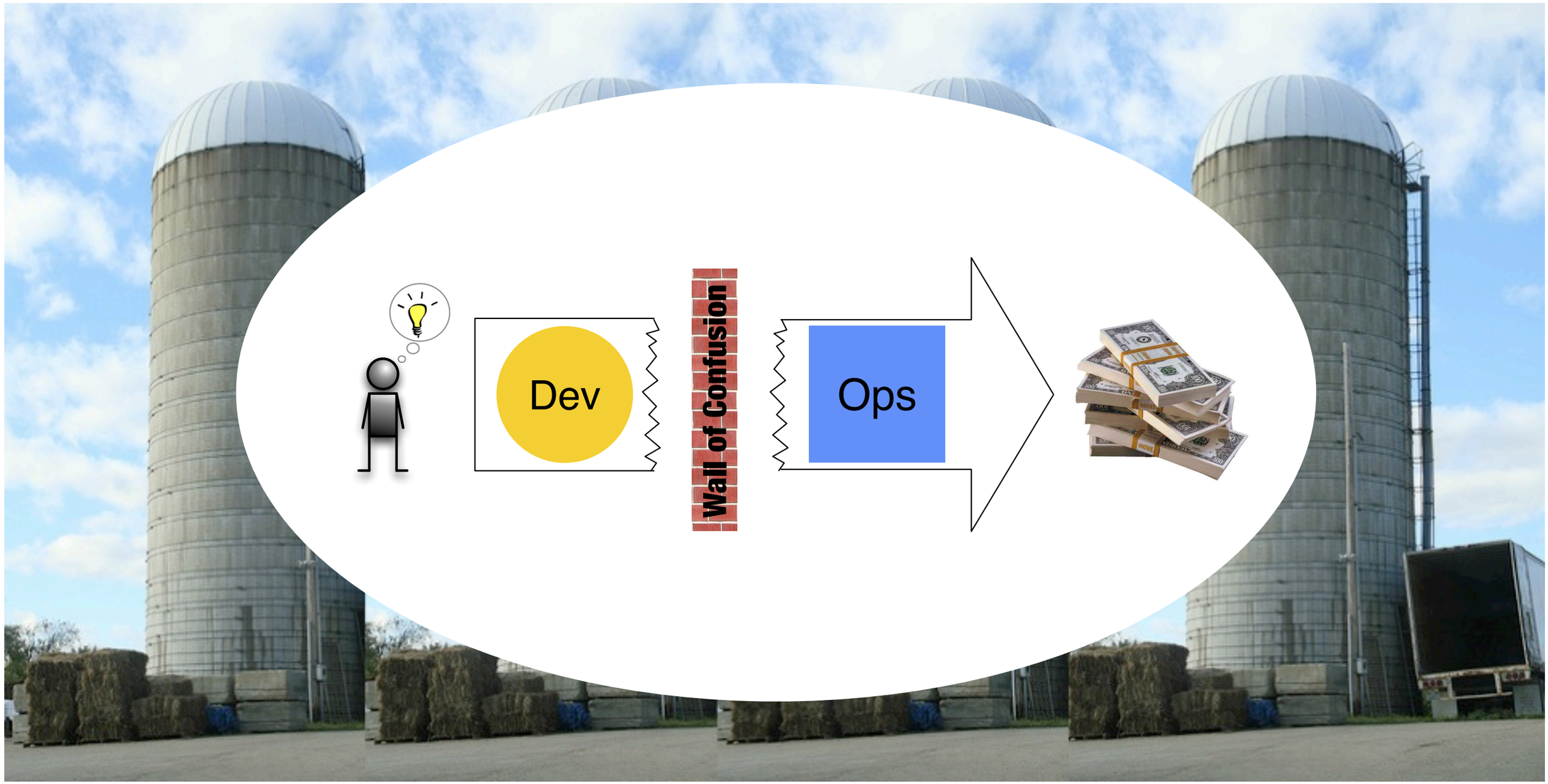


Photo credit: Doc Searls on Flickr

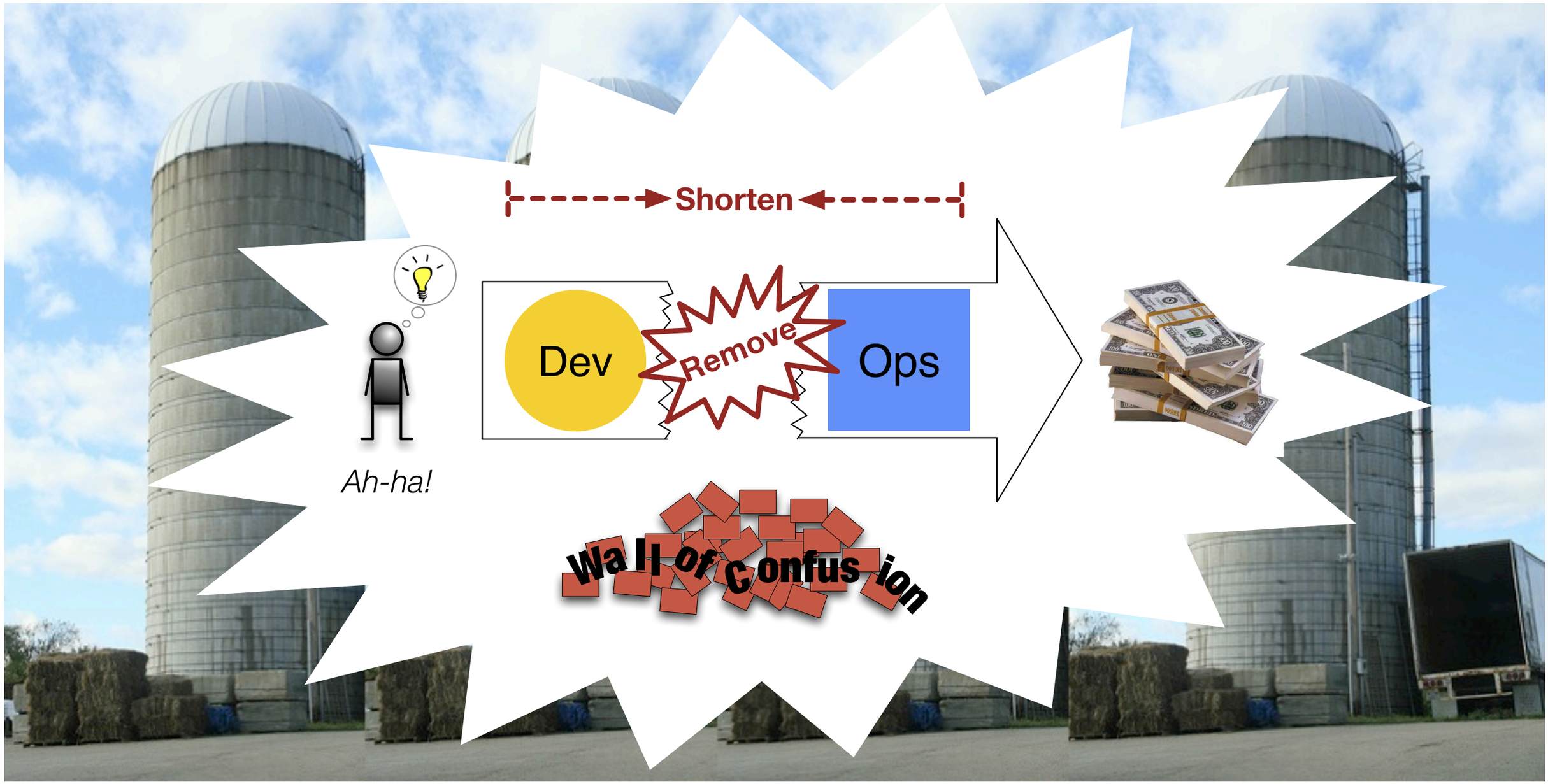


Photo credit: Doc Searls on Flickr

Photo credit: Doc Searls on Flickr

You are here

"...but how can we start a DevOps transformation from our Dev silo?"

Dev initiated DevOps Transformation

Dev initiated DevOps Transformation

1. Take an “Operations First” mindset

Understand the pressure on Ops

What is the product?

What is the product?

- Are customers paying for a running service?

What is the product?

- Are customers paying for a running service?
- Then the running service is the product

What is the product?

- Are customers paying for a running service?
- Then the running service is the product
    - Operations is your “factory floor” and “storefront”

What is the product?

- Are customers paying for a running service?
- Then the running service is the product
    - Operations is your “factory floor” and “storefront”
    - Everything else is a “parts supplier”

What is the product?

- Are customers paying for a running service?
- Then the running service is the product
  - Operations is your “factory floor” and “storefront”
  - Everything else is a “parts supplier”
    - (Yes, that includes Developers)

What is the product?

- Are customers paying for a running service?
- Then the running service is the product
  - Operations is your “factory floor” and “storefront”
  - Everything else is a “parts supplier”
    - (Yes, that includes Developers)
- If the service isn’t running, there is no product or business

“-ilities” are product features

“-ilities” are product features

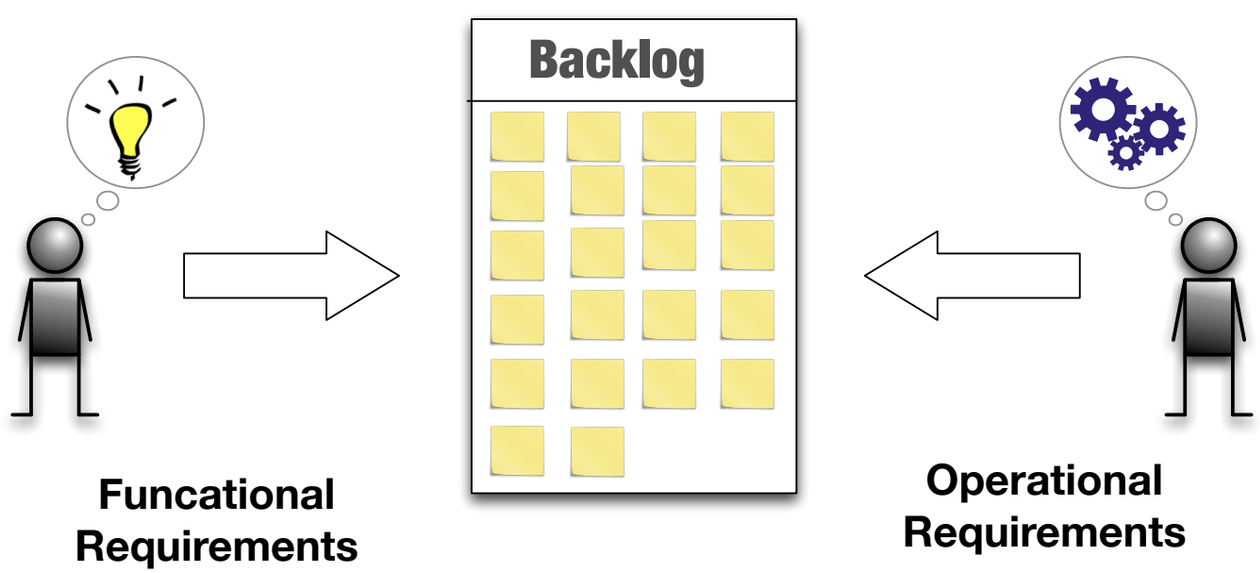
- Stability, scalability, availability, security, etc... are all features

“-ilities” are product features

- Stability, scalability, availability, security, etc... are all features
  - Like all features, you get what you invest

“-ilities” are product features

- Stability, scalability, availability, security, etc... are all features
    - Like all features, you get what you invest
- Operations Requirements should be first class citizens in your backlog



You are developing services (not software)

You are developing services (not software)

- Software is a service when it is running and managed

You are developing services (not software)

- Software is a service when it is running and managed
  - Deployment and configuration is automated

You are developing services (not software)

- Software is a service when it is running and managed
  - Deployment and configuration is automated
  - Standard operating procedures are automated

You are developing services (not software)

- Software is a service when it is running and managed
  - Deployment and configuration is automated
  - Standard operating procedures are automated
  - Tests/Health Checks/Monitoring at service level

You are developing services (not software)

- Software is a service when it is running and managed
  - Deployment and configuration is automated
  - Standard operating procedures are automated
  - Tests/Health Checks/Monitoring at service level
- You are developing a service... so this is part of *your* deliverable

Redefine “done”

Redefine “done”

Old “Done”

Redefine “done”

Old “Done”

- Work is "done" when it moves downstream

Redefine “done”

Old “Done”

- Work is "done" when it moves downstream
- No shared sense of "done"

Redefine “done”

Old “Done”

- Work is "done" when it moves downstream
- No shared sense of "done"
  - Developers commit code to repository

Redefine “done”

Old “Done”

- Work is "done" when it moves downstream
- No shared sense of "done"
  - Developers commit code to repository
  - Time for a party: code complete!

Redefine “done”

Old “Done”

- Work is "done" when it moves downstream
- No shared sense of "done"
  - Developers commit code to repository
  - Time for a party: code complete!
  - The work just begins for Operations to figure out how to run it in production

Redefine “done”

Old “Done”

- Work is "done" when it moves downstream
- No shared sense of "done"
  - Developers commit code to repository
  - Time for a party: code complete!
  - The work just begins for Operations to figure out how to run it in production

New “Done”

Redefine “done”

Old “Done”

- Work is "done" when it moves downstream
- No shared sense of "done"
  - Developers commit code to repository
  - Time for a party: code complete!
  - The work just begins for Operations to figure out how to run it in production

New “Done”

- It's running

Redefine “done”

Old “Done”

- Work is "done" when it moves downstream
- No shared sense of "done"
  - Developers commit code to repository
  - Time for a party: code complete!
  - The work just begins for Operations to figure out how to run it in production

New “Done”

- It's running
- It's managed

Redefine “done”

Old “Done”

- Work is "done" when it moves downstream
- No shared sense of "done"
  - Developers commit code to repository
  - Time for a party: code complete!
  - The work just begins for Operations to figure out how to run it in production

New “Done”

- It's running
- It's managed
- Customer accessible

Redefine “done”

Old “Done”

- Work is "done" when it moves downstream
- No shared sense of "done"
  - Developers commit code to repository
  - Time for a party: code complete!
  - The work just begins for Operations to figure out how to run it in production

New “Done”

- It's running
- It's managed
- Customer accessible
- Behaving properly

Redefine “done”

Old “Done”

- Work is "done" when it moves downstream
- No shared sense of "done"
  - Developers commit code to repository
  - Time for a party: code complete!
  - The work just begins for Operations to figure out how to run it in production

New “Done”

- It's running
- It's managed
- Customer accessible
- Behaving properly

Development of a feature can be “done”. But a service is never “done” until it is turned off!

If you want freedom, take responsibility

If you want freedom, take responsibility

Old Way

If you want freedom, take responsibility

Old Way

- Developers owned feature requirements

If you want freedom, take responsibility

Old Way

- Developers owned feature requirements
- Operations owned performance and uptime

If you want freedom, take responsibility

Old Way

- Developers owned feature requirements
- Operations owned performance and uptime
- QA owned quality

If you want freedom, take responsibility

Old Way

- Developers owned feature requirements
- Operations owned performance and uptime
- QA owned quality
- Security owned security

If you want freedom, take responsibility

Old Way

- Developers owned feature requirements
- Operations owned performance and uptime
- QA owned quality
- Security owned security
- etc...

If you want freedom, take responsibility

Old Way

- Developers owned feature requirements
- Operations owned performance and uptime
- QA owned quality
- Security owned security
- etc...

New Way

If you want freedom, take responsibility

Old Way

- Developers owned feature requirements
- Operations owned performance and uptime
- QA owned quality
- Security owned security
- etc...

New Way

- Developers own their application

If you want freedom, take responsibility

Old Way

- Developers owned feature requirements
- Operations owned performance and uptime
- QA owned quality
- Security owned security
- etc...

New Way

- Developers own their application
- Operations owns infrastructure and common tooling

If you want freedom, take responsibility

Old Way

- Developers owned feature requirements
- Operations owned performance and uptime
- QA owned quality
- Security owned security
- etc...

New Way

- Developers own their application
- Operations owns infrastructure and common tooling
- Everybody owns quality, availability, security for the thing they produce and shared responsibility for things that consume it

If you want freedom, take responsibility

Old Way

- Developers owned feature requirements
- Operations owned performance and uptime
- QA owned quality
- Security owned security
- etc...

New Way

- Developers own their application
- Operations owns infrastructure and common tooling
- Everybody owns quality, availability, security for the thing they produce and shared responsibility for things that consume it

Dev initiated DevOps Transformation

1. Take an “operations first” mindset
2. **Build organizational alignment**

What is organizational alignment?

What is organizational alignment?

- 1000’s of small decisions made daily
  - How do we harness that?

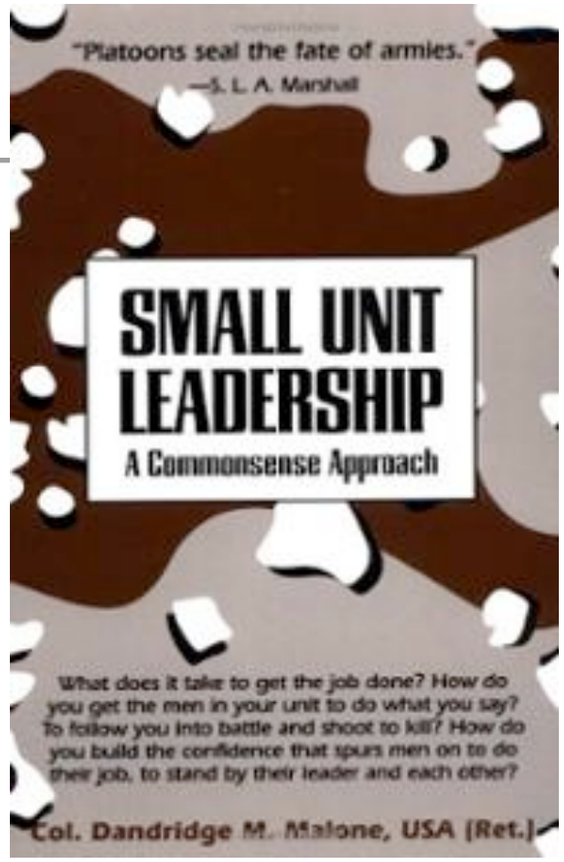
What is organizational alignment?

- 1000’s of small decisions made daily
  - How do we harness that?
- Top down is brittle
  - Conditions, people, tools are always changing... how do we handle that?

What is organizational alignment?

- 1000's of small decisions made daily
  - How do we harness that?
- Top down is brittle
  - Conditions, people, tools are always changing... how do we handle that?
- Alignment is when you know that different individuals would independently see a set of conditions and arrive at the same decision that is correct for the company's goal

What is organizational alignment?



- 1000's of small decisions made daily
  - How do we harness that?
- Top down is brittle
  - Conditions, people, tools are always changing... how do we handle that?
- Alignment is when you know that different individuals would independently see a set of conditions and arrive at the same decision that is correct for the company's goal

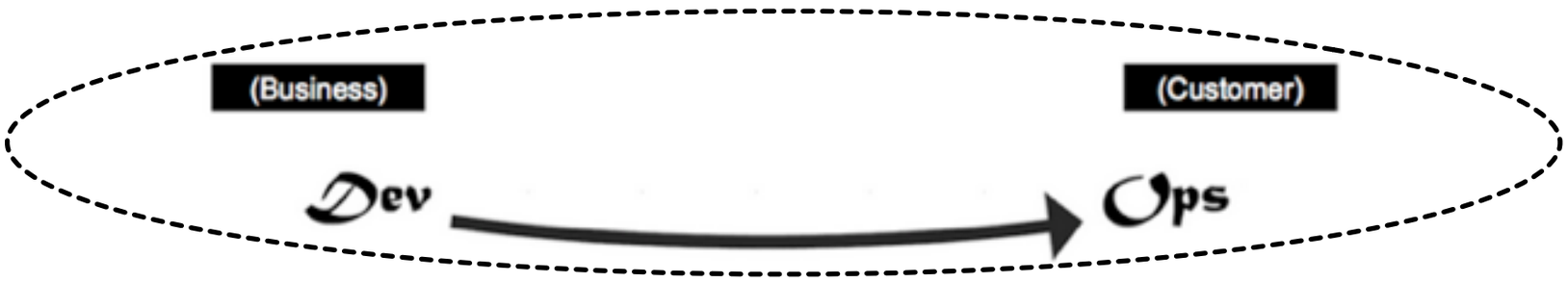
What does an aligned organization “see”?

What does an aligned organization “see”?

expanded from

What does an aligned organization “see”?

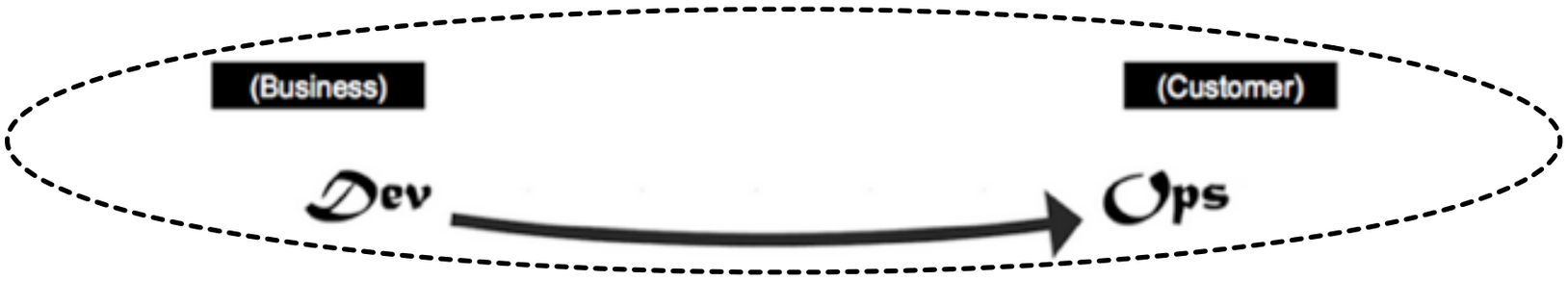
1. See the system



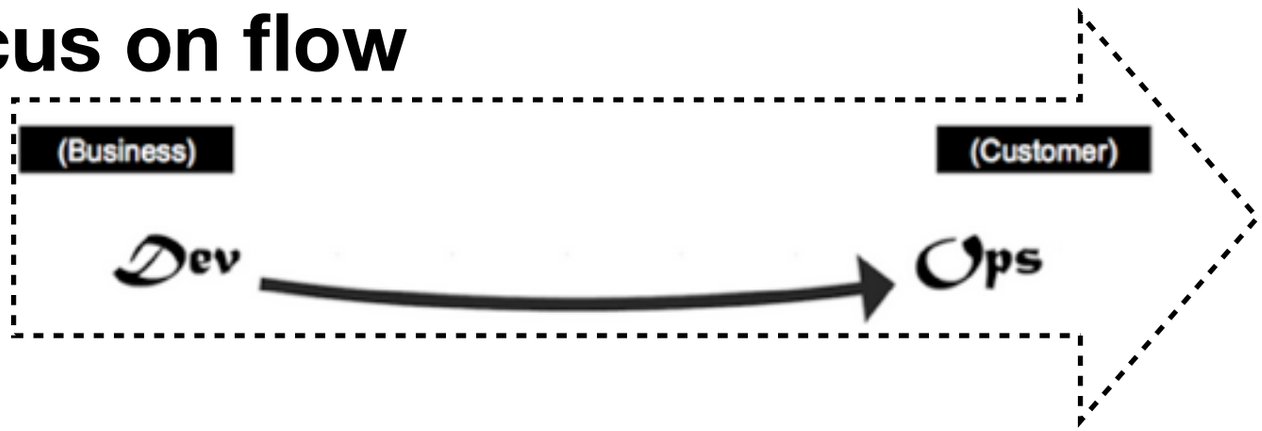
expanded from

What does an aligned organization “see”?

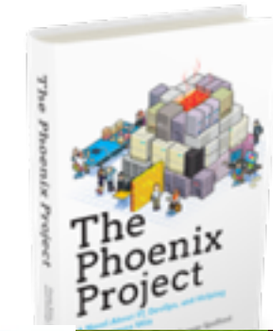
1. See the system

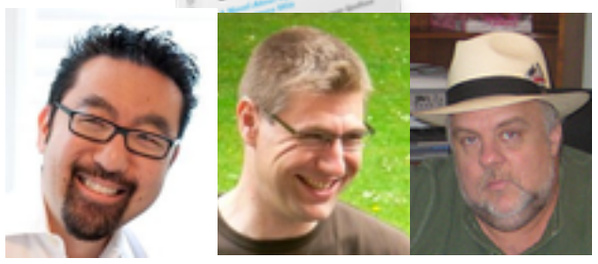


2. Focus on flow



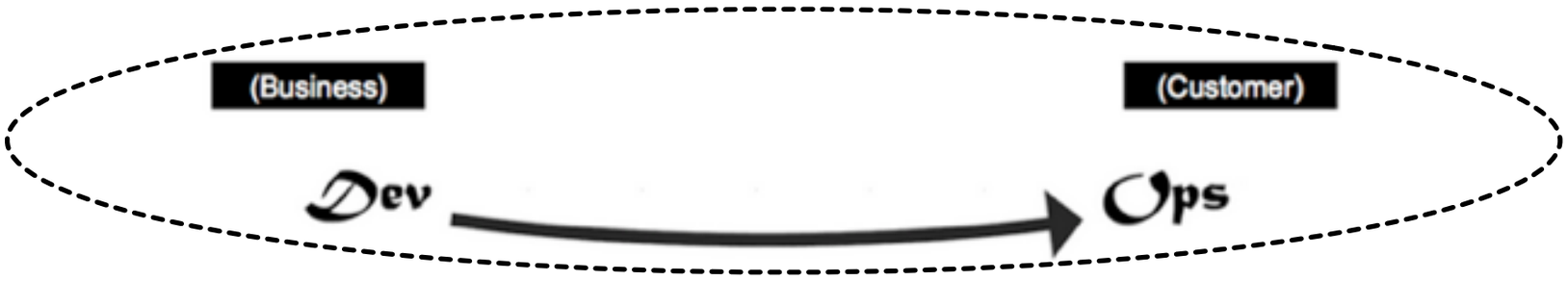
expanded from





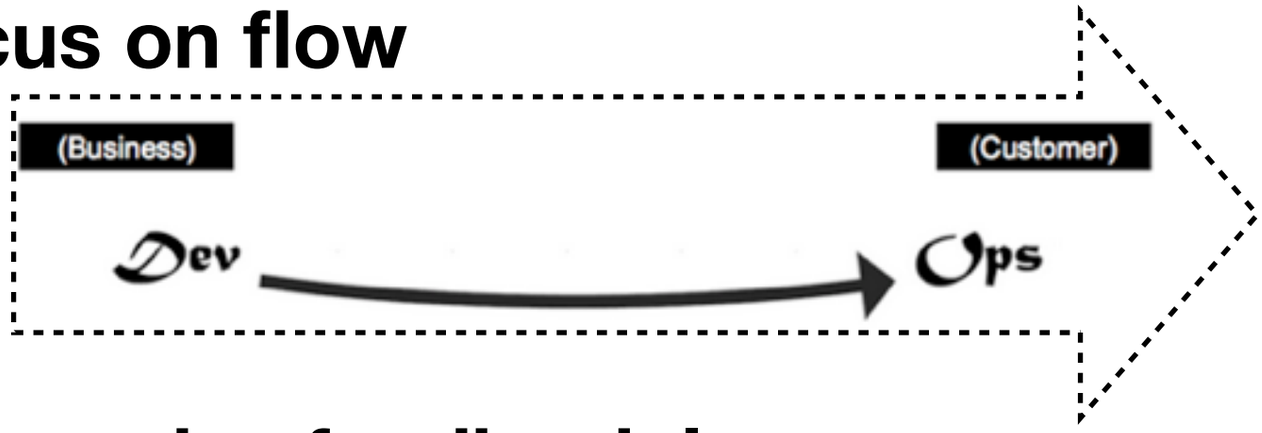
What does an aligned organization “see”?

1. See the system



expanded from

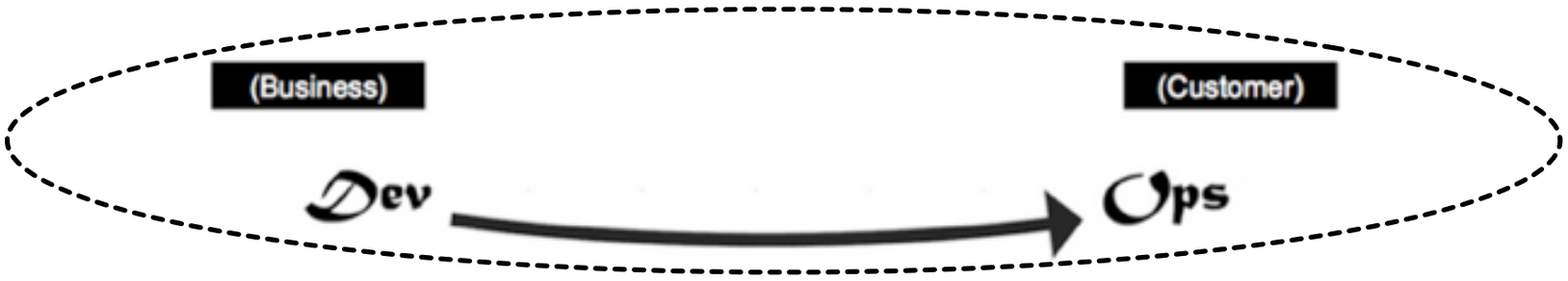
2. Focus on flow



3. Recognize feedback loops

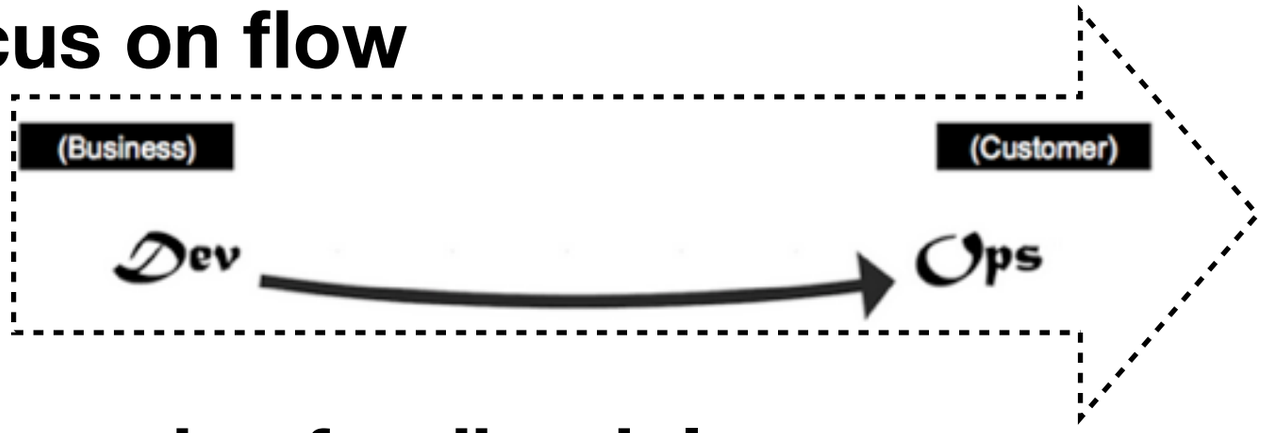
What does an aligned organization “see”?

1. See the system

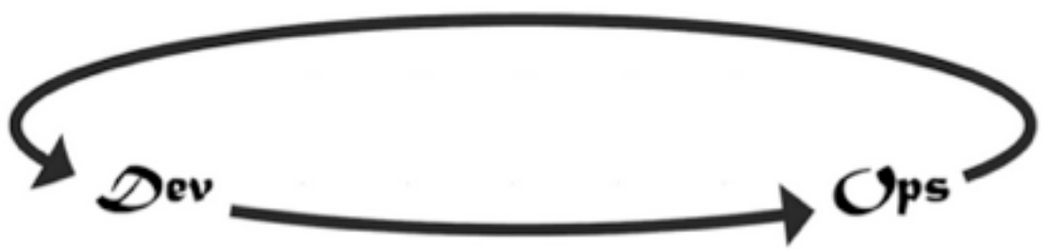


expanded from

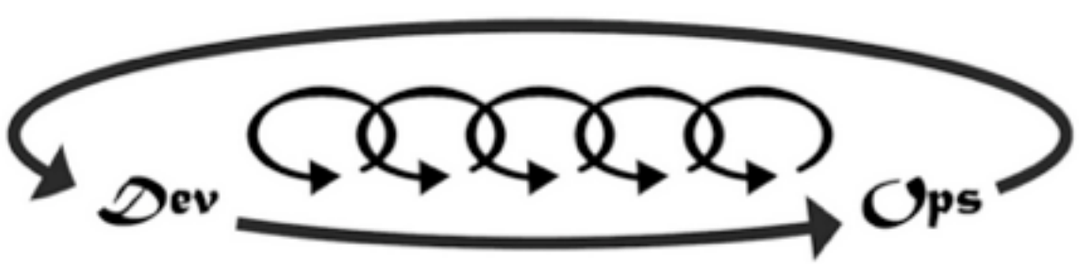
2. Focus on flow



3. Recognize feedback loops



4. Look for continuous improvement opportunities



Sure... but how do you do that?

Building organizational alignment

Building organizational alignment

1. Socialize the concepts and vocabulary

Building organizational alignment

1. Socialize the concepts and vocabulary
2. Visualize the system

Building organizational alignment

1. Socialize the concepts and vocabulary
2. Visualize the system
   a. value stream mapping

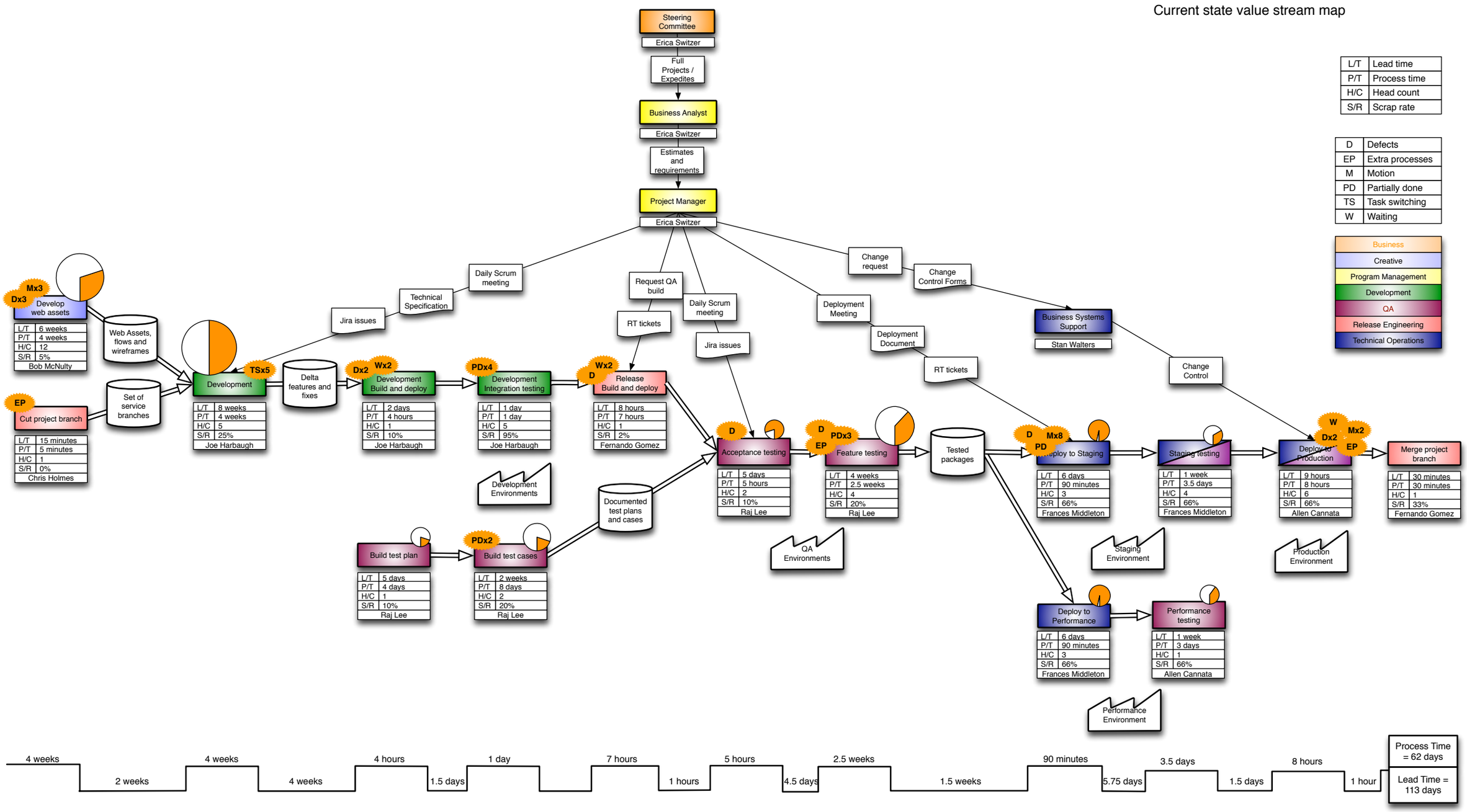
Building organizational alignment

1. Socialize the concepts and vocabulary
2. Visualize the system
   a. value stream mapping
   b. timeline analysis

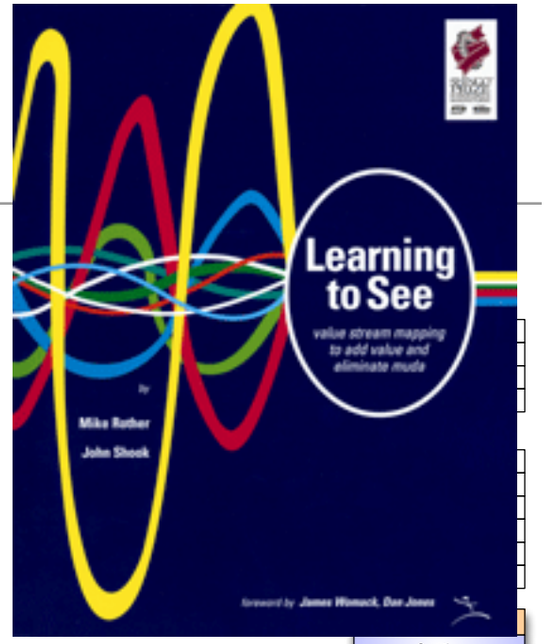
Building organizational alignment

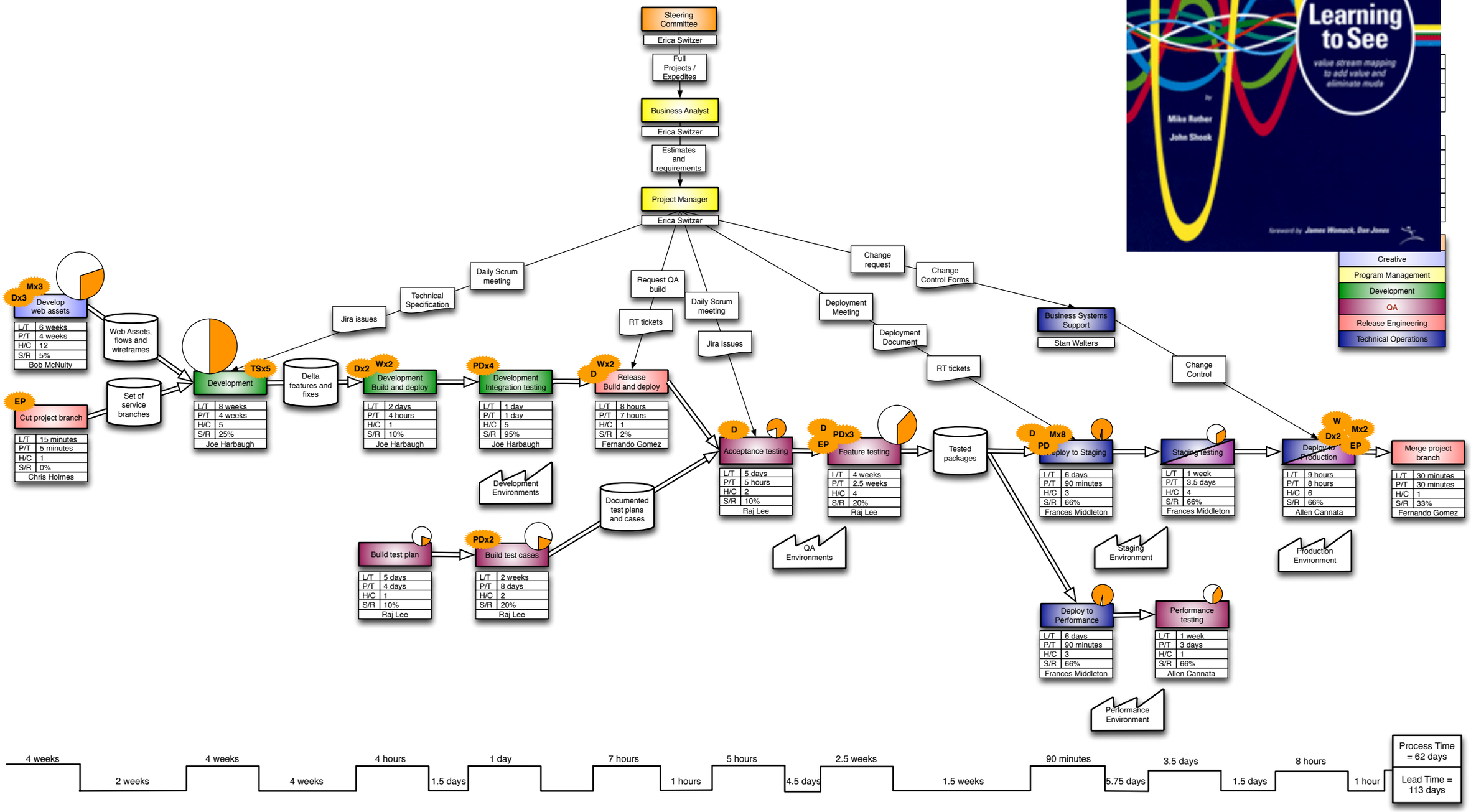
1. Socialize the concepts and vocabulary
2. Visualize the system
   a. value stream mapping
   b. timeline analysis
   c. waste analysis

Value Stream Mapping

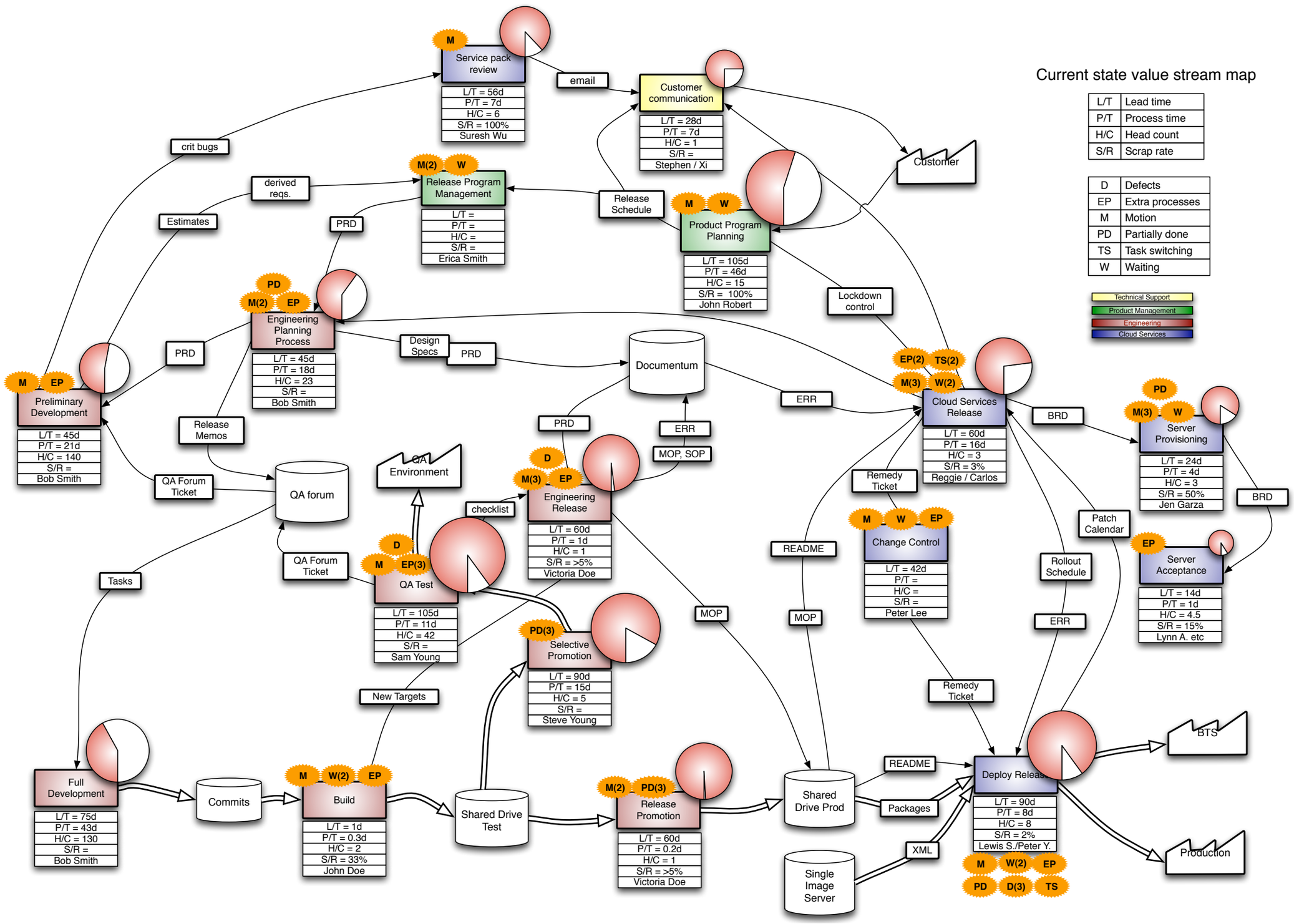


Value Stream Mapping

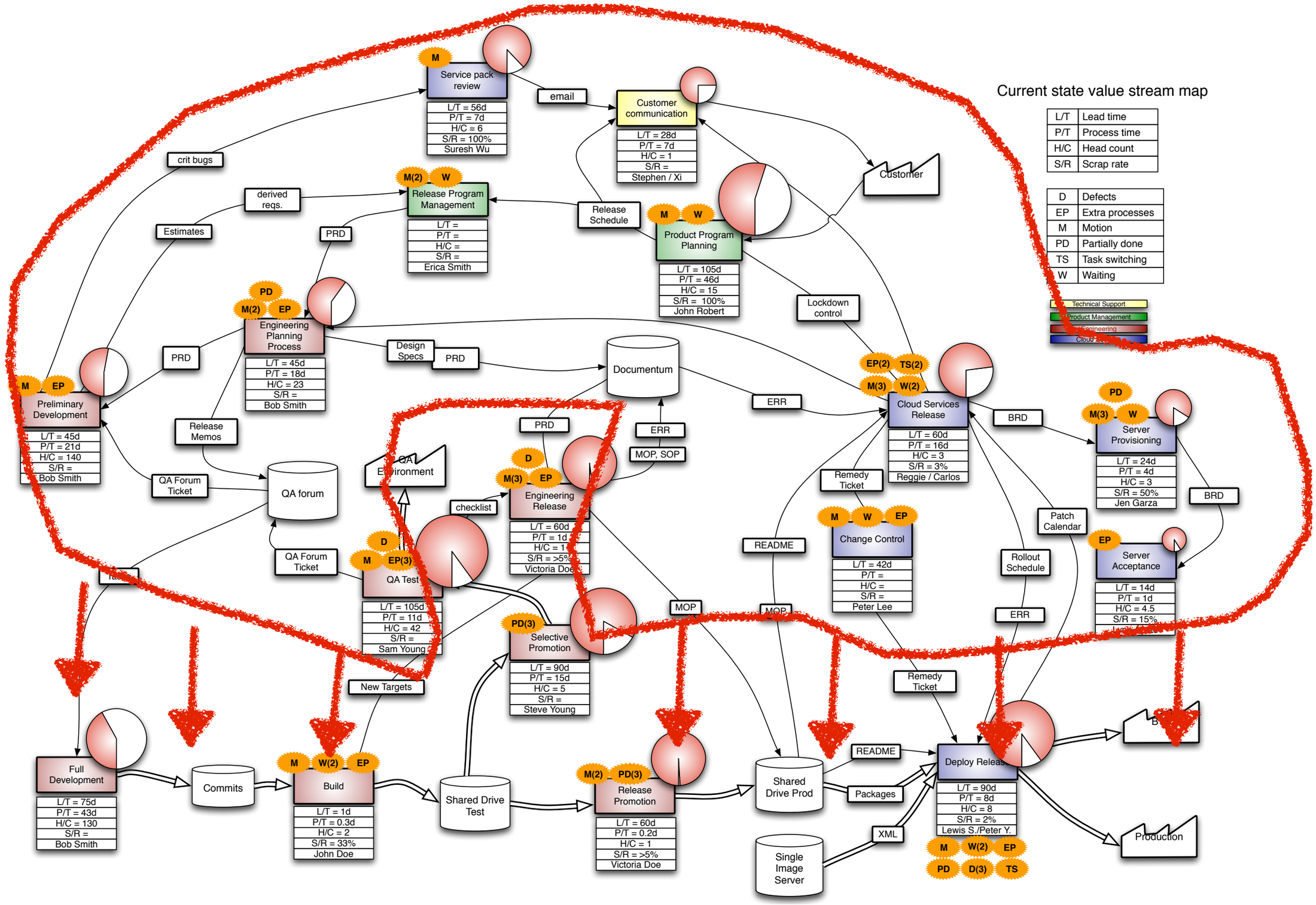




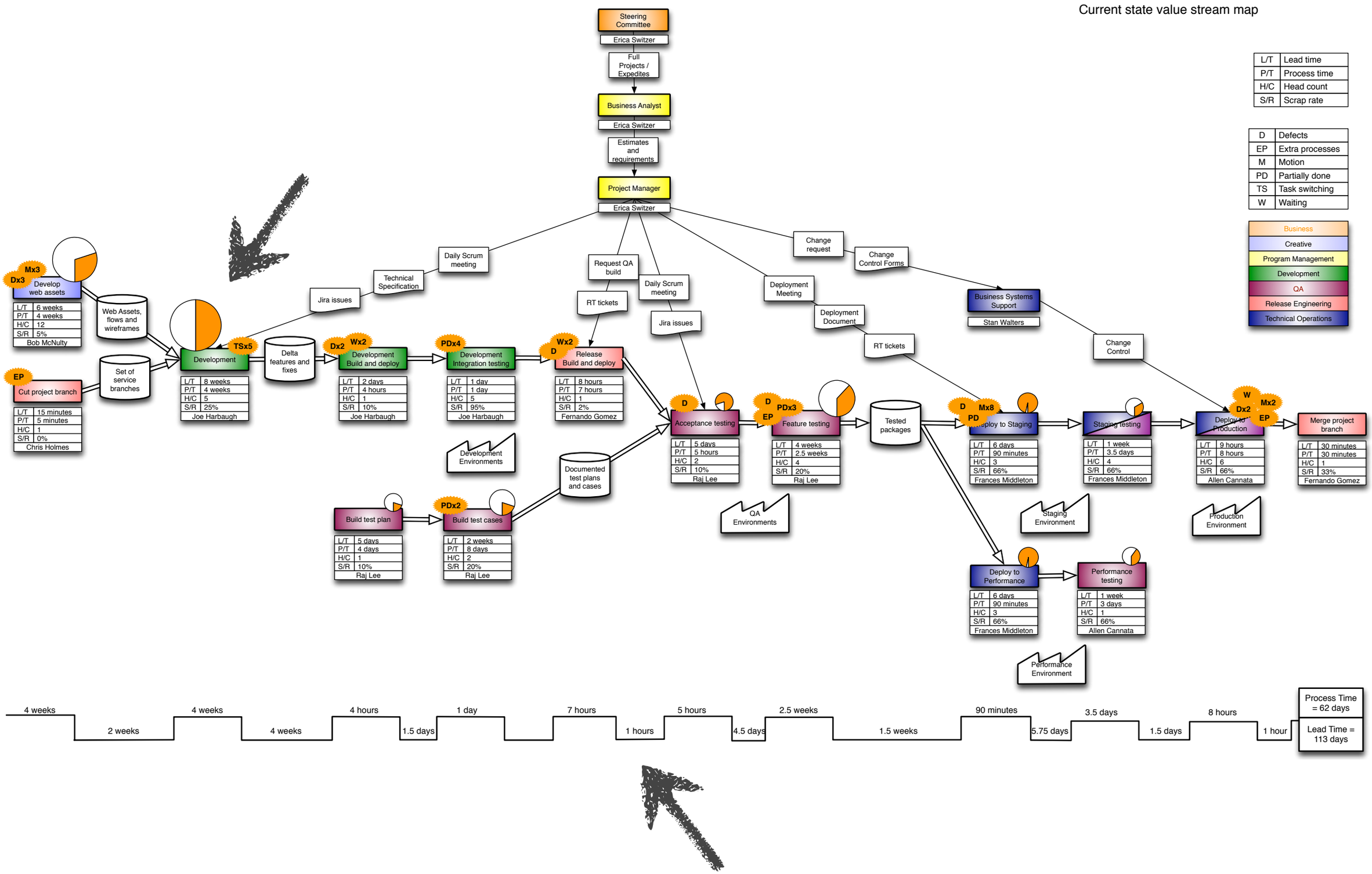
Value Stream Mapping



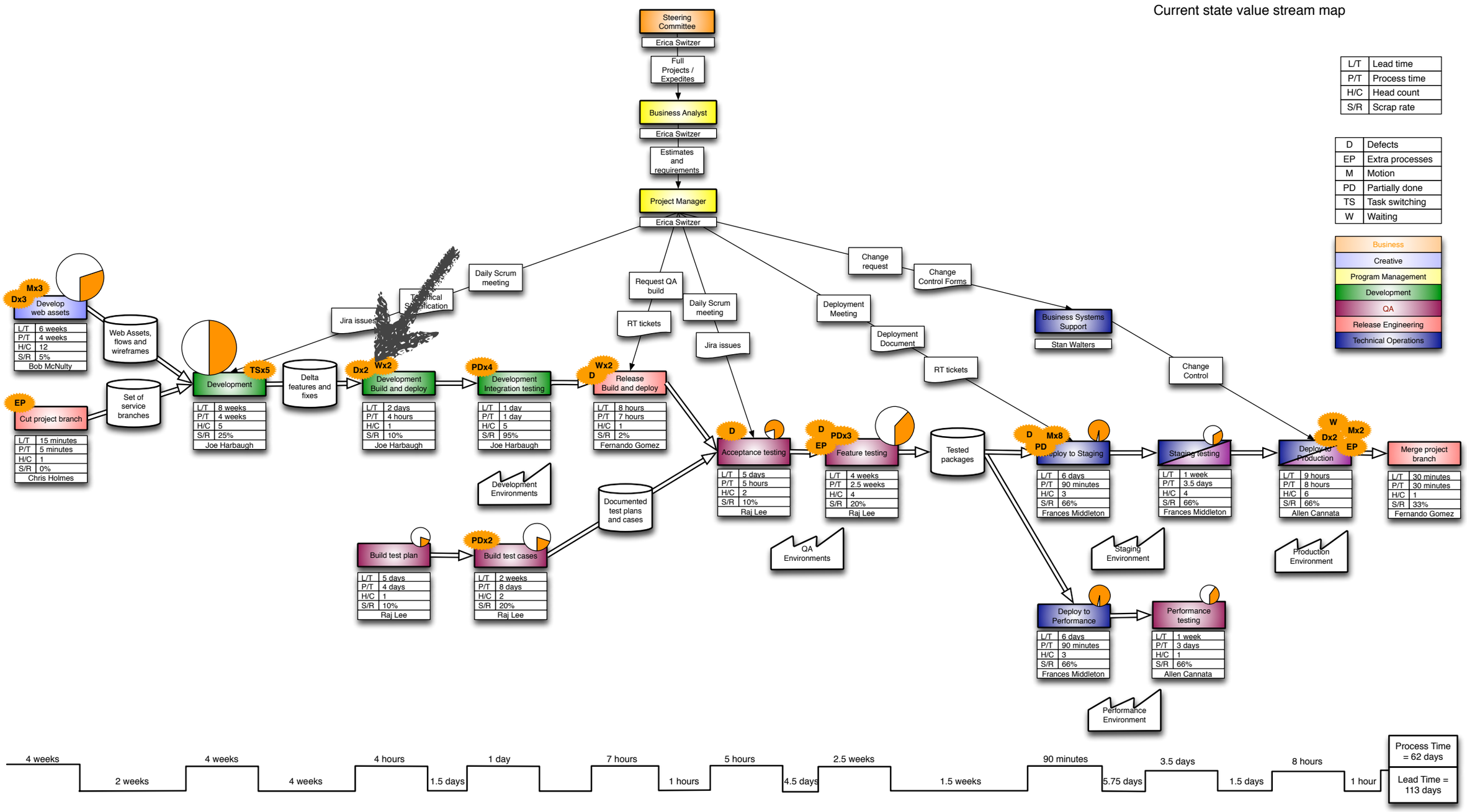
Value Stream Mapping



Timeline Analysis



Waste Analysis

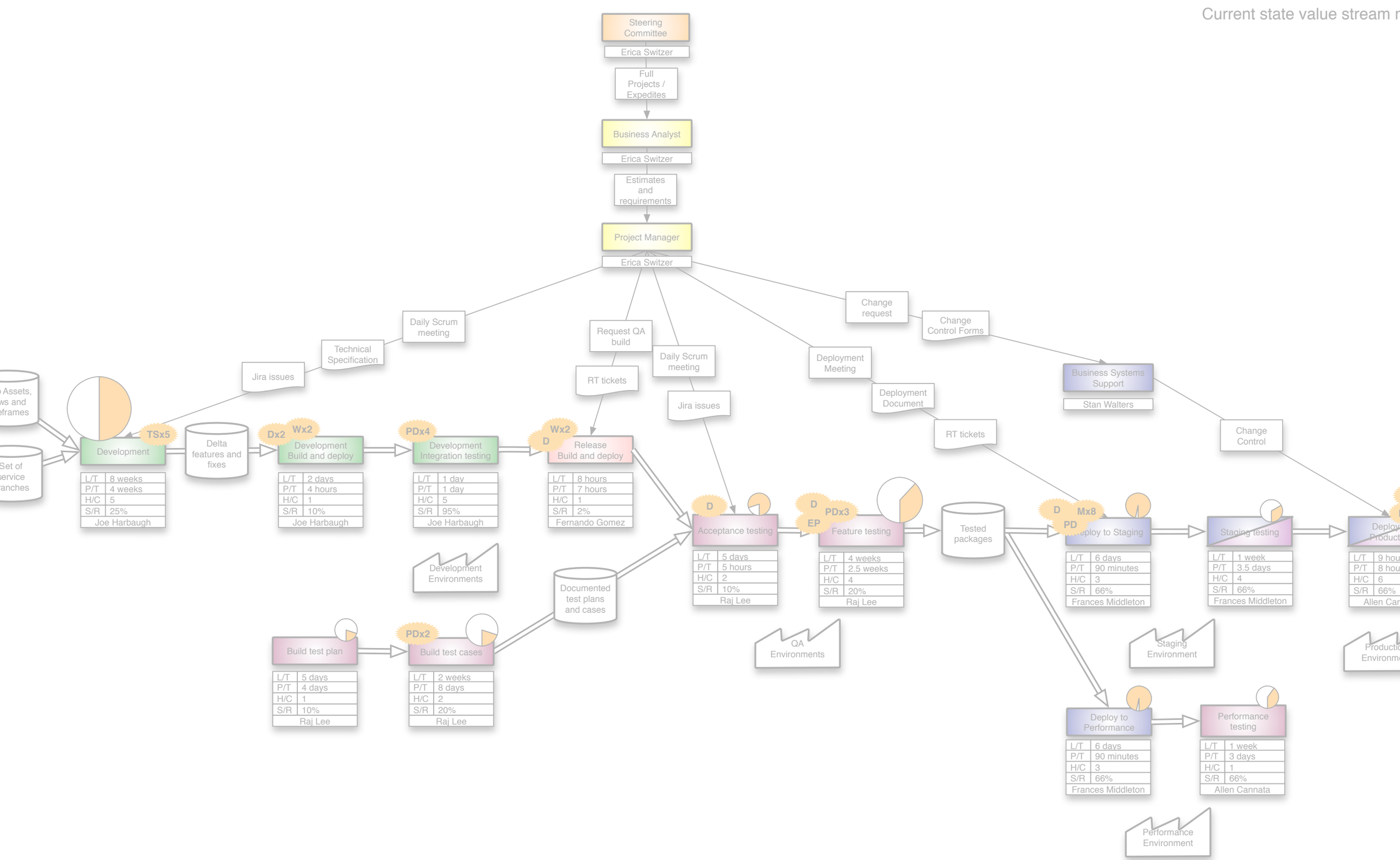


Waste Analysis

Waste	Icon	Description	Examples
Partially Done Work	PD	Any work item that is produced in the solution delivery process that has not been completed. This includes both partially done work within a process (ex: not reviewed requirements document) and work sitting in an inventory (ex: code waiting for QA). Partially done work becomes obsolete and loses value as time progresses.	• Documentation waiting for review • Untested code • Undeployed code
Extra Processes	EP	Any additional work that is being performed in a process that does not add value to the client. This may be documentation that is not used by the downstream processes or reviews/approvals that do not add any value to the output. Extra processes add effort and time to the value stream.	• Unused Documentation • Unnecessary reviews/approvals
Extra Features	EF	Gold plating or other additional features built into the solution that is not needed by the business. This may be features driven by technology or when the business asks for everything but the kitchen sink. Extra features adds complexity and effort to test and manage the functionality.	• Features driven by technology only • Features not likely to be used
Task Switching	TS	When people are assigned to multiple projects/streams requiring them to multi-task. The time required for context switching and managing dependencies between work adds additional effort and time to the value stream.	• People on multiple projects • Running concurrent streams with high dependencies
Waiting	W	Any delays between work requiring resources to wait until they can complete the current work item. Delays increase cycle times and prevent the client from realizing the value from the product as soon as possible.	• Delays from review/approvals
Motion	M	Amount of effort to move information/materials from one process to another. If people have to frequently communicate but are not co-located this is a form of motion waste. As well, hand-offs are another form as they require effort to move from one group to another and may require additional communication to resolve ambiguities	• Distributed teams • Hand-offs
Defects	D	Incorrect, missing and/or unclear information/materials/products create waste as effort is needed to resolve these issues. The amount of time from when the defect is created to when it's discovered increases impact of it to the value stream.	• Build defects • Requirement defects

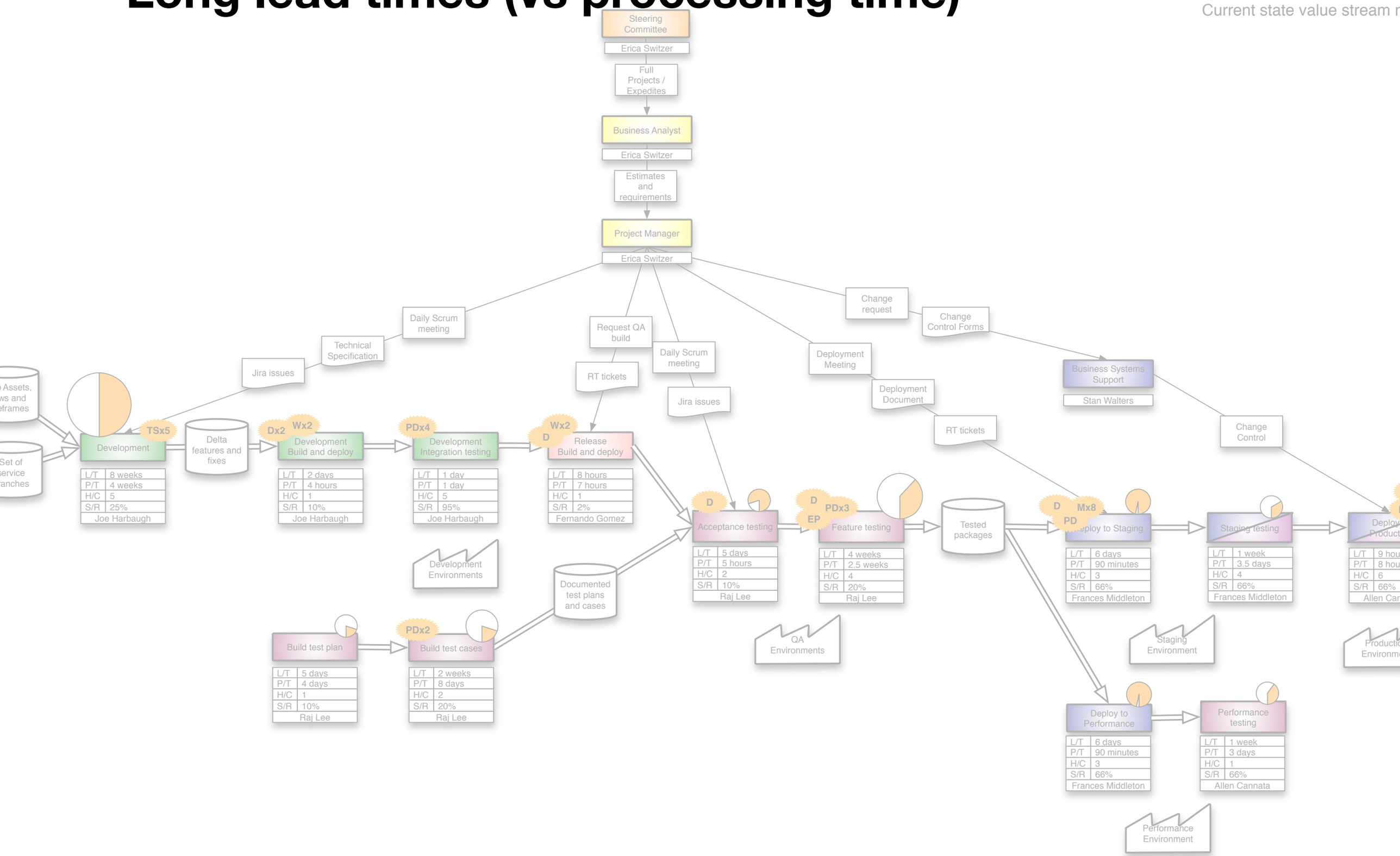
(Mary Poppendick's "Seven Wastes of Software Development")

What should we be looking for?



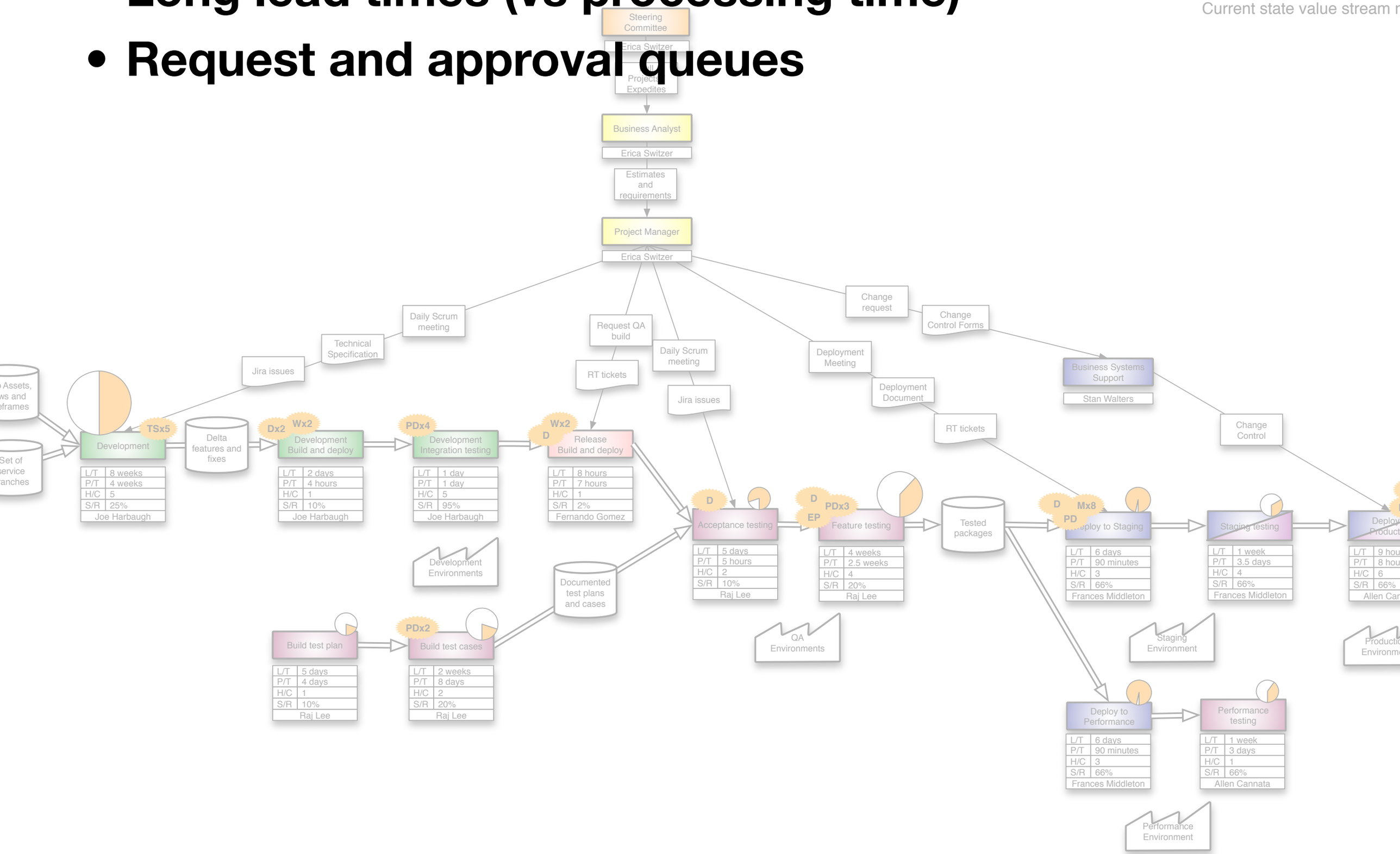
What should we be looking for?

- **Long lead times (vs processing time)**



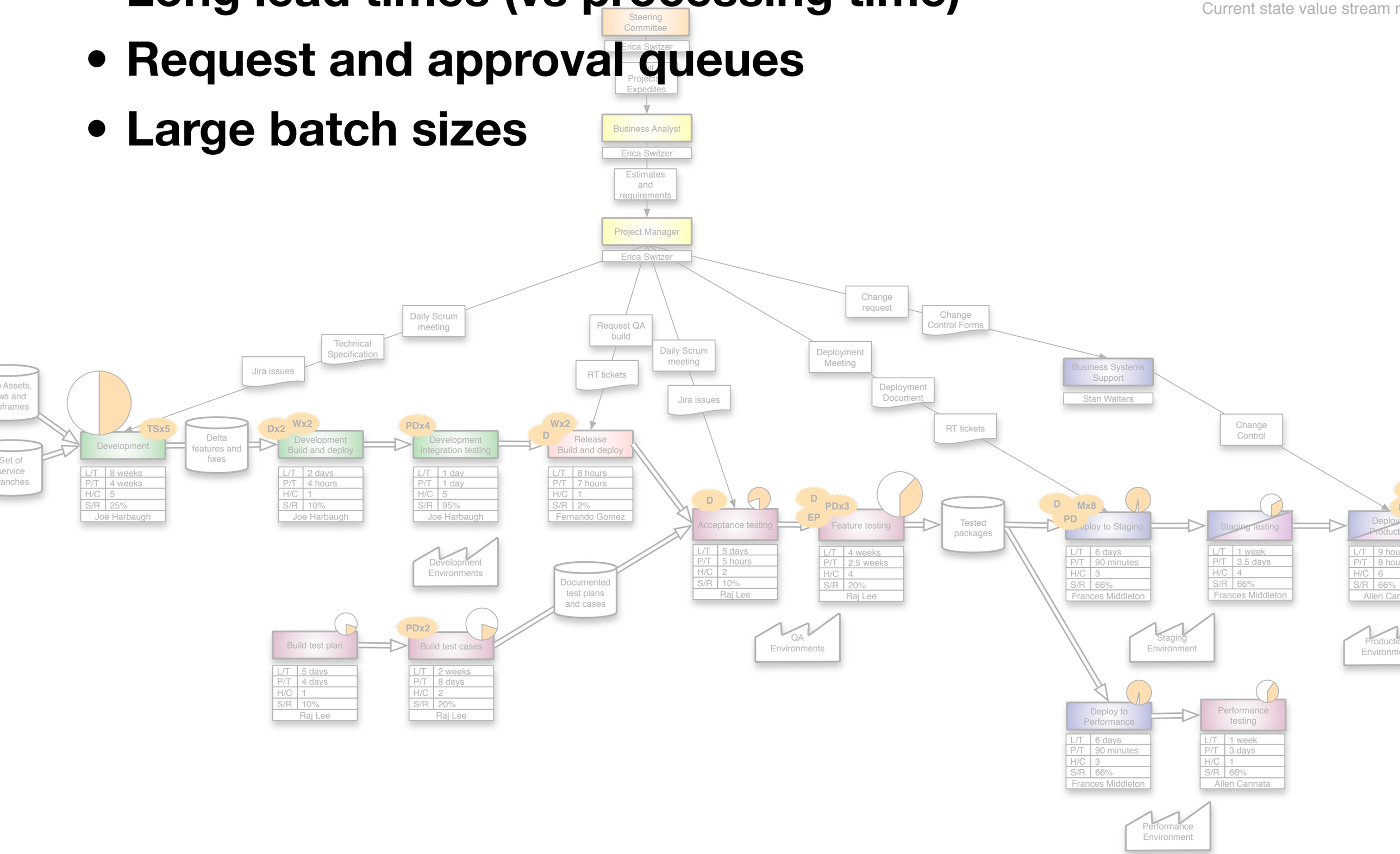
What should we be looking for?

- **Long lead times (vs processing time)**
- **Request and approval queues**



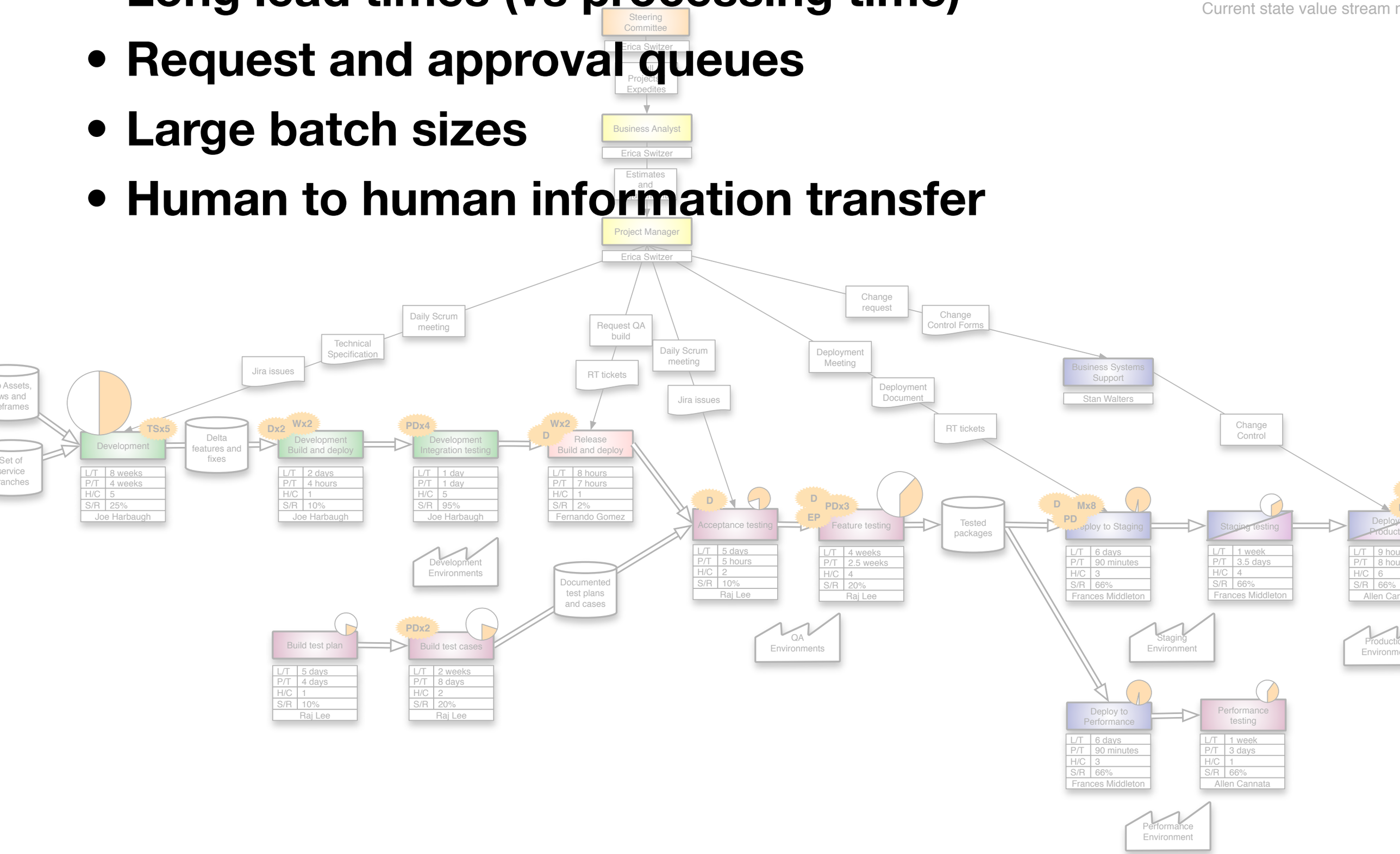
What should we be looking for?

- Long lead times (vs processing time)
- Request and approval queues
- Large batch sizes



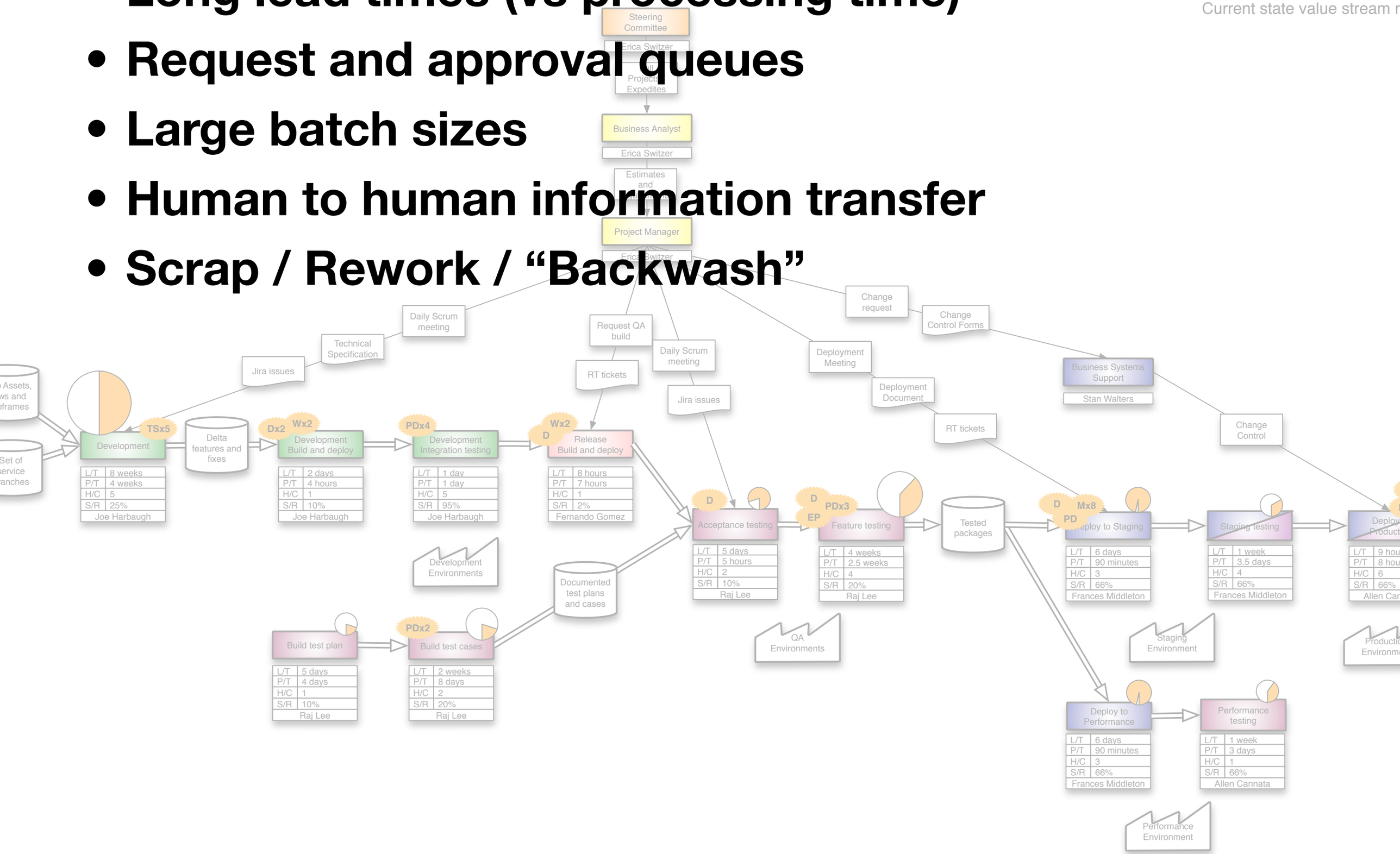
What should we be looking for?

- Long lead times (vs processing time)
- Request and approval queues
- Large batch sizes
- Human to human information transfer



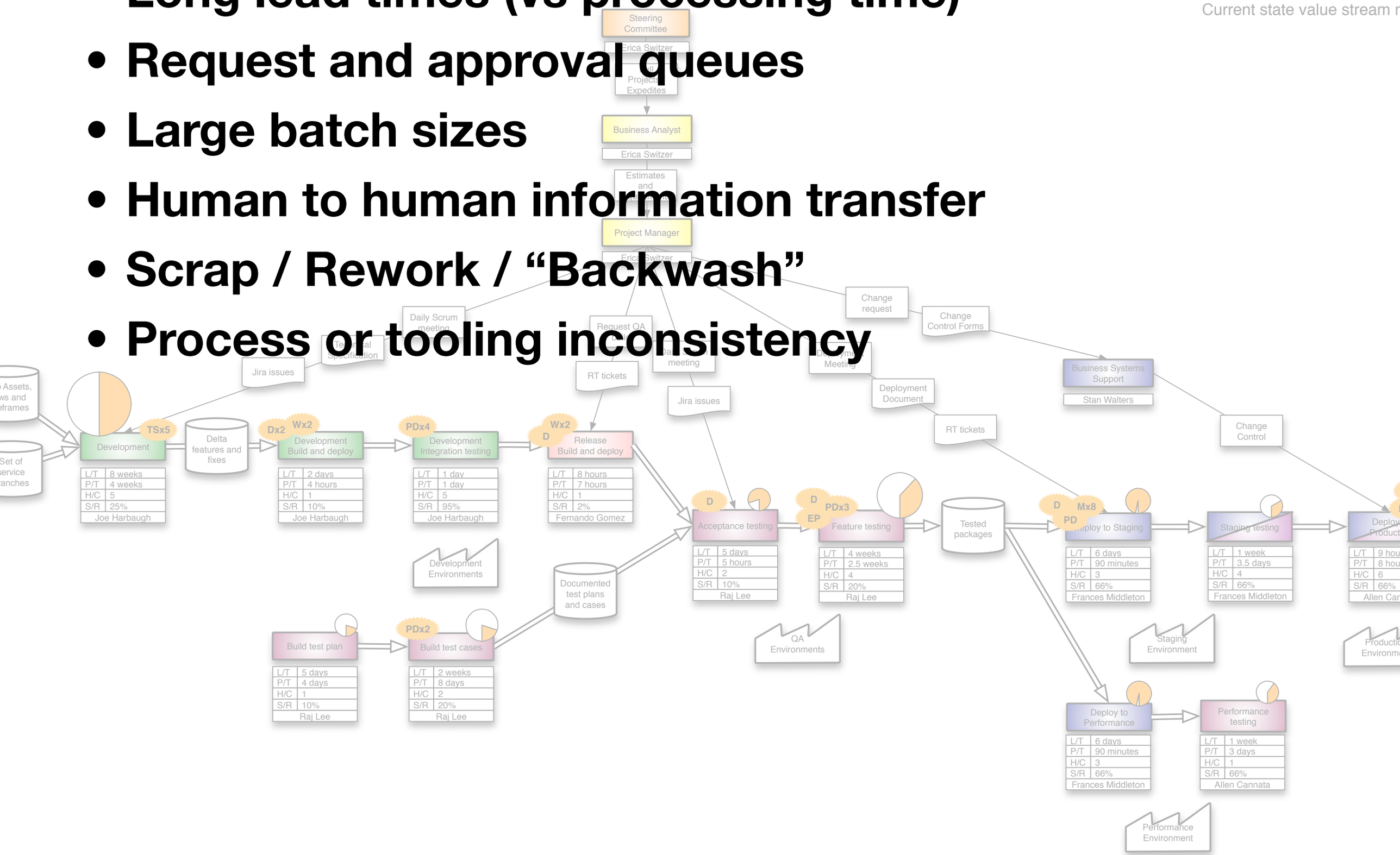
What should we be looking for?

- Long lead times (vs processing time)
- Request and approval queues
- Large batch sizes
- Human to human information transfer
- Scrap / Rework / “Backwash”



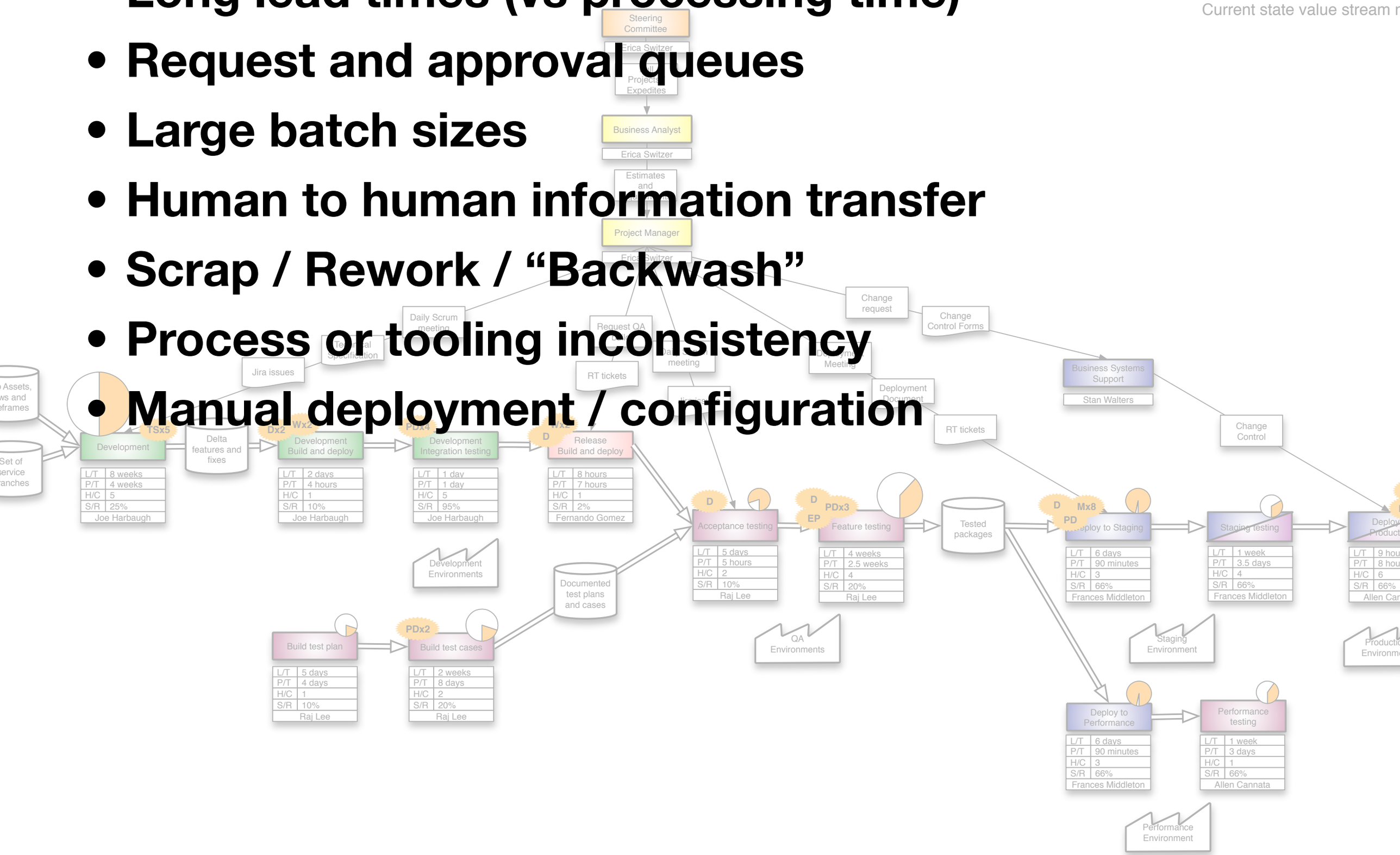
What should we be looking for?

- Long lead times (vs processing time)
- Request and approval queues
- Large batch sizes
- Human to human information transfer
- Scrap / Rework / “Backwash”
- Process or tooling inconsistency



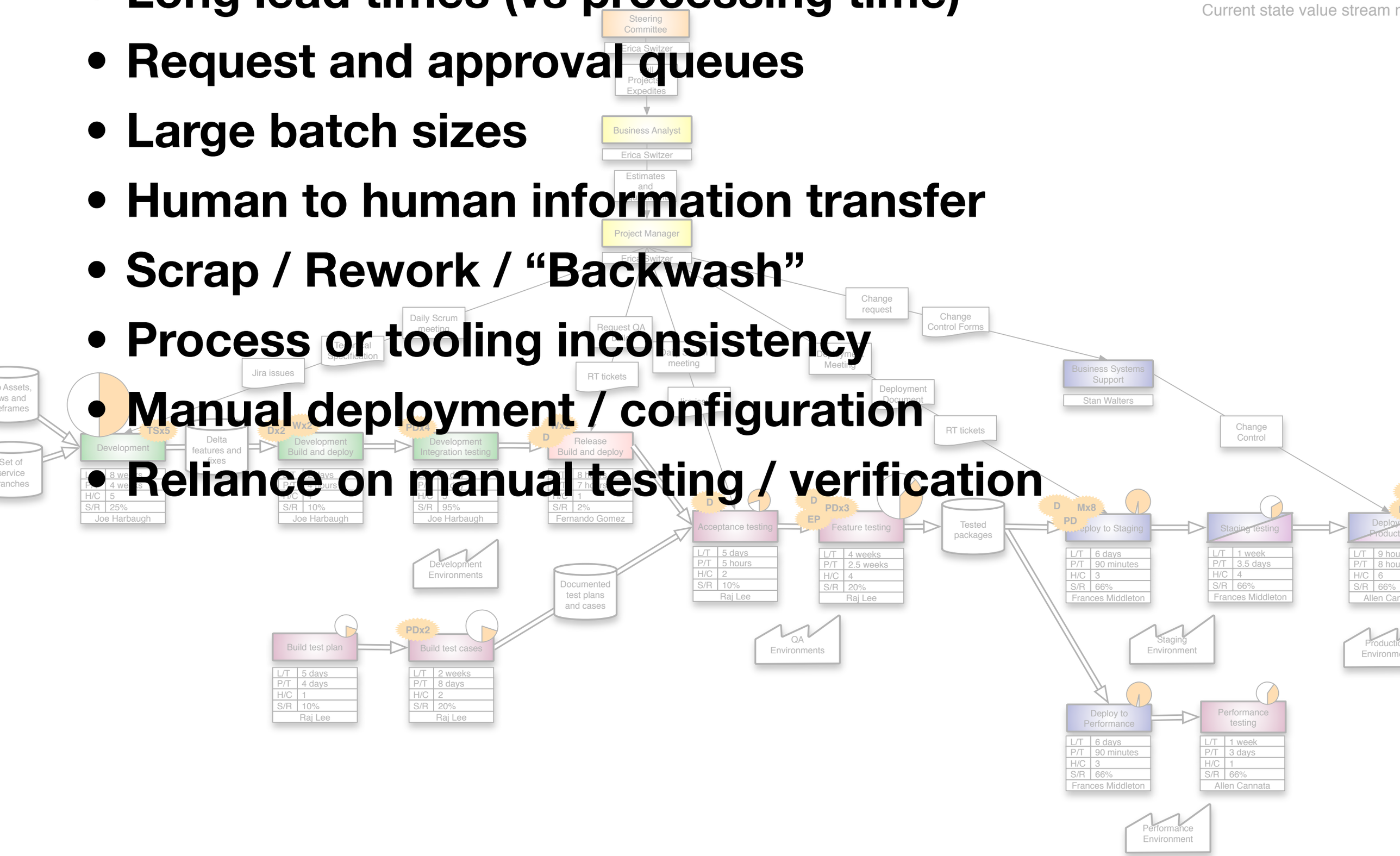
What should we be looking for?

- Long lead times (vs processing time)
- Request and approval queues
- Large batch sizes
- Human to human information transfer
- Scrap / Rework / “Backwash”
- Process or tooling inconsistency
- Manual deployment / configuration



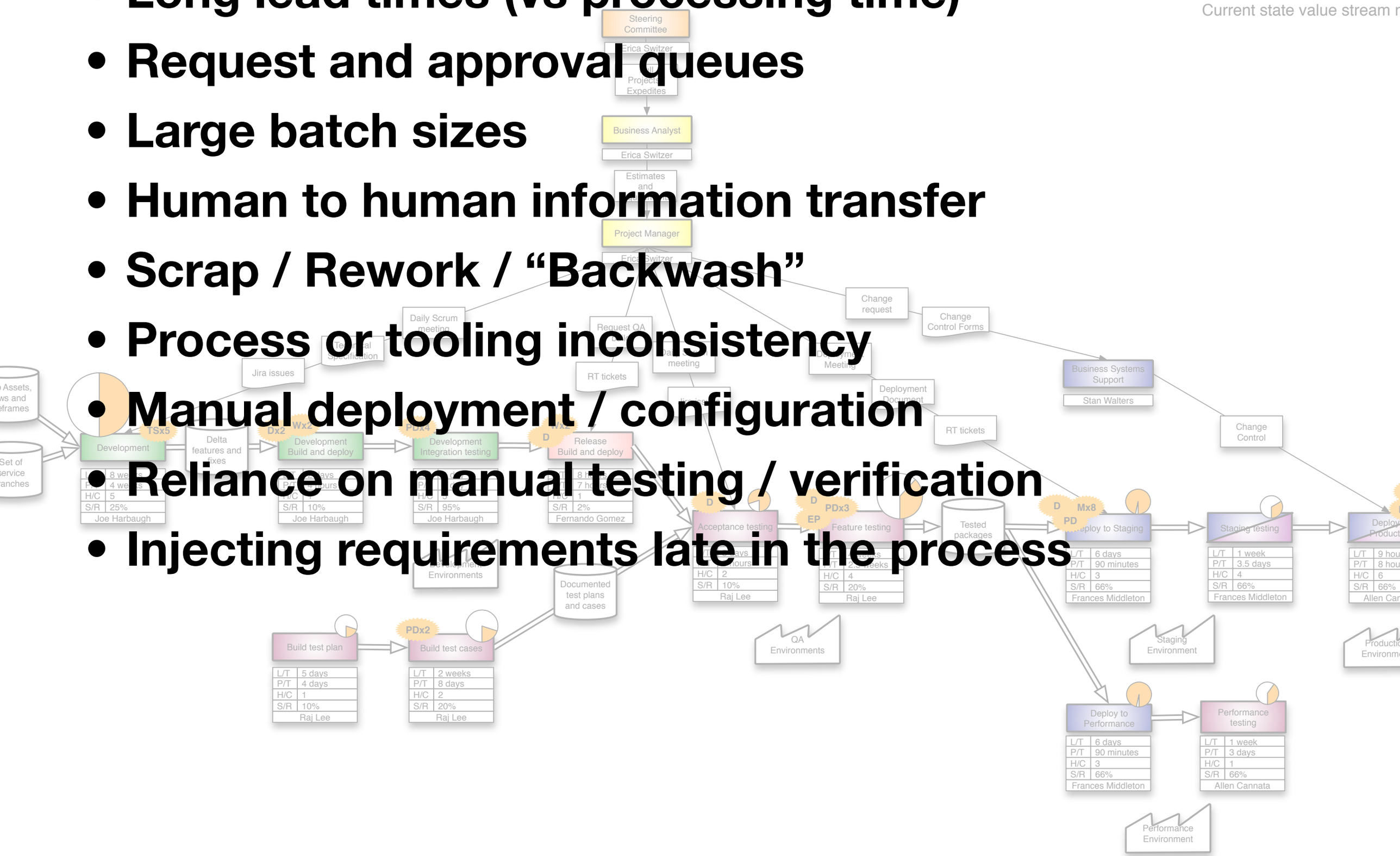
What should we be looking for?

- Long lead times (vs processing time)
- Request and approval queues
- Large batch sizes
- Human to human information transfer
- Scrap / Rework / “Backwash”
- Process or tooling inconsistency
- Manual deployment / configuration
- Reliance on manual testing / verification



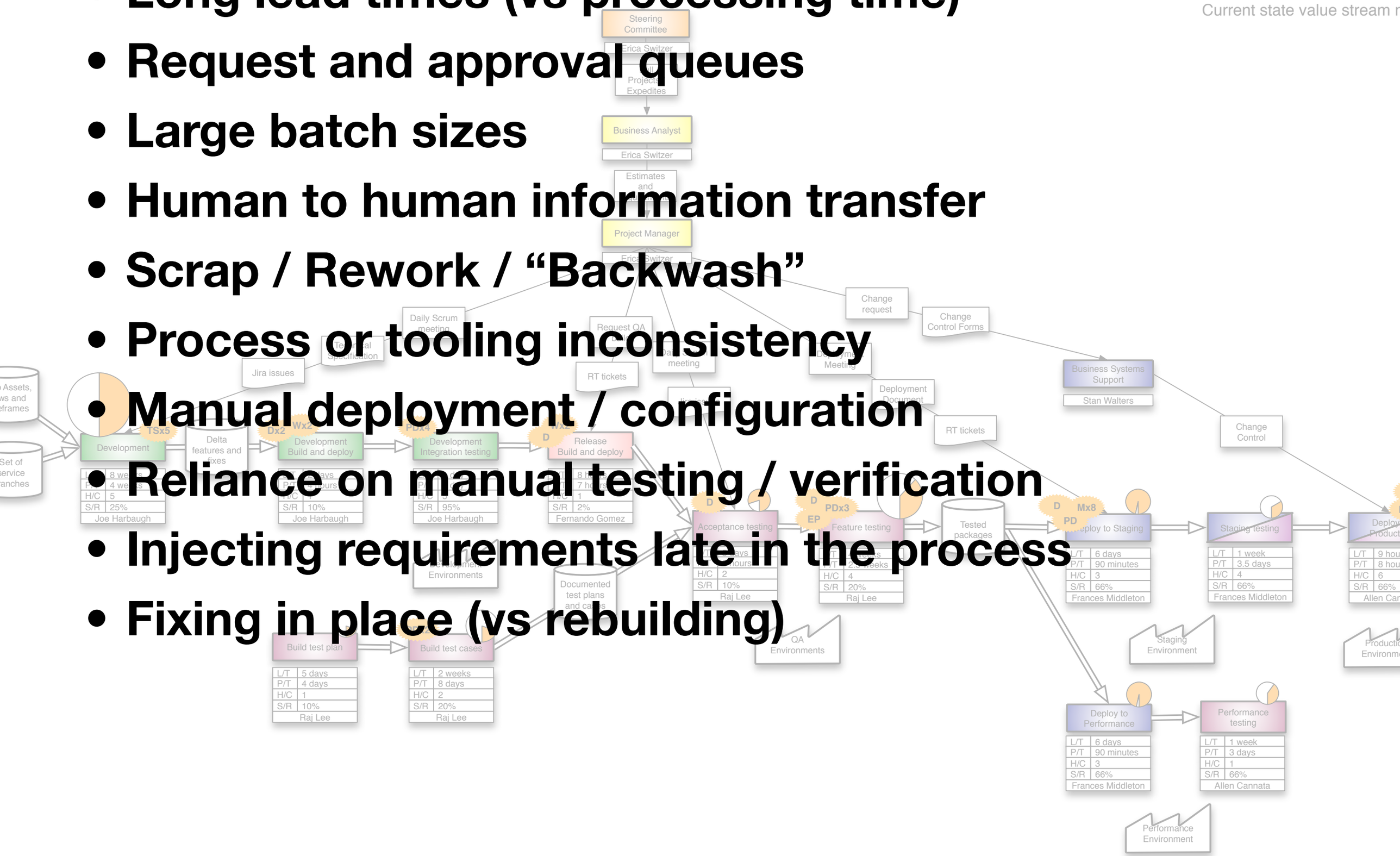
What should we be looking for?

- Long lead times (vs processing time)
- Request and approval queues
- Large batch sizes
- Human to human information transfer
- Scrap / Rework / “Backwash”
- Process or tooling inconsistency
- Manual deployment / configuration
- Reliance on manual testing / verification
- Injecting requirements late in the process



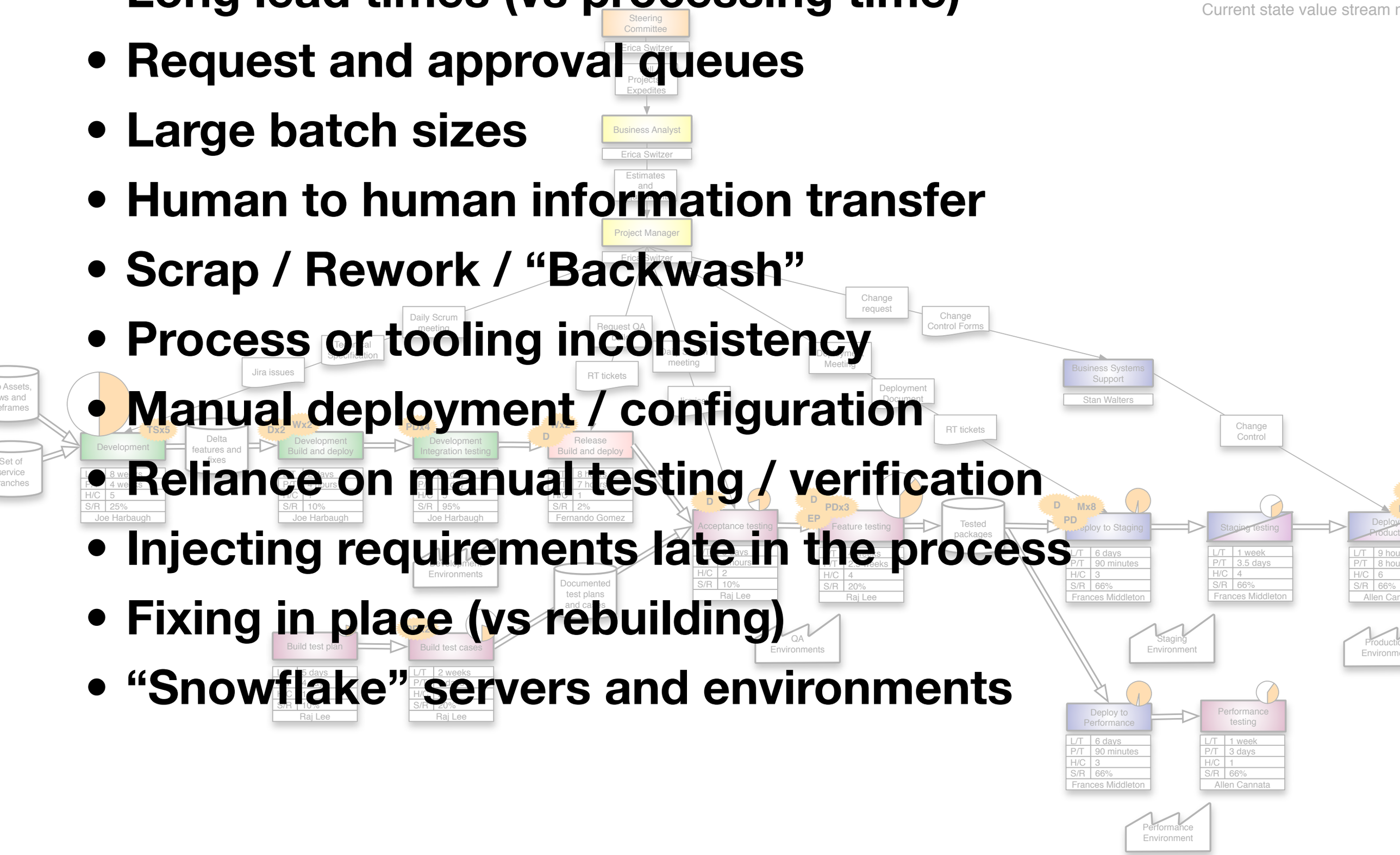
What should we be looking for?

- Long lead times (vs processing time)
- Request and approval queues
- Large batch sizes
- Human to human information transfer
- Scrap / Rework / “Backwash”
- Process or tooling inconsistency
- Manual deployment / configuration
- Reliance on manual testing / verification
- Injecting requirements late in the process
- Fixing in place (vs rebuilding)



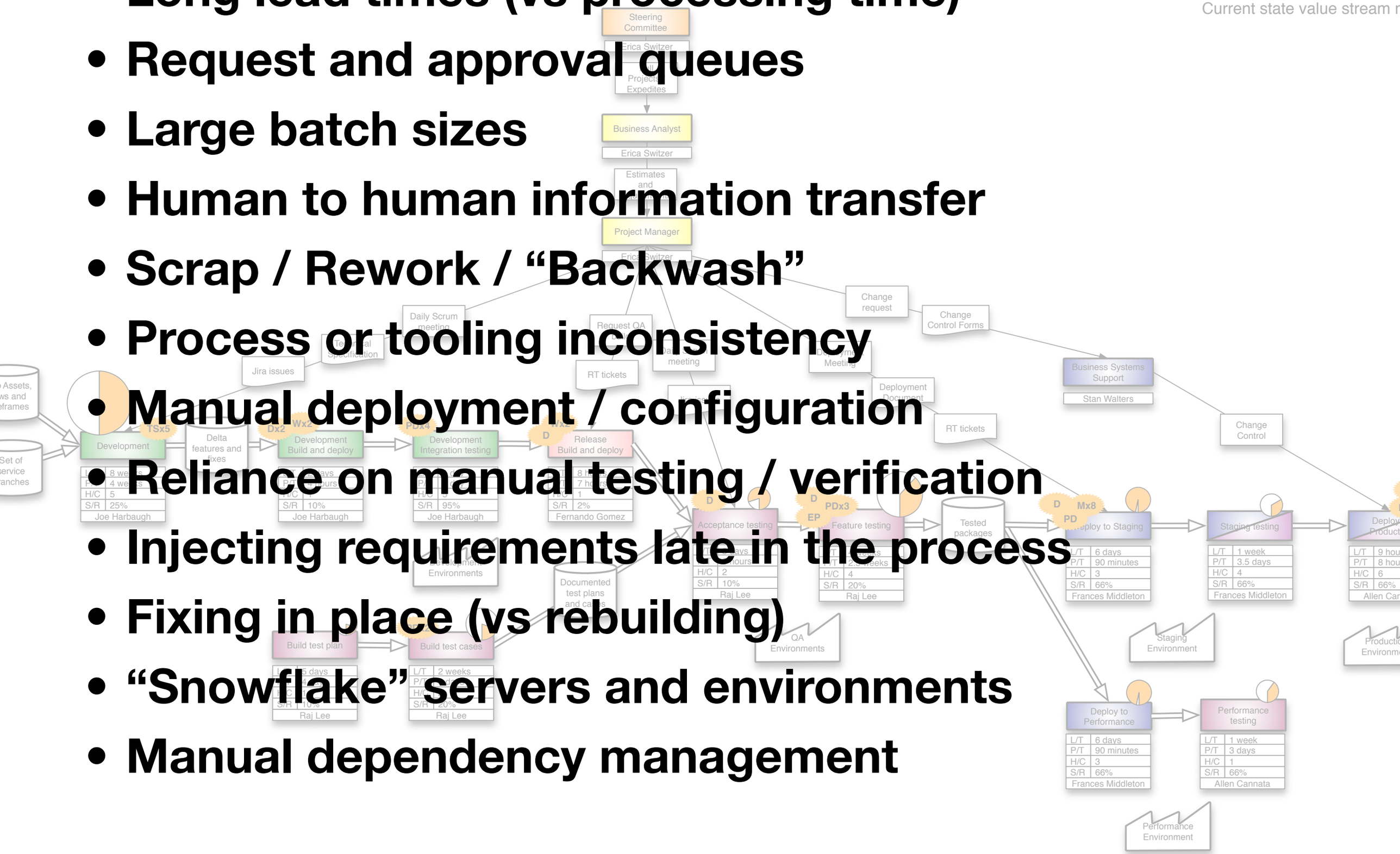
What should we be looking for?

- Long lead times (vs processing time)
- Request and approval queues
- Large batch sizes
- Human to human information transfer
- Scrap / Rework / “Backwash”
- Process or tooling inconsistency
- Manual deployment / configuration
- Reliance on manual testing / verification
- Injecting requirements late in the process
- Fixing in place (vs rebuilding)
- “Snowflake” servers and environments



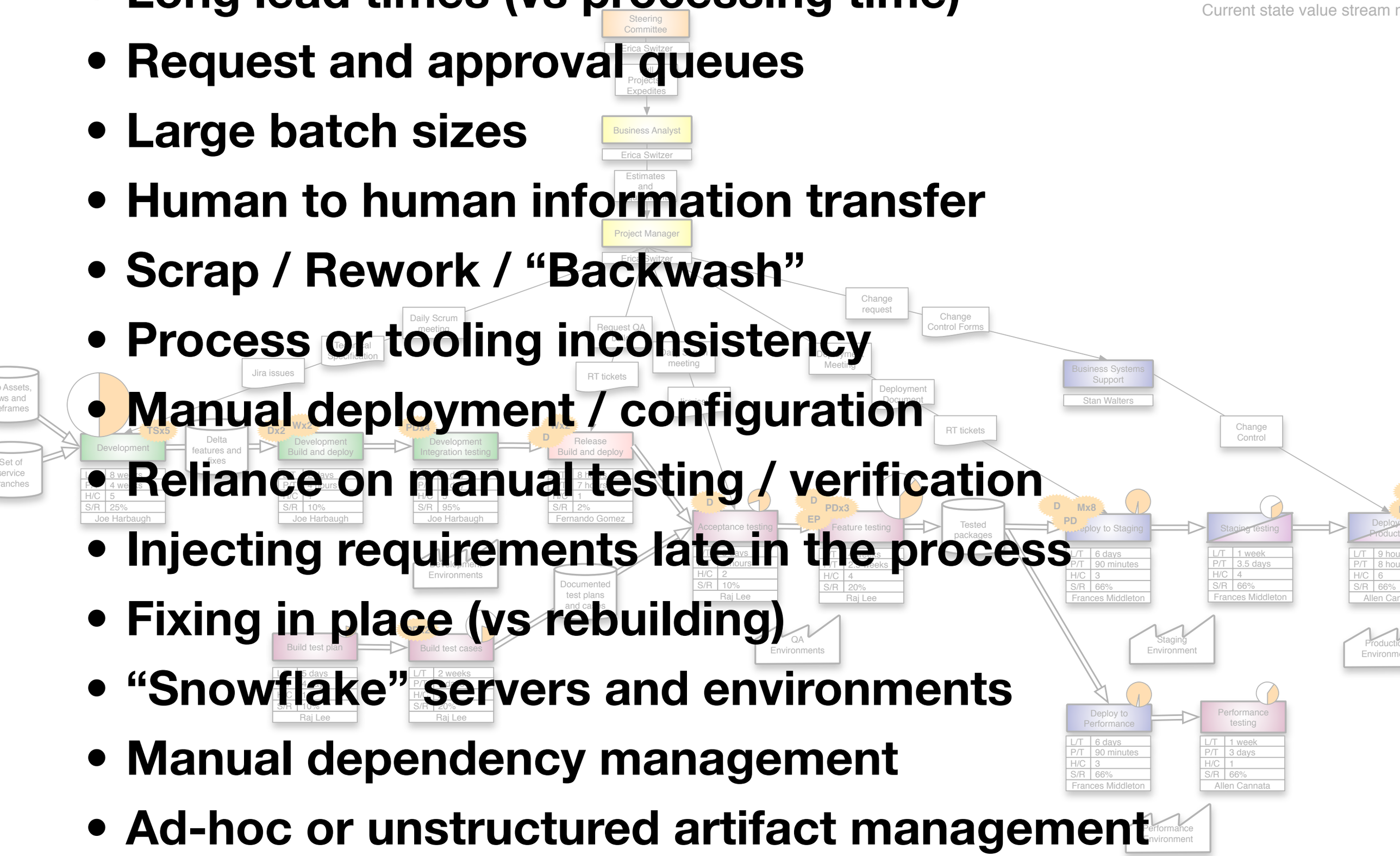
What should we be looking for?

- Long lead times (vs processing time)
- Request and approval queues
- Large batch sizes
- Human to human information transfer
- Scrap / Rework / “Backwash”
- Process or tooling inconsistency
- Manual deployment / configuration
- Reliance on manual testing / verification
- Injecting requirements late in the process
- Fixing in place (vs rebuilding)
- “Snowflake” servers and environments
- Manual dependency management



What should we be looking for?

- Long lead times (vs processing time)
- Request and approval queues
- Large batch sizes
- Human to human information transfer
- Scrap / Rework / “Backwash”
- Process or tooling inconsistency
- Manual deployment / configuration
- Reliance on manual testing / verification
- Injecting requirements late in the process
- Fixing in place (vs rebuilding)
- “Snowflake” servers and environments
- Manual dependency management
- Ad-hoc or unstructured artifact management



Building organizational alignment

1. Socialize the concepts and vocabulary
2. Visualize the system
   a. value stream mapping
   b. timeline analysis
   c. waste analysis

Building organizational alignment

1. Socialize the concepts and vocabulary
2. Visualize the system
   a. value stream mapping
   b. timeline analysis
   c. waste analysis
3. Pick metrics that matter

Are you getting better as an organization?

Are you getting better as an organization?

- Cycle Time

Are you getting better as an organization?

- Cycle Time
- MTTD (Mean Time To Detect)

Are you getting better as an organization?

- Cycle Time
- MTTD (Mean Time To Detect)
- MTTR (Mean Time to Repair)

Are you getting better as an organization?

- Cycle Time
- MTTD (Mean Time To Detect)
- MTTR (Mean Time to Repair)
- Quality at the Source (Scrap)

Metrics chains tie the individual to the goal

What matters to the business

Capability that influences what matters to the business

Activity over which an individual can cause/influence outcomes

Building organizational alignment

1. Socialize the concepts and vocabulary
2. Visualize the system
   a. value stream mapping
   b. timeline analysis
   c. waste analysis
3. Pick metrics that matter

Building organizational alignment

1. Socialize the concepts and vocabulary
2. Visualize the system
   a. value stream mapping
   b. timeline analysis
   c. waste analysis
3. Pick metrics that matter
4. Identify projects / experiments against baseline

Building organizational alignment

1. Socialize the concepts and vocabulary
2. Visualize the system
   a. value stream mapping
   b. timeline analysis
   c. waste analysis
3. Pick metrics that matter
4. Identify projects / experiments against baseline
5. Repeat steps 2 - 4
(continuous improvement program)

Start with a burst of energy

DevOps Workshop

Agenda	Day 1	Day 2	Day 3	Day 4
	Kickoff DevOps Goals Key Concepts	Current State Analysis	Future State / Solution Design	Future State / Solution Design
	Lunch	Lunch	Lunch	Lunch
	Case Study Discussion	Current State Analysis	Future State / Solution Design	Wrap-up

= Principles = Analysis = Design

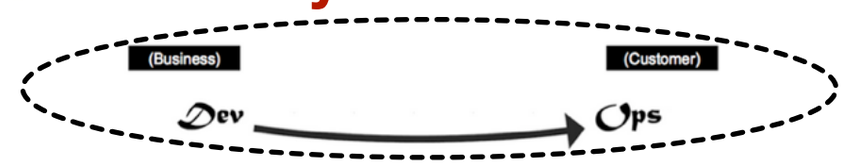
Building organizational alignment

1. Socialize the concepts and vocabulary
2. Visualize the system
   a. value stream mapping
   b. timeline analysis
   c. waste analysis
3. Pick metrics that matter
4. Identify projects / experiments against baseline
5. Repeat steps 2 - 4
(continuous improvement program)

Building organizational alignment

1. Socialize the concepts and vocabulary
2. Visualize the system
   a. value stream mapping
   b. timeline analysis
   c. waste analysis
3. Pick metrics that matter
4. Identify projects / experiments against baseline
5. Repeat steps 2 - 4
(continuous improvement program)

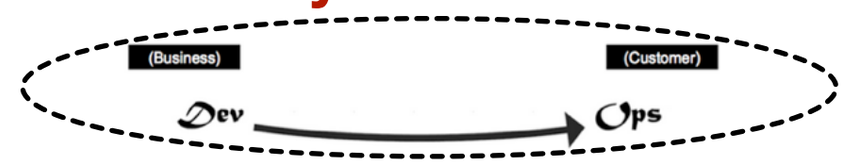
1. See the system



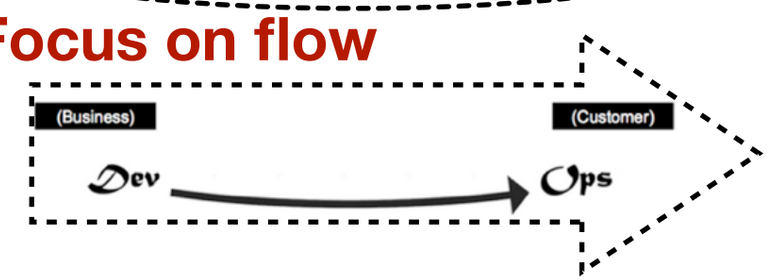
Building organizational alignment

1. Socialize the concepts and vocabulary
2. Visualize the system
   a. value stream mapping
   b. timeline analysis
   c. waste analysis
3. Pick metrics that matter
4. Identify projects / experiments against baseline
5. Repeat steps 2 - 4
(continuous improvement program)

1. See the system



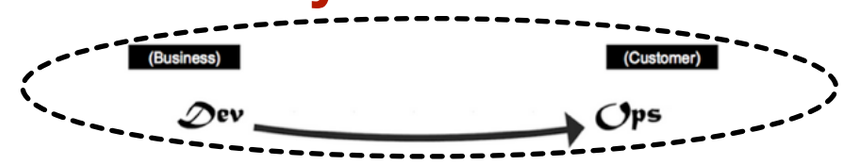
2. Focus on flow



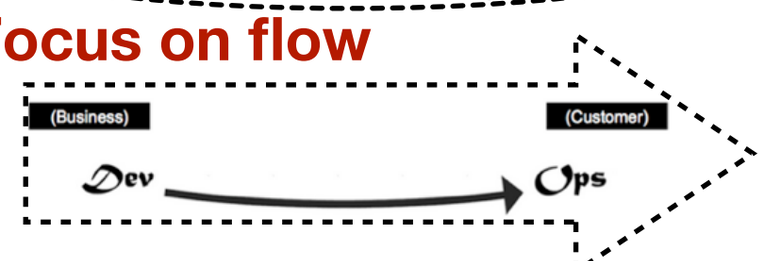
Building organizational alignment

1. Socialize the concepts and vocabulary
2. Visualize the system
   a. value stream mapping
   b. timeline analysis
   c. waste analysis
3. Pick metrics that matter
4. Identify projects / experiments against baseline
5. Repeat steps 2 - 4
(continuous improvement program)

1. See the system



2. Focus on flow

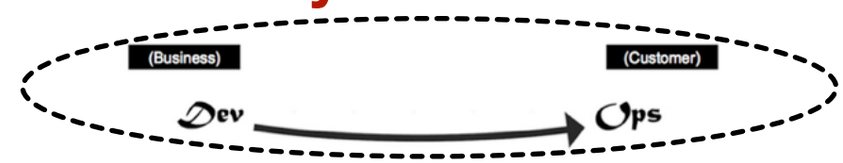


3. Recognize feedback loops

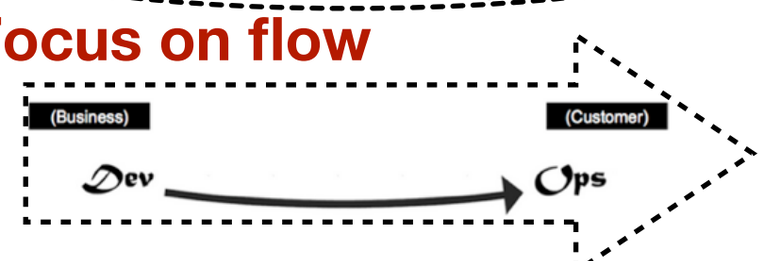
Building organizational alignment

1. Socialize the concepts and vocabulary
2. Visualize the system
   a. value stream mapping
   b. timeline analysis
   c. waste analysis
3. Pick metrics that matter
4. Identify projects / experiments against baseline
5. Repeat steps 2 - 4 (continuous improvement program)

1. See the system

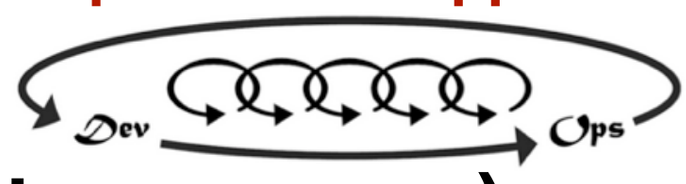


2. Focus on flow



3. Recognize feedback loops

4. Look for continuous improvement opportunities



Dev initiated DevOps Transformation

1. Take an “operations first” mindset
2. Build organizational alignment
3. **Establish a new model for working with Ops**

Example

What does Dev want?

What does Dev want?

- **What they need to get their job done**

What does Dev want?

- What they need to get their job done
- When they need it

What does Dev want?

- What they need to get their job done
- When they need it
- Fast feedback

What does Dev want?

- What they need to get their job done
- When they need it
- Fast feedback
- Dependable and predictable systems to integrate with

What does Dev want?

- What they need to get their job done
- When they need it
- Fast feedback
- Dependable and predictable systems to integrate with
- Limit extraneous information or tasks

What does Dev want?

- What they need to get their job done
- When they need it
- Fast feedback
- Dependable and predictable systems to integrate with
- Limit extraneous information or tasks
- For everyone to get out of their way

What does Ops want?

What does Ops want?

- Enough time to do their work
  - Deployment / provisioning
  - Stability and performance engineering
  - Hardening and security
  - Paying down technical debt
  - Compliance

What does Ops want?

- Enough time to do their work
  - Deployment / provisioning
  - Stability and performance engineering
  - Hardening and security
  - Paying down technical debt
  - Compliance
- To have their requirements considered earlier in the lifecycle

What does Ops want?

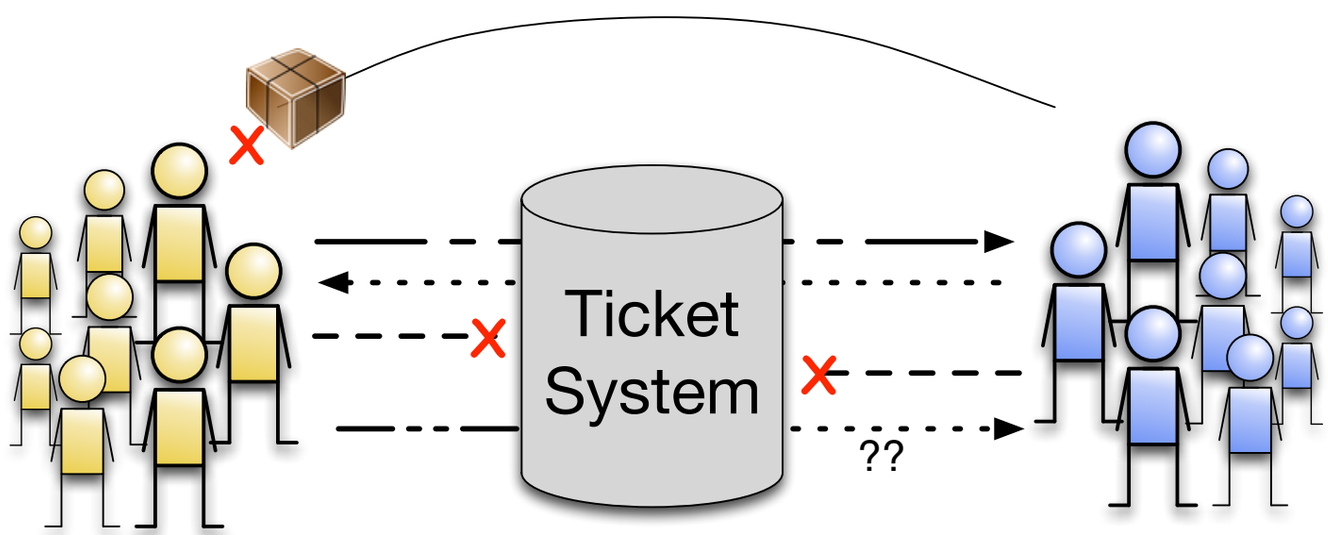
- Enough time to do their work
  - Deployment / provisioning
  - Stability and performance engineering
  - Hardening and security
  - Paying down technical debt
  - Compliance
- To have their requirements considered earlier in the lifecycle
- Confidence that changes are not going to break the system or create a vulnerability

Dev and Ops interact through request queues

Dev and Ops interact through request queues

Leads to...

- Bottlenecks
- Increased lead times
- Reinforces organizational silos
- Misinterpretation or omissions



Replace request queues with self-service interfaces

Replace request queues with self-service interfaces

- **Fully automate what used to be done by humans**

Replace request queues with self-service interfaces

- Fully automate what used to be done by humans
- Put behind self-service interfaces for on-demand consumption

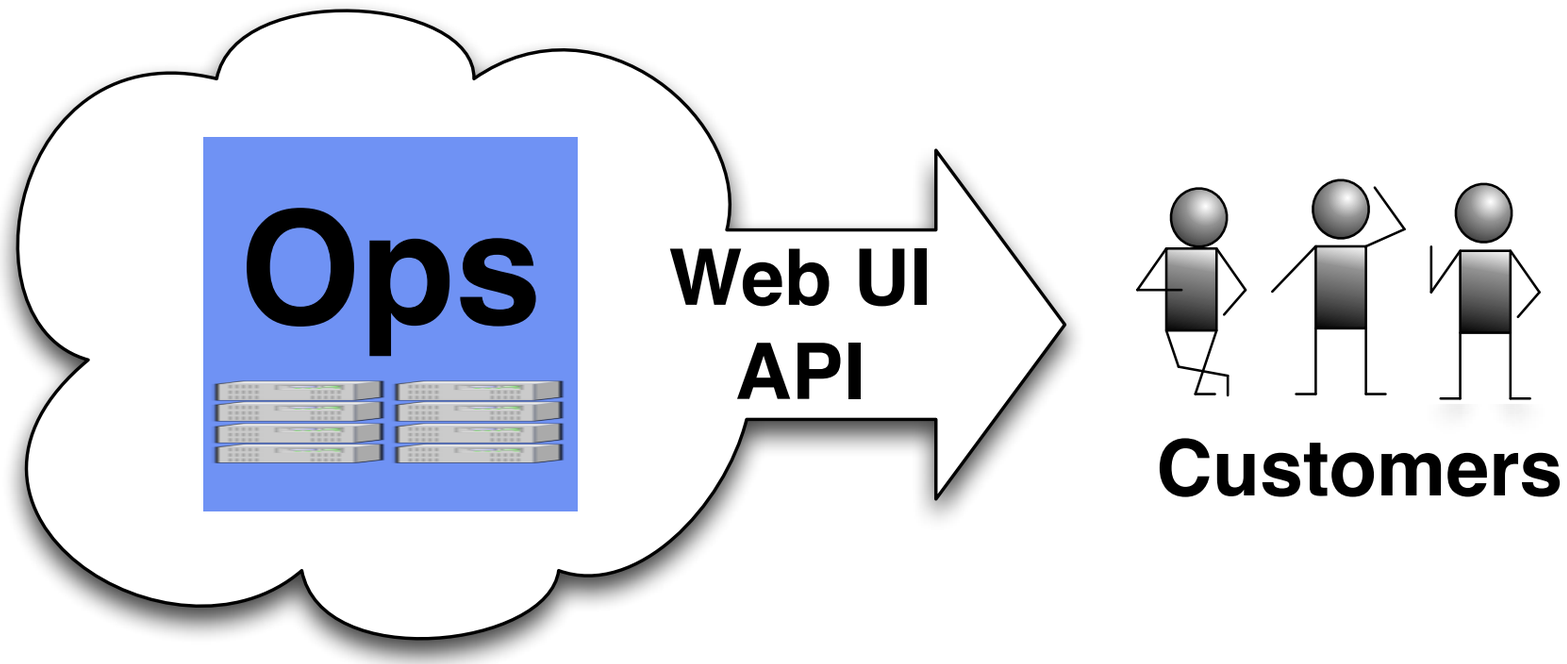
Replace request queues with self-service interfaces

- Fully automate what used to be done by humans
- Put behind self-service interfaces for on-demand consumption
- Benefits to Ops
    - Less time spent “doing”, more time adding value
    - Stop being the blocker

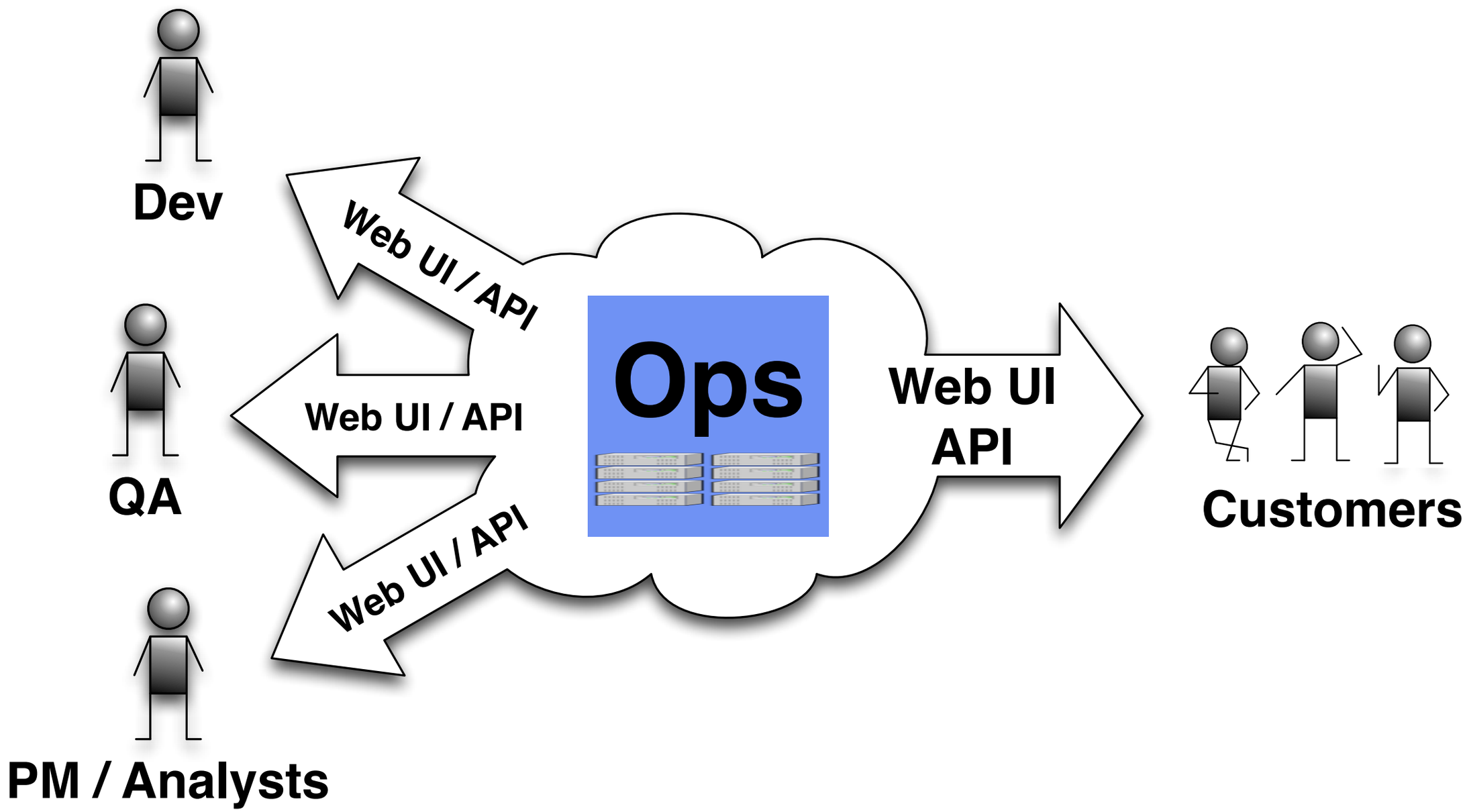
Replace request queues with self-service interfaces

- Fully automate what used to be done by humans
- Put behind self-service interfaces for on-demand consumption
- Benefits to Ops
  - Less time spent “doing”, more time adding value
  - Stop being the blocker
- Benefits to rest of organization
  - Decouple processes and avoid bottlenecks
  - Each team can move at their own pace
  - Cuts down on scrap and communication overhead
  - Enables a pull-based lifecycle

Service provider mindset is already familiar



Extend concept to internal interfaces as well

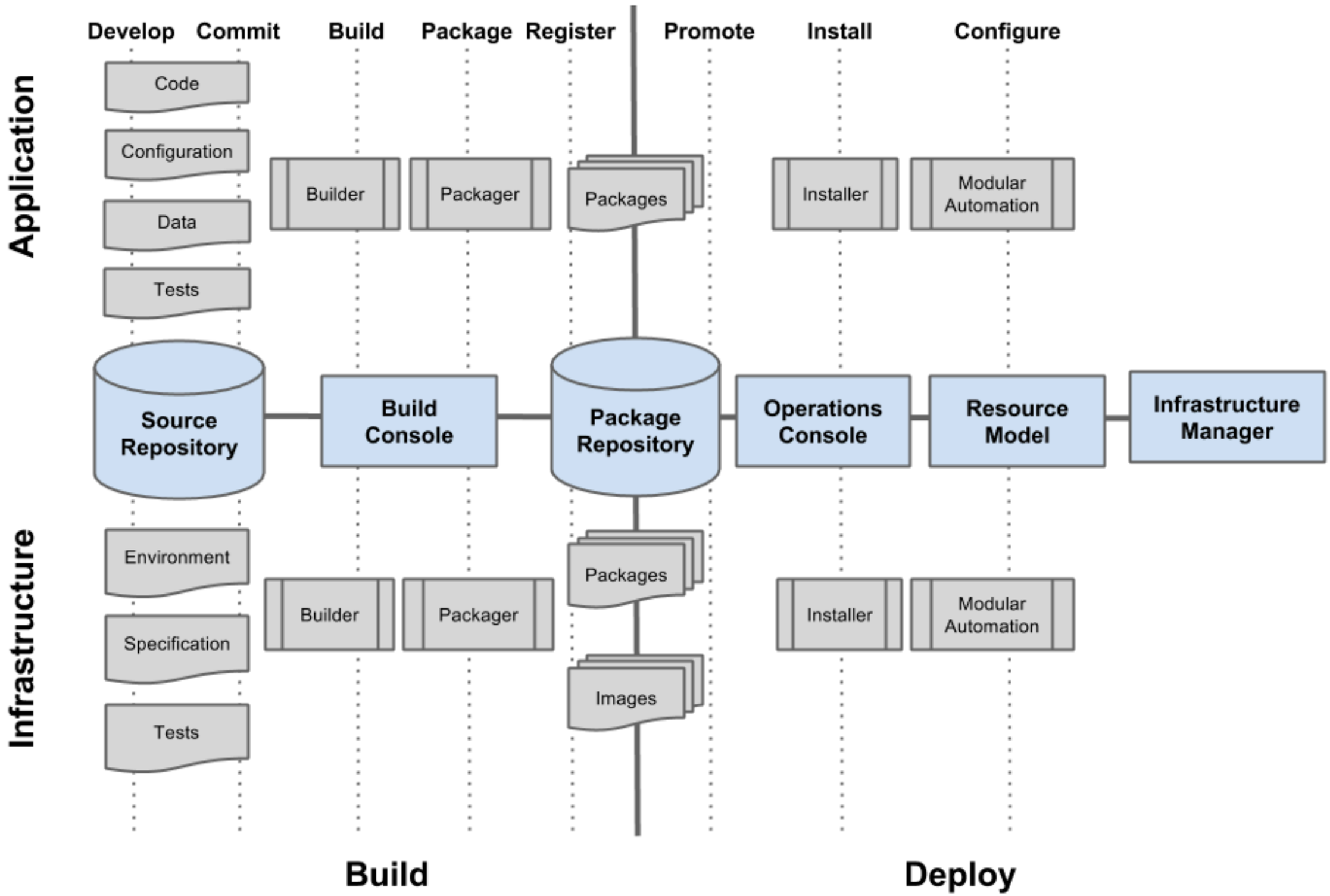


Why Ops will initially say no

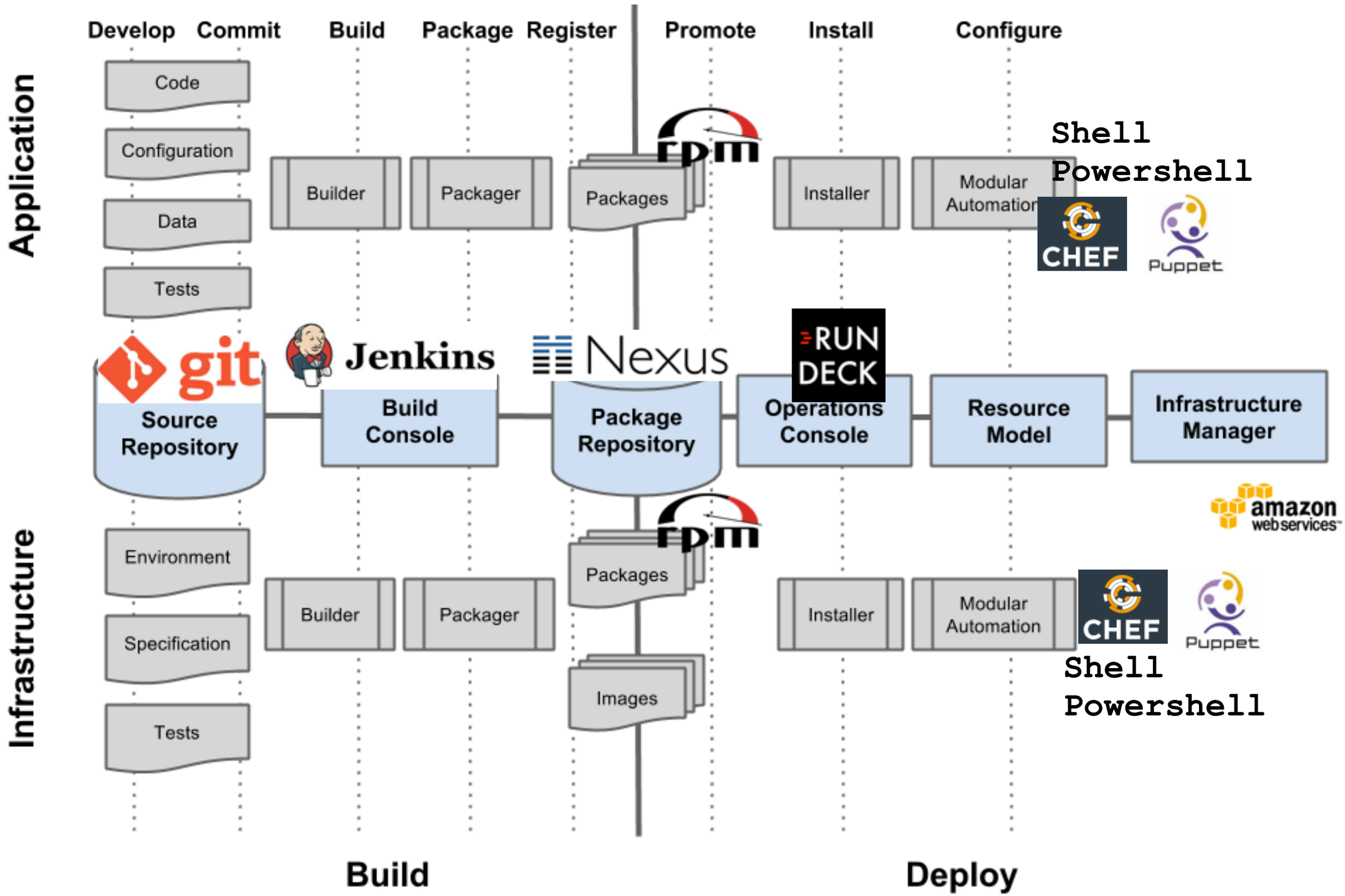
- Low confidence that new changes won’t break things
- Governance / Compliance
  - Auditing
  - Access Control
  - Accounting

...so lets show them what's possible step-by-step

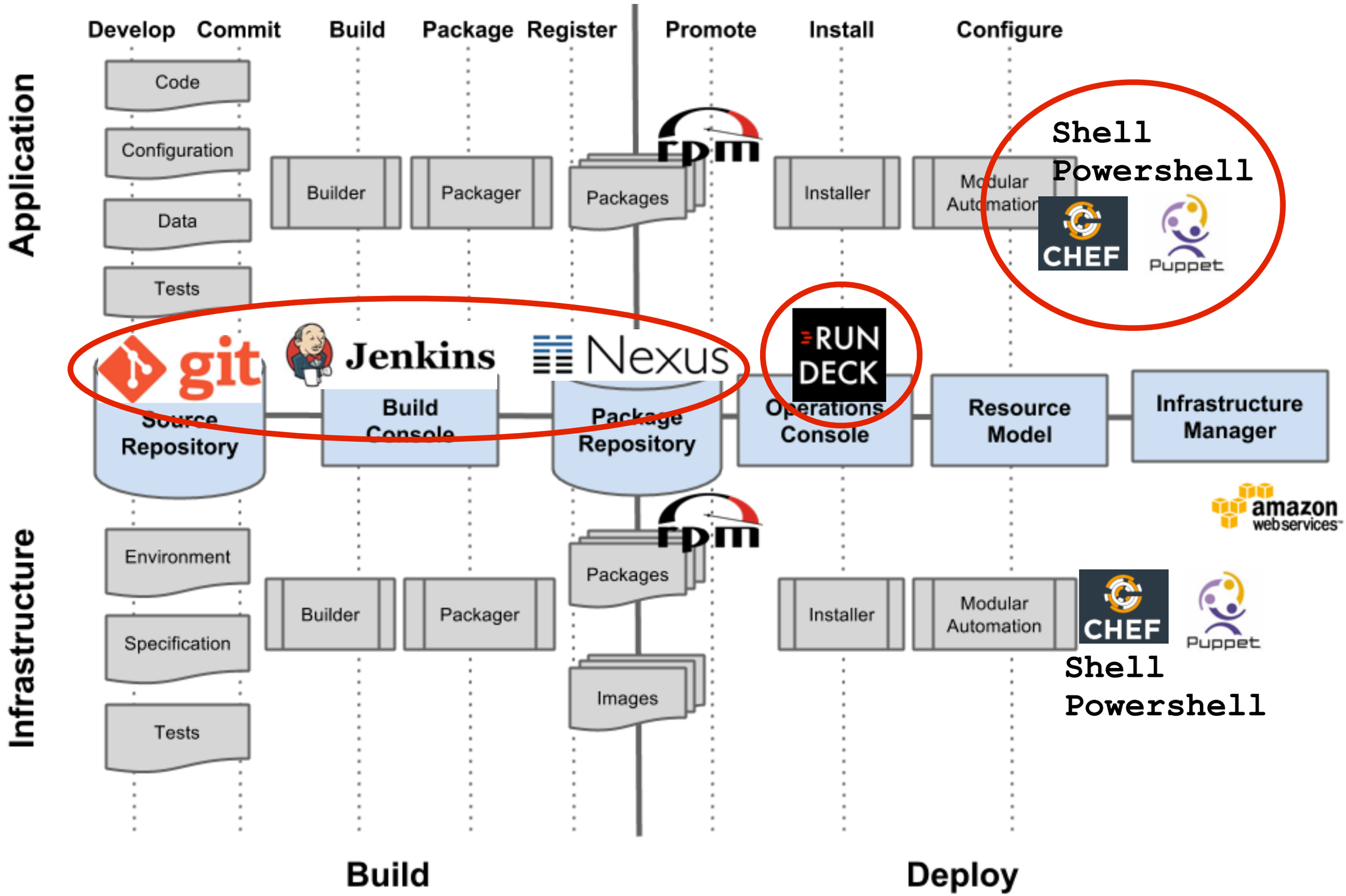
Fully automated, specification driven lifecycle



Example


Develop
Commit
Build
Package
Register
Promote
Install
Configure
Application
Code
Configuration
Data
Tests
Builder
Packager
Packages
Installer
Modular Automation
Shell
Powershell
CHEF
Puppet
git
Jenkins
Nexus
RUN DECK
Source Repository
Build Console
Package Repository
Operations Console
Resource Model
Infrastructure Manager
amazon webservices
Infrastructure
Environment
Specification
Tests
Builder
Packager
Packages
Images
Installer
Modular Automation
CHEF
Puppet
Shell
Powershell
Build
Deploy

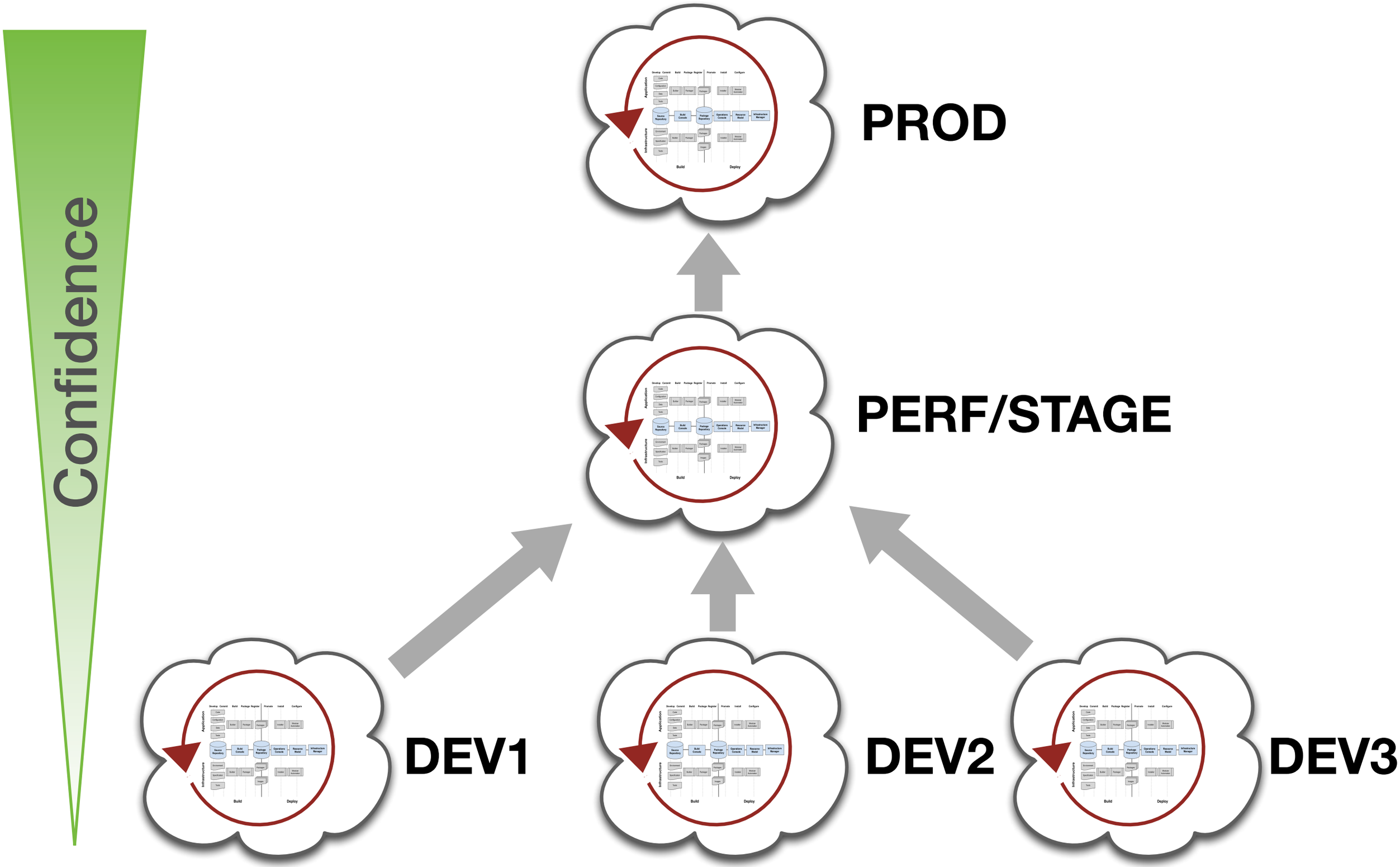
Example Governance / Compliance



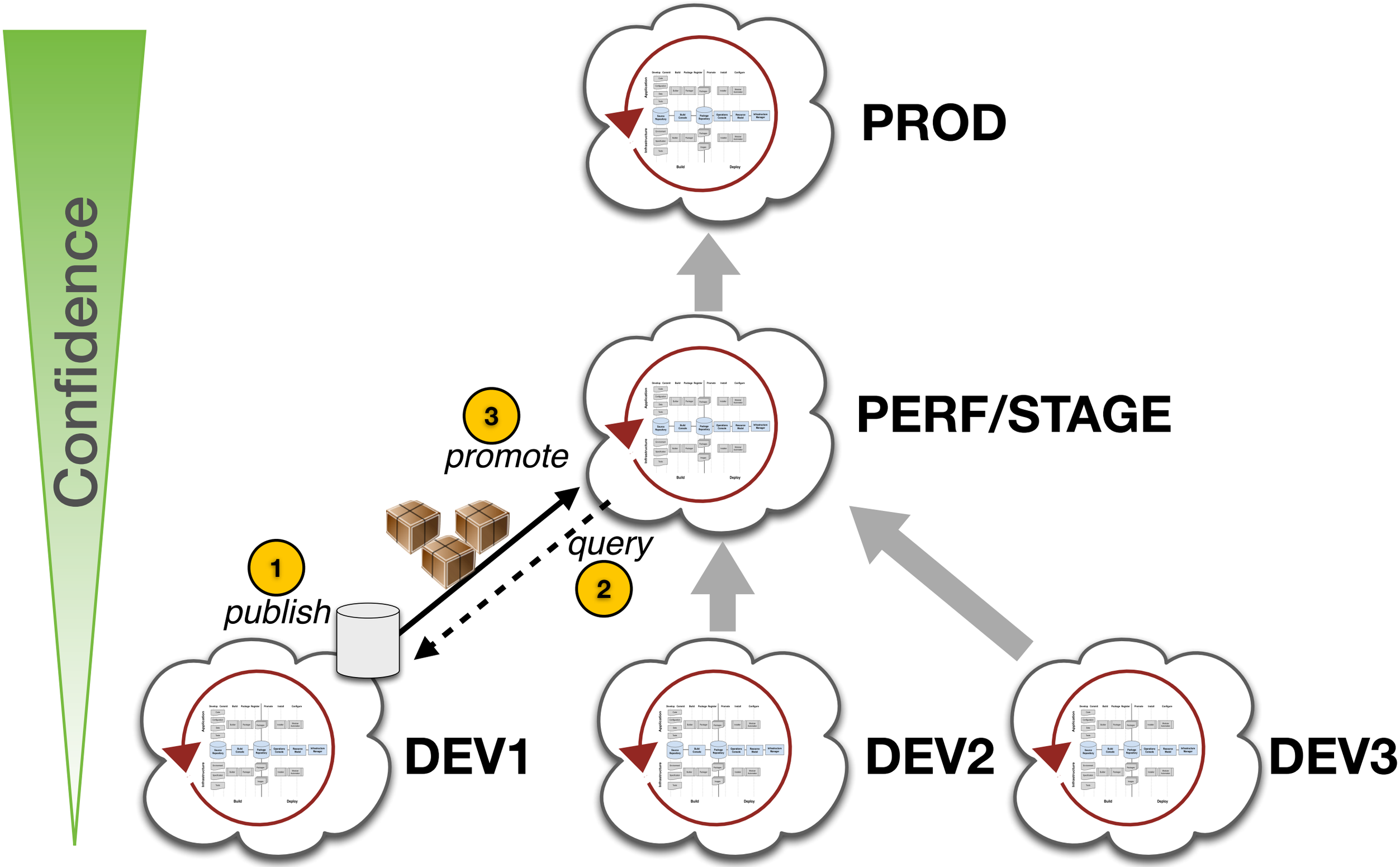
Mitigate quality and security risks

- Repeatedly “rehearse” all operations from earliest possible point in lifecycle
- Everybody should be deploying and testing with the same toolchain, automation, and tests
- QA and InfoSec provide standards and tooling that are used by Dev upfront

Build confidence from the start



Pull-based model to control promotions



Don’t forget to give it a name

Operations as a Service (OaaS)

Ticketless IT

Don’t forget to give it a name

Operations as a Service (OaaS)

More web / cloud friendly

Ticketless IT

Bigger impact with traditional enterprise IT

Dev initiated DevOps Transformation

1. Take an “operations first” mindset
2. Build organizational alignment
3. Establish a new model for working with Ops

Damon Edwards

@damonedwards

dev2ops.org

DevOps Cafe

http://www.dtosolutions.com

http://www.simplifyops.com

QCon

Please evaluate my talk via the mobile app!