

A CASE STUDY IN LARGE SCALE LEAN-AGILE ADOPTION

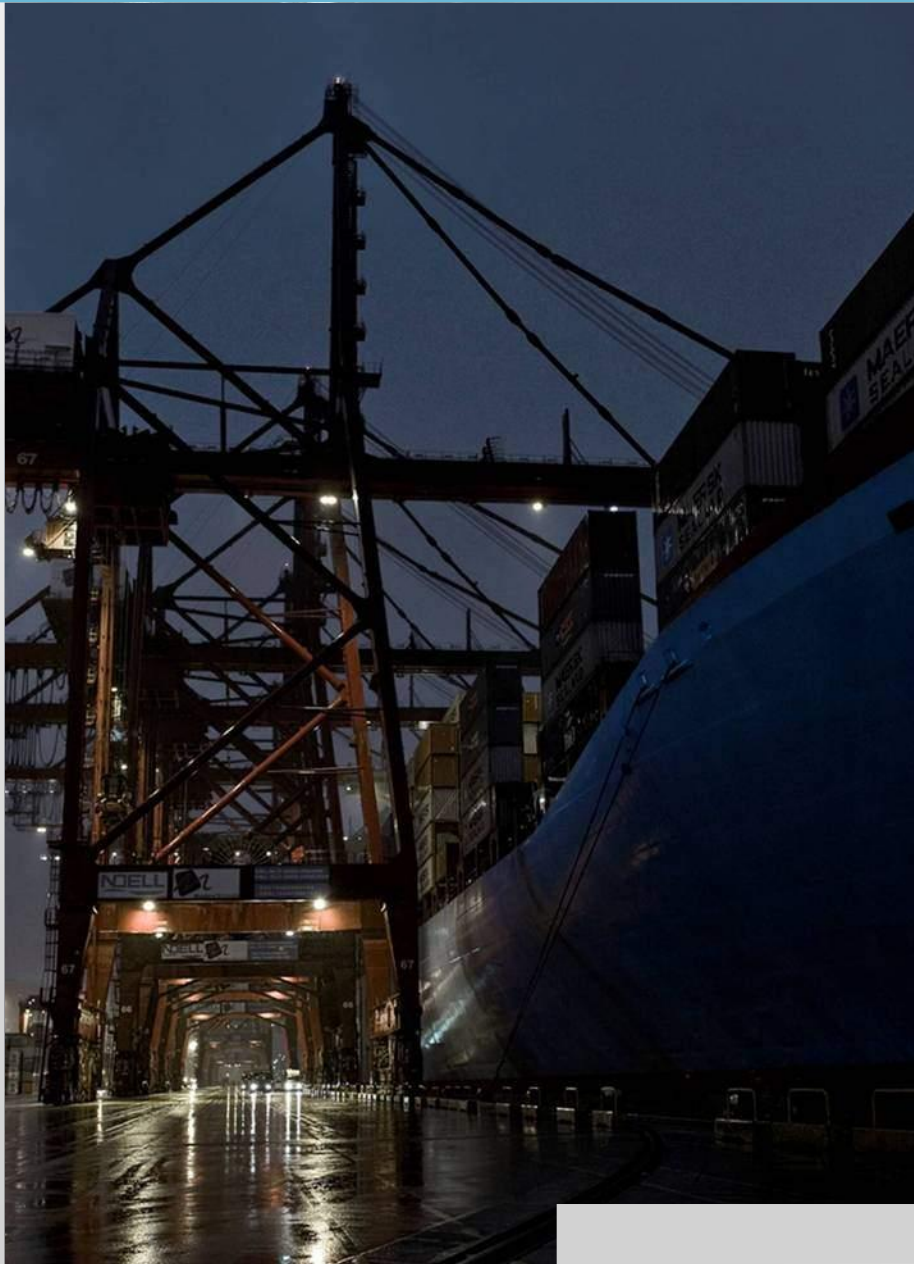
Chris Berridge
Maersk Line

About Maersk Line

- World's largest container fleet
- Truly global business
 - 325 offices in 125 countries
 - 25,000 employees (7,600 seafarers)
- 14.5% world market share [1]
 - 570 container vessels
 - Turnover \$26 billion [2]



[1] Source: Alphaliner Jan 2011
[2] Source: Annual Report 2011



Fragmented IT Landscape

- Thin outsourcing model
- Tier 1 vendors only
- 2,500 applications
- Core applications are tightly coupled
- 23,000 bookings/day

How we started our lean-agile journey?

New

Project, Platform, Team

Revolutionary



Existing

Project, Platform, Team

Evolutionary

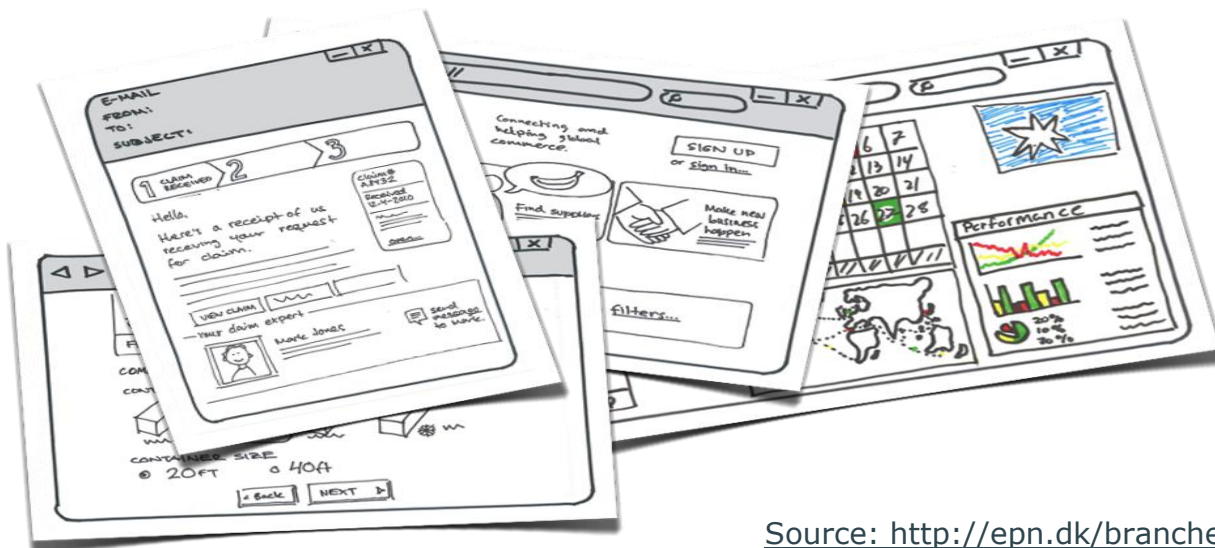


X-leap: The goal

Under Maersk Lines paraplystrategi - streamLINE - er der i værksat en række initiativer, der sikre at rederiet bliver endnu mere konkurrencedygtige gennem industriens bedste leveringssikkerhed, fortsatte CO2-reducerende initiativer og sidste men ikke mindst ved at sætte kunden i fokus

X-Leap er Maersk Lines største og vigtigste af disse programmer.

Formålet er at gøre det ligeså enkelt at booke en container hos os som en bog hos Amazon.com

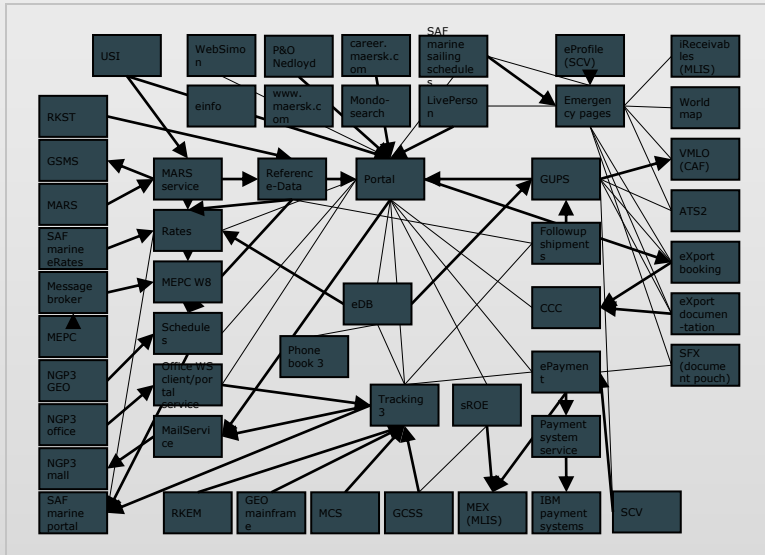


**Maersk Line CEO
(at the time)**



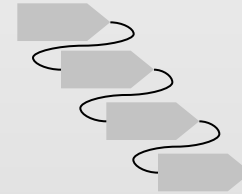
X-leap: How we sold agile to our stakeholders

Maersk is complex



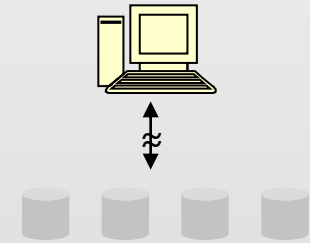
Two delivery approaches are common

1. Waterfall



No customer facing functionality for the first 18-24 months

2. Prototyping

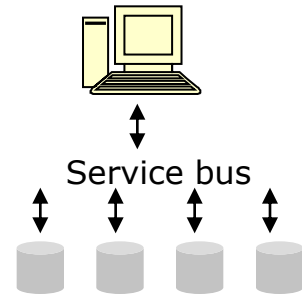


Lots of functionality early, but no connection to backend

Our approach is fundamentally different

Agile SOA

Minimal set of customer facing functionality delivered with true backend connections as early as possible (in our case 9-10 months)



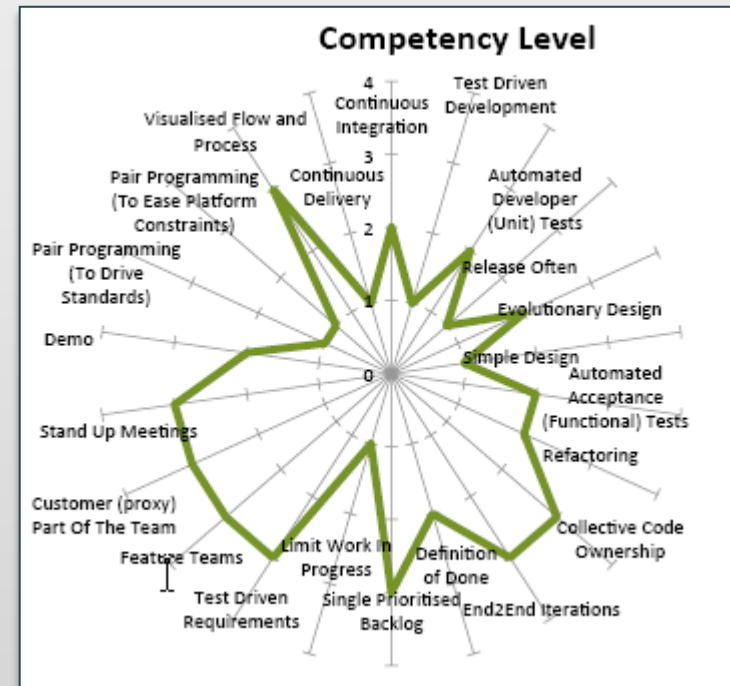
- 100's of backend systems
- Convoluted and unstable application architecture
- Inconsistent master data
- High product complexity
 - More than 20 000 lines in some contracts
 - More than 500 commodity types

X-leap: What we got right from the outset

- Strong customer focus
 - Clear customer experience vision created
- Co-location
- Shared Key Performance Indicators for whole team
- Onboard experienced people
- Willingness to experiment with new approaches
- Great senior leadership support

X-leap: 22 practices we (now) know that need to master

- Visualised Flow and Process
- Continuous Delivery
- Continuous Integration
- Test Driven Development
- Automated Developer (Unit) Tests
- Release Often
- Evolutionary Design
- Simple Design
- Automated Acceptance (Functional) Tests
- Refactoring
- Collective Code Ownership
- Definition of Done
- End2End Iterations
- Single Prioritised Backlog
- Limit Work-in-Progress
- Test Driven Requirements
- Feature Teams
- Customer (proxy) Part Of The Team
- Stand Up Meetings

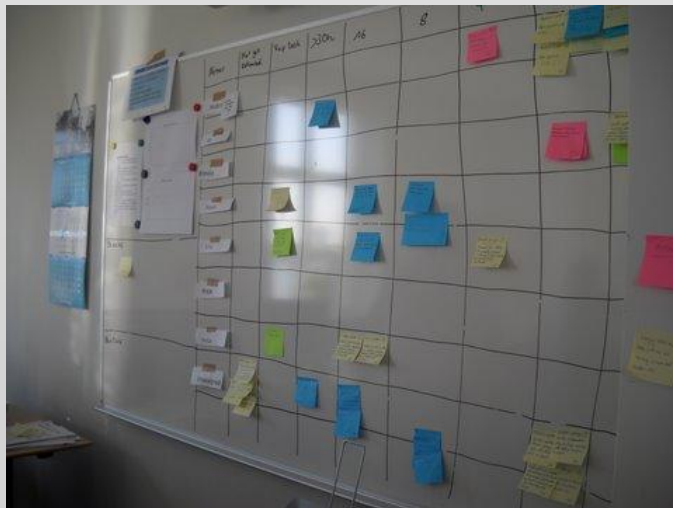
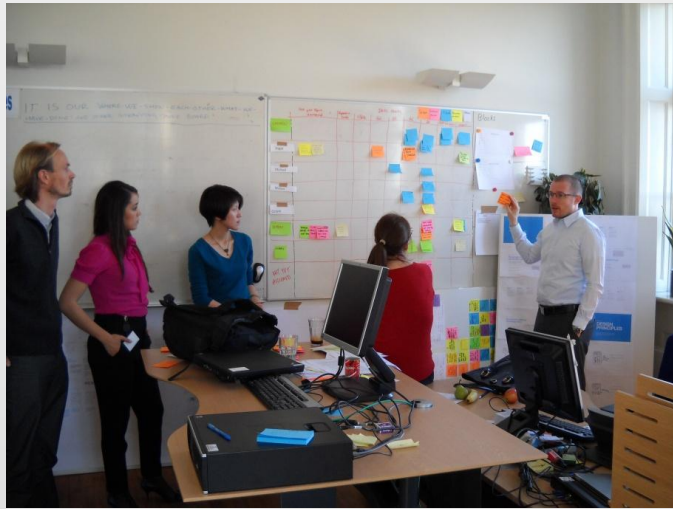


- Demo
- Pair Programming (To Drive Standards)
- Pair Programming (To Ease Platform Constraints)

Continuous Delivery Compliance Matrix

Acceptance Criteria	Compliance Level			
	None/ Never	Some/ Sometimes	Most/ Usually	All/ Always
All Code, Configuration, Test scripts, database schema/data migration etc is source controlled.				
All platform config, application config, database schema/data migration is automated.				
Any environment can be built from scratch on demand.				
All environments from development/test stations through to production are built using the same automated process.				
Any single check-in can pushed through all environments with single-click promotion at each stage.				
Binaries are only built once (outside local dev/test builds) before being pushed to subsequent delivery stages.				
All build artifacts can be traced directly back to all source controlled assets that were used to build them.				
There should be an audit log of builds and test runs.				
Source control branching is not used to create long lived branches.				

X-leap: A feature team in action



X-leap: Learnings within team

Manage requirements

- Prioritise effectively between functional & non-functional requirements
- Break down requirements and agree on what size is appropriate
- Need a process vision to support a customer experience vision

Iteration 0 is surprisingly large

- e.g. Reducing hardening phase took forever

X-leap: Value stream analysis for a feature



VALUE STREAM MAPPING

Process: User Story Development Lifecycle



Story: XL2806

Team: Purple

Title: Route details display

Sprint: Delphi 2

Unit of Time measurement: 1 Day

Findings:

1. Total cycle time from when the story is put into play unto the time it is in PROD = 68 days
2. Out of these there are approx 9 days of VALUE ADDING ACTIVITIES, 11 days of VALUE ENABLING ACTIVITIES & 48 days of NON VALUE ADDING ACTIVITIES
3. This results in a Process Efficiency of only 13%
4. The cycle time for the story from 'Done' to 'Delivered' is 56 days

X-leap: Root cause analysis for why hardening phase takes so long



X-leap: Learnings within team

Manage the change

- Engage advisors who focus on optimising the whole
- Own and manage practice adoption progress

Minimise thrashing

- E.g. Struggle to measure velocity due to constant changes

X-leap: Learnings outside team

Stakeholders need careful management

- Reluctant to exchange predictability for speed
- Difficult to explain refactoring & technical debt
- High expectations of delivering fast

Dependencies external to the development team are a headache

- Feature teams help but are no silver bullet
- There's no replacement for good project management to identify and manage external dependencies
- Others have to change their working practice (architects, infrastructure, other applications)

How we are completing the lean-agile journey.

New

Project, Platform, Team

Revolutionary



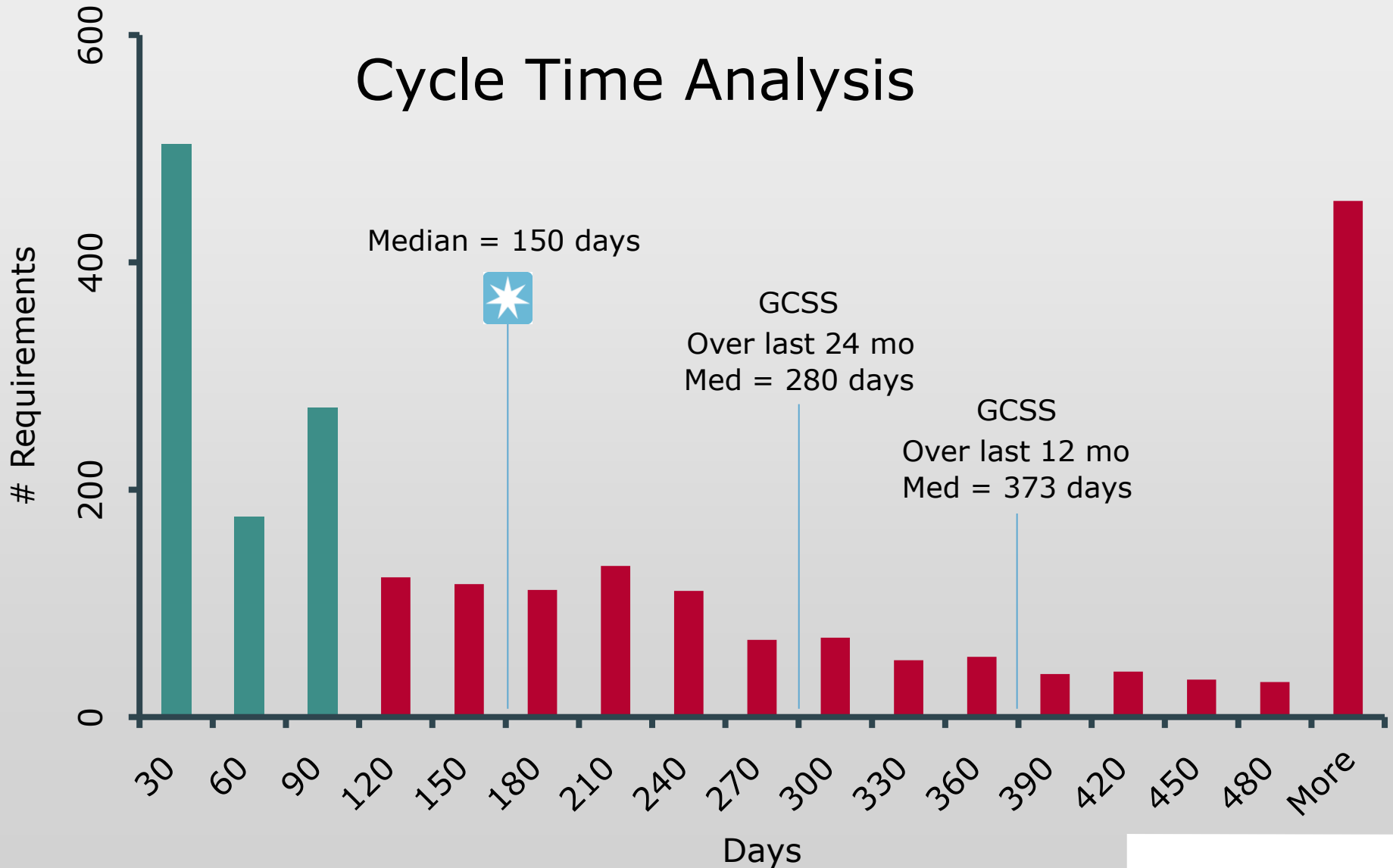
Existing

Project, Platform, Team

Evolutionary

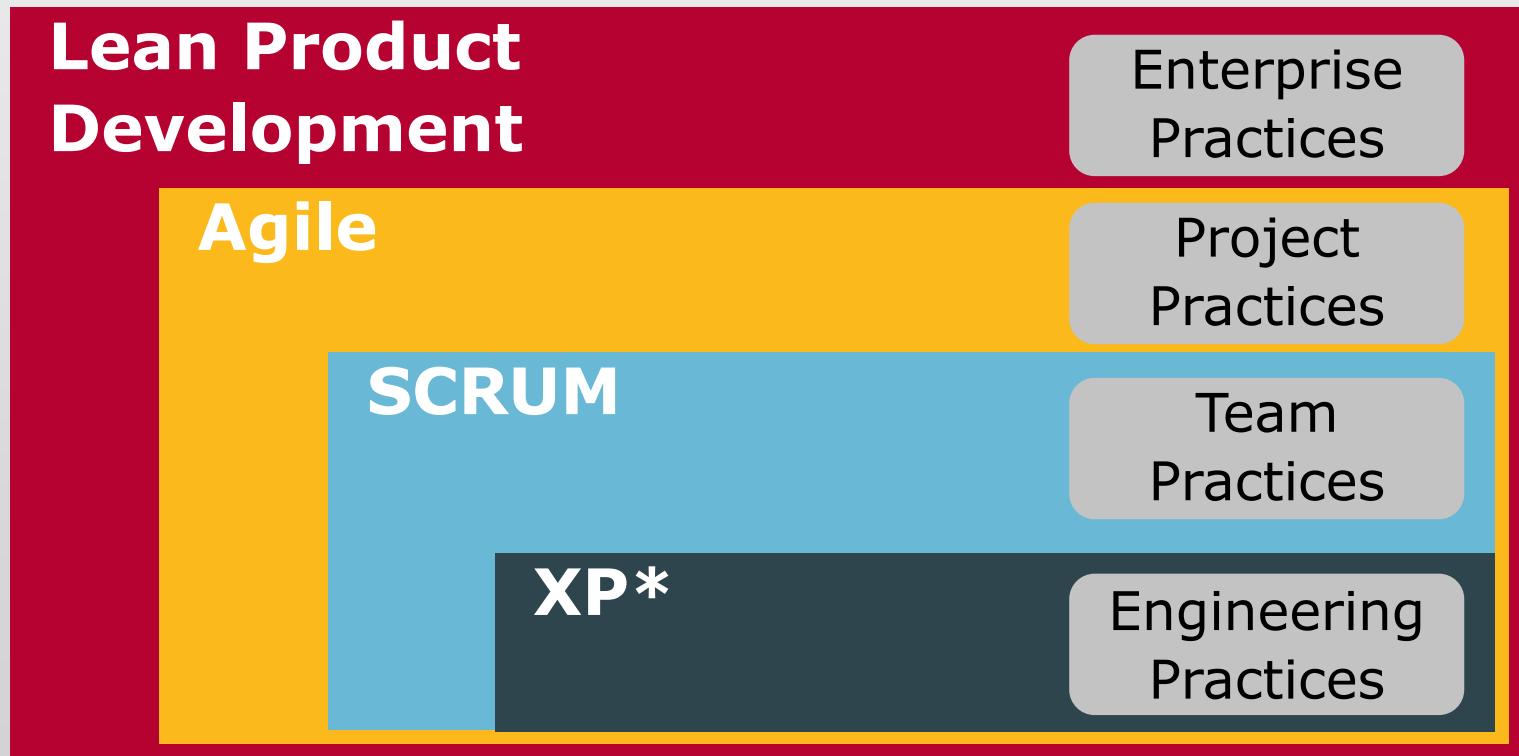


Cycle Time Analysis



Source: Focal Point – requirements that have been put into production over the last 2yrs, measured from date of creation to when set to working-in-production

Framing the methodologies



* Extreme Programming

The Starter Pack: 8 selected practices

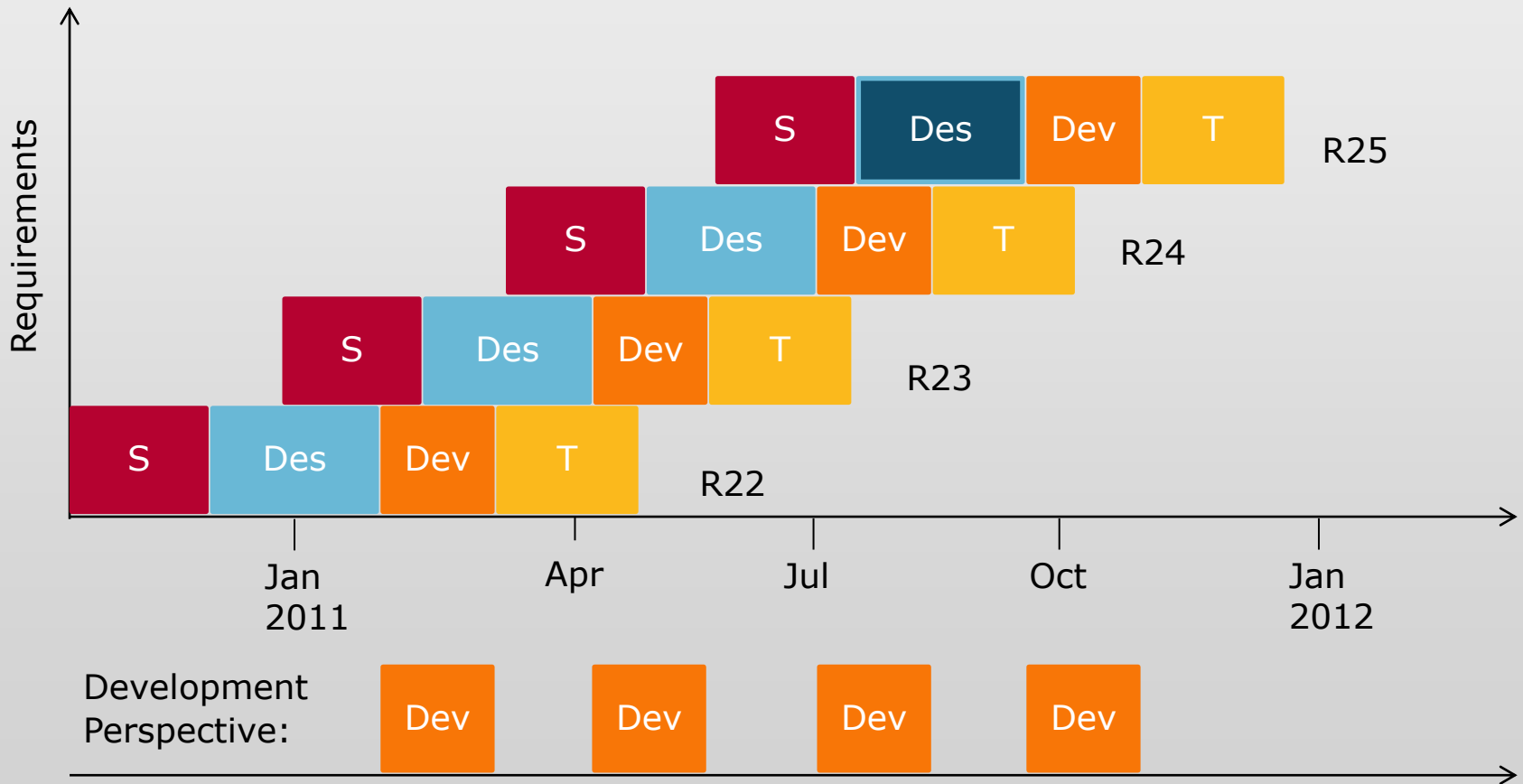
*Starter Pack** *Suitable for all applications*

1. Get to initial prioritisation faster
2. Improve prioritisation
3. Pull Requirements from Dynamic Priority List
4. Reduce size of requirements
5. Get to the point of writing code quickly
6. Actively manage Work-In-Progress (WIP)
7. Enable Faster Feedback
8. Enable more frequent releases

**Not
Scrum!*

GCSS: Release Frequency

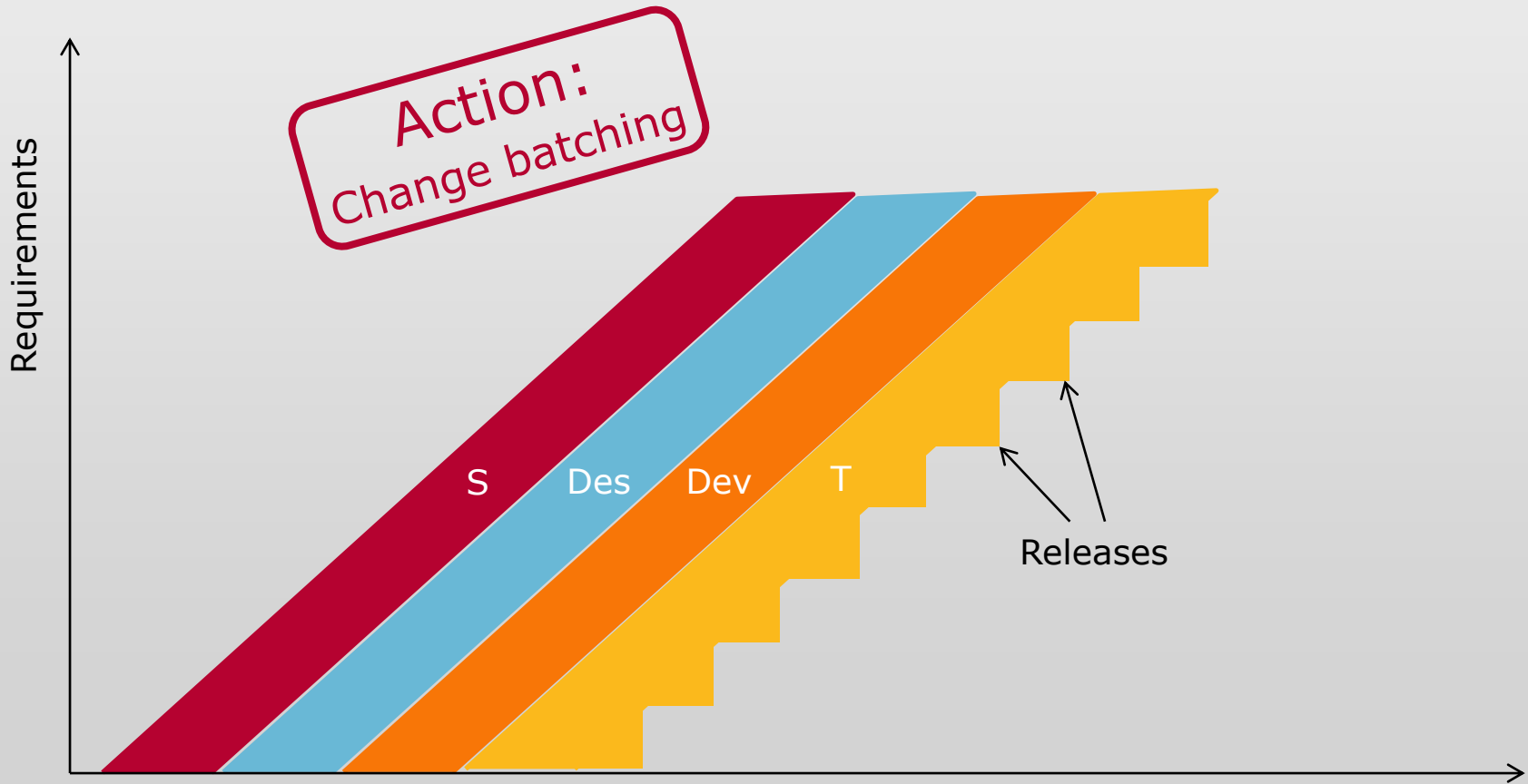
The effect of creating large release batches upstream



Estimated ~10,000 hours of idle time in 2010

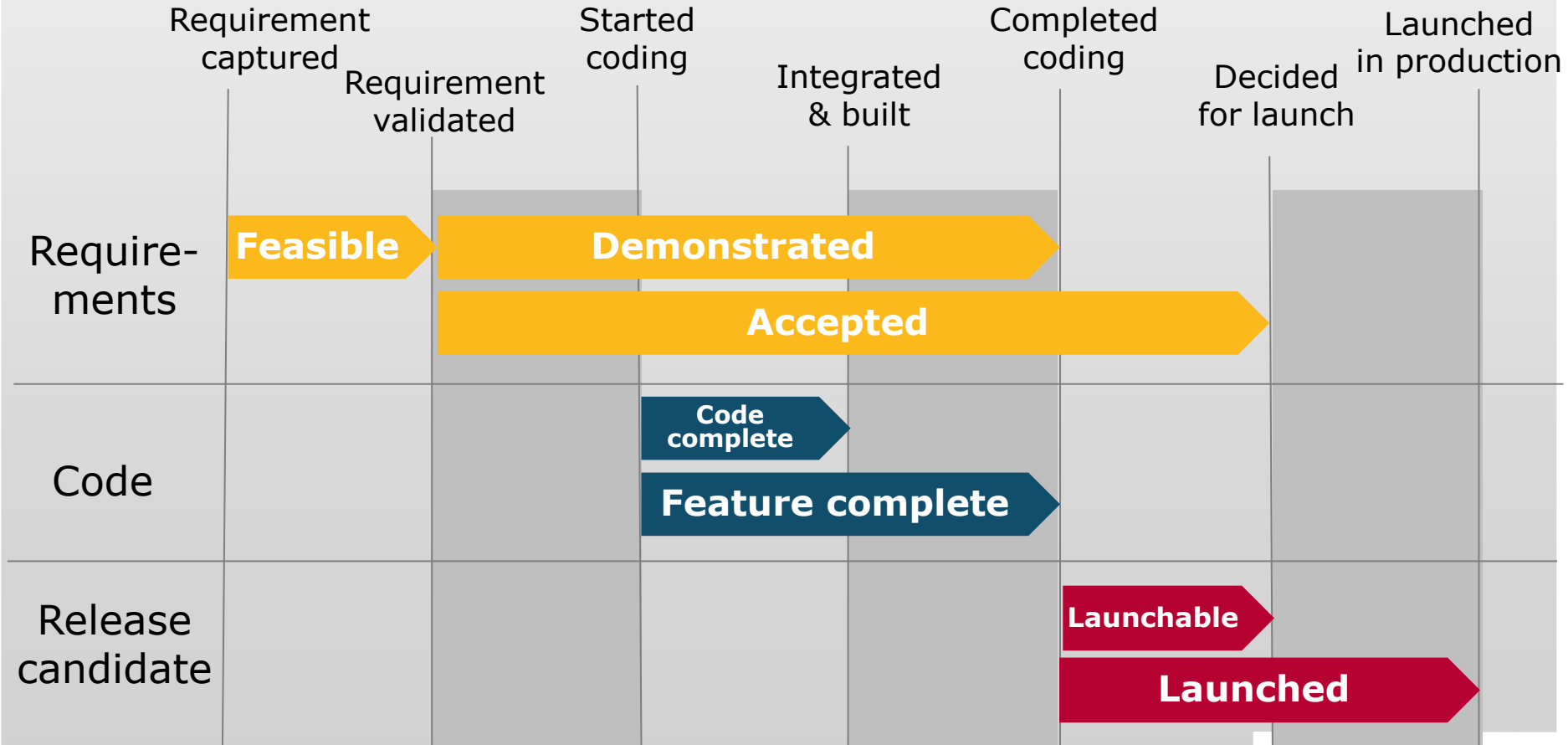
GCSS: More Frequent Releases

Enable the smooth flow of requirements



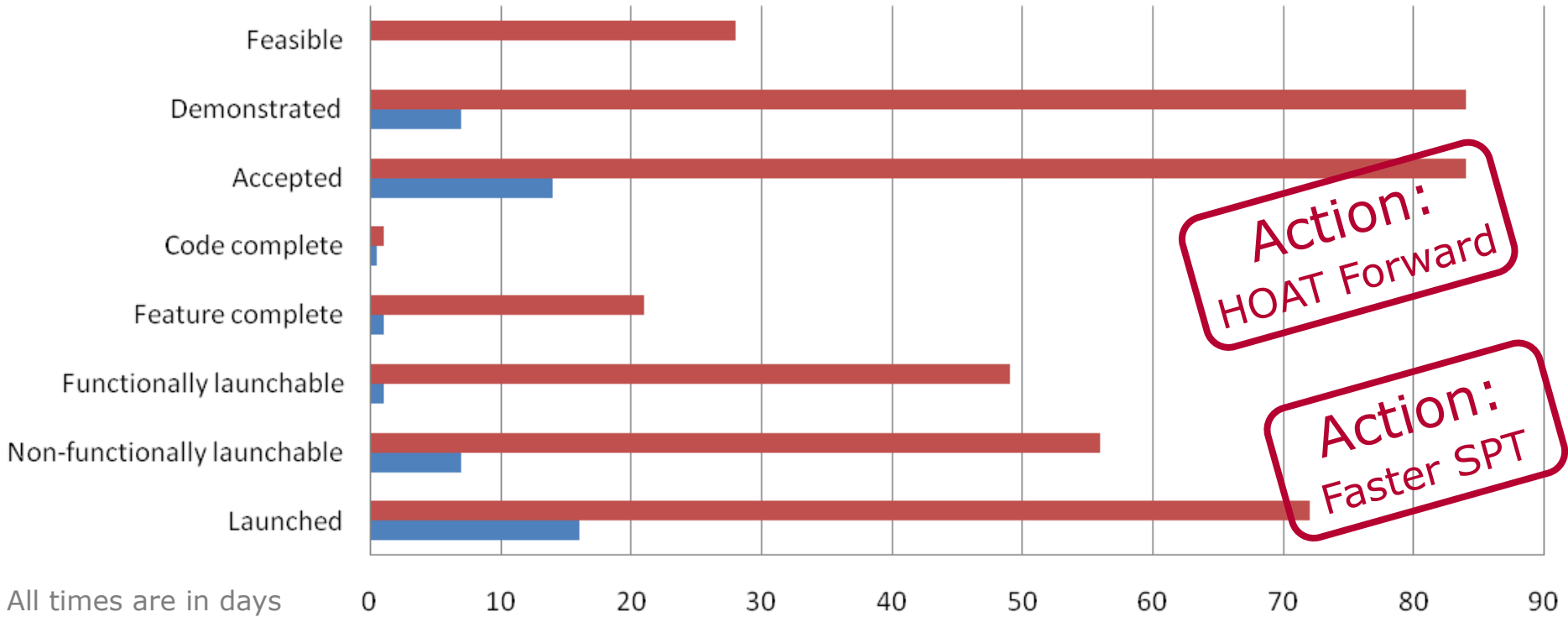
Faster Feedback

Eight Standard Measures



Faster Feedback

Comparing GCSS with the X-leap on the Eight Measures



**Action:
HOAT Forward**

**Action:
Faster SPT**

	Launched	Non-functionally launchable	Functionally launchable	Feature complete	Code complete	Accepted	Demonstrated	Feasible
■ GCSS	72	56	49	21	1	84	84	28
■ X-Leap	16	7	1	1	0,5	14	7	

**WIP LIMIT of 8
on bottleneck**



Daily standup

Changes to
how code is
handled

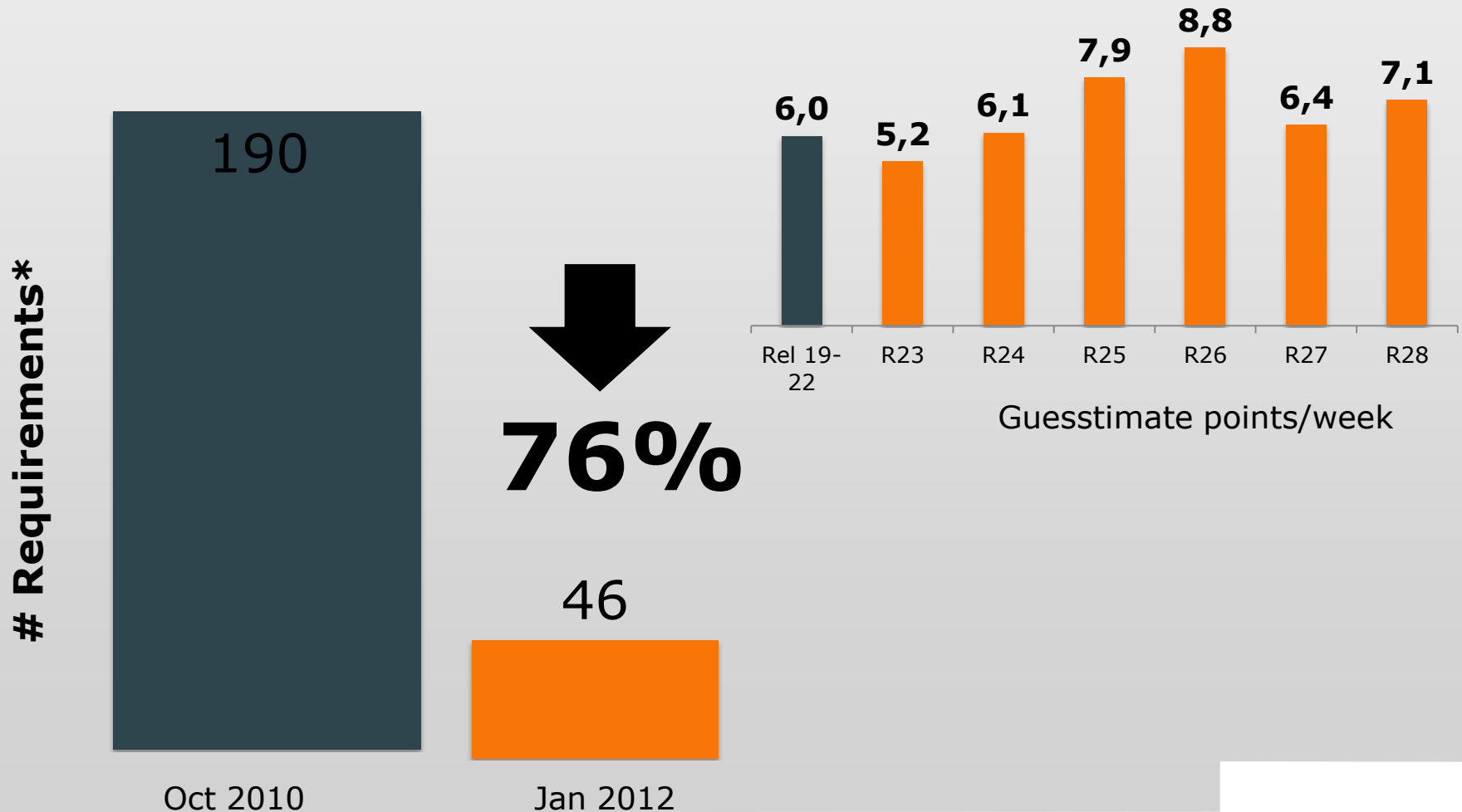
Streamline
impact
assessment

Improve
error tracing

GCSS: Actively Manage Work-in-Progress

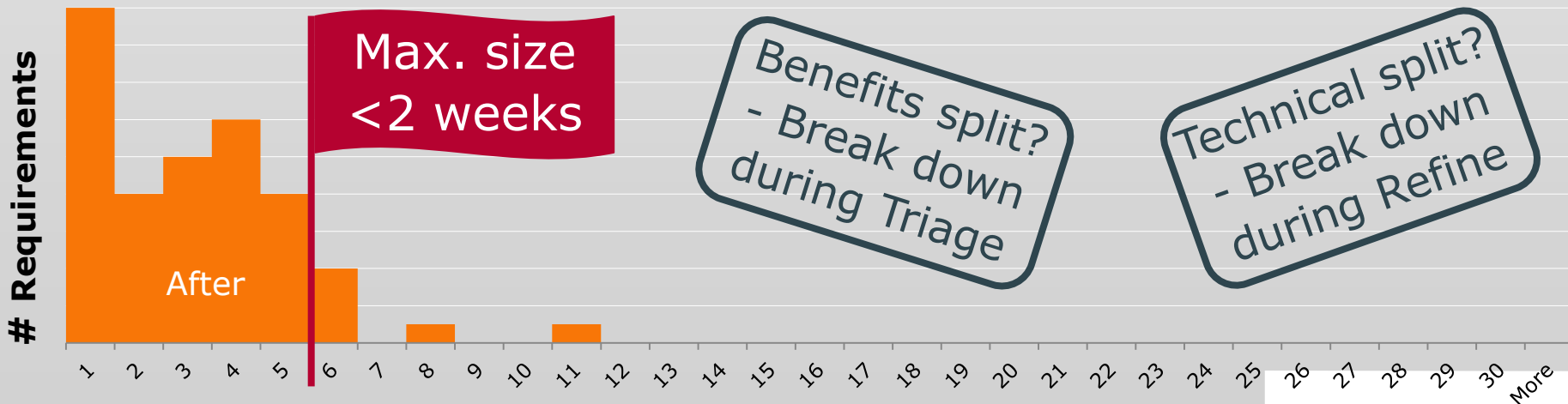
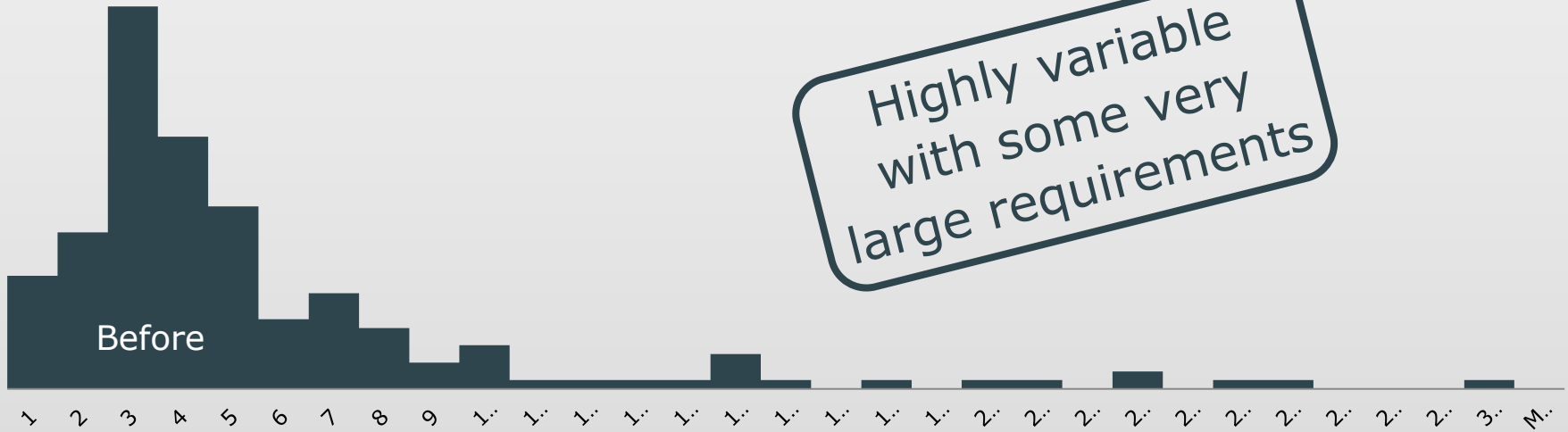
GCSS: Work-in-Progress reduced

...whilst at least maintaining throughput



*"Authorized" to "Launched"

GCSS: Requirement size variability

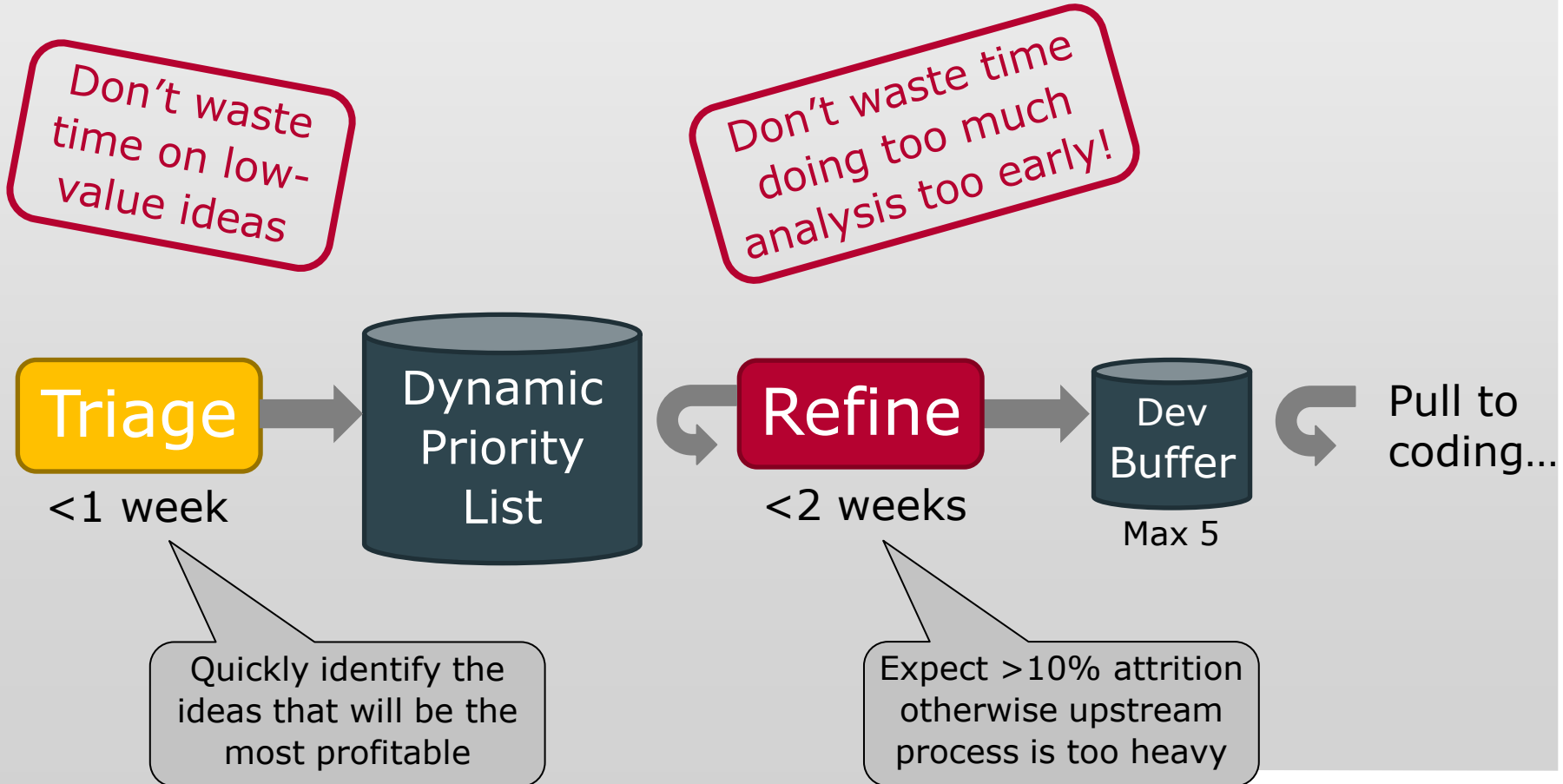


Guesstimate Points

GCSS: Standardized Upstream Process

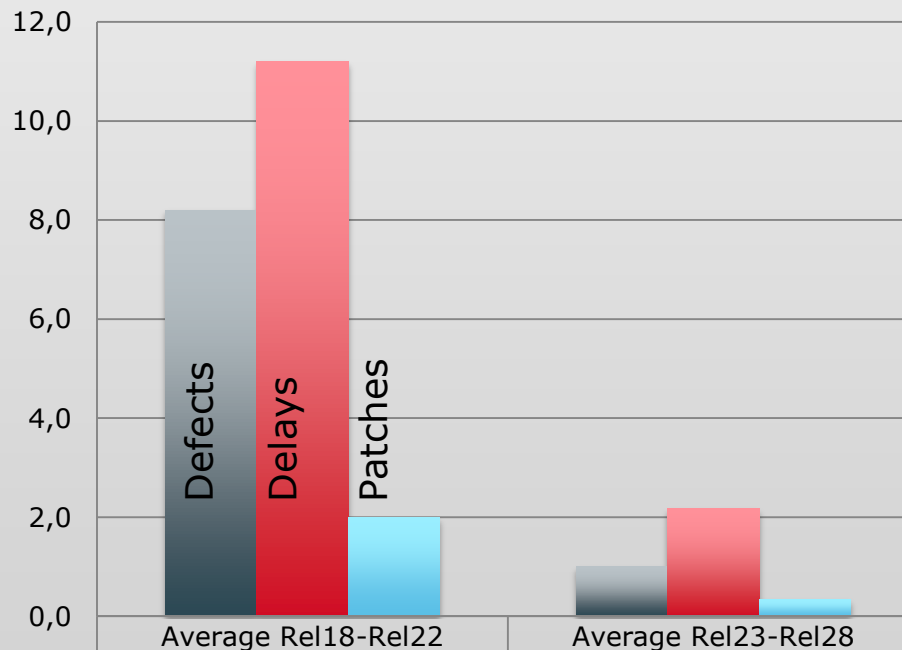
Get to initial prioritisation faster

Get to point of writing code quickly



GCSS: Quality improvements

Releases 2010-2011



■ E1+E2 Defects raised in HOAT	8,2	1,0
■ Production slippage (in days)	11,2	2,2
■ Patches 2wks after Prod	2,0	0,3

Up to June 2011

Since July 2011



-88%

Defects

-80%

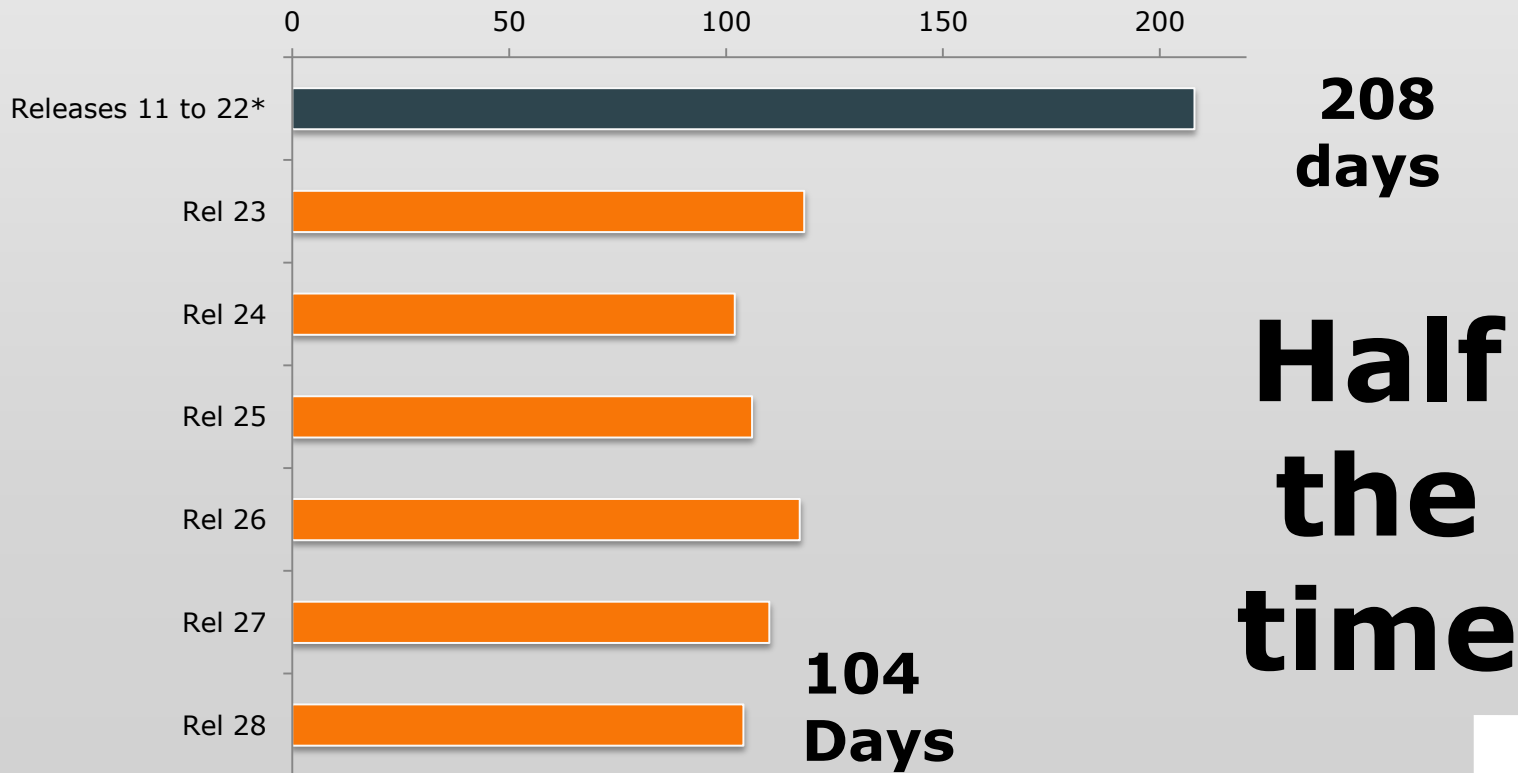
Delays

-85%

Patches

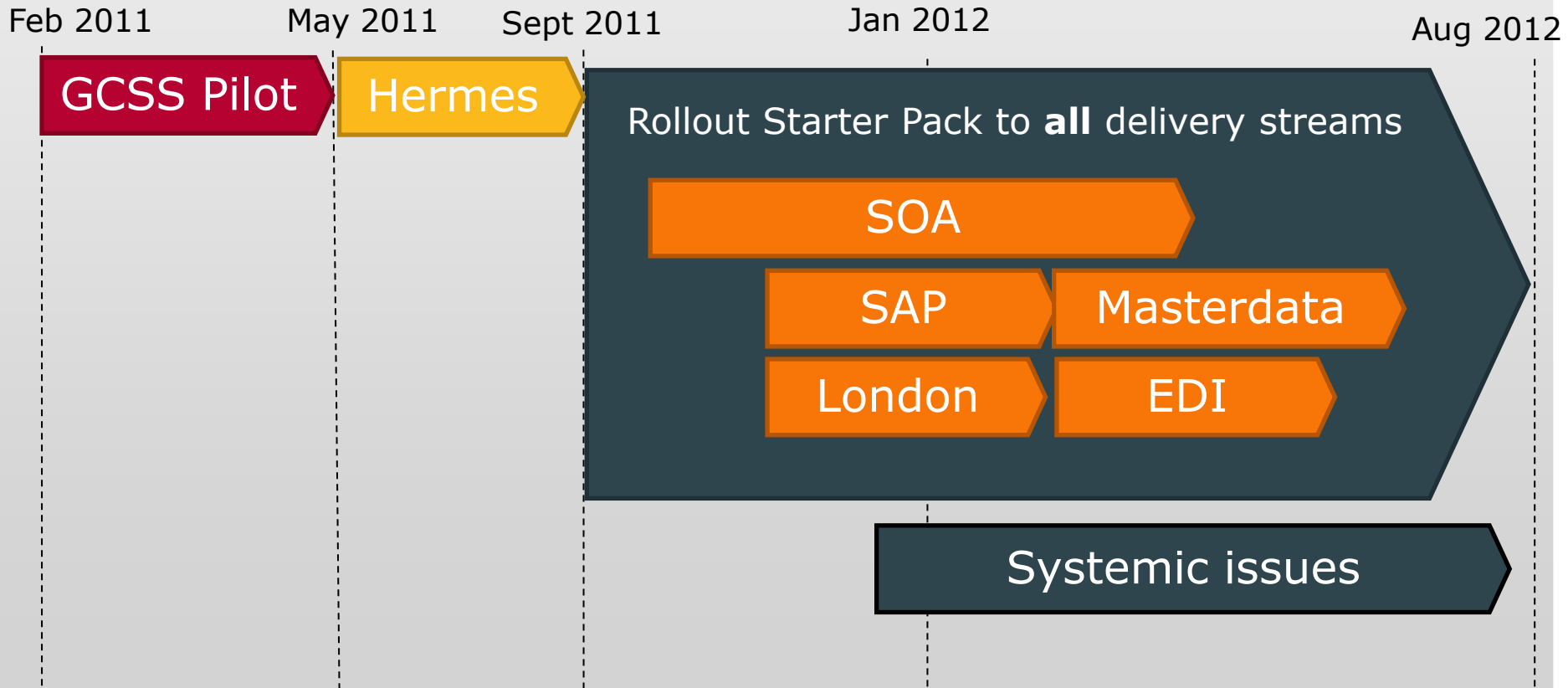
GCSS: Cycle time

Average time elapsed from starting work to released



*No data for R18, R19

Rolling out!



Lean Product Development Checklist



Start with these... The following are central to Lean Product Development

Bottom Line:

A working solution is delivered less than 90 days after starting work on a requirement (when you've achieved this, go lower. For new projects, aim for less than 30 days)

New requirements are prioritised within a week of being captured

Each requirement is supported by its own \$ benefits assessment

Each requirement is prioritised using Cost of Delay Divided by Duration (CD3)

Requirements are pulled from a dynamically prioritised list when the development team have capacity

The estimated size of a requirement is such that coding can be completed within two weeks

The team starts writing code within two weeks of pulling a requirement from the dynamically prioritised list

The team actively manages the number of requirements being worked on in at least one part of the process

Developers get immediate feedback about whether changes they are making work or not

When a requirement is completed it is demonstrated to the BPO for feedback and acceptance within a week

Feedback about whether a release candidate is launchable is obtained within a week of code completion

The team have regular Retrospectives where they capture learnings and drive improvements in how they work

Then go further with these...

If you are a new project, try to incorporate as many of these as possible from the beginning, *especially* the quality items

Projects or major new enhancements have a Vision Document that describes the high-level purpose and scope

All requirements are expressed as "User Stories" that form a placeholder for a conversation

Someone with business knowledge who represents the customer is always available to the team for ad-hoc direction

Risky items are prioritised by recognising the information value that completing them will generate

The team sits together

The solution is delivered in either small increments or short iterations *see other side

The team have a daily review of progress where impediments are identified

The team is made up of people with all the skills required to develop a solution

Requirements have defined Acceptance Criteria before development starts

Test coverage and code quality metrics are continuously monitored

The design is simple to start with and evolves as new functionality is added

All changes to the code base are immediately tested and reported on

Developers work together in pairs and there is collective code ownership

Executable tests are written before the solution is developed in code

Value

Quickly focus on the most profitable ideas

Flow

Smooth sustainable and just-in-time

Quality

Fast Feedback and flexible design

Engineering Quality Checklist

New delivery teams need to adopt these as soon as possible in order to build quality in and establish a foundation for sustainable delivery of value.

Development

- All assets are checked into a single repository (code, config., test scripts, schemas, migration scripts etc) 1
- Developers check-in code to the repository at least daily 2
- Developers have collective code ownership & responsibility 3
- Non-functional requirements are identified and prioritised alongside other requirements 4
- User interface tests & unit tests are run by the developer before code check-in 5
- Source control branches are frequently merged (every 2 weeks or less) 6

Technical debt

- The team regularly takes time to identify and record technical debt 7
- Repaying technical debt is prioritized alongside other requirements 8

Build & test

- The build runs all unit tests, regression tests and all non-manual acceptance tests 9
- Broken builds are fixed (or the check-in is reverted) before more code is checked-in 10
- Test stubs ensure all automated tests are independent of other systems (excl. network & integration tests) 11
- A build is completed within 20 mins of code check-in and is then deployed to a non-production environment 12
- Test coverage and code quality metrics are monitored 13
- Testing is prioritised using a risk-based approach 14
- Some performance tests are run at least daily 15
- The load-to-failure threshold is identified 16
- All programmatic interfaces are permanently available to other systems for integration testing 17

Deployment

- All batch testing of requirements and the subsequent deployment to production takes 7 days or less 18
- All environments can be recreated using the same automated process 19
- Updates are deployed to production without customer downtime 20
- All deployments are automated (including schemas, migrations & platform/application configuration) 21
- A developer's environment & tools are built from a standard configuration within 2 hours 22

Environment provisioning

- Any new environments (excluding production) required are provisioned within a week 23
- Any standard production environments required are provisioned within a month 24

Monitoring & improvement

- Build, test & deployment process performance is measured and continually improved upon 25
- How to monitor production health is an integral part of the design 26

Learning from rollout so far

- Practices seem to work everywhere
- Mature teams are generally more receptive than newer ones
 - They know their process and that it needs improvement
- As with all change programmes, a couple of key individuals in the team can make a huge difference
- Personnel turnover makes changes hard to stick
- There are systemic issues which need addressing

Potential lean-agile adoption barriers

SYSTEMIC ISSUES

LOCAL ISSUES

Lean-agile adoption at a team level helps with these

Senior Management
Programme Management

Give me a date for this feature
We don't really know.

Whats in the release?
Don't really know until it's ready

Business change management

We're gonna release often

We're not set up for that.

Funding governance

We need to know the costs & benefits up front

We focus on speed not predictability

You changed your mind again?

Yep! Because we learnt something new

We need you to make some changes now

Ready in 12 months!

Other systems

We want to test every day

Book a slot with our test manager at least 3 months in advance

Hardware and licences now please

Ready in 12 months!

Infrastructure

We want to release often using the same process as we use in development

We are not set up to work like this

Release management

Where's the normal documentation?

We only do documentation when it adds value

IT QA

Business-folk

Sign here so I can pass the blame if it goes wrong
Er. OK. We'll use loads of time creating massive spec. Then we'll add loads of padding to the schedule/ budget. Then we're willing to sign!

We've got our KPIs
And we've got ours!

Write better specs!

We don't know how

Tell us what it will cost and when we can have it
We don't really know

IT-folk

We want to try this technology/ architecture in this sprint

I have some time next month to review the documentation you need to submit

Architectural governance

Slow burn - stakeholder education

Have you got the "lean-agile mindset"? - Lean Product Development Blog - Windows Internet Explorer

http://team.apmoller.net/sites/leanproductdevelopment/blog/Lists/Posts/Post.aspx?ID=30

File Edit View Favorites Tools Help

Home @maersk Maersk Group Business units Tools Berridge, Chris Site Actions

Browse

MAERSK LINE | enable

LPD Blog | LPD Vision | LPD Ambassadors | LPD Reading List | LPD Healthcheck | IT Newsroom

You are here: Lean Product Development Blog

Archives

- April
- March
- February
- January

show more >

Recycle Bin

All Site Content

March 26

Have you got the "lean-agile mindset"?

by Berridge, Chris on 3/26/2012 2:45 PM

This week the LPD coaches team spent some time evaluating a potential LPD training course being developed by the agile consultancy [Emergn](#). The course provoked some reflections on what adopting a lean-agile mindset at Maersk Line means...

Intuitive Decision Making

The course asserts that way we make decisions is largely based on our intuition. Studies of how CEO's make decisions shows that, although they often believe they are making rational fact-based decisions, it's actually coming "from the gut." This kind of expert intuition is powerful stuff and can almost be magical - fire-fighters who run out of the house seconds before the floor collapses, chess masters who can glance at a board and declare that white can win in 3 moves, etc.

It was argued in the course that good intuitions take a long time to build up. Within IT development, the high variation both in the type of work, the length of time of most IT implementations and the short lifetime of most teams makes it

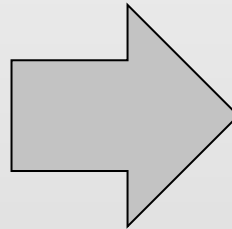
Key Performance Measures for IT



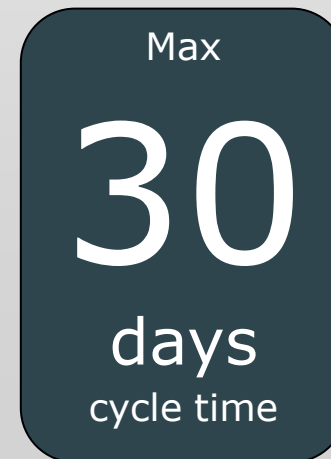
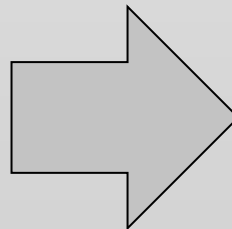
Variable	Typical measures	Usual outcomes	Alternative measures
Time	Delivering on a predicted date	Incentivises hidden time buffers and slower delivery	Maximise speed in getting to the point where value starts to be realised
Scope	Delivering all of the originally predicted scope	Incentivises gold plating and discourages exploitation of learning.	Minimize size of work packages to maximize both learning and early release of value
Cost	Delivering at or below a predicted development cost	Incentivises hidden cost contingencies, pushing costs up.	Maximize value delivered (trade development cost against the opportunity cost of delay)
Quality	Delivering changes with zero downtime and no errors	Resistance to making any changes. Overinvestment in testing & documentation.	Shorten feedback cycles at many levels (coding, defects...)

What next for Maersk Line?

- **Legacy:** Complete rollout 8 starter pack practices for all legacy applications



- **New:** Additional practices for our new Service Oriented "vision platform"



Enabling Agility

Business Agility

**Fast cycle
Time**

**Smooth
Flow**

**Fast
Feedback**

**Value
Maximised**

Discovery Mindset

Customer doesn't really
know what they want

The developer doesn't
really know how to build it

Things
change

Questions?



Chris Berridge

Programme Manager
Lean Product Development
Maersk Line IT

+45 3363 8165
chris.berridge@maersk.com

Agile Project/Programme Manager of the Year 2011

