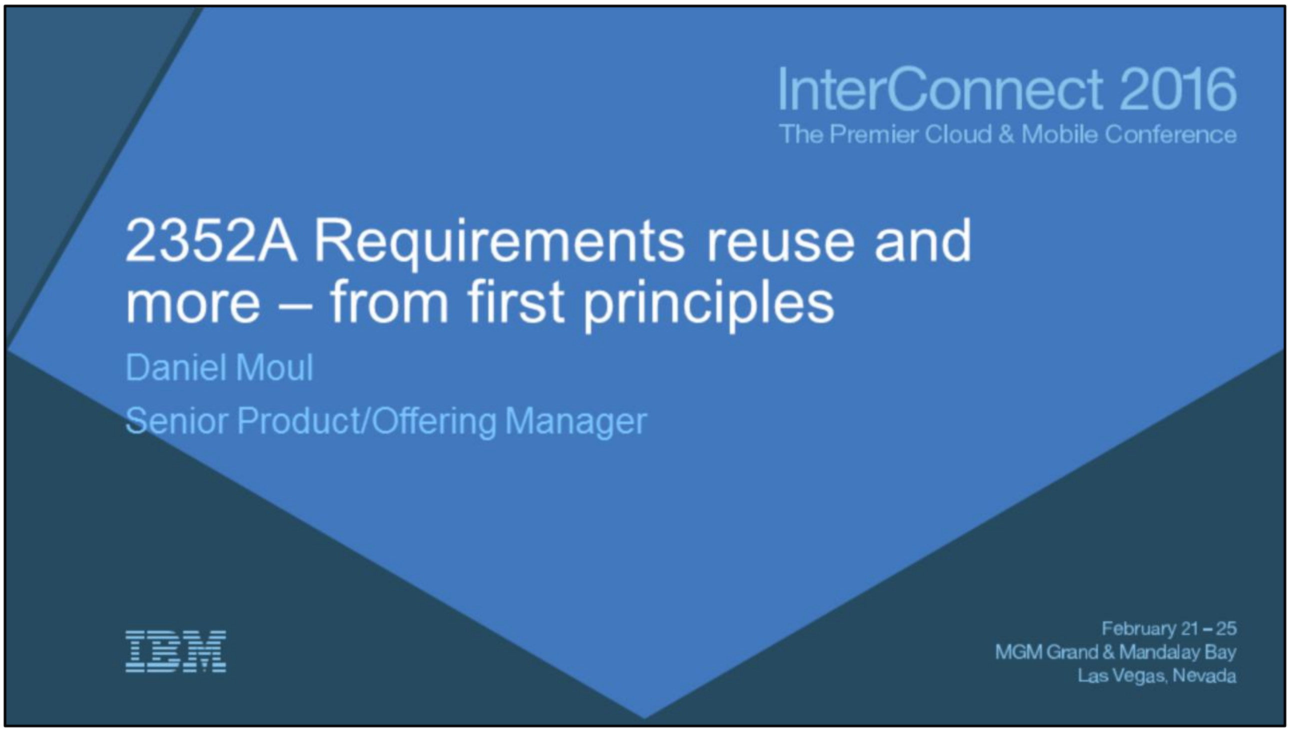


InterConnect 2016

The Premier Cloud & Mobile Conference

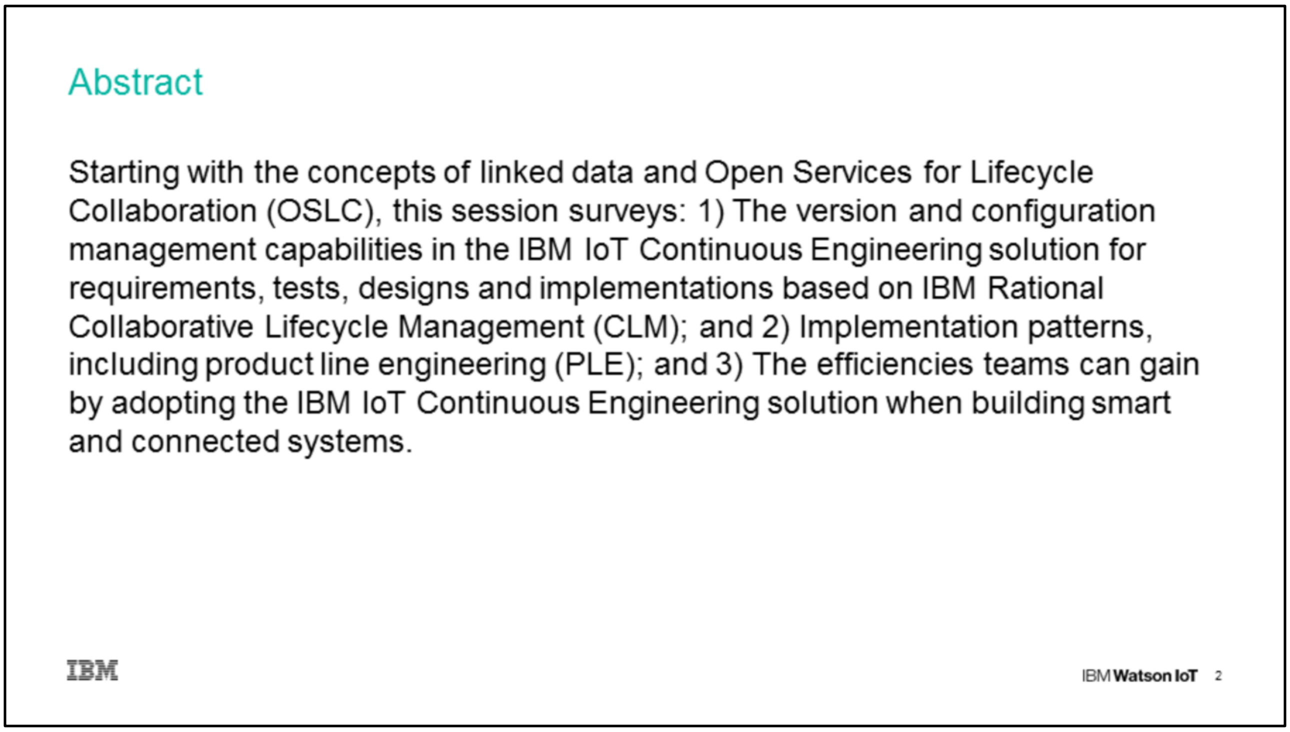
2352A Requirements reuse and more – from first principles

Daniel Moul

Senior Product/Offering Manager

IBM

February 21 – 25
MGM Grand & Mandalay Bay
Las Vegas, Nevada



Abstract

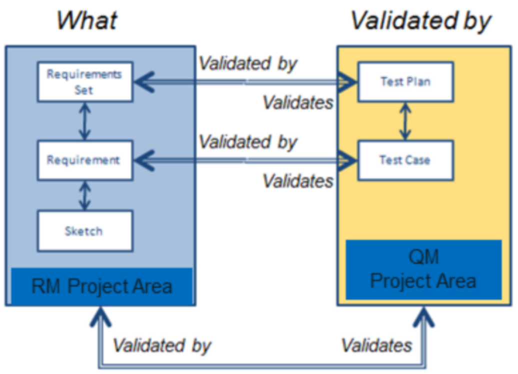
Starting with the concepts of linked data and Open Services for Lifecycle Collaboration (OSLC), this session surveys: 1) The version and configuration management capabilities in the IBM IoT Continuous Engineering solution for requirements, tests, designs and implementations based on IBM Rational Collaborative Lifecycle Management (CLM); and 2) Implementation patterns, including product line engineering (PLE); and 3) The efficiencies teams can gain by adopting the IBM IoT Continuous Engineering solution when building smart and connected systems.

1. Basic building blocks
2. Using streams and baselines
3. Using global configurations

IBM

IBM **Watson IoT** 3

Linked lifecycle data



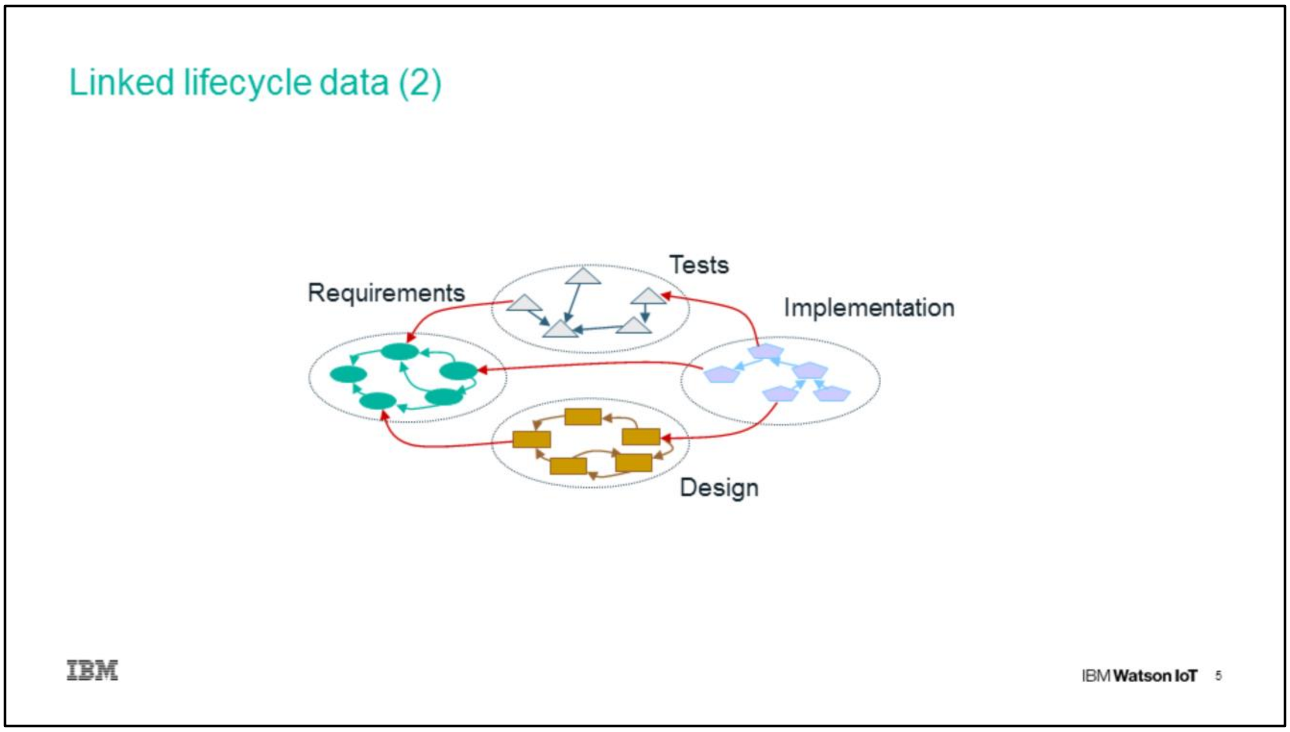
IBM

IBM Watson IoT 4

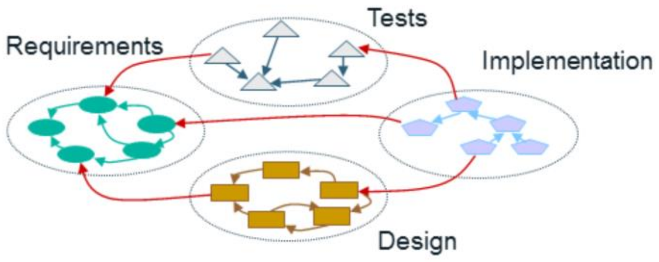
Engineering data exist in multiple tools and repositories.

Relationships express relationships among the data.

In this case test artifacts validate requirements.



Linked lifecycle data (2)



IBM

IBM Watson IoT 5

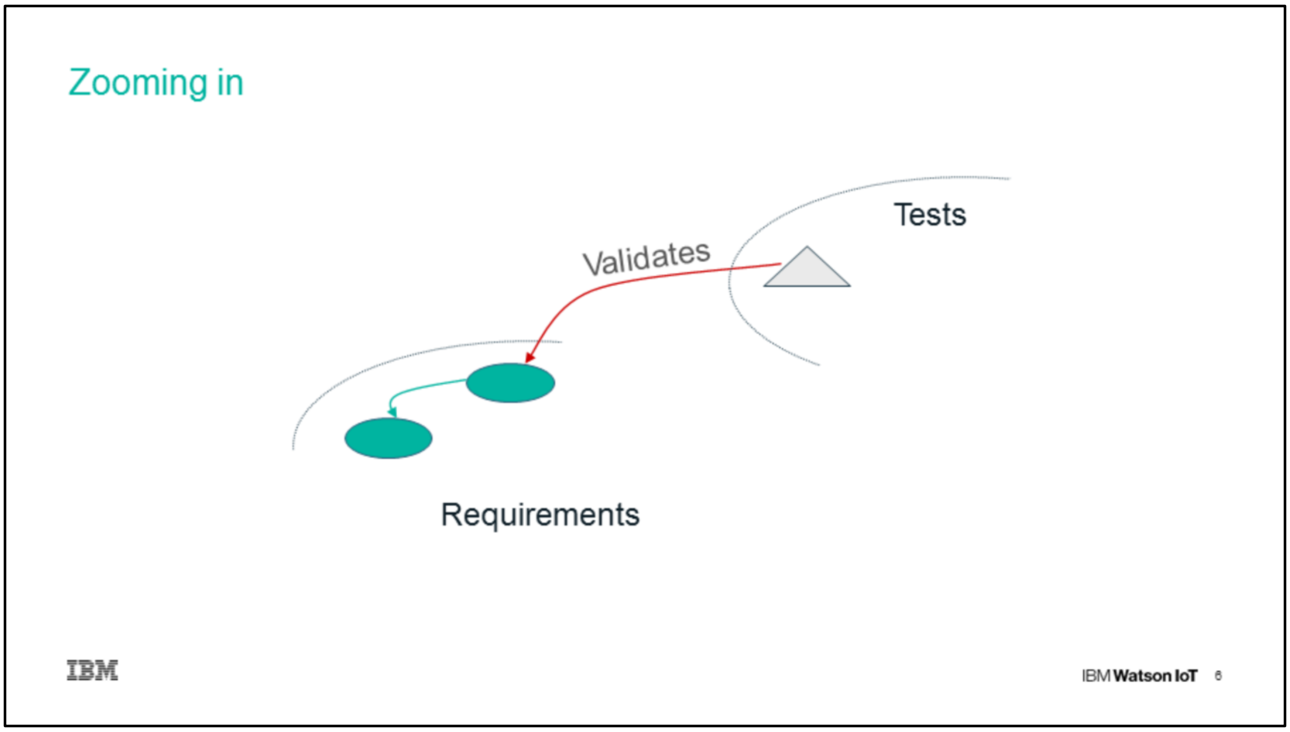
This pattern exists on a larger scale.

There are links within tools, for example, between requirements

And links across tools

Typically links have specific meaning. Some are defined in OSLC specifications;

others are custom-defined to as part of an information model that supports a team's development practices.



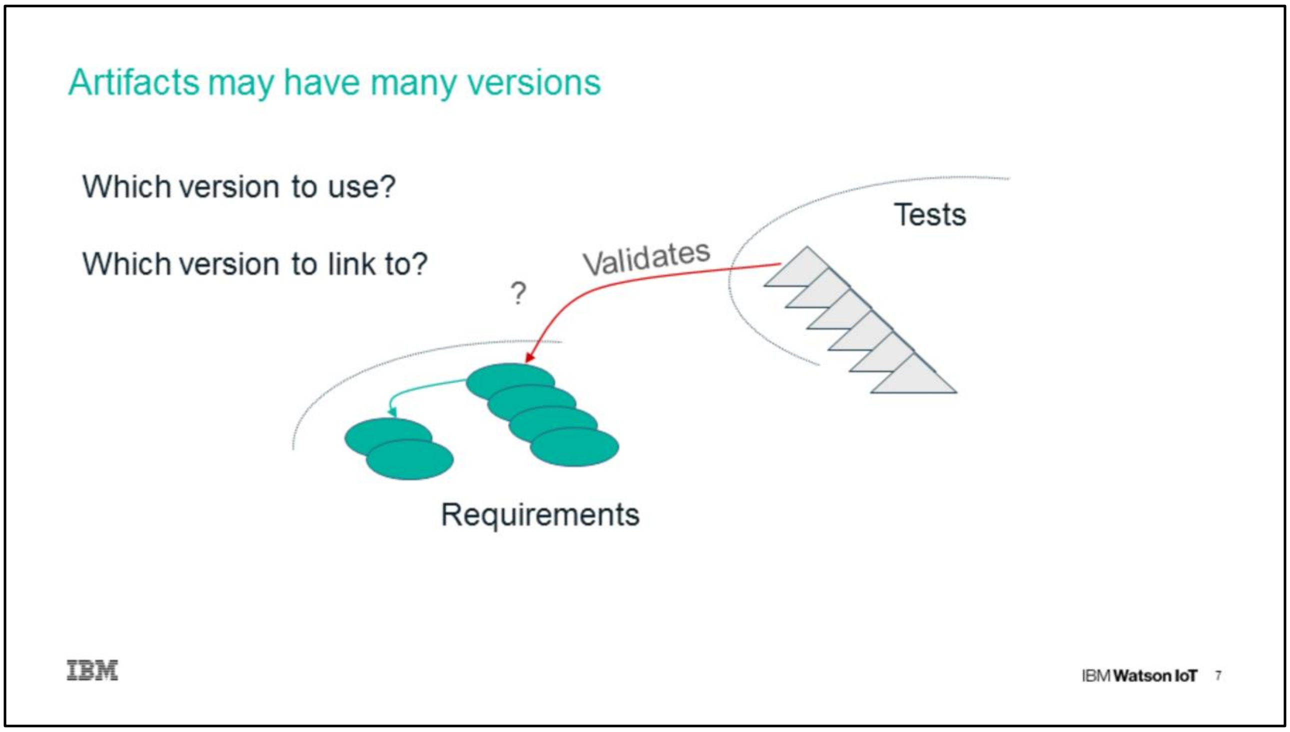
Zooming in

Tests
Validates
Requirements

IBM

IBM Watson IoT 6

Let’s zoom in and consider one test case that validates a requirement that satisfies a higher-level requirement.



Artifacts may have many versions

Which version to use?

Which version to link to?

IBM

IBM Watson IoT 7

The engineers make edits to the requirements and test case.

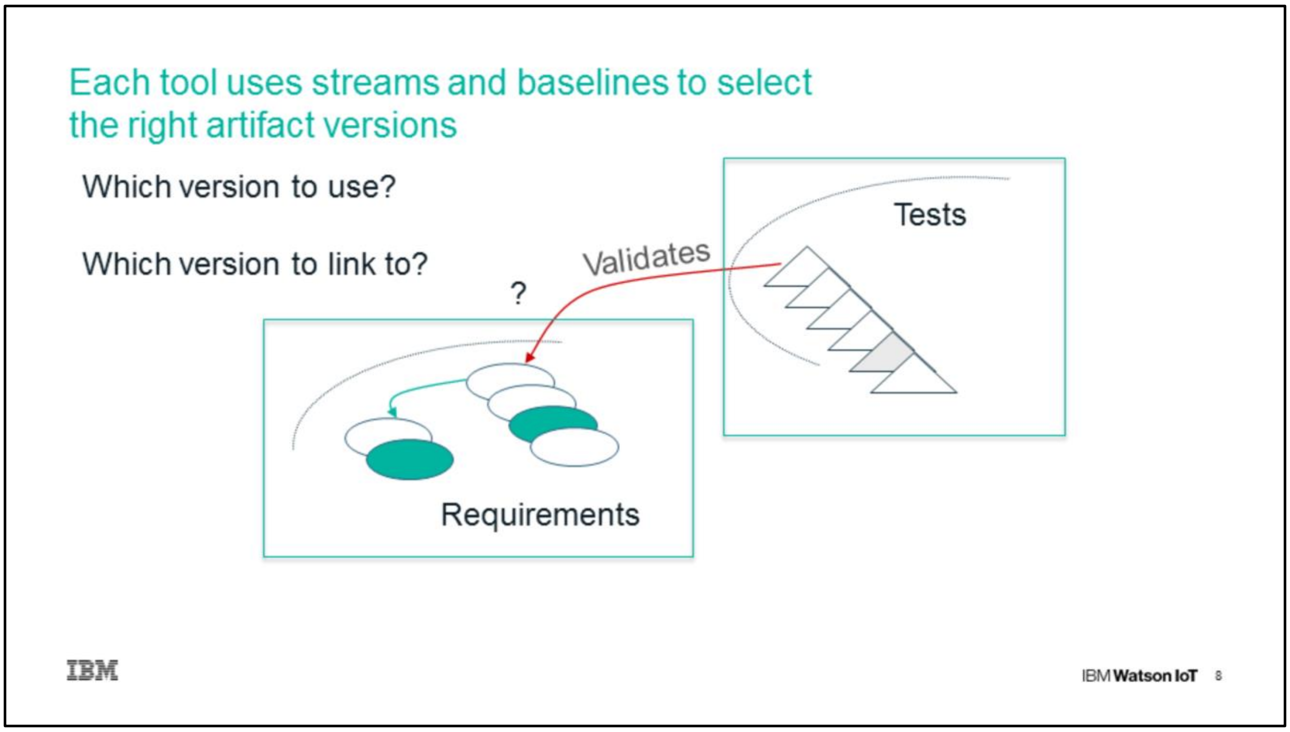
Now we have ambiguity: which version of the test case validates which version of the requirement?

The engineers make edits to the test

And the engineers don’t want to manually manipulate the links – that’s too much overhead, and it’s too easy to make mistakes.

We’d really like the system to do this for us. After all, computers are good at keeping track of details like this – and people are not.

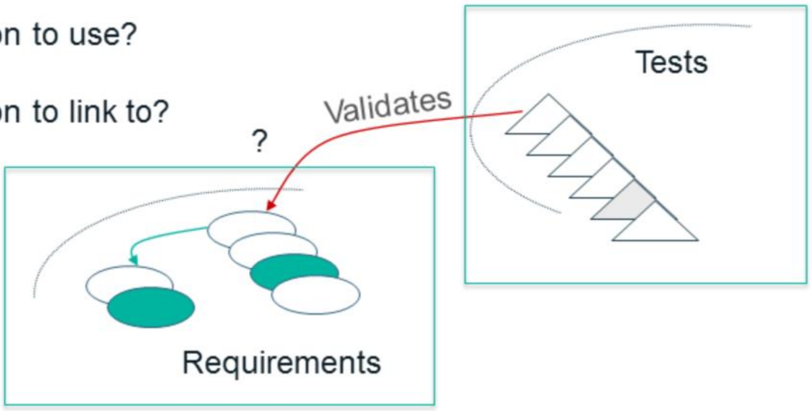
Let’s look at how the IBM solution, based on a new OSLC specification for configuration management, solves this.



Each tool uses streams and baselines to select the right artifact versions

Which version to use?

Which version to link to?



IBM

IBM Watson IoT 8

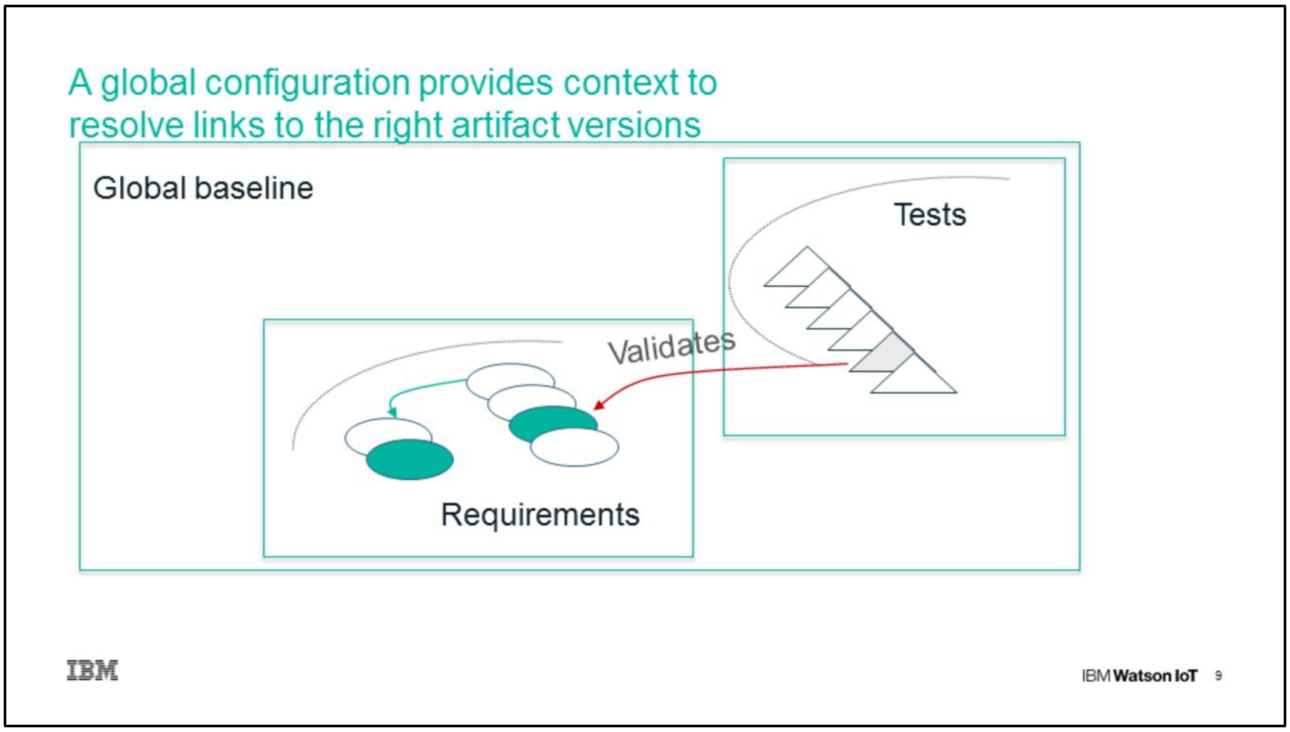
Each tool is responsible for it’s own baselines. This provides a way of saying “exactly these artifacts at exactly these versions”

In 2015 we’ve added this ability to do configuration management to Rational DOORS Next Generation and Rational Quality Manager.

But it doesn’t solve the question of data outside the scope of the tool’s responsibility – in particular, links to data in other tools.

In the past teams have tried to work around this by keeping track outside the tool:

For example, “Baseline X in the requirement tool is related to Baseline Y in the SCM …”



A global configuration provides context to resolve links to the right artifact versions

IBM

IBM Watson IoT 9

We really want the system to keep track of this detail. The information system of record should be the information that drives the tools.

Global configurations provide a context beyond single tools.

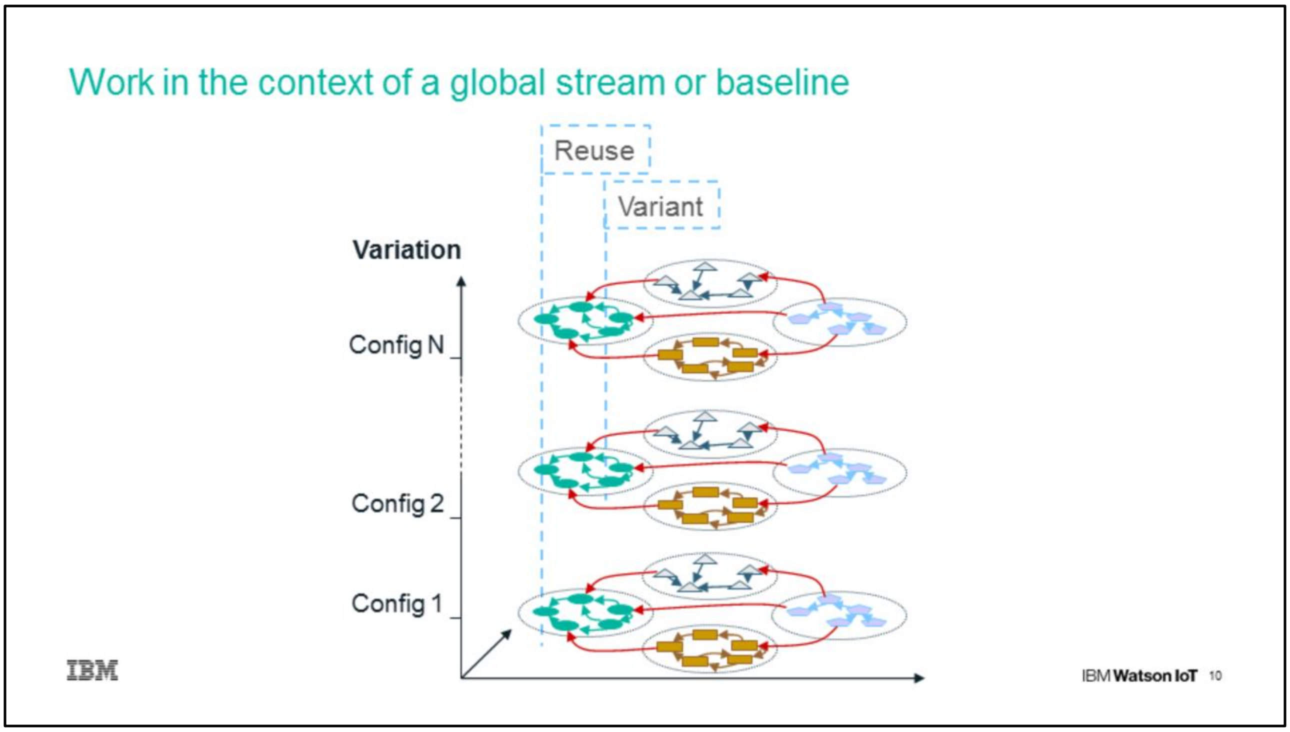
They can include baselines or streams from other tools, and they provide the context for links.

Links work differently in this configuration-aware environment.

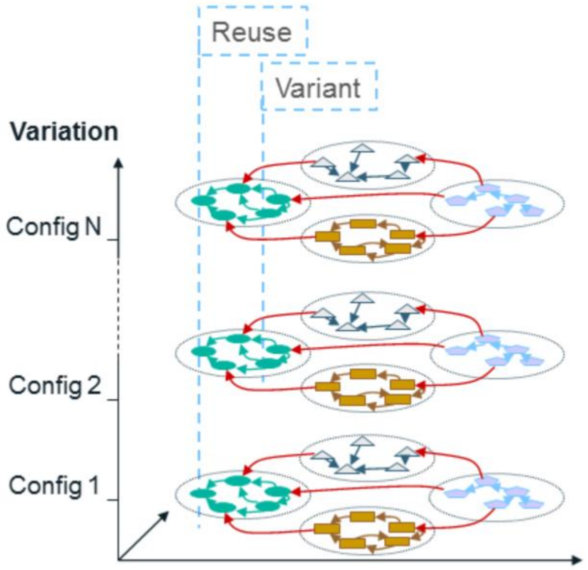
Instead of pointing to a specific version of an artifact, they point to the artifact in general. We call this a “concept link”.

The requester of an artifact at the other end of a link provides the link (as they did in CLM V5), and then also a configuration.

With the additional configuration context, the tool can resolve the link to a specific version of the requested artifact.



Work in the context of a global stream or baseline



IBM

IBM Watson IoT 10

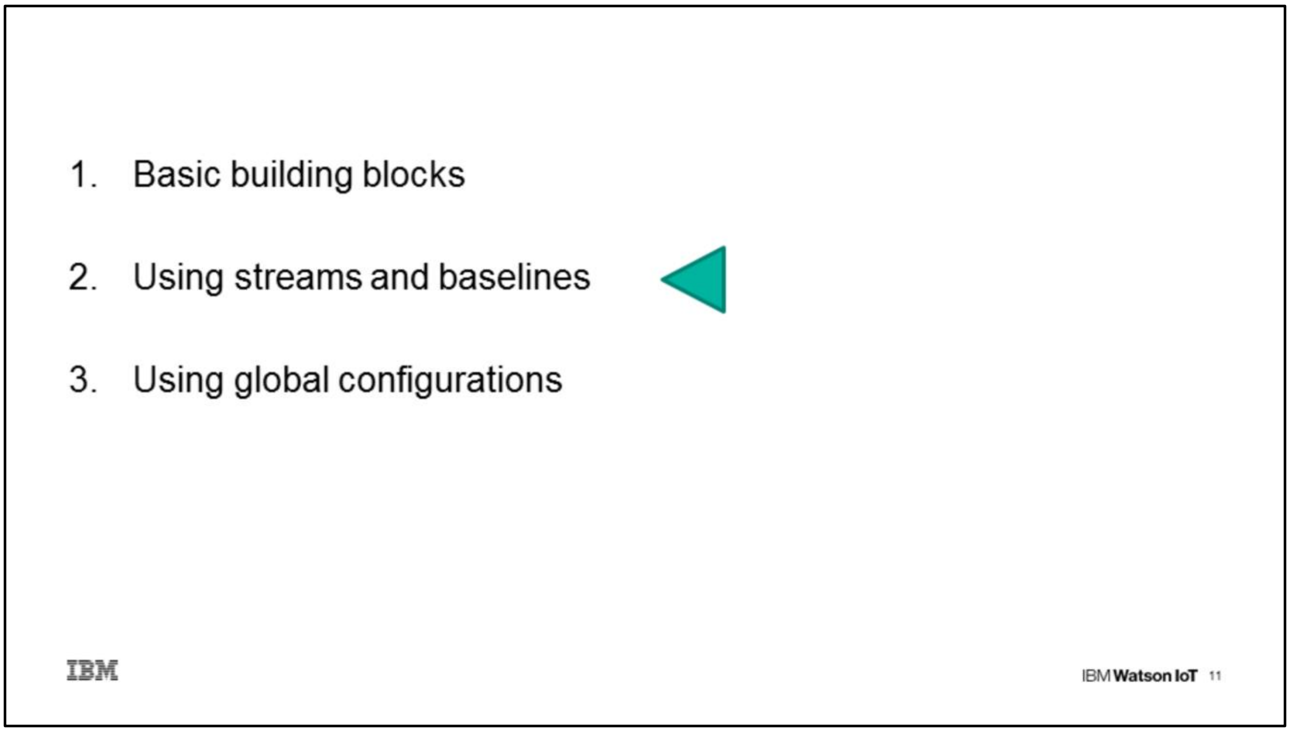
The engineer selects a configuration in which to work, and the tools do all the rest:

- Selecting the right version of each artifact
- Resolving all links across tools in the context of that configuration

Thus, from the engineer’s point of view, wherever he or she goes – following links from one tool to the next – he or she is always in the right place.

And when creating a baseline or branching, there is no extra work required to create copies or deal with the links – everything just works.

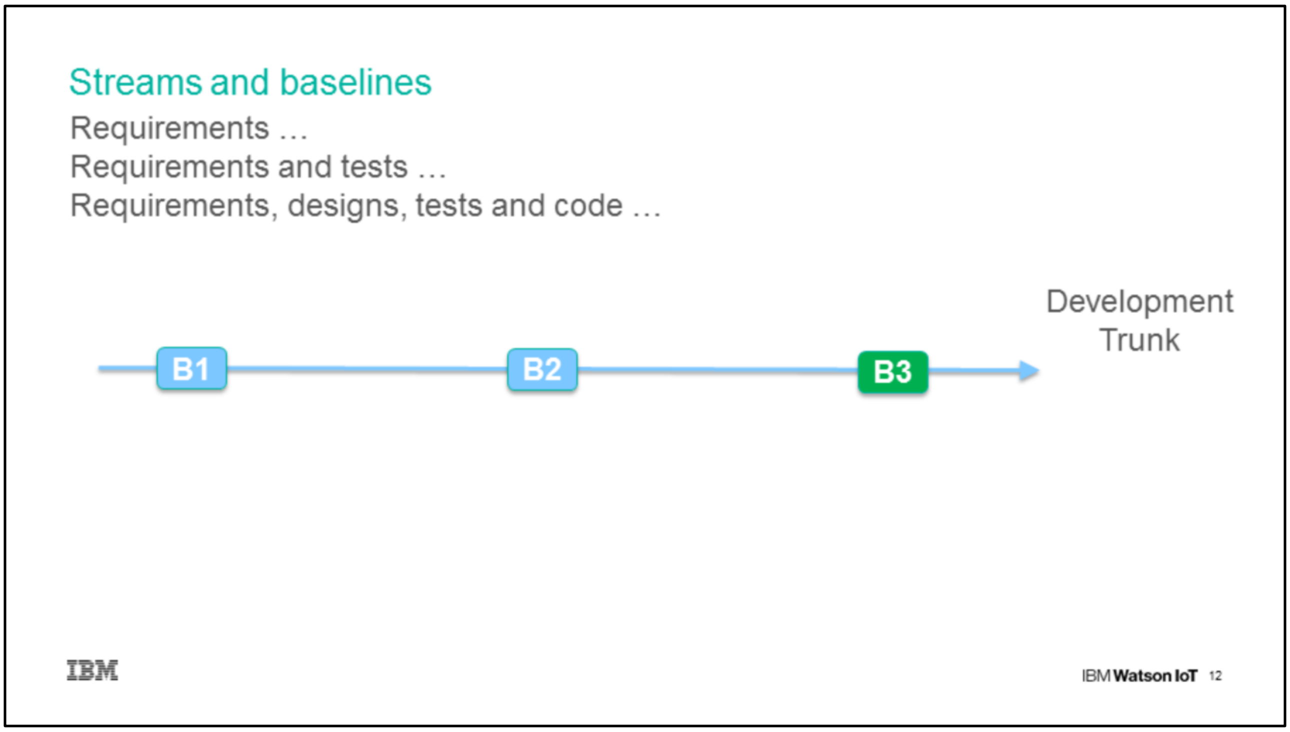
This is a great reduction in complexity and overhead compared to many solutions.



1. Basic building blocks
2. Using streams and baselines ◀
3. Using global configurations

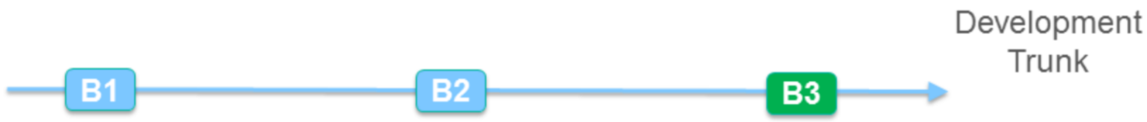
IBM

IBM Watson IoT 11



Streams and baselines

Requirements …
Requirements and tests …
Requirements, designs, tests and code …



IBM

IBM Watson IoT 12

Here we have a stream of requirements development in what this team calls a "development trunk".

A development stream is a context in which artifacts are created and modified. Change happens.

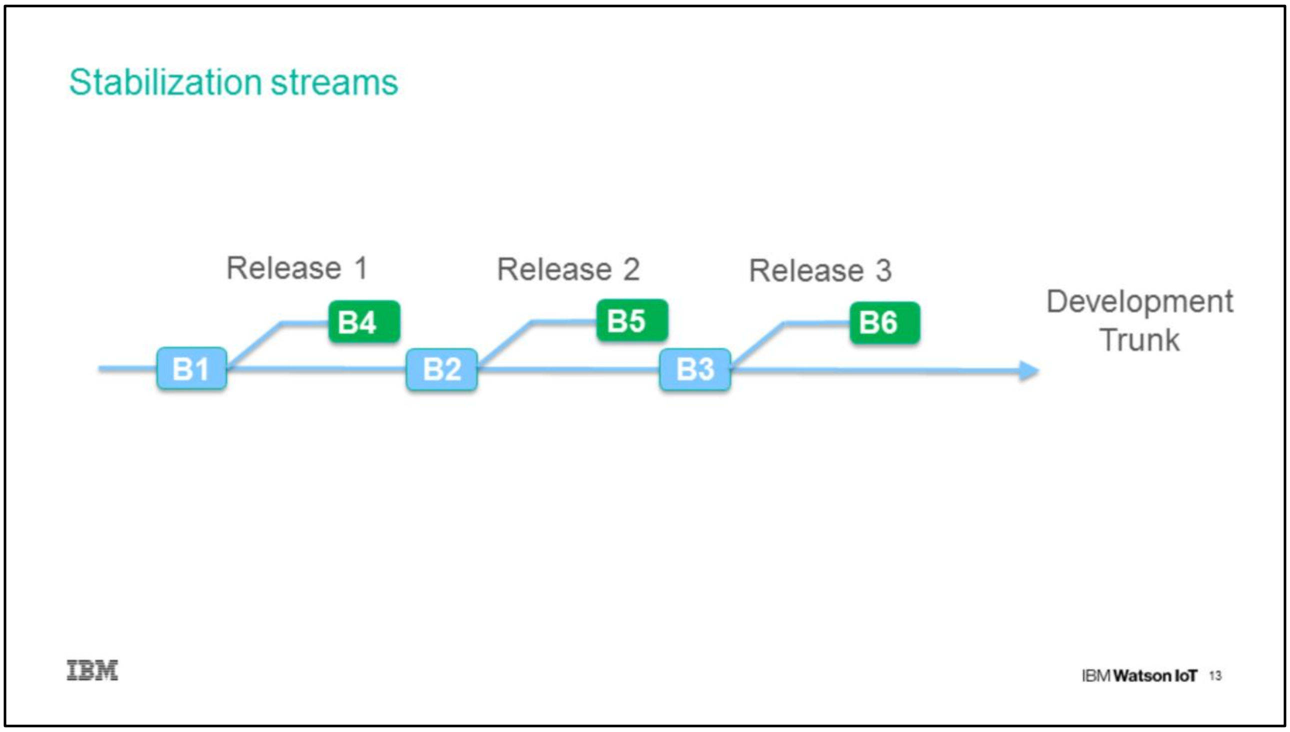
A baseline encompasses artifacts at specific versions that cannot be changed – it's immutable.

Both streams and baselines are configurations.

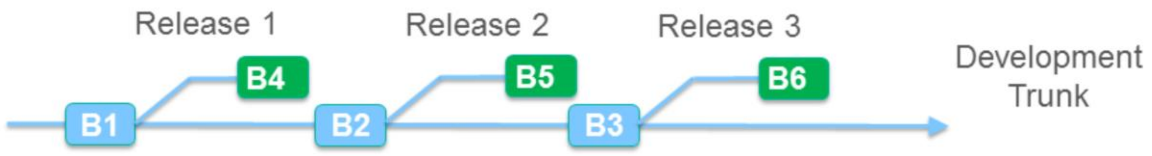
This team makes two intermediate baselines (blue) then a release baseline (green).

And this can include designs, tests, calibration data and implementation artifacts under source control.

Even though they are in different tools, they depend on each other and traceability needs to be maintained as we discussed earlier.



Stabilization streams



IBM

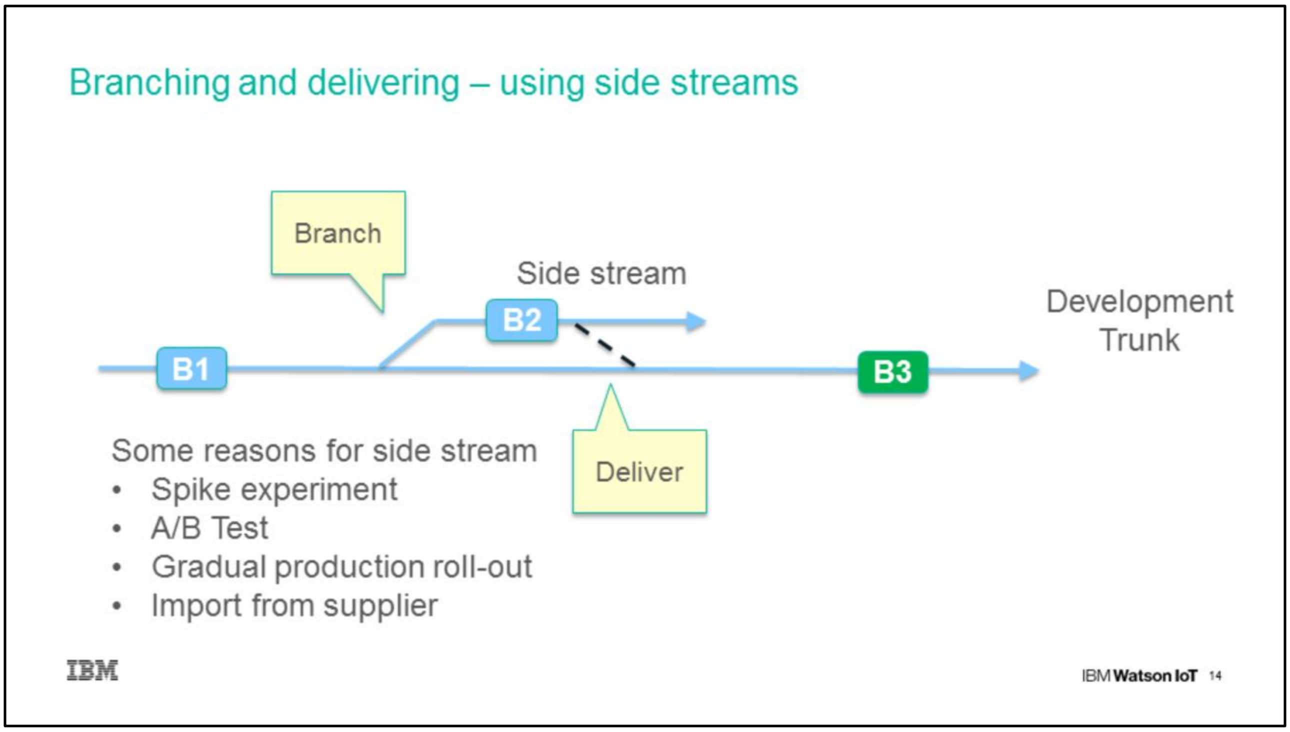
IBM Watson IoT 13

Here we have a common development pattern: stabilizing the work before releasing it.

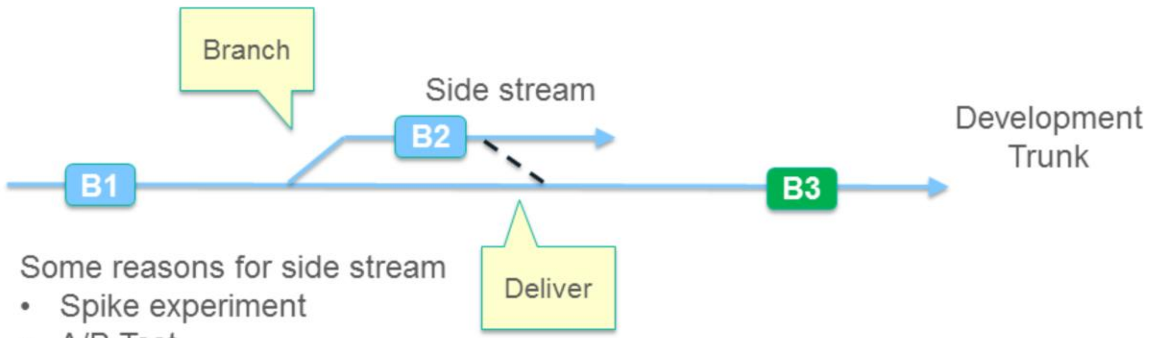
This stabilization happens in a separate release stream, so work on future releases can continue unhindered,

and new development doesn't destabilize the release nearing its end.

This shows one baseline per stabilization stream. Of course you could make as many baselines as you need.



Branching and delivering – using side streams



Some reasons for side stream
- Spike experiment
- A/B Test
- Gradual production roll-out
- Import from supplier

IBM

IBM Watson IoT 14

Here we branch the development trunk to create a side stream.

There are lots of reasons to do that beyond creating a stabilization stream …

Note that a branch reuses the current artifact in the development stream (not copies of all the artifacts).

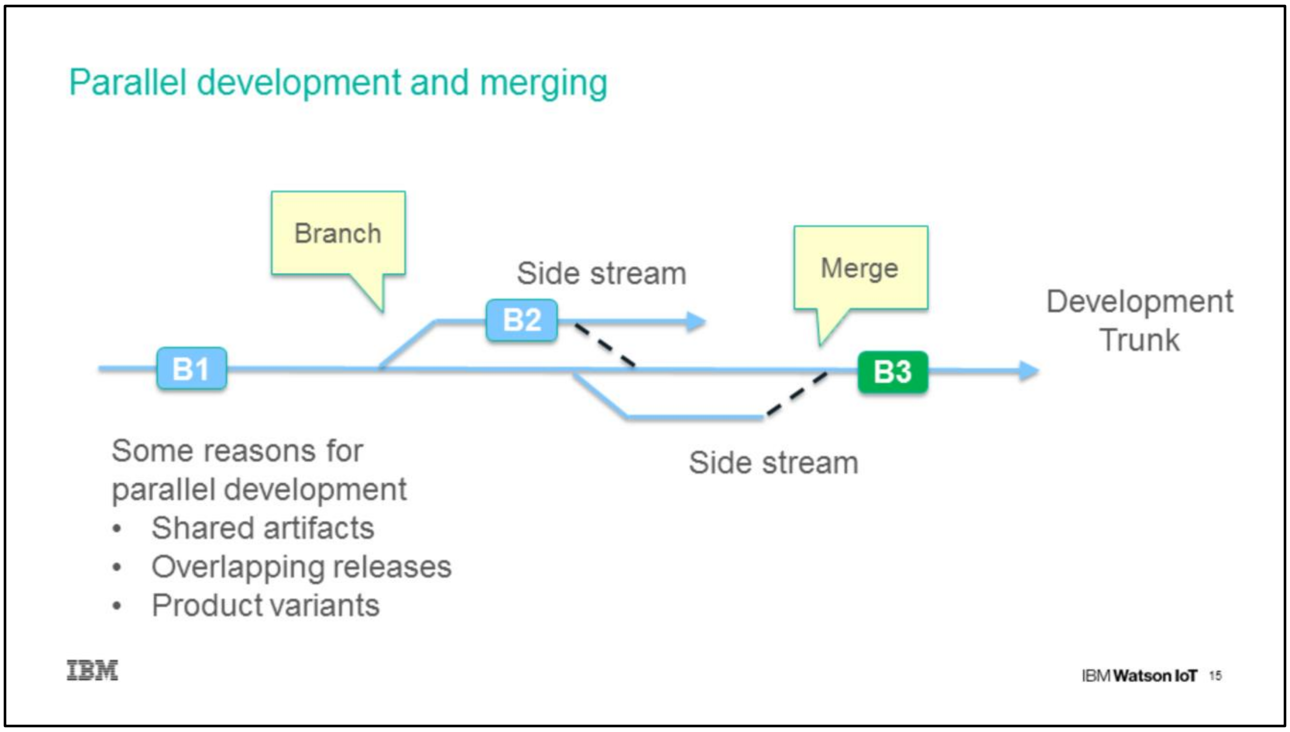
In this case, work is done in the side stream then after the B2 baseline, it's delivered into the development stream

Some reasons for side streams:

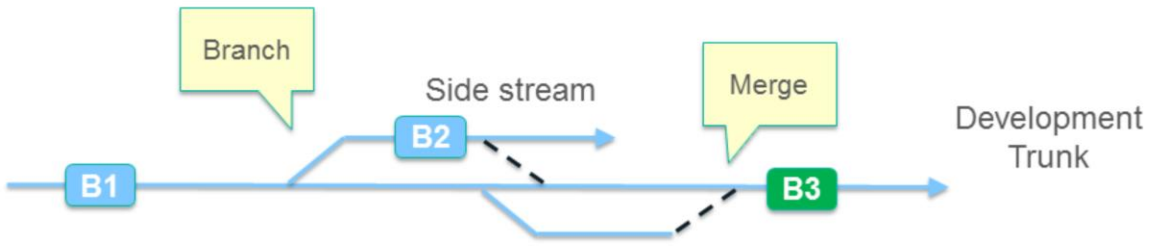
- Spike – an experiment to reduce risk, done separately from the main development stream.
- A/B test – a separate set of changes to your application for some portion of users. You'll measure the results of the test, and if they are positive, roll out the changes to larger portions of your users. If not, you'll stop offering it to your users.
- Gradual production roll-out – you've made some changes, and you are rolling out to your user community in a measured cadence. As the change proves to be stable, then you will roll out it out to the larger population. Could be store-by-store for your chain of stores; could be percentages of our your user community

coming to your web application.

- Import requirements and tests from a supplier – Perhaps in this case someone wanted to do some work on new and existing requirements and tests without disturbing the approved ones. The requirements have been shared with a supplier in a ReqIF file, and the supplier has added requirements for a subsystem the supplier is developing. These new requirements need to be reviewed and probably approved before delivering them to the main development trunk. Then tests may be developed in the side stream and approved before rolling them into the main development stream.



Parallel development and merging



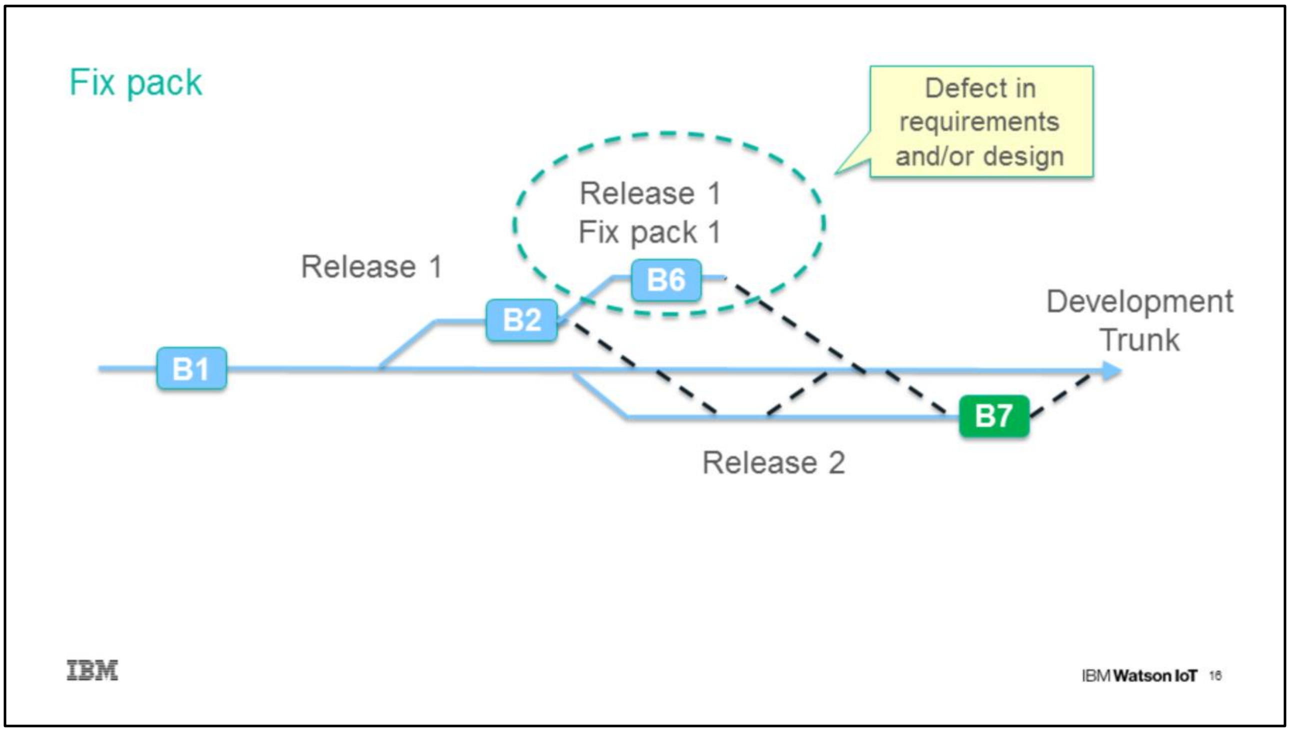
Some reasons for parallel development

- Shared artifacts
- Overlapping releases
- Product variants

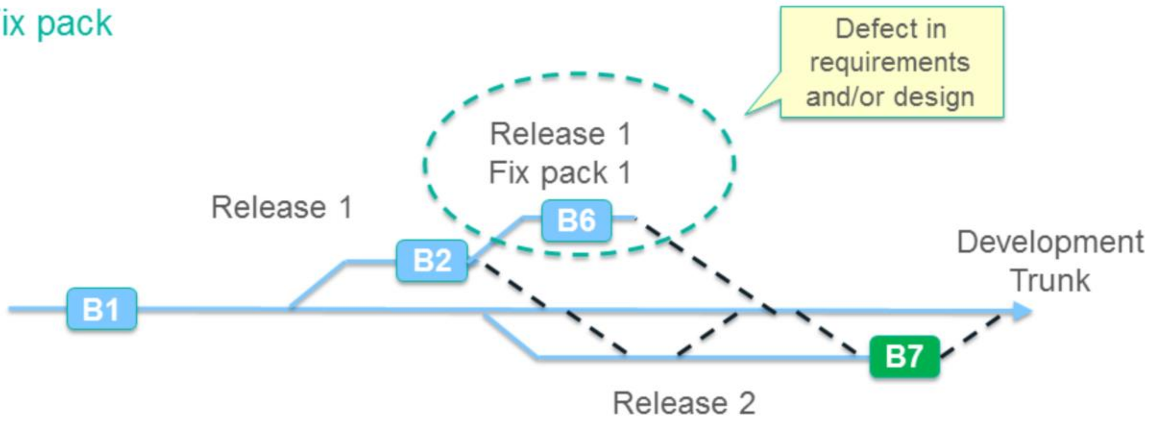
IBM Watson IoT 15

Now things are getting more elaborate. Two different people (or groups) have side streams going, and now and again (in this case at the end of their work), they deliver their changes to the development trunk.

But now we have a conflict. An artifact has been changed in both side streams. The system needs to let you know that there is a conflict, so you can address it, choosing which change "wins" or enabling you to manually reconcile the changes.



Fix pack



IBM

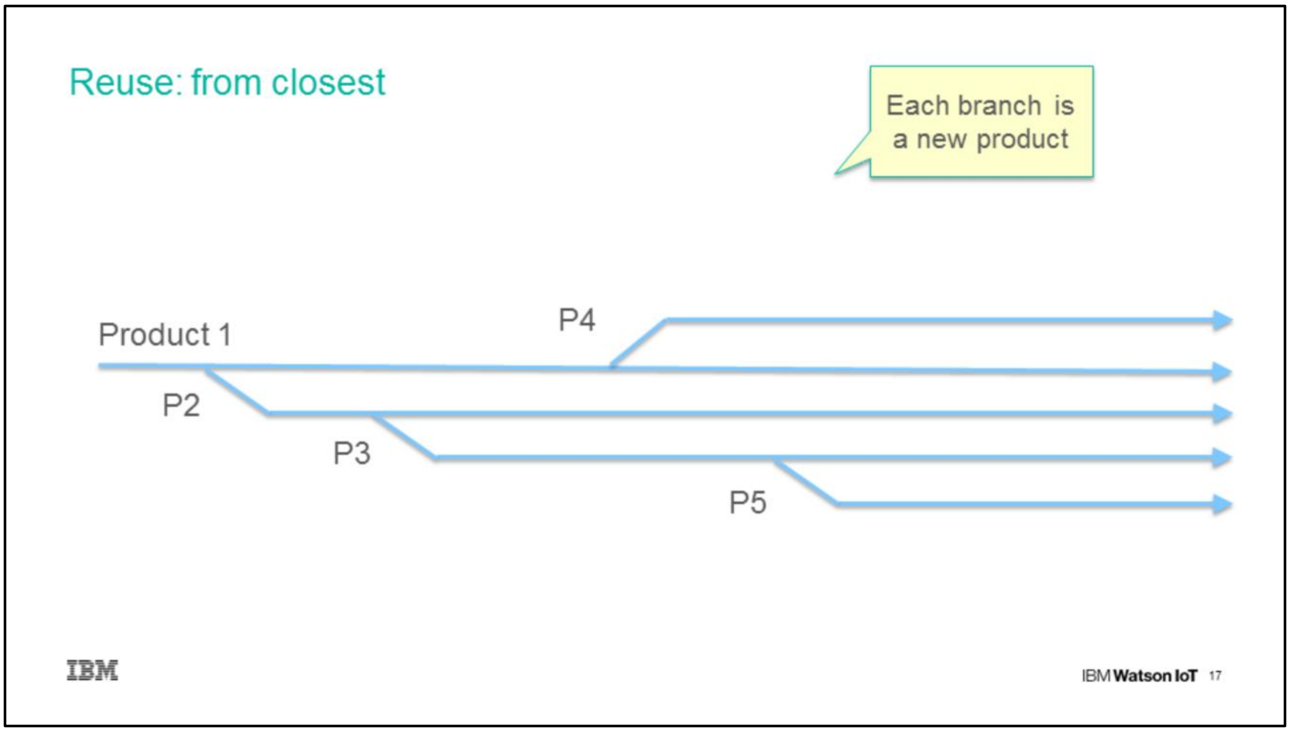
IBM Watson IoT 16

Here’s a common case: a defect has been found in release 1 – in the field -- and a fix is needed.

The team need to go back to the exact artifacts that were included in release 1 to troubleshoot and then apply the fix.

In this case the defect was in the requirements and associated tests as well as the code.

A new fix pack stream enables corrections to be made, then eventually delivered to every other relevant stream.



Reuse: from closest

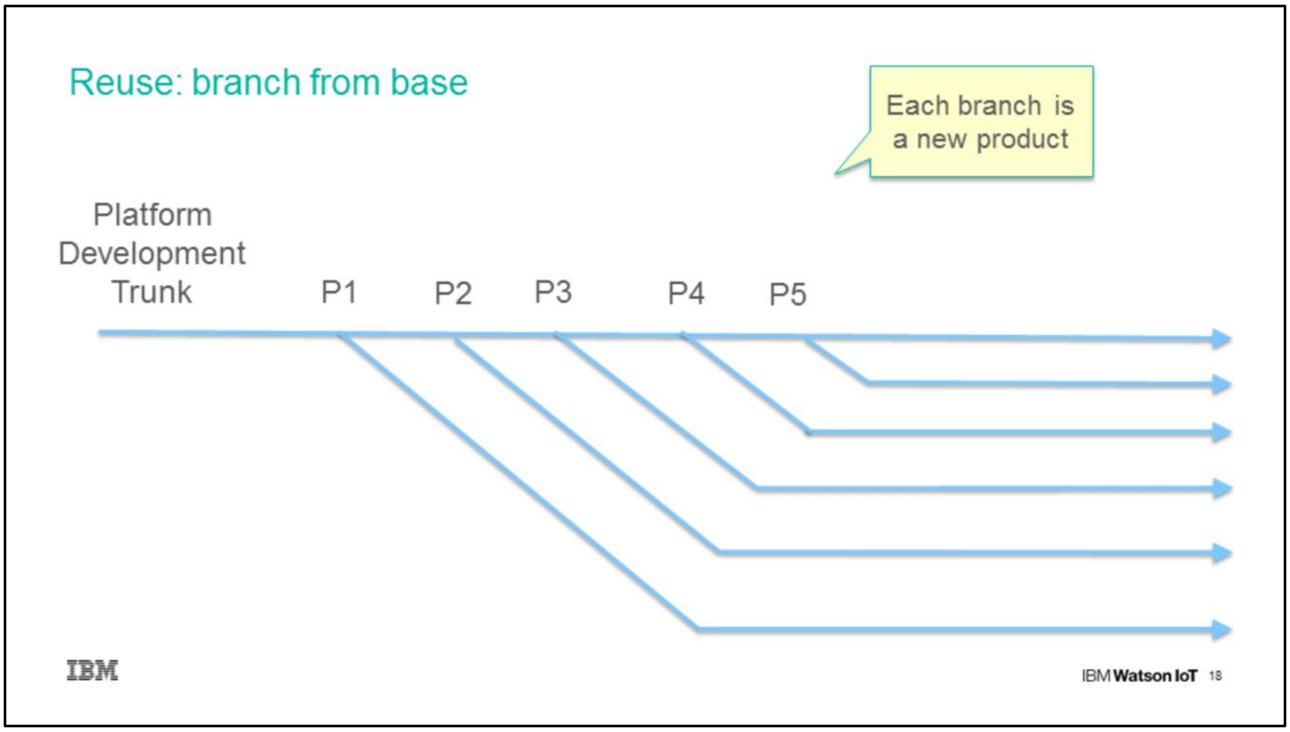
Now let's consider some reuse scenarios.

In this case a horizontal row represents a stream of development, one each for a particular product variant.

There is no grand reuse scheme thought out ahead of time.

The team is opportunistic in branching from the closest product when they need to start a new one.

(This can work, but it only scales so far before teams get bogged down in managing all the variants.)

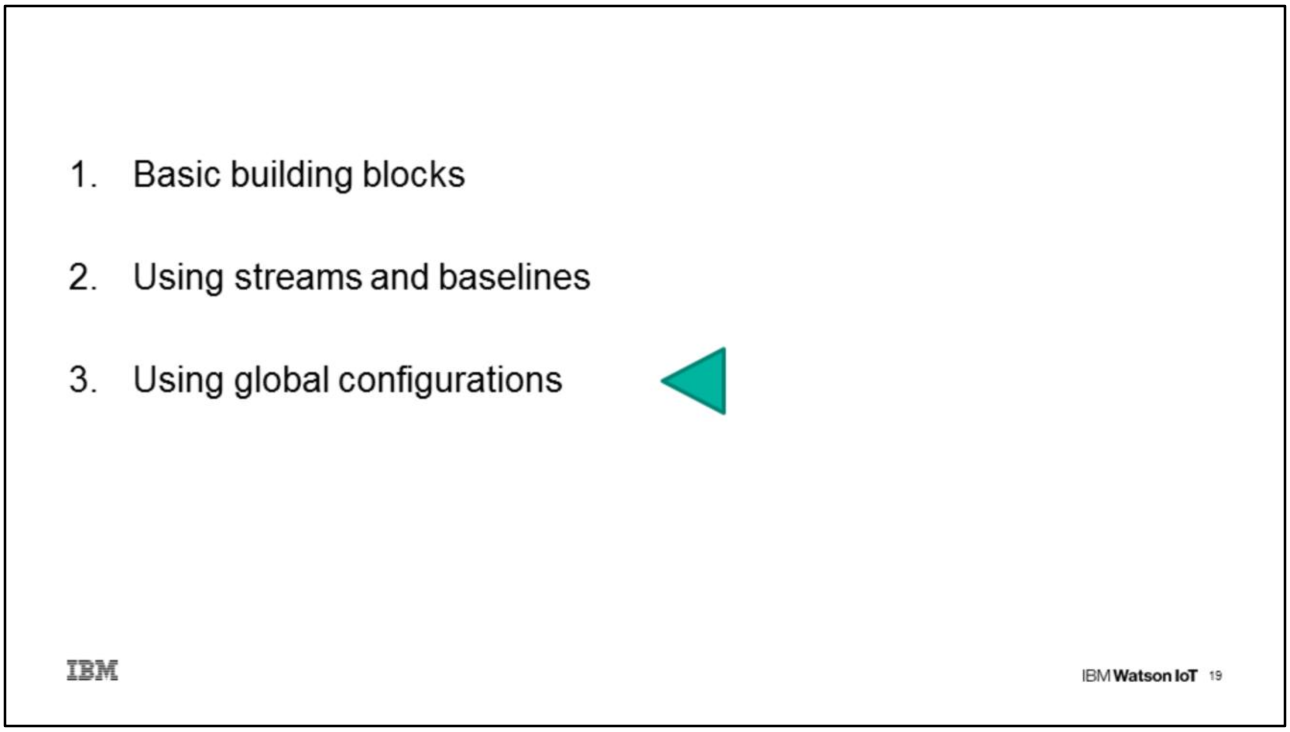


In this case all new product variants are derived from the development trunk.

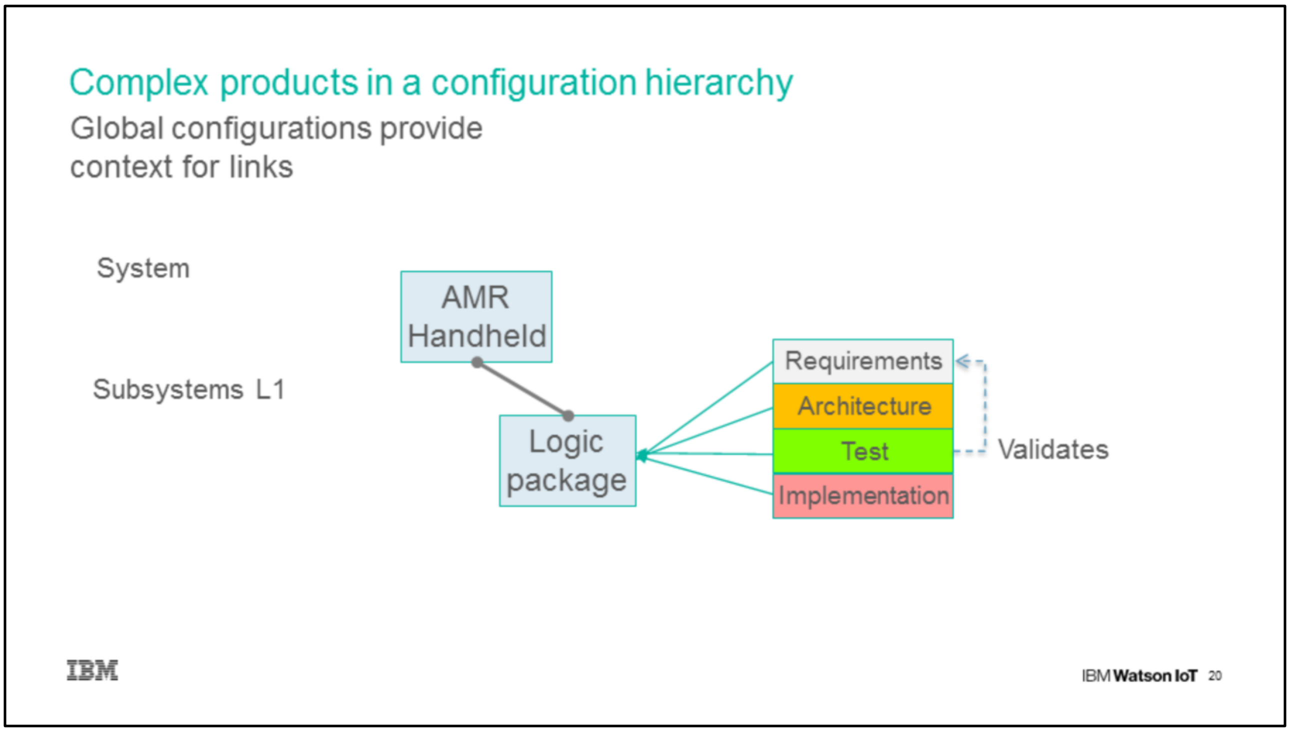
The development trunk may represent a library, catalog or “superset stream” of all reusable artifacts

New development may happen exclusively in the development trunk,

or some new features may developed in product streams then made available in the development trunk for other products to use.

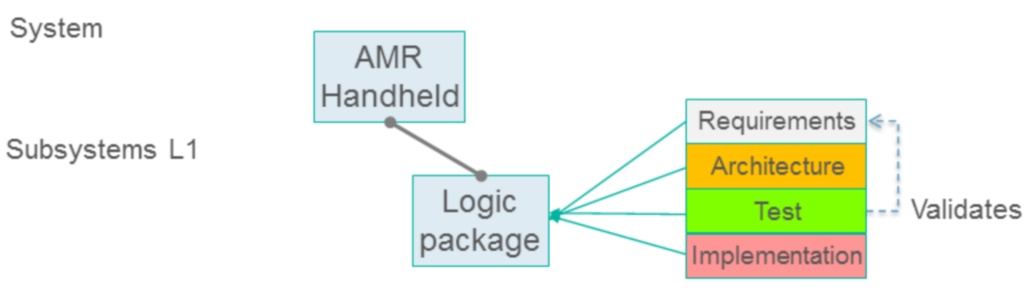


1. Basic building blocks
2. Using streams and baselines
3. Using global configurations ◀



Complex products in a configuration hierarchy

Global configurations provide context for links



IBM

IBM Watson IoT 20

Now let’s generalize as we look at an example:

- What we’ve looked at thus far (streams, baselines, branches, etc.) has been about one set of related engineering art.
- But most complex products and systems have many subsystems (and sub-sub-systems). Let’s call them “components”.
- Each component is under configuration management.

Let’s consider reuse of subsystems in the AMR Manual Handheld product.

In this picture all the dark grey boxes are global baselines. Together this hierarchy defines one release of one AMR Handheld product.

Here a Level 1 subsystem is represented by a component with a global baseline,

which includes baselines from each contributing tool and provides context for the links within and across the tools.

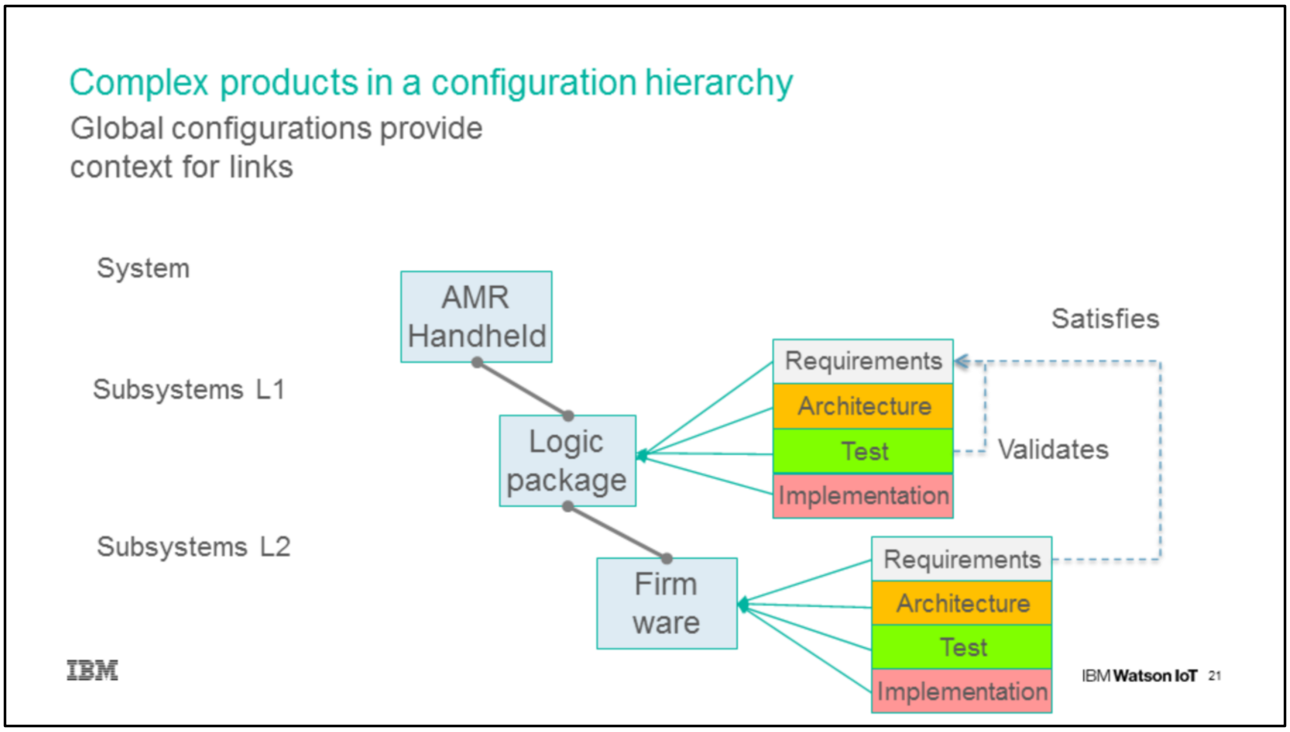
We see one “validates” link illustrating this.

In reality there would be many – hundreds or thousands – between test cases and the requirements they validate.

The top level AMR global baselines includes the level 1 subsystem baseline for the logic package

Thus an engineer can select a specific baseline of the top-level AMR Hand-held configuration, and the right baseline for all subsystems will be included automatically.

In this example (so far) there is only one.



Complex products in a configuration hierarchy

Global configurations provide
context for links

IBM

IBM Watson IoT 21

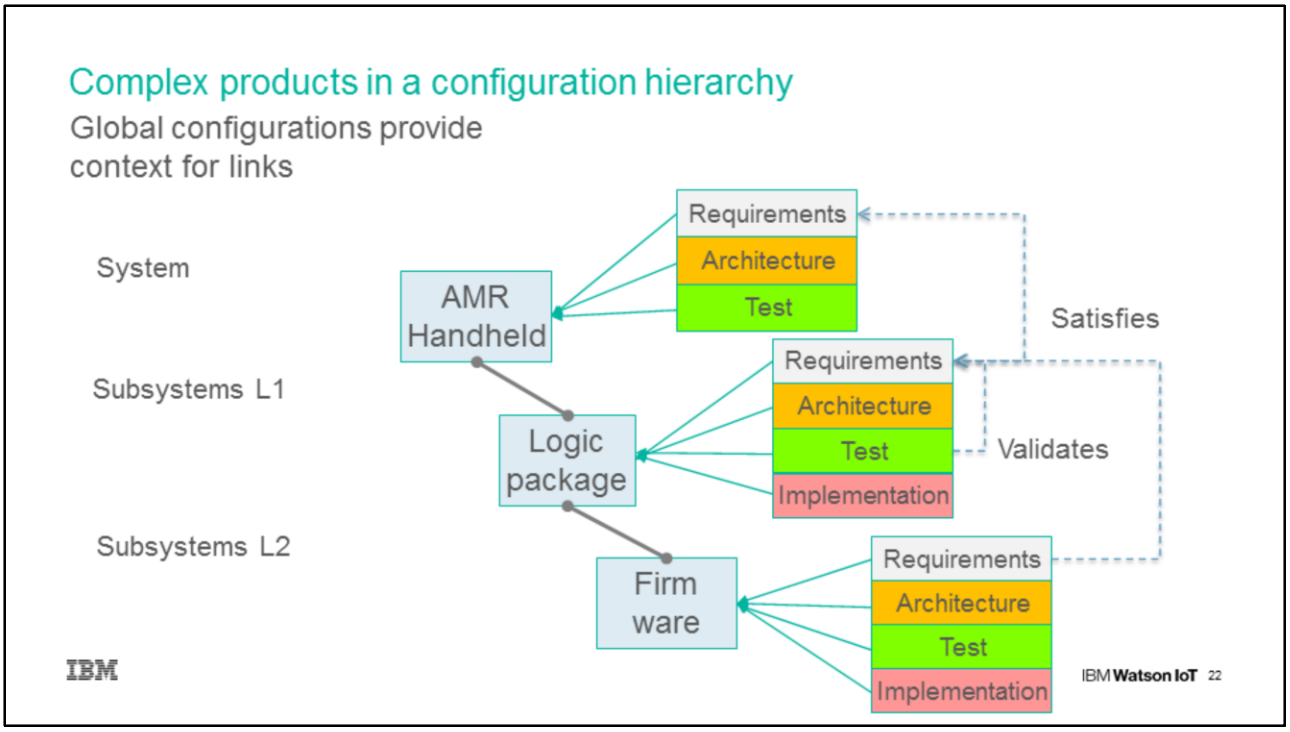
So let’s add a level 2 subsystem configuration (“Firmware”) to the Logic package configuration.

The firmware configuration has it’s own tool configurations for requirements, design, etc. … and its own baselines

Now links between the levels will resolve correctly…

in this case the satisfies link between a lower-level requirement in the firmware and a higher-level requirement in the Logic package.

Again, I’m showing only one link. In practice there may be hundreds or thousands.

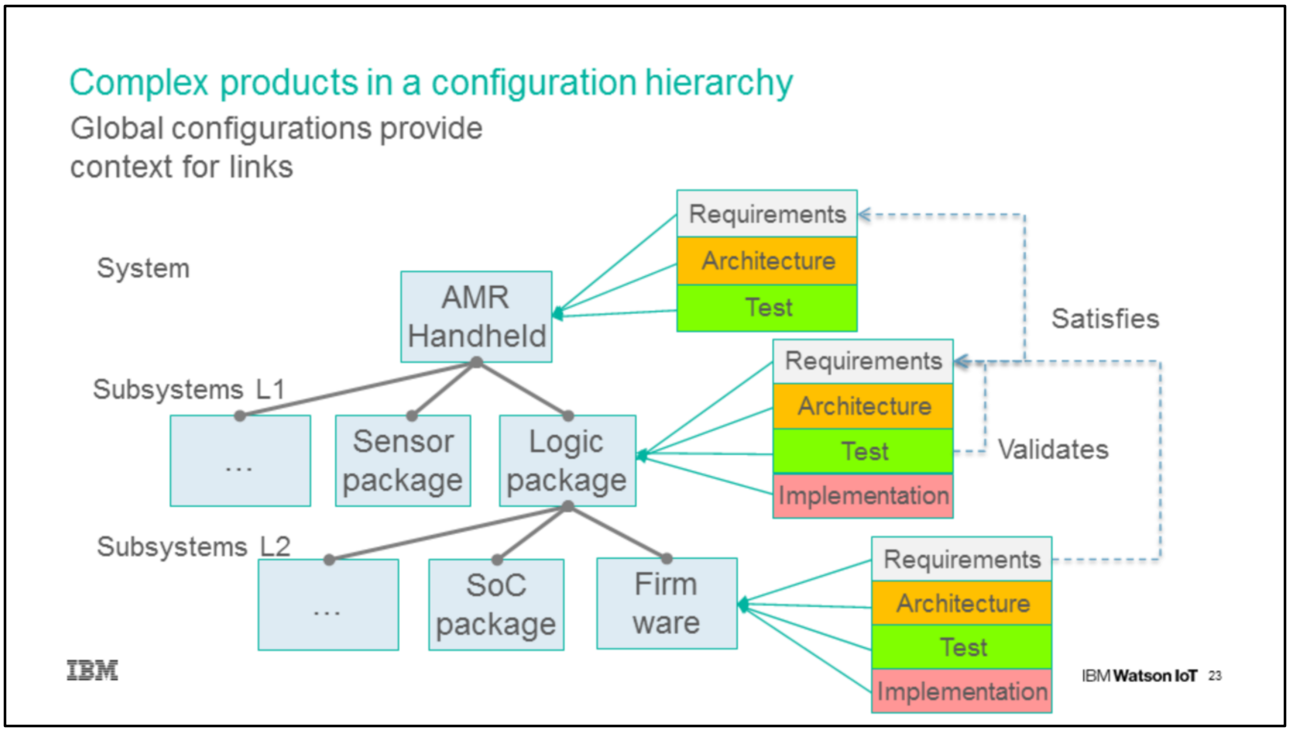


Complex products in a configuration hierarchy

Global configurations provide context for links

And this pattern continues for additional levels.

Here we added a satisfied link from a logic package requirement to the corresponding top-level system requirements



Complex products in a configuration hierarchy

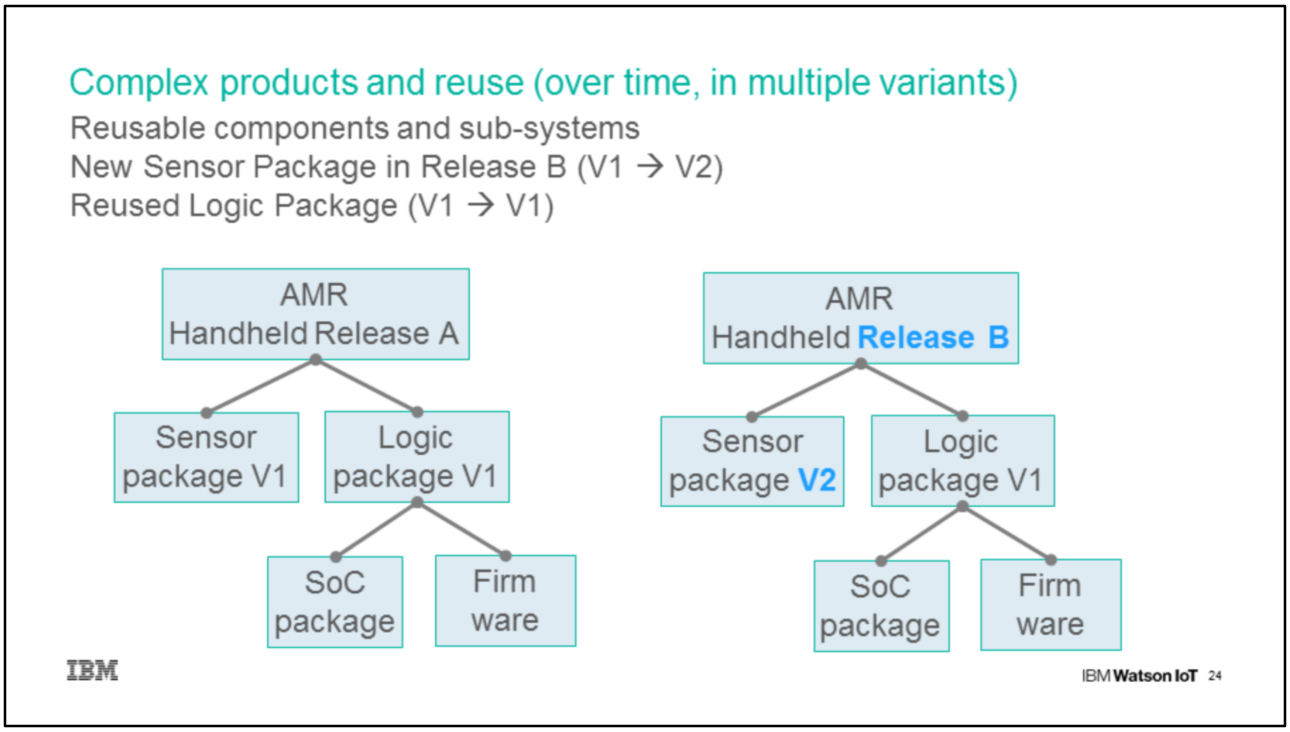
Global configurations provide context for links

And of course, there are typically many sub-systems. And there may be even more levels … 5, 6, even 10 levels deep.

You get the idea.

[following are more details about our implementation]

- We define a component as a unit of configuration; for example, the span of a baseline.
- Within each of the Jazz tools, RM, QM, RTC SCM, and the GC application, components provide this span.
- In RM and QM at the moment the each project area corresponds to one component.
- Within an RTC SCM stream there can be many components, each with its own baseline (the aggregate can be snapshotted)
- Within the GC application you can create many components, as shown in this example.


Complex products and reuse (over time, in multiple variants)

Reusable components and sub-systems
New Sensor Package in Release B (V1 → V2)
Reused Logic Package (V1 → V1)

When it's time for a new release of the manual handheld product, most components can be reused.

In this case only the sensor package has changed.

So in the Handheld Release B stream the sensor baseline is updated from V1 to V2, then a baseline is created of Handheld Release B.

In the future, someone could compare the definition of Release A and Release B to determine what changed in Release B.

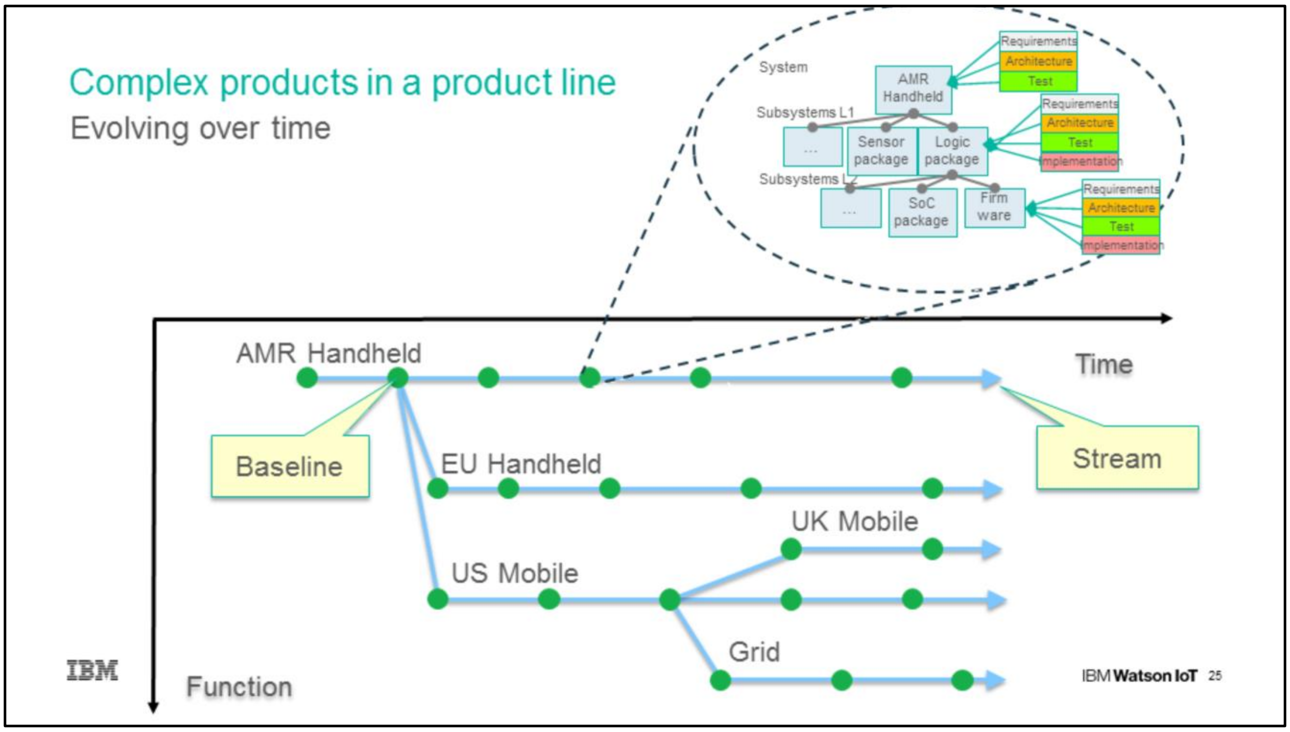
And for example, when a defect is found in the field, which products need a firmware update.

Often the different components are produced by different teams, each with their own schedules.

This approach provides a way to manage both the component release and the releases of the products that contain the component.

This example shows reuse with variation over time.

It's also possible to have reuse with variation in two products for different market segments, for example, the US and Europe.



Complex products in a product line

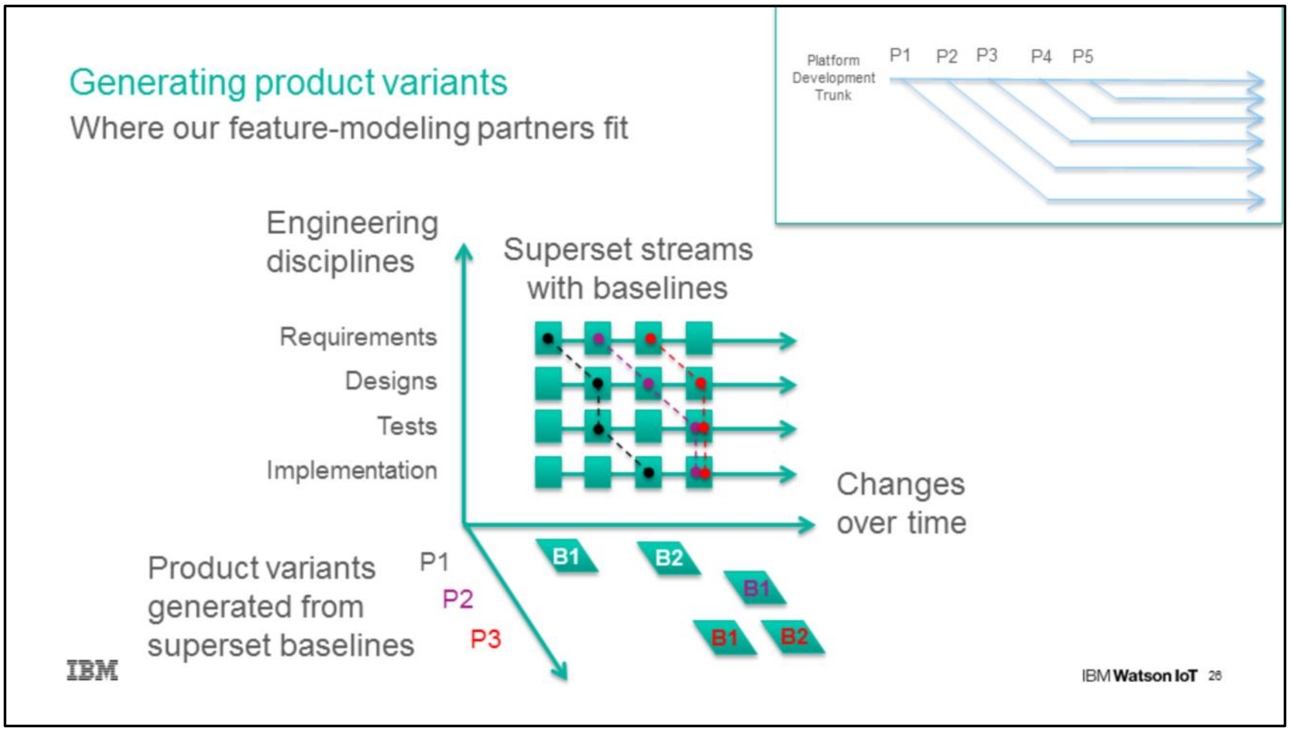
Evolving over time

So these complex products evolve over time as represented in the horizontal lines, with various baselines along the way.

And branches to create new product variants (or for any of the other reasons we've already looked at today).

To review:

- A stream evolves over time and is associated with a set of baselines
- Baselines record state in time and are immutable
- Streams reuse common artifacts and use different version where there is variability
- Artifact changes can propagate across streams
- Product variants are realized in different streams



Generating product variants

Where our feature-modeling partners fit

OK … one or two more big ideas then we are done.

This applies when you are doing platform-based development and generating product variants from that platform.

For example, when you are doing feature modeling with one of our partners.

Consider two dimensions … engineering disciplines on the vertical, and development through time on horizontal streams

Each tool has its set of baselines over time.

The content of the stream is the superset of all potential product variants

Let’s imagine we want to create three product variants.

We will do this on a third dimension, coming out from the page.

How do we know which superset baselines from each tool we should use when generating a product variant?

A global configuration specifies the answer.

In this example …

For P1, a global baseline containing only a requirements baseline was used for the first generation (B1), then all the disciplines participated in the generation at the second global baseline (B2).

For P2, we see only one generation that includes all the disciplines.

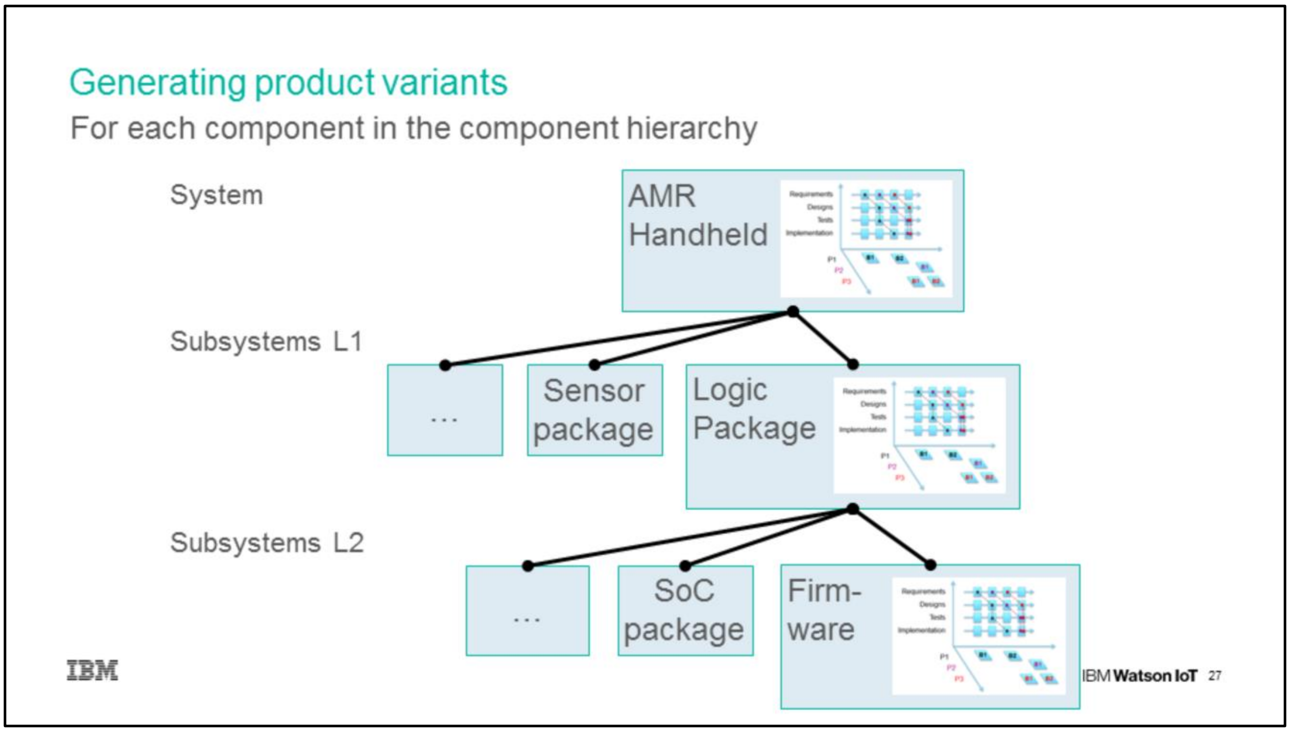
For P3 we again see one first with requirements only, then later with all disciplines.

Note that the same superset baselines of tests and implementation were used for the last generation of P2 and P3.

If the development trunk is branched for the variant just before the feature model is applied, then the system will maintain all the right links within and across the engineering disciplines.

And since the feature model is changing over time, it too needs to be under configuration management.

In fact the feature model and logs from the generation activity can be baselined in the SCM and included in a global baseline (part of the “implementation” stream).

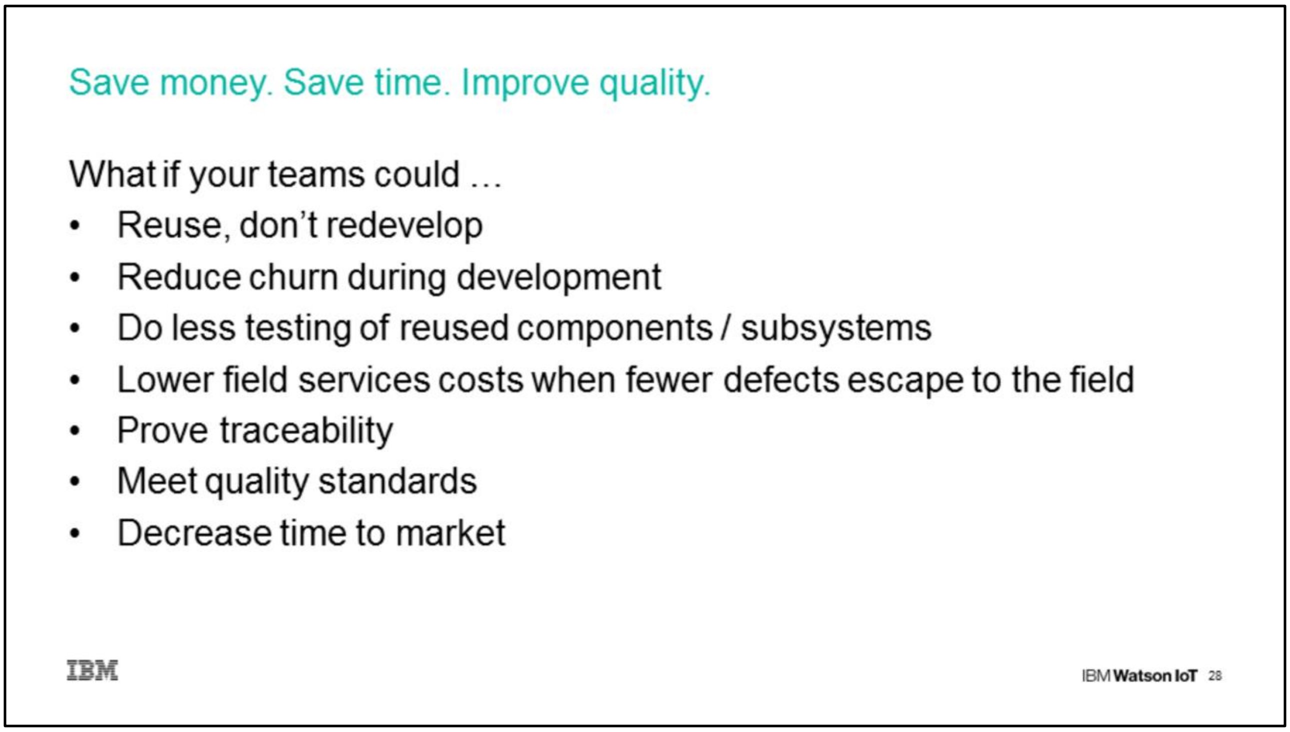


Generating product variants

For each component in the component hierarchy

Now consider that we need to generate a from the superset in each relevant component … all the way through the product hierarchy.

The use of global configurations can massively simplify the record keeping – and the automation – associated with creating each product variant.

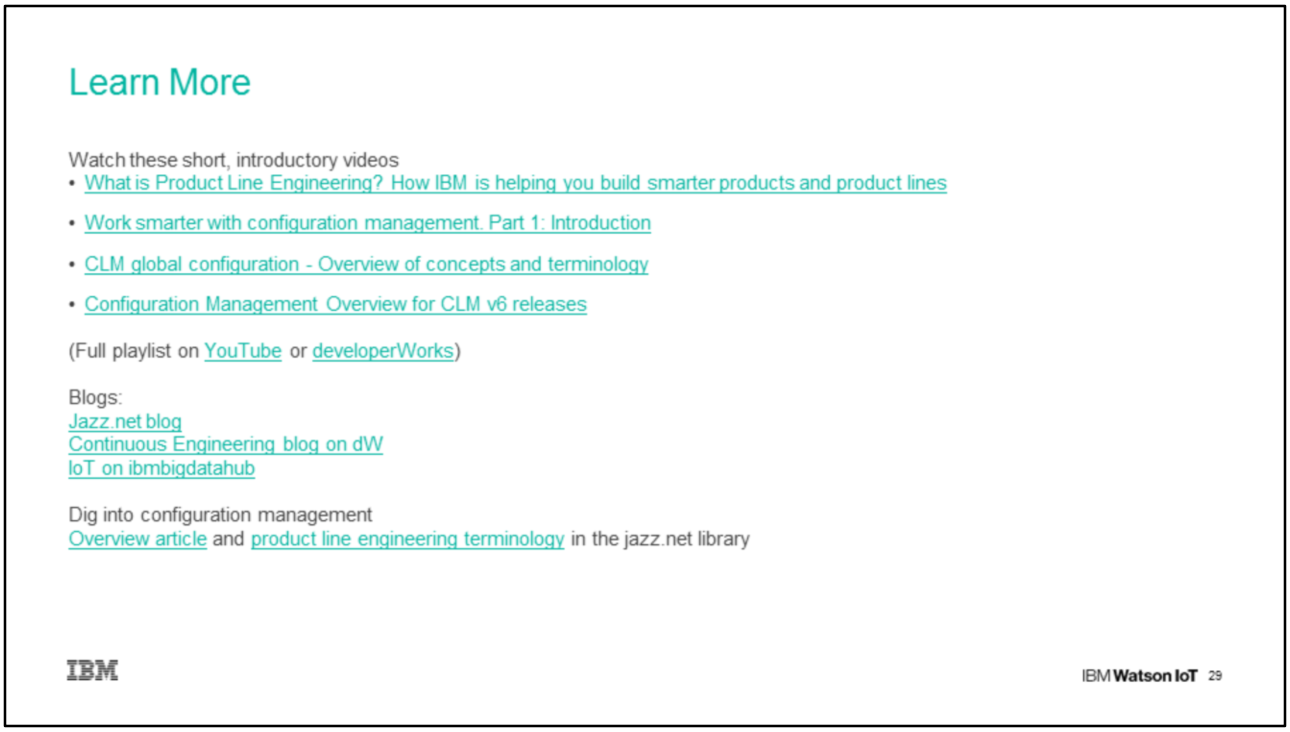


Save money. Save time. Improve quality.

What if your teams could …

- Reuse, don't redevelop
- Reduce churn during development
- Do less testing of reused components / subsystems
- Lower field services costs when fewer defects escape to the field
- Prove traceability
- Meet quality standards
- Decrease time to market

IBM



Learn More

Watch these short, introductory videos

- What is Product Line Engineering? How IBM is helping you build smarter products and product lines
- Work smarter with configuration management. Part 1: Introduction
- CLM global configuration - Overview of concepts and terminology
- Configuration Management Overview for CLM v6 releases

(Full playlist on YouTube or developerWorks)

Blogs:
Jazz.net blog
Continuous Engineering blog on dW
IoT on ibmbigdatahub

Dig into configuration management
Overview article and product line engineering terminology in the jazz.net library

IBM Watson IoT 29

Watch these short, introductory videos

- What is Product Line Engineering? How IBM is helping you build smarter products and product lines
- Work smarter with configuration management. Part 1: Introduction
- CLM global configuration - Overview of concepts and terminology

(Full playlist on YouTube or developerWorks)

Blogs:

Jazz.net blog

Continuous Engineering blog on dW

IoT on ibmbigdatahub

Dig into configuration management

Overview article in the jazz.net library

InterConnect 2016
The Premier Cloud & Mobile Conference

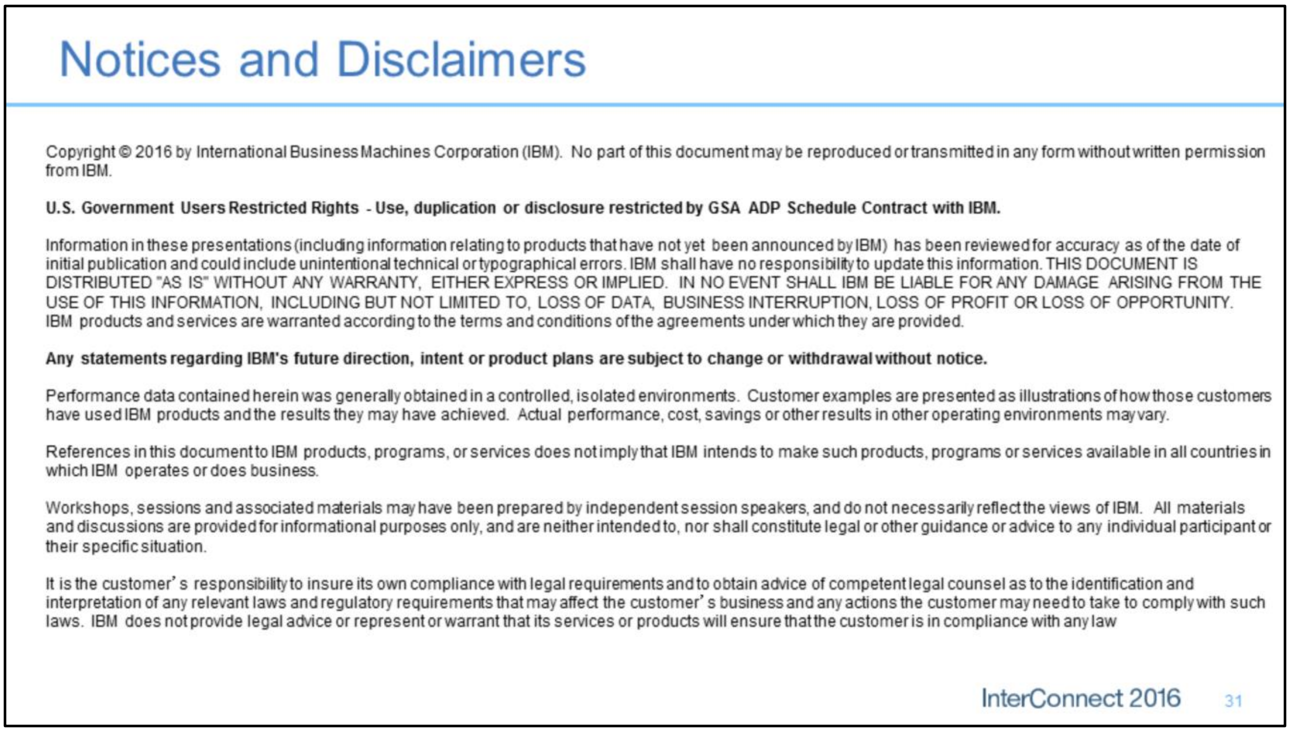
Thank You

Your Feedback is Important!

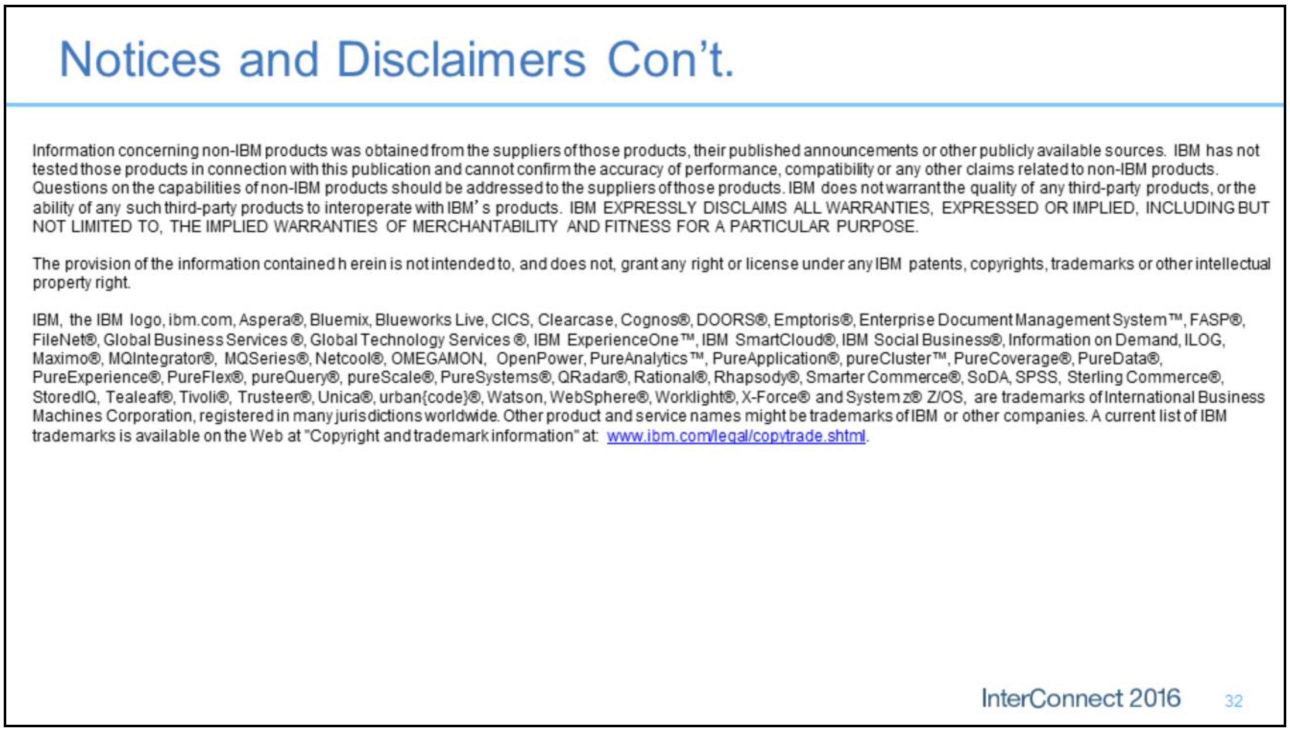
Access the InterConnect 2016 Conference Attendee Portal to complete your session surveys from your smartphone,
laptop or conference kiosk.

IBM

February 21 – 25
MGM Grand & Mandalay Bay
Las Vegas, Nevada



Notices and Disclaimers



Notices and Disclaimers Con't.