W4156

Designing for Production

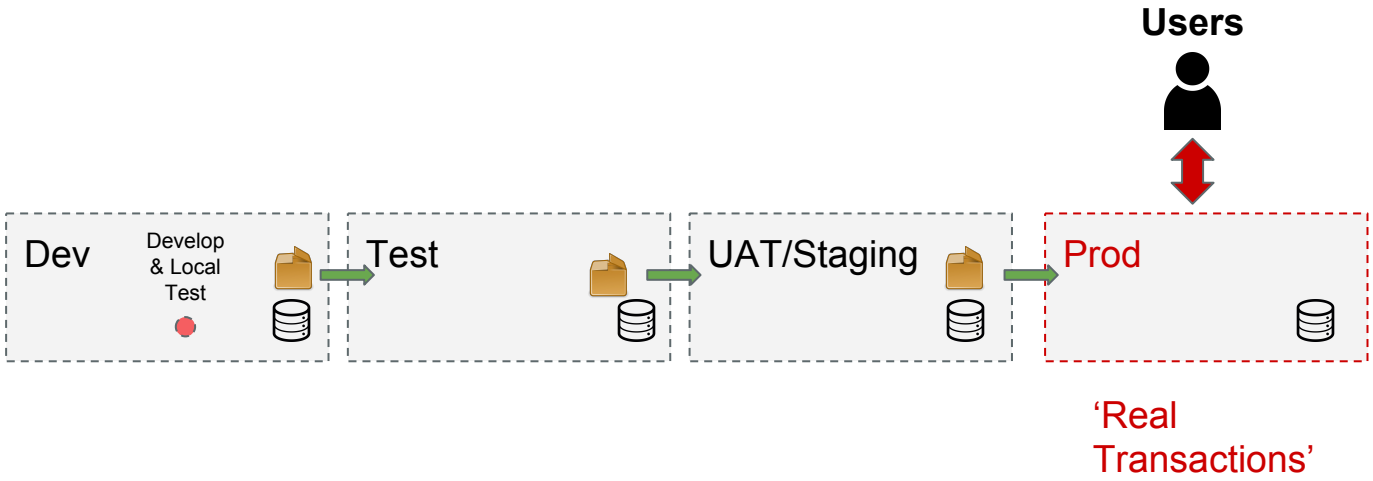
(A very seasoned engineering manager)
Hey, Ewan. I hear you are teaching at Columbia.
What's going on?

Yeah. I am teaching ASE. I would summarize it as the 'bridge' between how we learn to 'program': (alone, predefined problem, etc) to being prepared for fun of S.E. in industry (working out requirements, in teams, methodologies, CI/CD, testing, production, etc, etc)

"Ah. Nice. I think the ***production*** topic is critical. When I think back to transitioning from university to industry that was a huge huge shock: the fact my software runs 24x5, is deployed/released by others, has to be supportable and face all the things that can go wrong in prod was a shock. **The way I need to think as a software engineer and the responsibilities and discipline of writing production quality code.** I think it is remains one of the biggest shocks for graduates each year"

Completely agree. Am going to devote a lecture to it!

Environments Recap



Non-Prod environments may have users or developers behaving as users for purposes of testing. Not shown
Exact number and responsibility of environments can vary team to team

Production Environment

- Flow of value is idea to working software in hands of customer (↠ prod)
- Prod environment facing customers with real economic transactions
- Availability ***economically*** important and impacts your ***brand perception***

For the project thus far we did not add real world constraints:

1. Did not enforce prod 99.9N% uptime (or penalize for downtime!!)
2. *Including during* each upgrade from version N to version N+1

Agenda

- Getting code into production
- Who supports it when it is there?
- Is it working?
- Its broke. What do we do now?

Getting it to production

Key Terms

Change Request: Approval process to release new code/config into PRD

Deploy: The act of placing new binaries onto production hosts (but not necessarily running).

Release: Enabling users onto the new version of the software

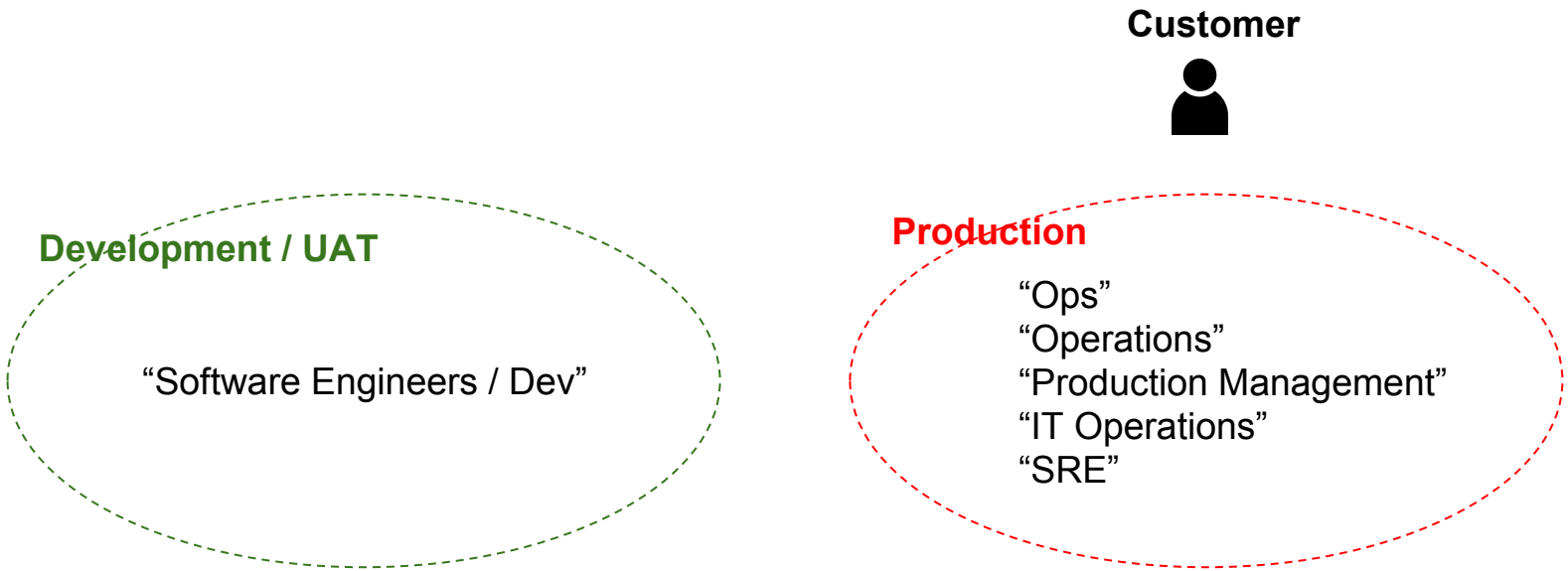
Rollback: Reverting to a previous version of code/config in event of issue

Architectures/approaches such as **Blue**/**Green** attempt to eliminate downtime, minimize risk of deploy/release and make rollback very rapid.

(I have spoiled you with your first introduction being end to end CD setup. Many/most teams are not CD)

Who supports it when it is in prod?

Who supports the service while I am here?



Different names (and operating models) but fundamentally an organization/team that *operate* the software produced by software engineers to run it as a service.

Who does what?

Environments:

Development	Production
● Run by ***Development***	● Managed by ***Operations***
● Incentivised to deliver ***new functionality***	● Incentivised by ***stability of the service***

Rules:

- 30 seconds simulate a month
- Features represented by a ‘ball’
- Developers release new features by placing ‘ball’ in production environment
- Production has two versions (balls) at any one time (which must be juggled to signify support)
- At the end of the experiment
  - Development paid by # of features from development to production
  - Operations paid by uptime of service

  What happens?

Dysfunctional Situation

Common meme denotes a ***dysfunction*** relationship between dev and ops

Out of scope but briefly:
SRE and DevOps are different ways of establishing ***culture, processes and tools*** between developers and operations to have ***functional*** relationship (features && availability)

Is it working?

Is your product working right now?

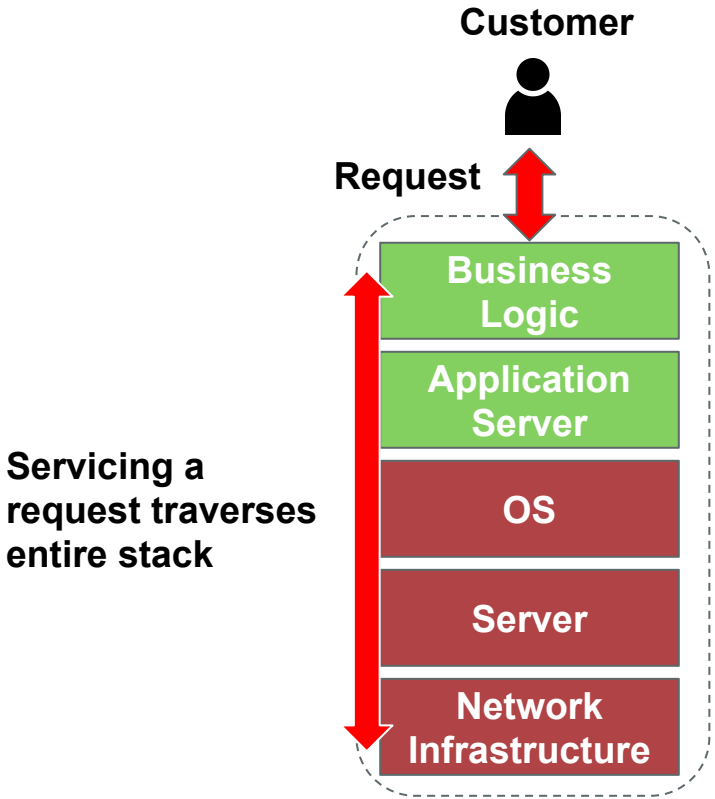
(irrespective of whether you are in lecture/sleeping/coding/travelling/whiteboarding/drinking coffee)

OR

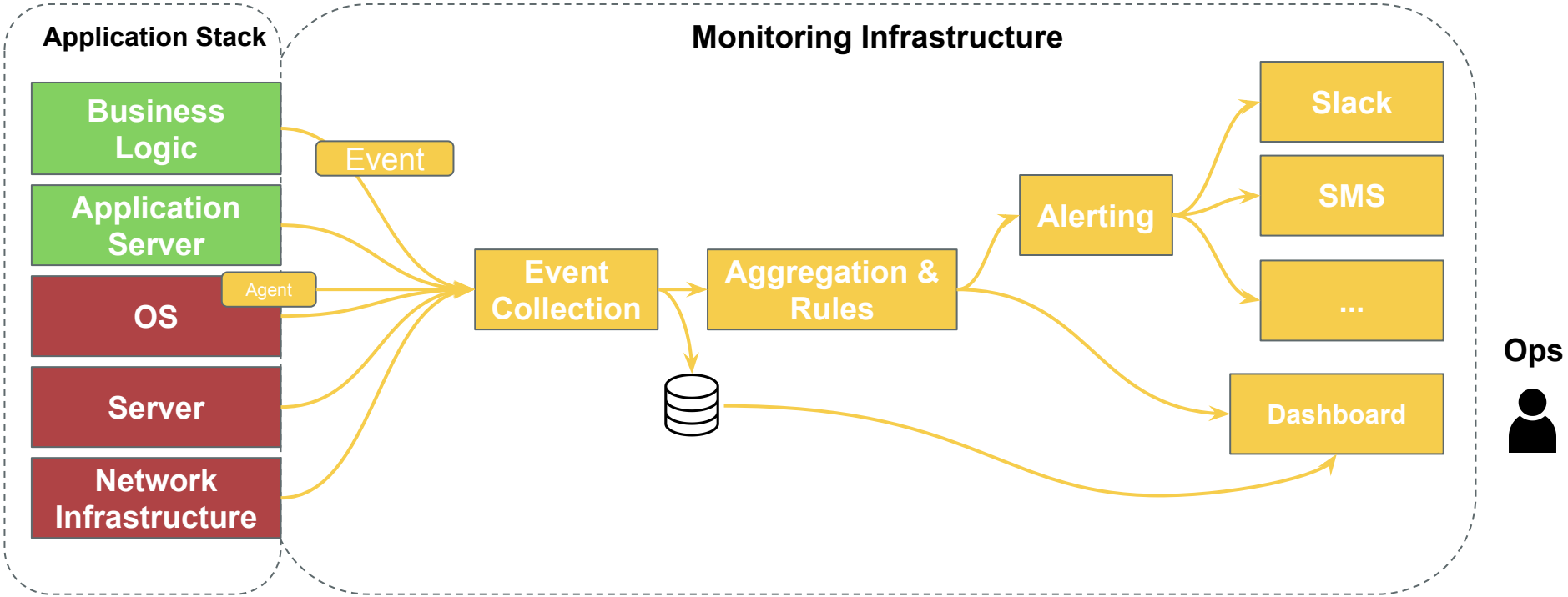
Do you find out when your customer tells you that your product is not working?

Is that embarrassing?

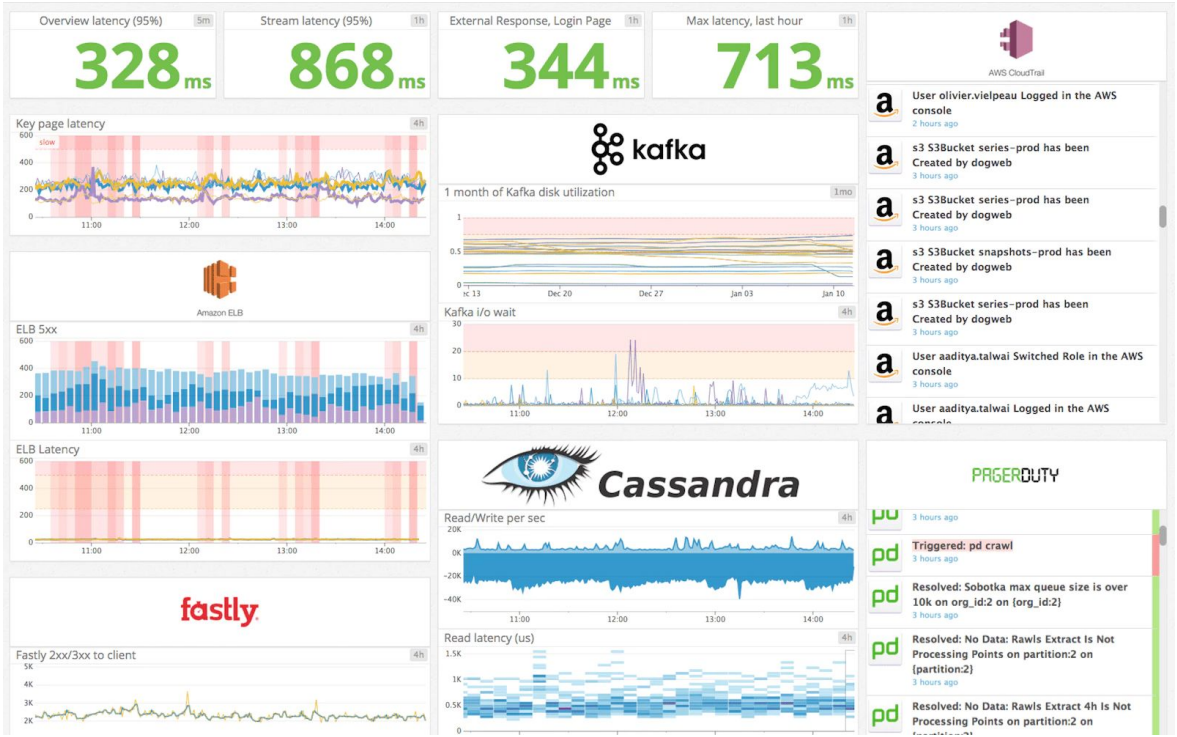
Conceptual Stack



Monitoring Architecture

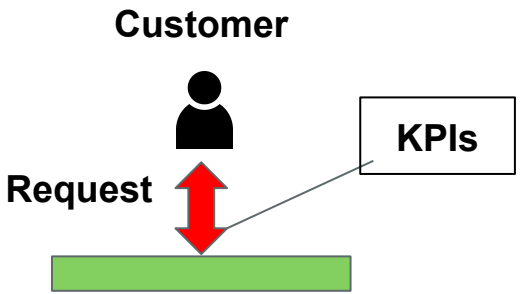


Monitoring Dashboard



Datadog

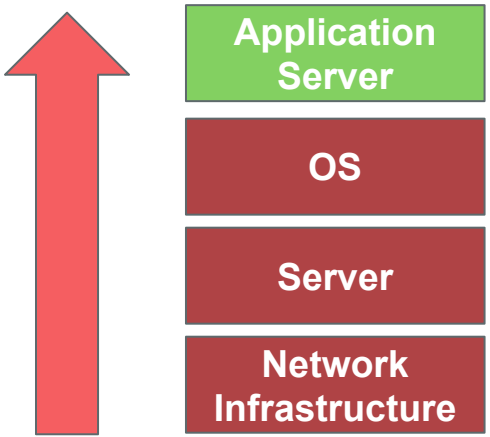
Approaches: Top Down



Product working ~=
working from the customer perspective
(proxied via defining KPIs)

Can you give some more specific examples of top down?
Can you guess some top down metrics for well known companies?
Can you think of pros/cons of top down?

Approaches: Bottom Up



Product working ~=
servers + CPU + disk + network +
no exceptions + app queue size

Can you give some more specific examples of bottom up?
Can you think of pros/cons of bottom up?

Top Down vs Bottom Up

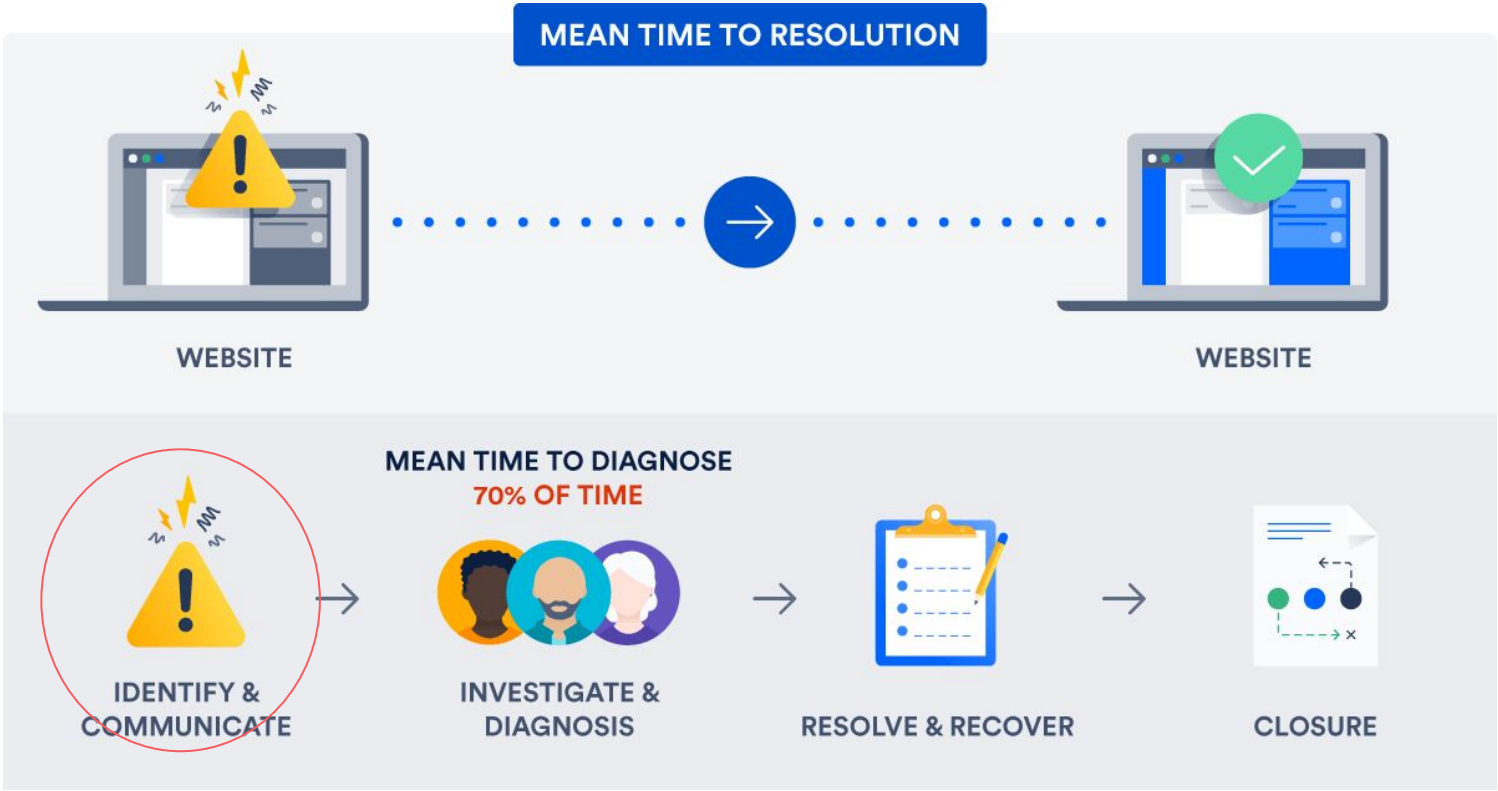
	Approach	
	Top Down	**Bottom Up**
Pros	• Lets you know customer experience / SLA is breached • Generally lower noise • Can be cheaper to implement	• Helps quickly diagnose causes of SLA breaks

Cons	• Says ‘something is wrong’ but doesn’t help isolate	• Noisy
		  ○ False negatives [1]
		  ○ False positives [2]
		• Costly/difficult to be exhaustive

[1] Application can hang but unless this manifest in an infrastructure metric then your app is non-responsive but no alert
[2] CPU can spike but it didn’t impact customers.
In my experience teams seem to focus on ‘bottom up’ rather than ‘top down’ despite the need for both and benefits of top down

Oh bless my buttons.
Product is down / US-East-1 is down / etc.

Activity I: Incident Management



Atlassian

Activity II: Post Mortem

Irrespective of design, testing, monitoring, preparation, etc - we will have outages (or near misses).

Post-mortem is a meeting facilitated by a senior engineer where we understand what happened and try to prevent ‘it’ and similar issues from recurring.

Format varies:

- What happened / impact?
- How did we respond?
- What was the root cause?
- Which factors enabled this to happen?
- What did we learn and what do we need to do to prevent?

Summary

The flow of value is idea to working software in the hands of the customer. Life does not end at commiting ‘done code’ it ends with a production environment.

This was a flavor/taster of issues related to running prod. There are approaches/ fields of study/methodologies including DevOps and SRE.

Key takeaways

- Be an engineer who understands the criticality of production
- Consider *supportability* in the design/architecture of systems
- Be responsible and don’t ‘chuck code over the fence’
- Post-mortems are about learning not blame

Reading	Optionality
Beginning S.E. Chapter 9	Required
Blue Green	Required
The art of monitoring chapters 1,2,3,9	Required
Netflix Views (focus on the key issue of them defining a top down metric vs the maths of that KPI)	Required
Incident Management 101	Required
SRE, SRE Post Mortem	Required
Blameless Post Mortems	Required
Post Mortem Facilitation Guide	Optional
Google Cloud Monitoring, Cloudwatch, Datadog	Peruse / Skim

(if you look at many texts they imply the 'end of the story' is software in the repo or magically released to users as if this is the ***end*** of the story. ***No! No! No!.*** The software is finished its gestational period but now lives in production! It must be supported, changed, incidents managed and post mortems conducted, learning lessons and work fed back into development.