IT4IT™ Agile Scenario

Using the Reference Architecture

A White Paper by:

The Open Group IT4IT Forum Agile Work Group

February, 2016

IT4IT™ Agile Scenario

Document No.: W162

Published by The Open Group, February, 2016.

Any comments relating to the material contained in this document may be submitted to:

The Open Group, 44 Montgomery St. #960, San Francisco, CA 94104, USA

or by email to:

ogspecs@opengroup.org

Table of Contents

Boundaryless Information Flow™
achieved through global interoperability
in a secure, reliable, and timely manner

Executive Summary

This is one of a series of documents describing how to apply the IT4IT Reference Architecture, an Open Group Standard, to various different scenarios related to managing the business of IT.

This document describes the application of the IT4IT Reference Architecture to the area of Agile Development using techniques such as DevOps and Kanban.

Introduction

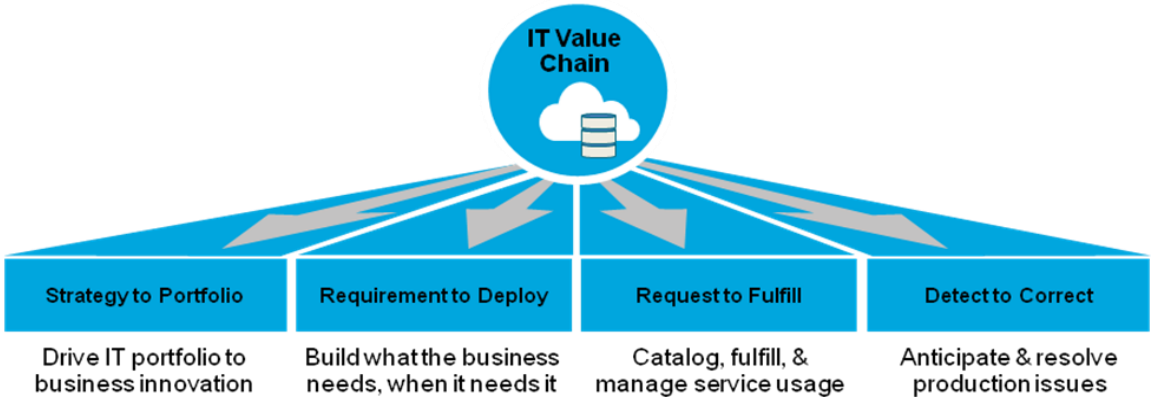


Figure 1: Value Stream Overview

The Open Group IT4IT Forum Agile Work Group is chartered with the following scope:

- To develop patterns, scenarios, and perspectives demonstrating utility of the IT4IT Reference Architecture for Lean and Agile delivery, including DevOps
- To identify specific changes to the IT4IT Reference Architecture as needed to better support Agile delivery
- To contribute to positioning IT4IT specifically with reference to SAFe, Kanban, SCRUM, and other Agile methods

This document is the Agile Work Group's first deliverable.

Version of the Reference Architecture

This document is based on Version 2.0 of the IT4IT Reference Architecture.[1]

[1] The Open Group IT4IT™ Reference Architecture, Version 2.0, Open Group Standard (C155), October 2015, published by The Open Group; refer to: www.opengroup.org/bookstore/catalog/c155.htm.

Business Context: Agile and Lean IT

The Agile movement represents many years of successful industry application, supported by robust theory, and increasingly defines both the goals of Information Technology (IT) management as well as its execution.

Agile methods and philosophy, originating from the challenges of IT, are informing business strategy well beyond the confines of traditional IT. Business strategies are turning to an emphasis on business experimentation; e.g., "Fail Fast" and "Think Big, Start Small" [Ries 2011]. These strategies require corresponding IT agility: moving from large "batches" of project requirements to the software equivalent of Lean "single-piece flow".

There are many books and other resources available describing Agile and the reader is referred to the references, especially [Humble 2011], [Reinertsen 2009], and [Burrows 2015] as initial reading.

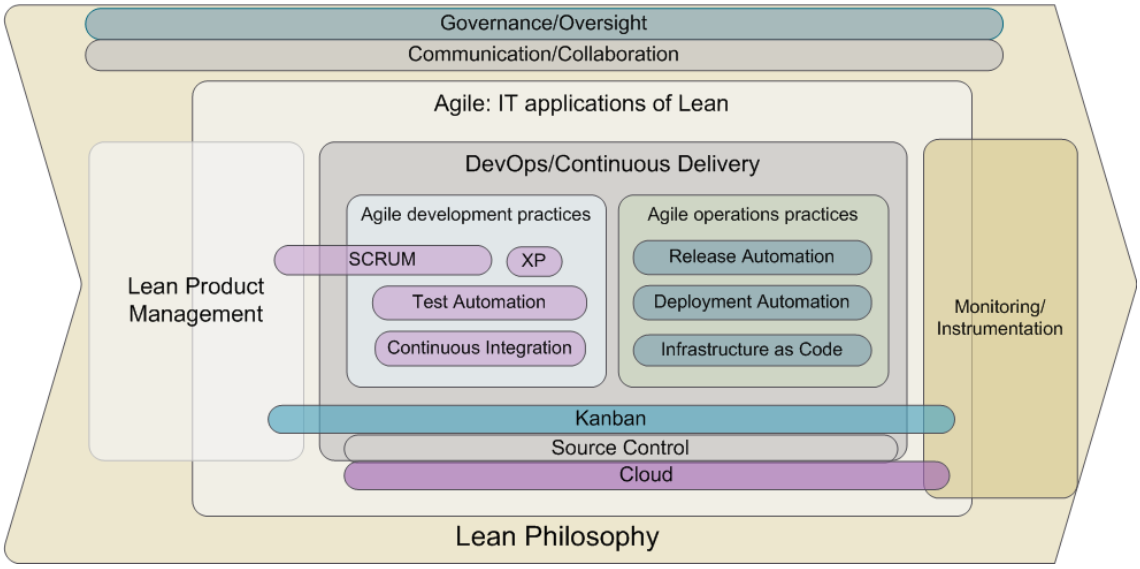


Figure 2: Lean, Agile, DevOps, and Related Conceptual Framing

"Lean" is the overall philosophical framework, generally credited to Ohno and others at Toyota [Ohno 1988], with the name "Lean" first applied by Krafcik [Krafcik 1988], and further Western development and popularization by Womack, Jones, Liker, and others [Womack 1990], [Womack 2003], [Liker 2004]. Another key related source is the "Theory of Constraints" developed by Eli Goldratt [Goldratt 1997], [Goldratt 2004].

The application of Lean philosophy to software engineering is closely related to the term "Agile," as in "Agile Development". The often-cited "Agile Manifesto" [Alliance 2001] states:

"We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation

- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more."

The Poppendiecks were early appliers of Lean principles to software development [Poppendieck 2003], [Poppendieck 2007]. David Anderson has popularized the Toyota Production System term "Kanban" as a form of collaborative work management for software product teams, focused on self-organizing teams strongly concerned with flow and limiting work in process [Anderson 2010]. Rational Unified Process author Dean Leffingwell has developed the Scaled Agile Framework (SAFe) [Leffingwell 2010].

Leffingwell, the Poppendiecks, and Anderson all cite the influence of Don Reinertsen [Reinertsen 1997], [Reinertsen 2009]. Reinertsen specializes in Lean product development and has developed a set of theoretical perspectives based on economics, queuing theory, statistics, and related topics.

The Phoenix Project is a notable exploration of the Lean, Agile, and DevOps themes [Kim 2013].

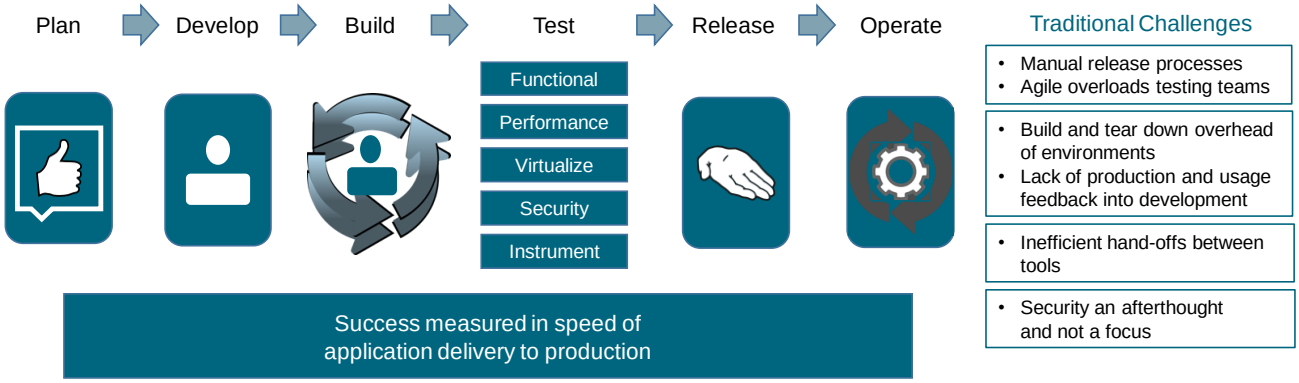


Figure 3: Challenges of the IT Value Chain

As seen in Figure 3, the IT Value Chain has a number of challenges. Manual processes, silo walls between development and operations, and poorly integrated tools are some of the problems seen in the legacy methods to IT product development.

Within the general framing of Lean and Agile methods for IT delivery, this scenario reflects and explores several major Agile currents:

- DevOps
- Bi-modal IT (Gartner)
- Kanban
- Scaled Agile Framework (SAFe)

DevOps

"DevOps" (the DevOps response) is a set of innovative practices emerging in reaction. Coined as a portmanteau representing "Development" and "Operations" (Patrick Debois is generally credited with originating the term), it represents the specific concerns of accelerating flow and software delivery from ideation through production, extending Agile philosophy from its traditional home in software development into operational domains. Fast feedback is a critical objective.

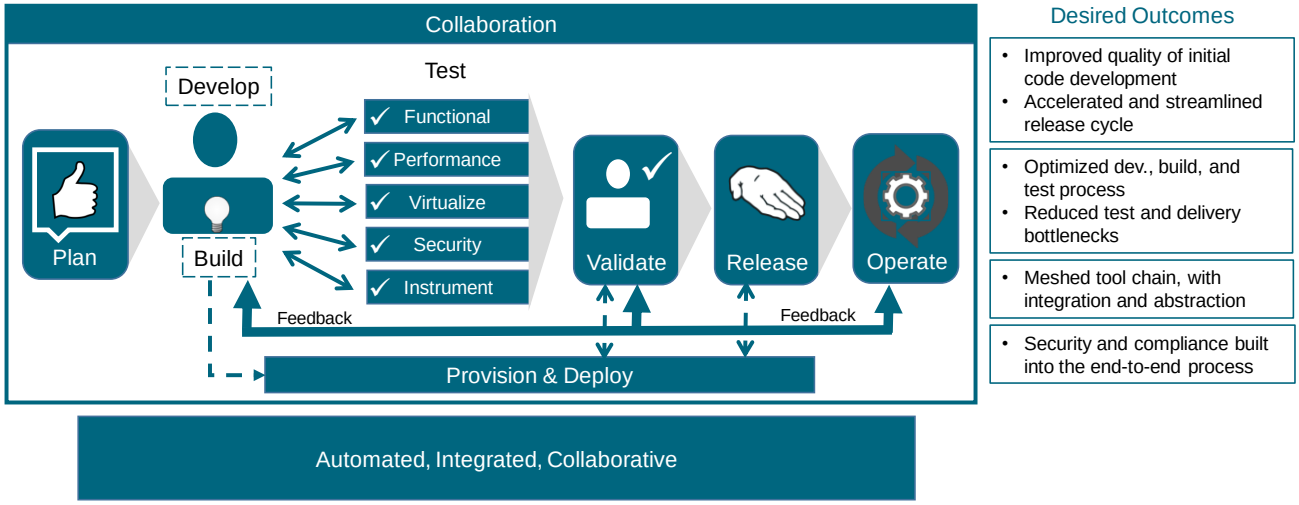


Figure 4: The DevOps Response

Hammond and Allspaw's "10 Deploys per Day: Dev & Ops Cooperation at Flickr" is a key document [Allspaw 2009] and Humble and Farley's "Continuous Delivery" is the first comprehensive text on the subject [Humble 2011].

For the purposes of this document, Kanban and DevOps, as with other terms like SCRUM and Extreme Programming, are seen as specific manifestations of Agile, which is in turn an IT-centric manifestation of Lean.

Some argue that DevOps must transcend Agile, as the broader term. However, if the Agile Manifesto does *not* apply to IT operations, it is difficult to see how DevOps can succeed in its goals. Such conceptual questions are of course difficult to settle definitively.

DevOps Definition

DevOps is a way of collaborating and industrializing using highly automated approaches to deploy solutions that evolve as fast as your business needs it. By adopting DevOps an organization can dramatically improve the value delivered by its business. The team-centric DevOps ethos tears down traditional silos to tightly integrate business, development, and operations to drive agility and service delivery excellence across the entire lifecycle.

IT4IT™ Agile Scenario

DevOps Maturity Model (DMM)

To achieve the DevOps vision, the following maturity model can be defined:

Level 1: Basic – At the basic level, the following characteristics are apparent:

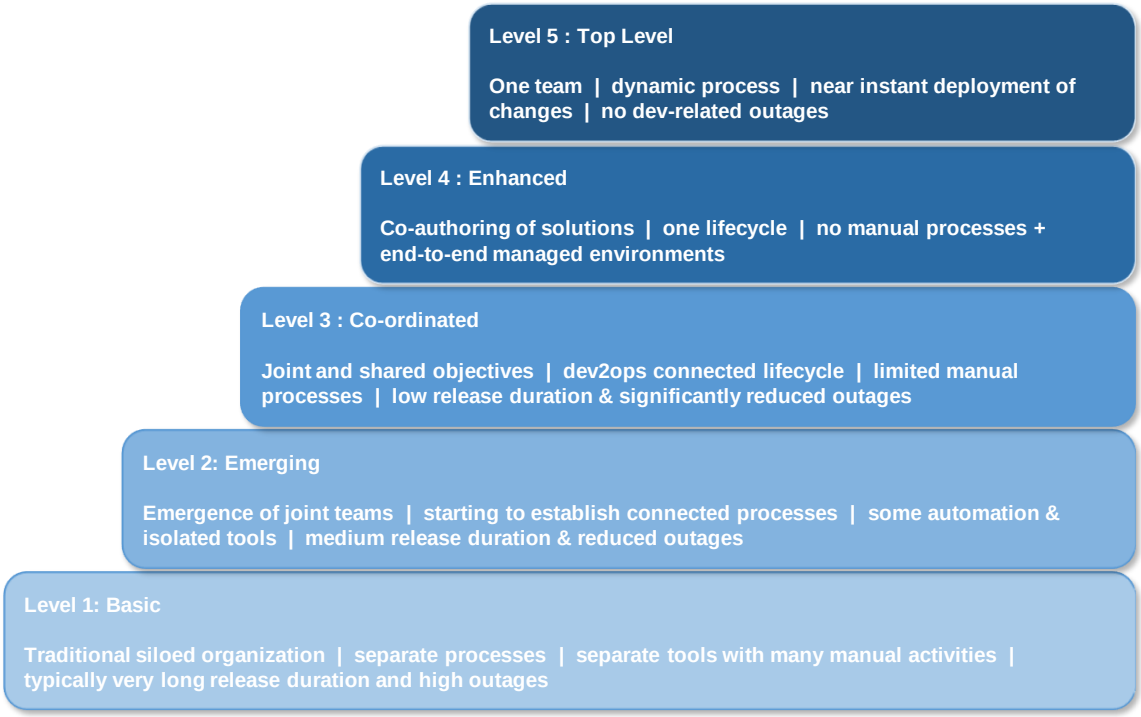


Figure 5: DevOps Maturity Model (Notional)

- People: Separate strategy, design, development, testing, and live operations teams. Complete lack of terms of references. No joint sessions, get-togethers. Teams focus on their own direct targets and objectives only. No joint or shared objectives and no overall reward system. People only feel accountable for their immediate area – no common or overarching ownership.
- Process: Separate and disconnected processes are place which are *ad hoc*, reactive, and chaotic. No common end-to-end process framework, no common sign-off criteria or any joint solution design characteristics that support appropriate "–ilities" (availability, stability, flexibility).
- Tools: No automation tools, majority of activities are manual, *ad hoc*, and unplanned. No integration between hardware provisioning, operating system installation/configuration, and middleware/application-related provisioning/installation. No sharing of joint configuration information with all information being stored and retained in different repositories.

Level 2: Emerging – At the emerging level, the following characteristics are apparent:

- People: Limited changes to basic – still very siloed and separate teams with no single team/person taking end-to-end responsibility. Developers mainly focus on functional requirements with very limited focus on non-functional requirements. However, there is the emergence of some shared/joint touch points where some developers and some operational staff engage.

- Process: Limited changes to basic – there are some attempts to establish better managed processes; however, these are restricted to specific environments only; i.e., covering development or User Acceptance Testing (UAT) only.
- Tools: Some minor changes to basic, mainly targeted at developing automatic scripts covering hardware and operating system-related aspects. As per basic these are mainly targeted at the development environments. Most other environments such as testing, training, System Integration Testing (SIT) are being manually installed.

Level 3: Coordinated – At the coordinated level, the following characteristics are apparent:

- People: Mainly siloed organization; however, lead architect/lead designer(s) increase their scope to also include operational aspects. Joint sessions are held to increase wider visibility – for instance, key operational staffs are actively engaged in the design and build phase. Developers are also measured on operational characteristics.
- Process: Still mostly separate processes covering the entire solution lifecycle; however, there are some joint process points where development and operational aspects are jointly covered. Better understanding of the entire environment set-up and characteristics.
- Tools: Most of the development environment set-up is being created automatically. Only application-related components are manually installed.

Level 4: Enhanced – At the enhanced level, the following characteristics are apparent:

- People: Joint teams that cover the entire solution lifecycle. Lead architect owns entire solution including functional and non-functional covering design, build, test, and run.
- Process: Single overall process covering the entire solution lifecycle – from design, build, test to run. Clear visibility of all projects/changes that are at different stages with a clear view on all compliance levels (functional and non-functional). Clear view on the entire environment set-up and characteristics.
- Tools: Most of the development environments; set-up, testing, and live are being created automatically. This now covers servers, operating system, operating system near, as well as most middleware and application-related components.

Level 5: Top Level – At the top level, the following characteristics are apparent:

- People: One team, co-located and extensive collaboration and knowledge sharing.
- Process: Single overall process covering the entire solution lifecycle – from strategy, planning to design, build, test to run.
- Tools: All environment set-ups are being created automatically from a single repository. This covers all aspects – servers, operating system, operating system near, as well as all middleware and application-related components. No manual processes in place.

DevOps Implementation Model (DIM)

DevOps is hampered by a number of key aspects:

- The lack of a standard definition for DevOps has created confusion for Infrastructure and Operations (I&O) leaders, trying to adopt this philosophy [Gartner 2014]
- There is no standardized or simplified approach regarding the adoption of DevOps by an enterprise I&O leader, causing confusion about how and where to start [Gartner 2014]
- Each DevOps implementation is unique and every customer requires a customized approach [Gartner 2014]

In practice, tools, methods, and technologies are seldom deployed on green-field sites and the key to success is to:

- Define a clear target
- Establish a clear transformation plan
- Actively manage the plan execution

DevOps implementation should not merely be perceived as deploying a new tool like CodeStream or Docker. It should be viewed from a wider perspective and should be planned and executed in an efficient manner. Poorly planned DevOps implementations may result in significantly higher costs.

A DevOps implementation starts with creating a rationale business case, mapping a way for code migration between environments (considering people, processes, and technology), and placing focus on the target. Understanding the "as-is" scenario, mapping the "to-be" scenario, and estimating the benefits of moving to the "to-be" are critical for success. A DevOps implementation should be backed by a strong business case. Every environment does not benefit from full or partial DevOps deployment. For instance, environments with little change requirements may not benefit from DevOps implementation at all.

DevOps aims to reduce the impact of changes, to reduce cost, and minimize impact to the live services. As applicable to every change project, the decision to change culture or processes and to deploy the right tools must be backed by a strong business case. Many businesses struggle to take the right decisions at this stage.

To estimate the benefits of DevOps implementation within their environment, they should analyze the existing situation; the existing tools, processes, resources, and their skills.

IT4IT™ Agile Scenario

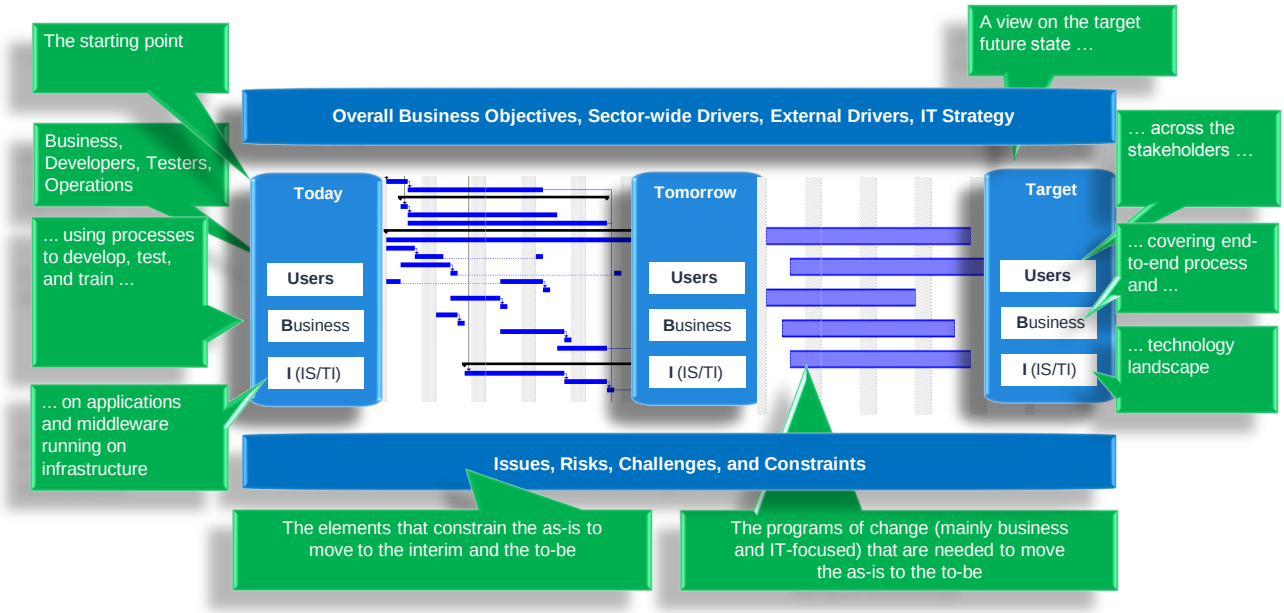


Figure 6: Transformation Over Time

Then, as the "snowflake" point in Gartner's paper [Gartner 2014], each client context is different and what works for one might not work for the next.

A strong business case for DevOps is the prime catalyst which should be thrown in for demonstration of the true value across the stakeholder value chain. That being said, there are numerous factors to be considered where costs are involved. Some of the major factors which impact costs are:

- The business processes-related changes and the changes in the way the information flows between humans
- The inconsistencies across process and tools impacting the changes towards new standard operating procedures
- The methods of automated environment provisioning through increased virtualization and cloud adoption
- A revamp of the change management process and the relevant systems which accounts for the ways in which deployment and configuration move towards being more automated than manual

DevOps Implementation Approach (DIA)

DevOps implementations do vary from client to client as there is no single silver bullet. To ensure successful adoptions, clients should execute the above activities. However, there are key enablers to consider when starting a DevOps engagement. Applying the experience from previous DevOps engagements, there are eight key aspects to consider:

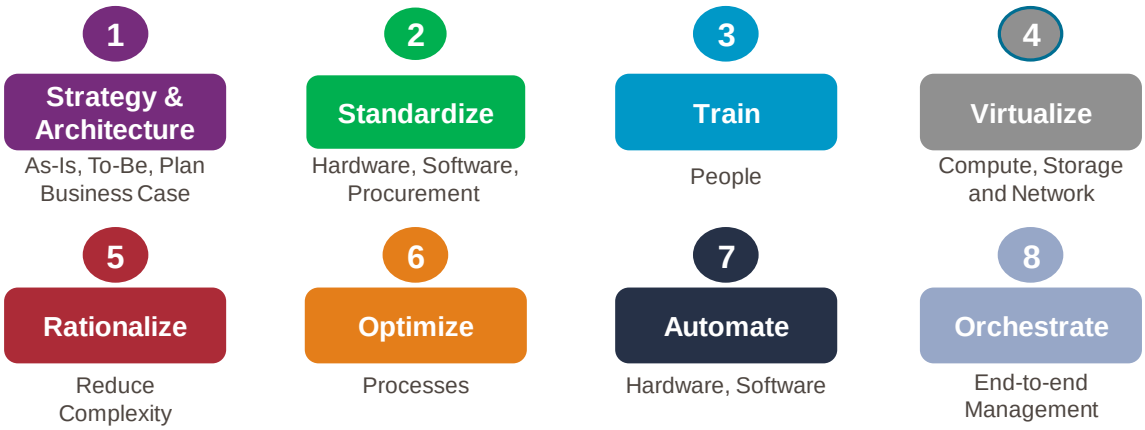


Figure 7: DevOps Implementation Concerns

1. Strategy and Architecture: Understand where you are; covering business, applications, information, and infrastructure. Know where you want to be and have a plan that drives real value.
2. Standardize: Focusing on hardware, hardware-near software (like operating system), as well as middleware and try to standardize as much as possible and reduce overall product variances.
3. Train: Focus on your people; up-skill and train in basic but also advanced subjects covering technology-specific, generic, and soft skills.
4. Virtualize: Not an absolute must; however, a great accelerator automating processes is abstracting from underlying hardware covering not just compute but also storage and network.
5. Rationalize: Reducing overhead will allow for better focus as well as reduce complexity.
6. Optimize: Eradicate unnecessary steps in key processes, in particular when these are of manual or semi-manual nature.
7. Automate: A key foundation block for DevOps – automate as much as possible. Moving in to a continuous deployment cycle will be accelerated if processes are automated.
8. Orchestrate: Moving, starting, creating, and deleting services in an orchestrated fashion; for instance, via a central capability, will increase the impact of DevOps.

To determine DevOps success it is necessary to be able to measure the impact across people, process, and technology for a successful implementation.

IT4IT™ Agile Scenario

Bi-Modal IT

Bi-modal or "two-speed" IT is a recent Gartner hypothesis that IT capabilities can usefully be separated into two modes: a slow, more deliberate mode for core systems and a faster, more Agile mode for systems requiring greater agility. The advantage of the IT4IT Reference Architecture is that one architecture supports various "speeds" of IT delivery.

Kanban

Kanban [Anderson 2010], [Burrows 2015] is a term for both a specific technique of visually managing work-in-process, as well as a broader methodology promoted by David J. Andersen. The method calls for self-organization, limited WIP, culture change, and other goals consistent with Lean and Agile approaches.

Scaled Agile Framework (SAFe)

SAFe [Leffingwell 2010], [SAFE 2015] is a more prescriptive framework that formalizes to some degree concepts such as epic, story, backlog, release train, and so on. SAFe is perhaps unique among Agile schools of thought in explicitly recognizing a role for architecture.

Agile and the IT4IT Framework

As the IT4IT framework is process and methodology-agnostic, DevOps can be mapped into the Reference Architecture (Figure 8, Figure 9).

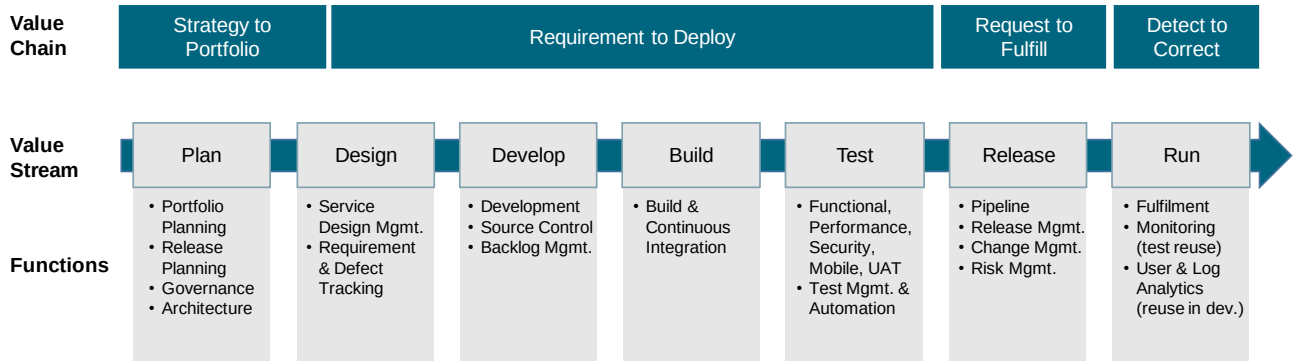


Figure 8: DevOps Across the IT Value Streams

IT4IT™ Agile Scenario

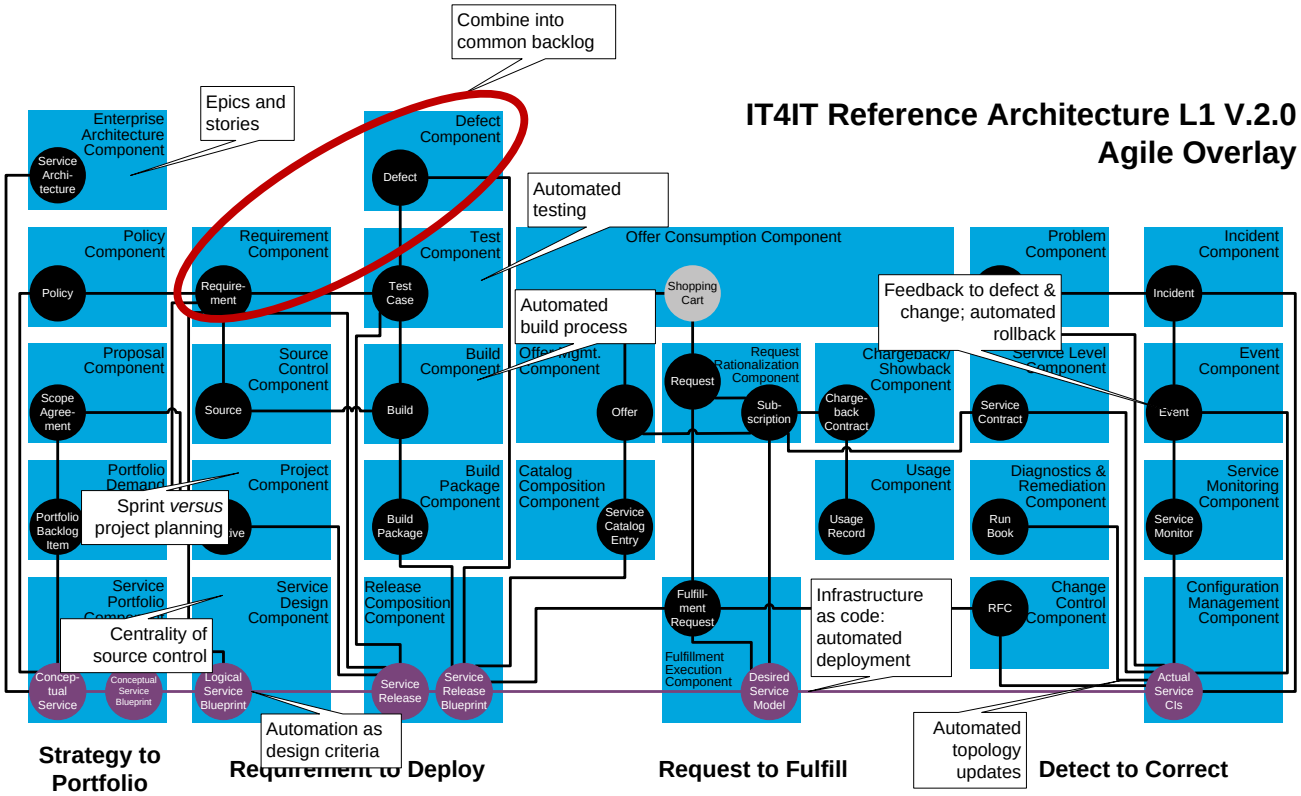


Figure 9: Agile Concerns Across the IT4IT Reference Architecture

As we can see in Figure 9:

- Proliferation of queues
- Enterprise Architecture and portfolio backlog
- Automation as design criteria
- Requirements as defects as combined product backlog
- Automated build process
- Automated testing
- Infrastructure as code and automated deployment
- Event feedback to product backlog
- Automatic configuration updates

There may be questions from the Agile community about the role, purpose, and desirability of any framework purporting to address some aspect of Agile. However, Lean and Agile authors, case studies, and presentations provide a wealth of tangible objectives that can inform the development of a reference architecture.

IT4IT™ Agile Scenario

The IT4IT standard is not a methodology. It is not primarily concerned with how work is done. This distinguishes it from CMMI, ITIL, SAFe, SCRUM, etc. It is a framework and prescriptive in the sense of it being a reference model. It is prescriptive in the same sense that design patterns are prescriptive – they prescribe a solution, but still need to be applied correctly.

Frameworks can be implementation methods or reference models. The IT4IT standard is a reference model and, as such, is similar to the patterns literature [Buschmann 1996], [Fowler 2003], [Fowler 1997], [Gamma 1995], [Hay 2006], [Larman 2002].

Without question, there are emergent consensus points within the Agile movement that can be stabilized into a reference architecture. The central importance of source control is one example, and the IT4IT Reference Architecture has been updated accordingly (the "Service Development" component was renamed the "Source Control" component). The need to universally identify and manage queues [Reinertsen 2009] is another overriding concern, and the basis of Kanban and Queueing (on page 54).

The IT4IT standard is primarily intended for large enterprise IT organizations. The intent is to enable loosely-coupled, decentralized Agile teams by prescribing simple IT coordination interfaces supporting accepted Agile implementation patterns, informed by documented Agile principles, providing end-to-end traceability of IT services.

The IT4IT standard may become prescriptive in a tighter sense when applied to IT management tool interoperability. However, the IT4IT standard is *not* prescribing particular tools, practices, or methods.

(Note: Agile/Lean has a strong emphasis on culture and organizational change management. Such concerns are outside the purview of the IT4IT standard in general.)

Business Goals

It is by now well established that the IT Value Chain is best seen as a product development and management process, as opposed to a truly repetitive production process [Reinertsen 2009]. As discussed in the previous section, Reinertsen, Poppendieck, and others have developed a variety of well documented and empirically proven principles that characterize the Agile/Lean approach for product development. These apply to software development, product development, and business development at its most general.

Essential Agile requirements can be inferred from these classic questions from Mary and Tom Poppendieck:

- How long would it take your organization to deploy a change that involved just one single line of code?
- Do you deploy changes at this pace on a repeatable, reliable basis? [Poppendieck 2007, p.92]

The Kanban movement addresses these business goals by limiting work in process through simple and effective shared visual models ([Anderson 2010], [Kniberg 2011], [Burrows 2015]) while the DevOps movement seeks to satisfy these business goals by accelerating and making the end-to-end software pipeline as automated as possible while bridging the cultural divide between Development and Operations ([Allspaw 2009], [Humble 2011], [Limoncelli 2014]).

IT4IT™ Agile Scenario

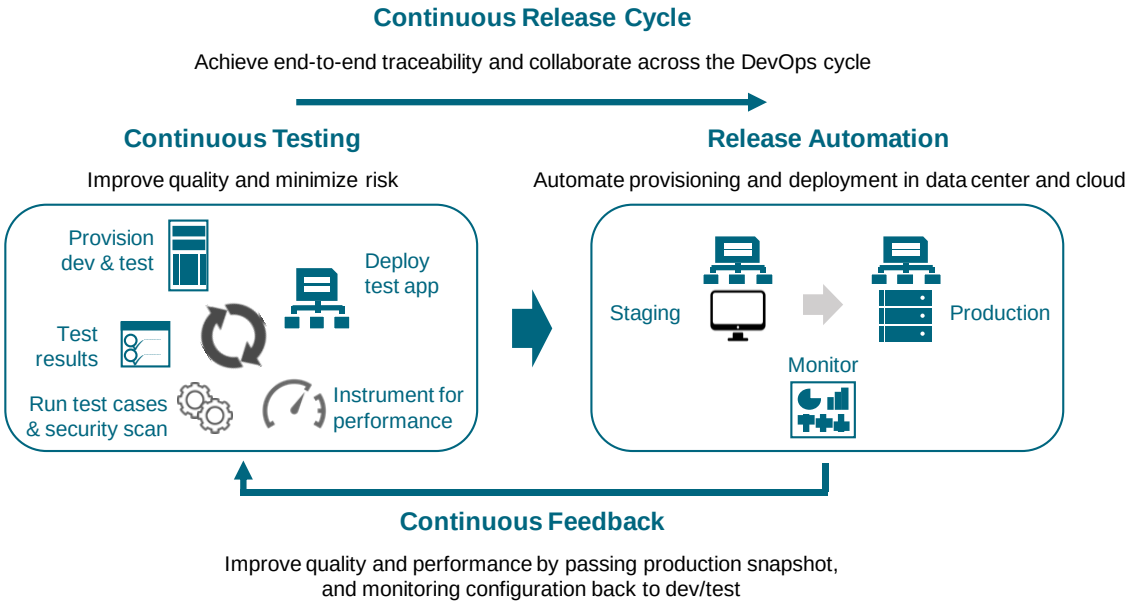


Figure 10: Importance of Feedback

Feedback is a critical objective in product development (e.g., software and systems development); as product development is essentially the generation of information [Reinertsen 2009].

Examination of various Agile sources supports the following matrix of business objectives for the Agile scenario:

(Note: There is one set of unified business goals for the Agile scenario as a whole, but each sub-scenario has its own requirements.)

ID	Goals	Notes
G.0.1	**Correctly apply economics**	[Reinertsen 2009]
G.0.1.1	Understand overall economic impact	[Reinertsen 2009]; e.g., leveraging beneficial variability and uncertainty
G.0.1.2	Understand cost of delay and value of information	[Reinertsen 2009], [Hubbard 2010]
G.0.1.3	Understand metrics as economic proxy variables	[Reinertsen 2009], [Hubbard 2010]
G.0.2	**Avoid waste**	[Womack 2003]
G.0.2.1	Limit work-in-process	[Kim 2013], [Goldratt 2004]
G.0.2.2	Reduce task switching or multi-tasking	[Kim 2013]
G.0.2.3	Eliminate non-value-add process	[Ohno 1988]

IT4IT™ Agile Scenario

ID	Goals	Notes
G.0.2.4	Minimize manual work	[Humble 2011]
G.0.2.5	Identify, make visible, and manage all work queues	[Kim 2013]
G.0.2.6	Avoid the antipattern of “optimizing” for utilization	[Goldratt 2004]
G.0.3	**Maximize information**	[Reinertsen 2009]
G.0.3.1	Seek fast feedback	[Reinertsen 2009]
G.0.3.2	Invest in options where possible (e.g., parallel development to explore different product directions simultaneously)	[Burrows 2015], [Scotland 2010]
G.0.3.3	Ensure information visibility across entire pipeline	
G.0.4	**Manage for flow under uncertainty**	[Reinertsen 2009]
G.0.4.1	Manage batch size	[Reinertsen 2009]
G.0.4.2	Synchronize complex activities	[Reinertsen 2009]
G.0.4.3	Minimize variability in repetitive processes	
G.0.4.4	Accept and manage variability in creative processes	
G.0.5	**Build effective culture**	
G.0.5.1	Manage for end state and intent	[Reinertsen 2009]
G.0.5.2	Encourage self-organization	[Anderson 2010]
G.0.5.3	Tolerate failure; maximize information captured and focus on problem-solving	[Limoncelli 2014]
G.0.5.4	Employ proven human motivators: autonomy, recognition, collaboration, flow	
G.0.5.5	Avoid demotivators: control, blame, non-value-adding work	
G.0.6	**Build effective software pipeline**	
G.0.6.1	Continuously integrate and test source code and system functionality at appropriate levels (e.g., unit, integration, and system)	[Duvall 2007]
G.0.6.2	Maintain complex system functionality in an “always shippable” state	[Allspaw 2009]

IT4IT™ Agile Scenario

ID	Goals	Notes
G.0.6.3	Fully automate the repetitive aspects of code development, build, testing, and deployment	
G.0.6.4	Seek end-to-end integration of “development” and “operations” tasks, resources, communication, and culture	
G.0.6.5	Build a product-centric mindset where the overriding goal is software functioning correctly in production	[Alliance 2001]
G.0.6.6	Limit variability by using consistent configurations, toolsets, and resources in building all environments across the pipeline	[Duvall 2007]
G.0.6.7	Roll back failed changes immediately with robust and well tested, automated approaches	

Portfolio and Product Backlog

Portfolio Backlog Scenario

Overview

Portfolio management is a common model for IT investment management [Kaplan 2005], [TSO 2011], [Maizlish 2005]. Portfolios are logical, large-grained, investment categories containing products (IT services) which are managed throughout the product (IT service) lifecycle.

The strategic plans and roadmaps for the business are *managed* at the portfolio level (portfolio backlog), and *detailed* at the product (IT service) level within the portfolios. SAFe uses the term "epic" [SAFE 2015]. User requirements, enhancements, open defects, improvement opportunities, and IT demand are tracked at the product (IT service) level within a product backlog (alternatively termed program backlog) which exists throughout the lifecycle of the product (IT service).

Management of portfolio backlogs (strategic demand/roadmaps) should be consistent across all portfolios. Likewise, product (IT service) backlogs should be managed consistently across business units. In this way, staff who support various portfolios and products use the same tools and processes, thereby reducing total time/cost on projects since different project delivery staff could be assigned to any project/effort. Managing consistently also enhances transparency.

As product stakeholders (e.g., "the business") identify needs, they work with product delivery (e.g., "IT") to validate those business needs against the service portfolio to determine whether the needs could be met by enhancing existing IT services, or if new IT services would be needed. The business works with IT to add the business need (portfolio demand) as a portfolio backlog item for a particular business unit (portfolio).

Product owners and delivery teams collaborate to determine when funding is available to allow the initiation of work (project). At that time they will approve work to be performed for one or more portfolio backlog items. The portfolio backlog items will be linked to specific products (IT services) by linking to requirements (product backlog items). Once the project funding is approved, work will begin to create or modify the product (e.g., an IT service).

Requirements

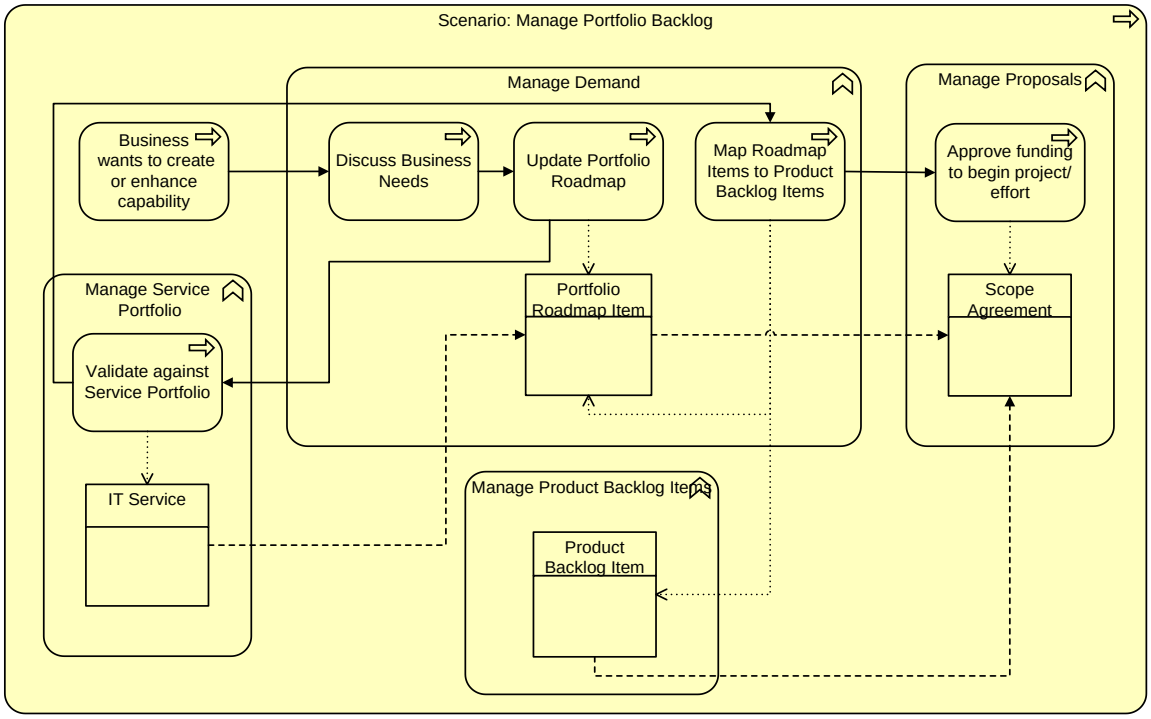
ID	Goals
PFBG.01	Utilize an integrated suite of tools and aligned processes for all efforts across the IT organization regardless of approach (methodology) or customer (business unit) resulting in lower training costs and increased flexibility in staffing initiatives.
PFBG.02	Manage and maintain authoritative repositories for key data elements within the IT organization.
PFBG.03	Visibility into planned, in-progress, and completed work across the IT organization.
PFBG.04	Roadmaps are derived from the portfolio backlog and should exist for every portfolio with a focus on the current fiscal year, view into the last fiscal year, and view into the next fiscal year.
PFBG.05	A product backlog exists for each IT service throughout the life of that IT service.

IT4IT™ Agile Scenario

ID	Goals
PFBG.06	The product backlog supports any software development methodology (i.e., Agile, waterfall, RUP).
PFBG.07	Product backlog contains all proposed changes to an IT service (i.e., user stories, enhancement requests, defects).
PFBG.08	Project delivery begins after funding has been approved by the applicable governing group.
PFBG.09	Ability to capture business needs in the portfolio backlog as a portfolio backlog item: • Portfolio • Summary • Description • Status • Priority • Planned fiscal year • Rough order of magnitude effort estimate
PFBG.10	Ability to link a portfolio backlog item to one or more conceptual services: • Portfolio backlog item • Conceptual service
PFBG.11	Ability to link a portfolio backlog item to one or more product backlog items: • Portfolio backlog item • Product backlog item (requirement)
PFBG.12	Ability to link an IT project to a portfolio backlog item(s): • IT project (scope agreement) • Portfolio backlog item

IT4IT™ Agile Scenario

Process Flow



PFBREQ.01: Process requirements record a business need in the portfolio backlog

The business identifies needs during annual planning and records those efforts in the roadmap (portfolio backlog). Annually, a budget is agreed by the business and the roadmap is updated to reflect what work is planned for the current fiscal year and what work will be moved to future fiscal years. Throughout the year, additional needs may be identified, in which case the business works with IT to update the roadmap and potentially defer existing planned work. The roadmap is kept in priority order as funding may be changed throughout the year and additional work may be brought into a given year, or pushed to a future year.

In some Agile shops, annual planning may not occur and the portfolio backlog is updated as business identifies needs. As work is planned, portfolio backlog items will be moved or linked to a program or team backlog and worked by the Agile teams.

PFBREQ.02: Roadmap items are validated against IT services

As business needs are identified, IT will work with the business to understand if an existing service(s) could be modified to meet the need, or if a new service(s) is needed to meet the need. As new services are conceived, portfolio backlog items may be linked to the new services.

PFBREQ.03: Map portfolio backlog items to product backlog items

IT links portfolio backlog items to one or more product backlog items (user stories). This allows traceability of product backlog items to the scope agreement when one or more portfolio backlog items are approved to work.

PFBREQ.04: Approve portfolio backlog item to be worked

The business reviews the roadmap (portfolio backlog) and approves funding for work to begin. The business may delegate funding authority to the product team and trust that the product team is focused on delivering business results.

The roadmap items are linked to one or more products (IT services) and the corresponding product backlog items for those IT services. The business approves a project, which is linked to the appropriate portfolio and product backlog items, to begin and with an initial budget.

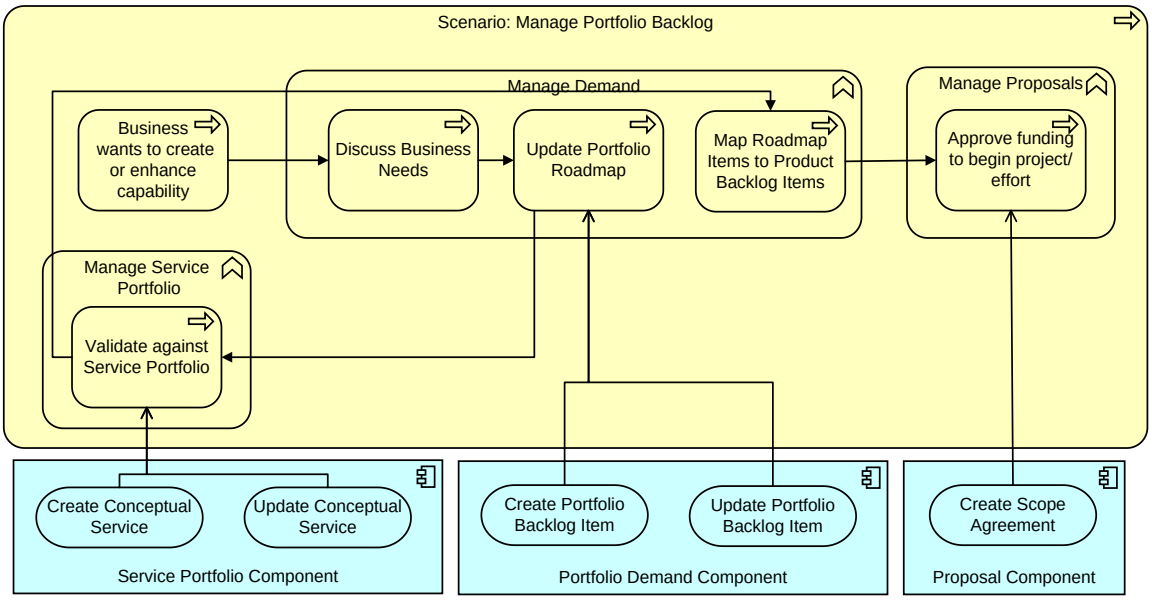
Goal	Explanation
G.01	Demand processes and management of the portfolio backlog are consistent across all methodologies and portfolios.
G.02	Single repository for the portfolio backlog is management and maintained for all portfolios.
G.03 G.04	A portfolio backlog exists for the key business portfolios and shows past roadmap items, planned roadmap items, and future roadmap items.
G.08	Projects can only start if there is agreement from the business (IT contract) for scope and cost.

Automation Specification Using the Reference Architecture

The portfolio backlog is composed of large-grained, thematic requirements (*aka* epics). Requirements come from business strategy in the face of market conditions.

The reference architecture supports the system of record functional components for service portfolio, portfolio demand, and proposal.

Essential Services Supporting the Scenario

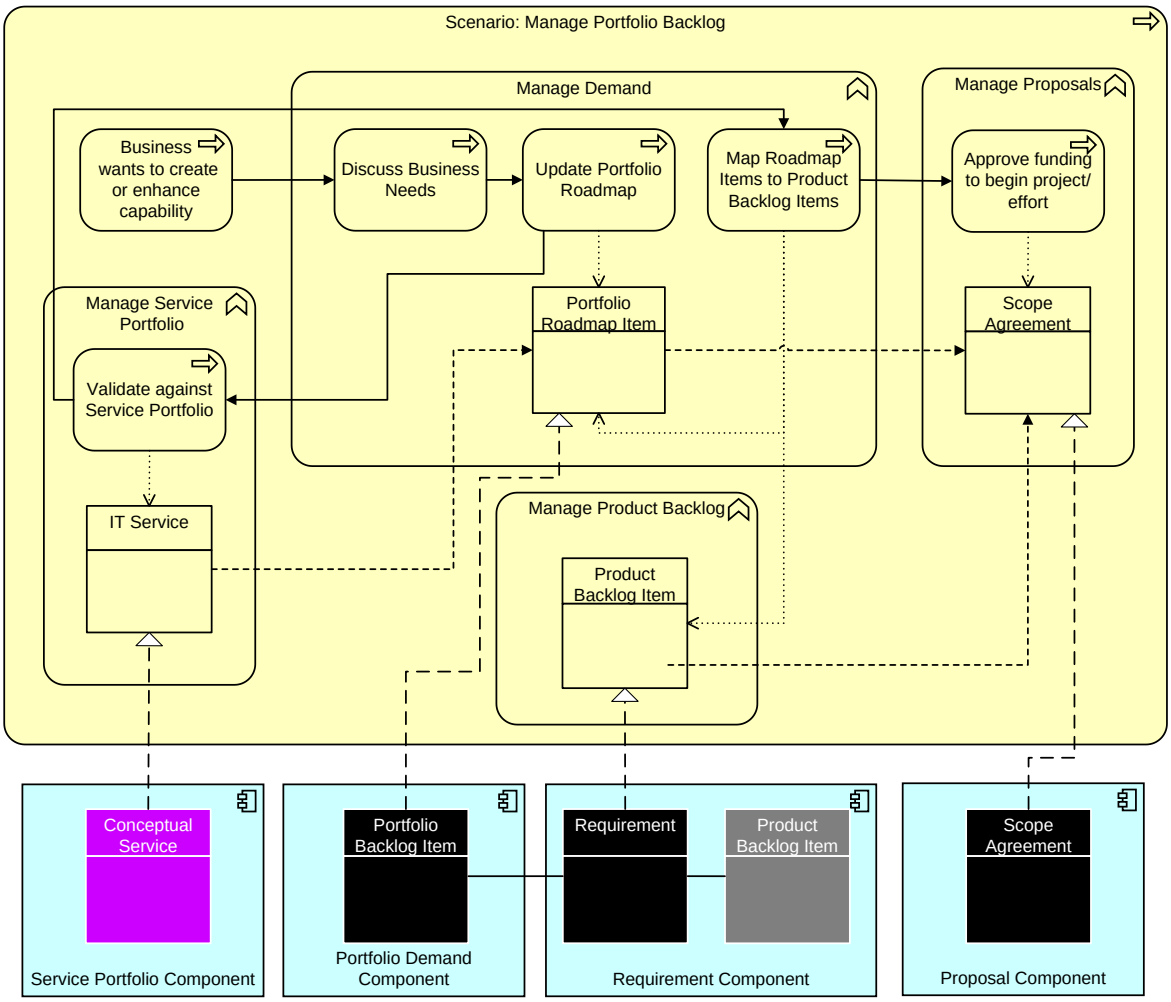


IT4IT™ Agile Scenario

The scenario calls for essential services that can:

1. Create/update a conceptual service
2. Create/update a portfolio backlog item
3. Create a scope agreement

Data Architecture



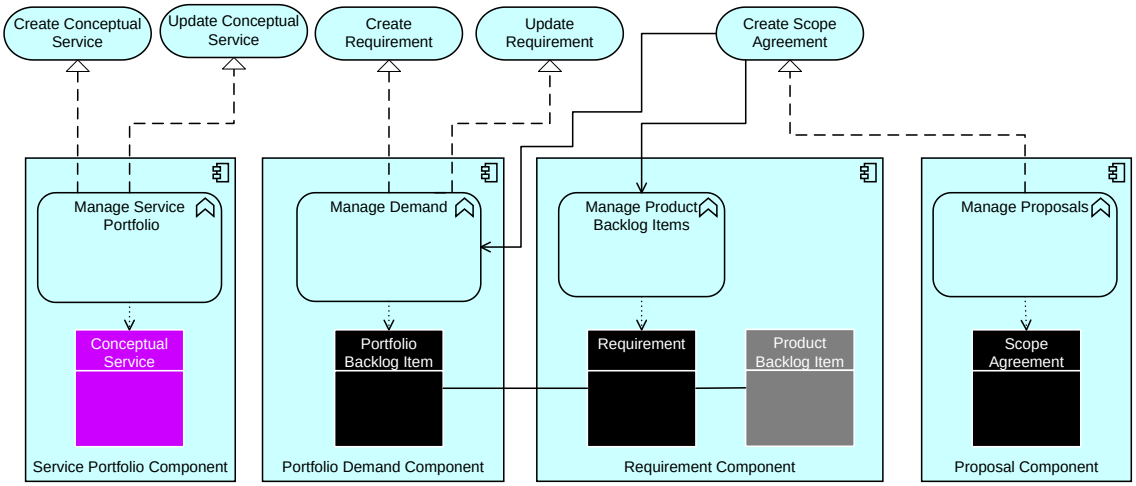
The major entities include:

1. Conceptual service
2. Portfolio Backlog Item
3. Requirement
4. Product Backlog Item

5. Scope Agreement

See Kanban and Queueing (on page 54) for a discussion of how requirements and backlog items might be consolidated with a common queuing interface.

Detailed Explanation of Reference Architecture Usage



1. The Manage Service Portfolio process creates/updates the Conceptual Service.
2. The Manage Demand process creates/updates portfolio backlog items (e.g., epics) (diagram shows that Manage Demand is also invoking Create Requirement?).
3. The Manage Product Backlog process creates requirements (e.g., user stories).

Key Attributes Required by this Scenario

Artifact	Attributes	Additional Information
Conceptual Service	ID	Unique ID of conceptual service.
	Name	This is the name easily identifying the service.
	Description	This would represent the short description of a service.
Portfolio Backlog Item	ID	Unique ID of portfolio backlog item.
	Portfolio	This would be the portfolio name to which the backlog item is related.
	Summary	This would represent the short description/title/summary of a given portfolio backlog item.
	Description	This would be the full description; ideally supports rich text.
	Backlog Priority	This priority is unique across all backlog items for a single portfolio.

IT4IT™ Agile Scenario

Artifact	Attributes	Additional Information
	Backlog Status	This status shows the state of the backlog item. Status would be used by both the business and IT to determine whether it was an agreed roadmap item, proposed, in progress, etc.
	Proposed Budget	This would be requirement, defect, or known error.
	Fiscal Year	This would indicate in which fiscal year the roadmap item (portfolio backlog item) is planned.
	IT Service ID/Name	This needs to be at least the IT service name, but ideally it would be the IT service ID and service name.
	Requirement ID(s)	This would be used to link a portfolio backlog item to one or more product backlog items (requirements).
	Scope Agreement ID	This would be used to link a portfolio backlog item to a scope agreement.
Requirement	ID	Unique ID of requirement.
	IT Service ID/Name	This needs to be at least the IT service name, but ideally it would be the IT service ID from CMDB and the IT service name.
	Summary	This would represent the short description/title/summary of a given requirement.
	Description	This would be the full description; ideally supports rich text.
	Portfolio Backlog ID	This would be used to link a product backlog item to a portfolio backlog item.
Scope Agreement	ID	Unique ID of a scope agreement.
	Portfolio Backlog ID	This would be used to link a scope agreement to a portfolio backlog item.
	IT Service ID/Name	This needs to be at least the IT service name, but ideally it would be the IT service ID and the IT service name.
	Summary	This would represent the short description/title/summary of a given scope agreement.
	Description	This would be the full description; ideally supports rich text.

Product Backlog Scenario

Overview

As made clear in the previous section, investments are divided into portfolios, which may have backlogs. This level is equivalent to the concept of "epics" within SAFe. Once product investments are authorized within a portfolio, each product has its own smaller-grained backlog, consisting of stories, defects, and so on.

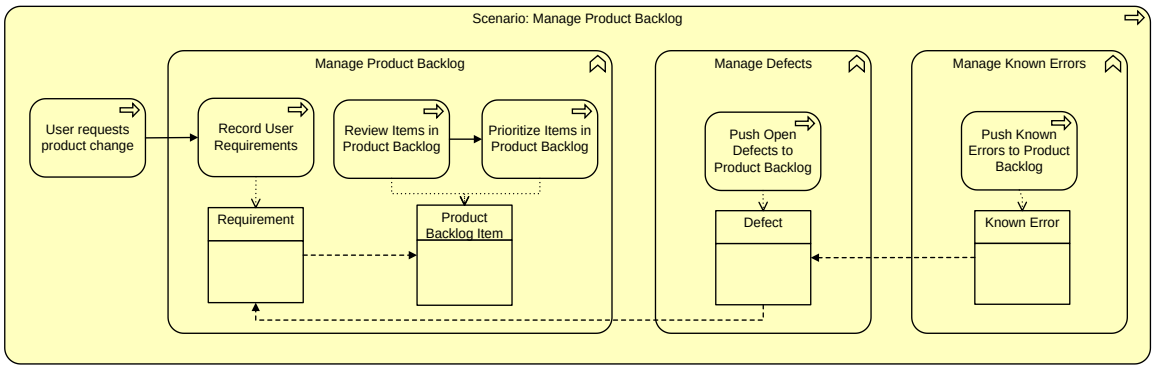
IT4IT™ Agile Scenario

New product backlog items can come from multiple sources, but should all be captured in a single repository for an individual IT service. The business may ask for enhancements to existing products, IT may identify known errors or operational issues that need to be remedied; there may be other defects in the products which need to be resolved.

Requirements

ID	Requirements
REQ.05	Ability to capture new enhancement requests or updates to existing enhancement requests in the product backlog: • IT Service • Summary • Description • Status • Priority
REQ.06	Ability to add/modify unfulfilled requirements on the product backlog for an IT service: • IT Service • Summary • Description
REQ.07	Ability to add/modify open project defects which are not resolved prior to going live on the product backlog for an IT service: • IT Service • Summary • Description
REQ.08	Ability to add/modify known errors to the product backlog for an IT service: • IT Service • Summary • Description

Process Flow



REQ.05: Process requirements; log IT request

Qualified product stakeholders may ask for a product to be enhanced. These requests need to be tracked in the product backlog which should exist for the entire lifecycle of the product.

The system should be able to track key information about the request such as the summary and description, status, and priority. The status should reflect information about the request such as if it is new, approved, rejected, or in-process. The priority of the request should indicate impact to the organization and reflect an order of importance with respect to other items in the product backlog. The product backlog should be able to be mapped to portfolio backlog items which are used to obtain approval by the business to fund an effort. Many companies have a standing portfolio backlog item representing a pre-approved fund to enhance a product based on the product owner's discretion as delegated by the business.

REQ.06: Record open requirements in product backlog

There are times where requirements which were in scope for the project are unable to be met during the course of a project. In these cases, those requirements should remain in an open status in the system so they can be grouped into a future release of the IT service. The project or delivery team is responsible for making sure the unfulfilled requirements are reflected in the product backlog.

REQ 07: Record open defect in product backlog

There are times where a project defect is not so severe as to prevent the release of an IT service. In these cases, the defect still needs to be resolved in a future release and should be added to the product backlog so it can be implemented in the future. The project or delivery team is responsible for making sure the open defects are captured in the product backlog.

REQ 08: Record open known error in product backlog

When a problem is identified with an IT service, and the root cause is determined highlighting the need for a change to the IT service, that known error should be reflected in the product backlog so it can be implemented in the future. The problem manager is responsible for capturing known errors in the product backlog.

Goal	Explanation
G.01 G.05	A product backlog should exist for an IT service throughout the service lifecycle. Since requests can come from many different sources such as the business and IT (projects and operations), it is important to have a single list (product backlog) of all potential changes to the IT service so the changes can be planned comprehensively. This promotes including operational changes and business changes in releases of an IT service which ultimately reduces the cost and effort as multiple items can be resolved with one project or initiative, thus preventing the ramp-up and deployment costs associated with multiple projects or initiatives to do the same.
G.02 G.06	Many different software development methodologies may be used to create or enhance an IT service. Since we share resources among our delivery teams, and since different resources may work on various changes to various IT services, it is critical that some key information is stored in a similar fashion without regard to which methodology may be used on a given project/effort. By storing the data in the same repository and IT service, it then becomes a delivery team determination on how best to meet the needs of a given project/initiative. There is also no time lost between modifying data to fit one methodology or another.
G.02 G.07	By adding end-user change requests, open items from the project delivery process, and open items from operational processes into a single repository, it allows the delivery team to go to a single centralized repository to see the details (or links to details) of all in-scope items, thus reducing time to otherwise review and consolidate various repositories.

IT4IT™ Agile Scenario

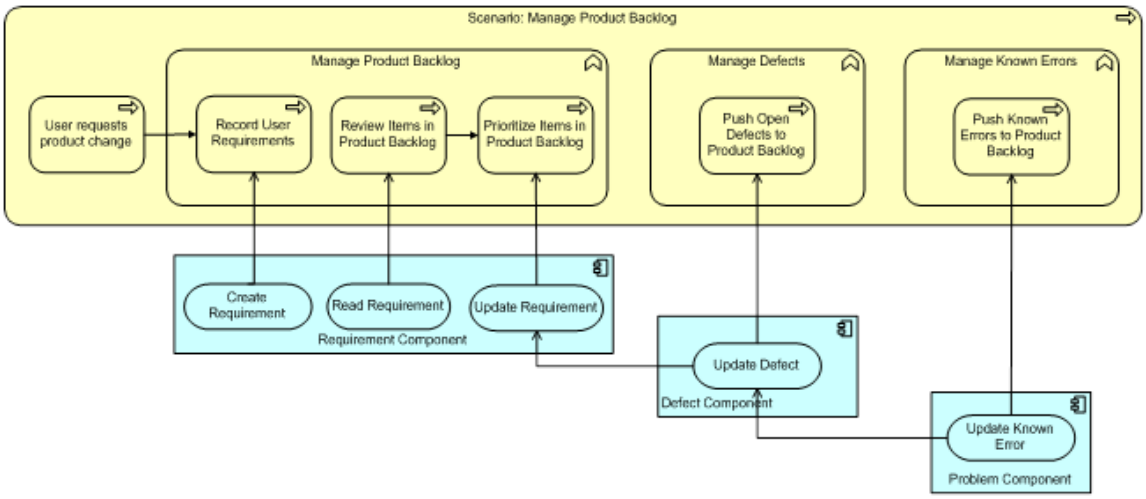
Automation Specification Using the Reference Architecture

The product backlog is a subset of requirements which are prioritized, associated with releases, and track status. Requirements come from business demand, defects, and known errors.

The reference architecture supports the system of record functional components for requirement, defect, and problem which ultimately are the source of the individual items comprising the product backlog.

The Requirement functional component becomes a key integration point between the Strategy to Portfolio and Requirement to Deploy value streams.

Essential Services Supporting the Scenario

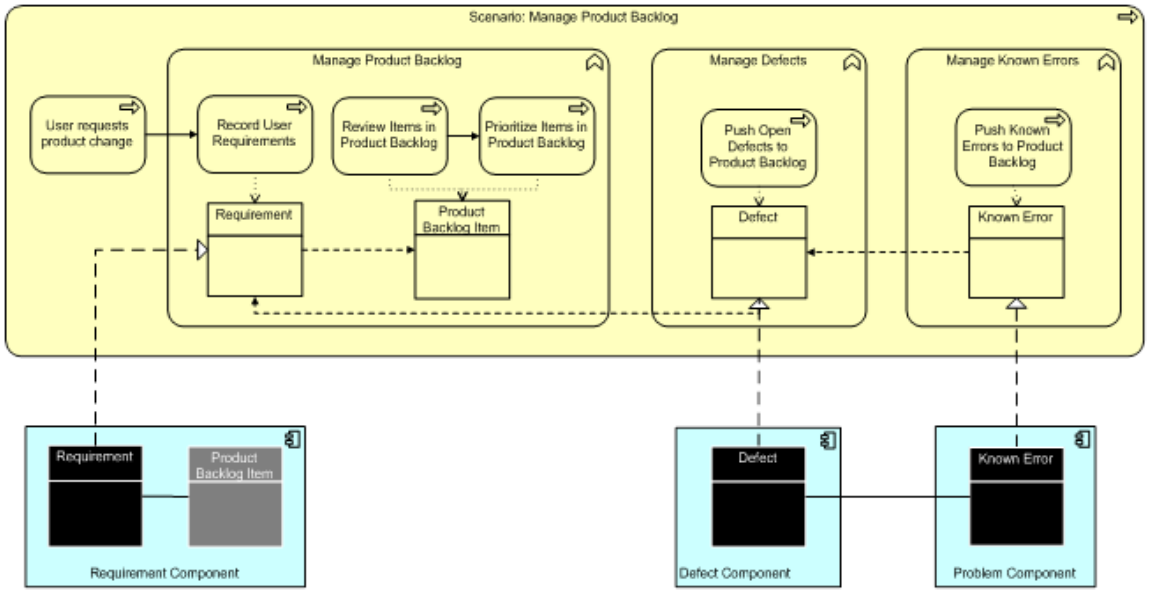


The scenario calls for essential services that can:

- Create/update a conceptual service
- Create/update a portfolio backlog item
- Create a scope agreement

IT4IT™ Agile Scenario

Data Architecture



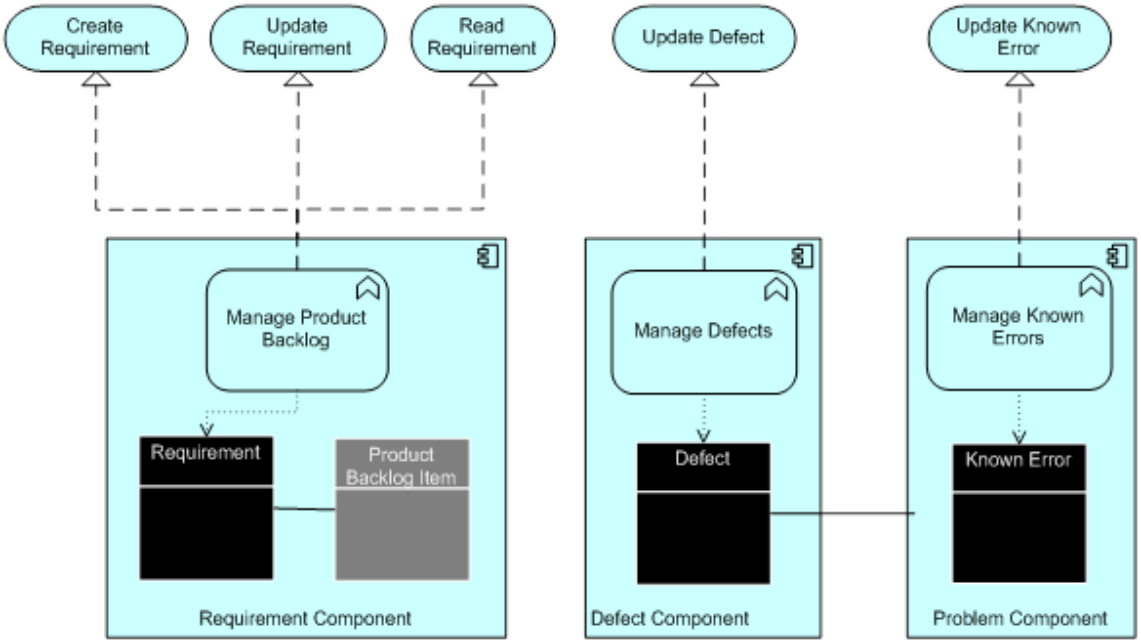
The major entities include:

1. Requirement
2. Portfolio Backlog Item
3. Defect
4. Known Error

See Kanban and Queueing (on page 54) for a discussion of how requirements, defects, and known errors might be consolidated with a common queuing interface.

IT4IT™ Agile Scenario

Detailed Explanation on Reference Architecture Usage



1. The Manage Service Portfolio process creates/updates the Conceptual Service.
2. The Manage Demand process creates/updates portfolio backlog items (e.g., epics) (diagram shows that Manage Demand is also invoking Create Requirement?).
3. The Manage Product Backlog process creates requirements (e.g., user stories).

Key Attributes Required by this Scenario

Artifact	Attributes	Additional Information
Requirement	ID	Unique ID of requirement.
	IT Service ID/Name	This needs to be at least the IT service name, but ideally it would be the IT service ID and the IT service name.
	Summary	This would represent the short description/title/summary of a given requirement.
	Description	This would be the full description; ideally supports rich text.
	Portfolio Backlog ID	This would be used to link a product backlog item to a portfolio backlog item.
Defect	ID	Unique ID of a scope agreement.
	IT Service ID/Name	This needs to be at least the IT service name, but ideally it would be the IT service ID and the IT service name.
	Summary	This would represent the short description/title/summary of a given defect.

IT4IT™ Agile Scenario

Artifact	Attributes	Additional Information
	Description	This would be the full description, ideally supports rich text.
Known Error	ID	Unique ID of a known error.
	IT Service ID/Name	This needs to be at least the IT service name, but ideally it would be the IT service ID and the IT service name.
	Summary	This would represent the short description/title/summary of the known error.
	Description	This would be the full description, ideally supports rich text.

Proposed Changes to the IT4IT Reference Architecture for the Portfolio and Product Backlog Scenario

ID	Proposed Change	Status
1	Add new attributes to portfolio backlog item within the Portfolio Demand component as follows: • ID • Portfolio • Summary • Description • Backlog Status • Backlog Priority • Proposed Budget • Fiscal Year • IT Service ID/Name • Requirement ID • Scope Agreement ID	Proposed
2	Add new attributes to conceptual service within the Service Portfolio component as follows: • ID • Name • Description	Proposed
3	Add new attributes to requirement within the Requirement component as follows: • ID • IT Service ID/Name • Summary • Description • Backlog Status • Backlog Priority • Portfolio Backlog ID	Proposed

IT4IT™ Agile Scenario

ID	Proposed Change	Status
4	Add new attributes to defect within the Defect functional component as follows: • ID • IT Service ID/Name • Summary • Description	Proposed
5	Add new attributes to known error in the Problem functional component as follows: • ID • IT Service ID/Name • Summary • Description	Proposed
6	Add new attributes to scope agreement in the Proposal component as follows: • ID • Portfolio Backlog Item ID • IT Service ID/Name • Summary • Description	Proposed

DevOps and Automation

DevOps and the IT4IT Reference Architecture

DevOps is a notable trend in IT management. It is grounded in Lean and Agile theory, and enabled by advances in IT computing power and software and systems design.

DevOps calls for a new mental model of change. Often, change and stability are seen as diametrically opposed. However, this can often result in deferring change, which results in larger accumulated "batches" of change. Complex systems do not respond well to such large perturbations. It is more effective to change them continually over time in small, controlled increments. This results in the paradoxical finding that more frequent change is better for stability. This can be counterintuitive for many IT professionals.

There is a great deal of material available on DevOps, some of it discussed in this document. For further information on DevOps see the cited references, especially Continuous Delivery by Humble and Farley [Humble 2011].

Goals

Goal	Explanation
DG.1	High automation of the code pipeline – build, test, migrate, release.
DG.1.2	Entire pipeline from "Dev" to "Ops" is viewed as an integrated system.
DG.2	Continuous integration.
DG.2.1	Eliminate/minimize "branching" from the "mainline" of source.
DG.2.2	Daily/ hourly code check-ins across all teams.
DG.2.3	Automated software testing.
DG.3	Develop against production-like environment.
DG.3.1	Common build approaches used for all environments.
DG.3.2	Focus on configuration management.
DG.4	Architect for resiliency.
DG.4.1	Focus on Mean Time to Recovery (MTTR), not Mean Time Between Failures (MTBF)
DG.5	"Infrastructure as code".
DG.5.1	Configurations are scripted assets in source repositories, part of build.
DG.5.2	Automation enables rapid provisioning and deployment.
DG.5.3	Automated rollback, where possible.

IT4IT™ Agile Scenario

The following matrix mapping People, Process and Technology to the IT4IT Value Streams is intended to illuminate how DevOps and the IT4IT standard interact.

	S2P Value Stream	R2D Value Stream	R2F Value Stream	D2C Value Stream
People	Establish common values and awareness Culture of systems engineering and collaboration	Ops cross-pollination	Training to develop and support maximum automation	Dev on support frontline
Process	Matrix organization and service-aligned virtual teams supported by shared services engineering All queues identified	Delivery and deployment cadence Reduced WIP	Services designed for automation and self-service through catalog and APIs	Problem management feedback to R2D
Technology	Lifecycle Management & Automation Planning, reporting Metrics management (KPIs, MTBF, MTTR)	Automate build, test, deployment	Infrastructure virtualization and provisioning automation	Automate monitoring, workload management Analytics

DevOps Reference implementation

An initial reference implementation of the DevOps scenario was undertaken using free and open source software [Betz 2015]. Products employed included:

Product	Role
Ubuntu Linux	Operating system
Vagrant	Virtual environment
Chef Zero	Virtual server provisioning
Java	Programming language & interpreter
Junit	Automated testing framework
Apache Ant	Language-specific build tool
Apache Tomcat	Application server
Git	Source control repository
Jenkins	Build choreography tool
Artifactory	Package repository

IT4IT™ Agile Scenario

Purpose

One of the purposes of this effort was to ensure that the DevOps scenario was well grounded in technical fundamentals. Abstractions (e.g., "source code control") need to have concrete examples, and interactions suggested by graphical lines need to be traceable to actual system relationships.

The initial version of the pipeline explored the relationship between development, testing, and production environments. Various learnings are presented here in support of the architecture approach in the following sections.

Learnings

There were a number of learnings and insights generated by this effort:

- The importance of representing subject systems and environments in the IT4IT pipeline (e.g., "developer workstation," "production server")
- Clarifying the relationship between artifact and data object
- Maven coordinates and semantic versioning (see Maven Coordinates and Semantic Versioning, below) as emergent metadata standards
- Relationship between source and package repository
- Role of testing practices versus test management systems

Systems and Environments

The IT4IT Reference Architecture has not formally represented the computing systems producing and being produced as a result of the IT Value Chain. However, in attempting to produce a well grounded DevOps reference architecture, it is helpful to represent concepts such as a "development environment", "build environment", and "production environment". These are intended abstractly. Particular technologies were chosen for the reference implementation (e.g., Java and Tomcat) but alternatives could also be used (e.g., Javascript and node.js).

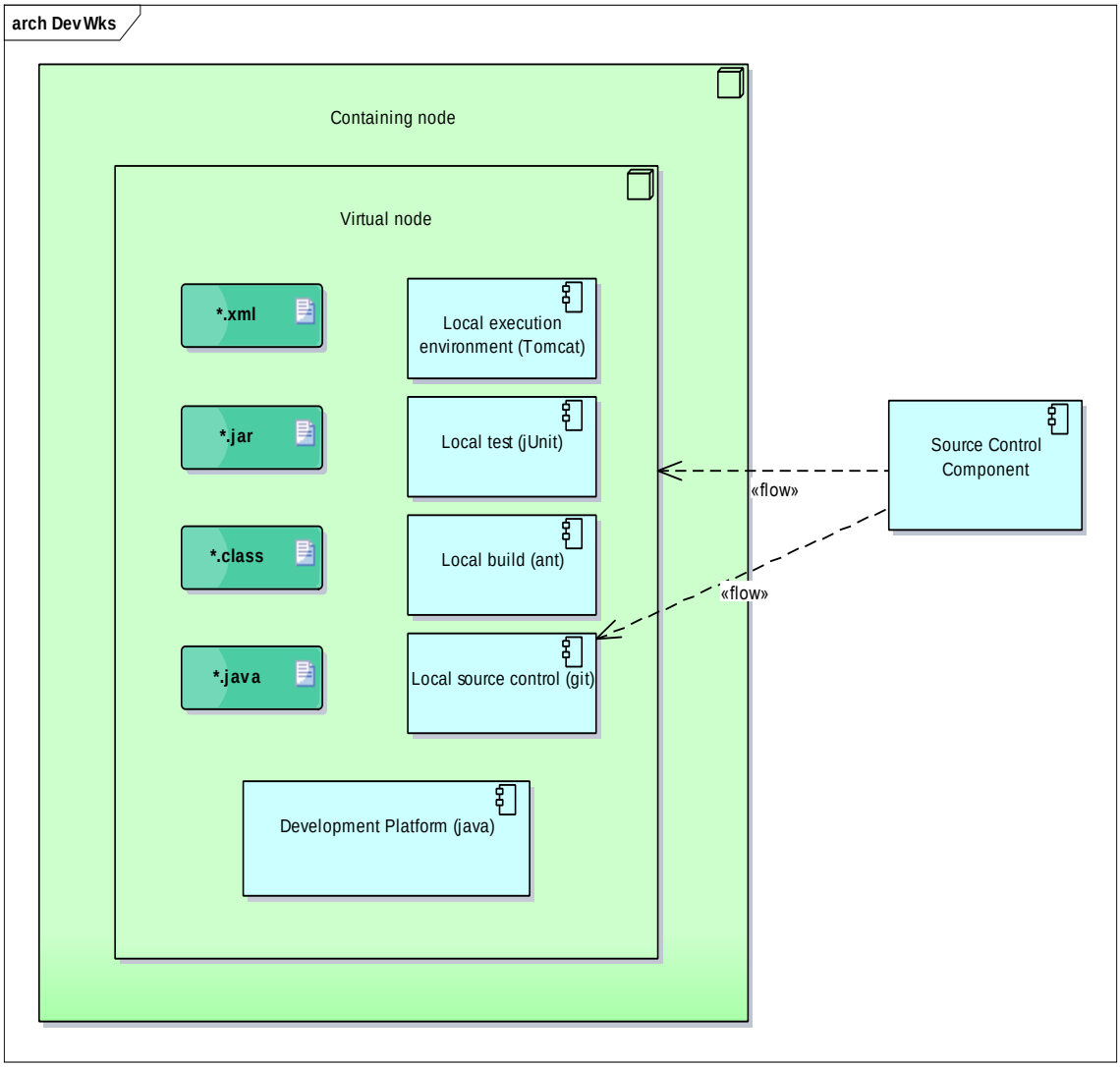


Figure 11: Example Development Environment

With the increasing power of infrastructure and reach of cloud, there are many variations on how development environments are constructed. A local physical PC used by a developer may employ a hypervisor (VirtualBox or boot2docker) with development and execution carried out in a virtual layer. The virtual layer increasing is built consistently with production environments through infrastructure as code (e.g., Chef recipes) under shared source control.

Others may still run an execution environment such as Tomcat directly on the base OS, without an intervening virtual layer; however, this approach seems to be falling out of favor.

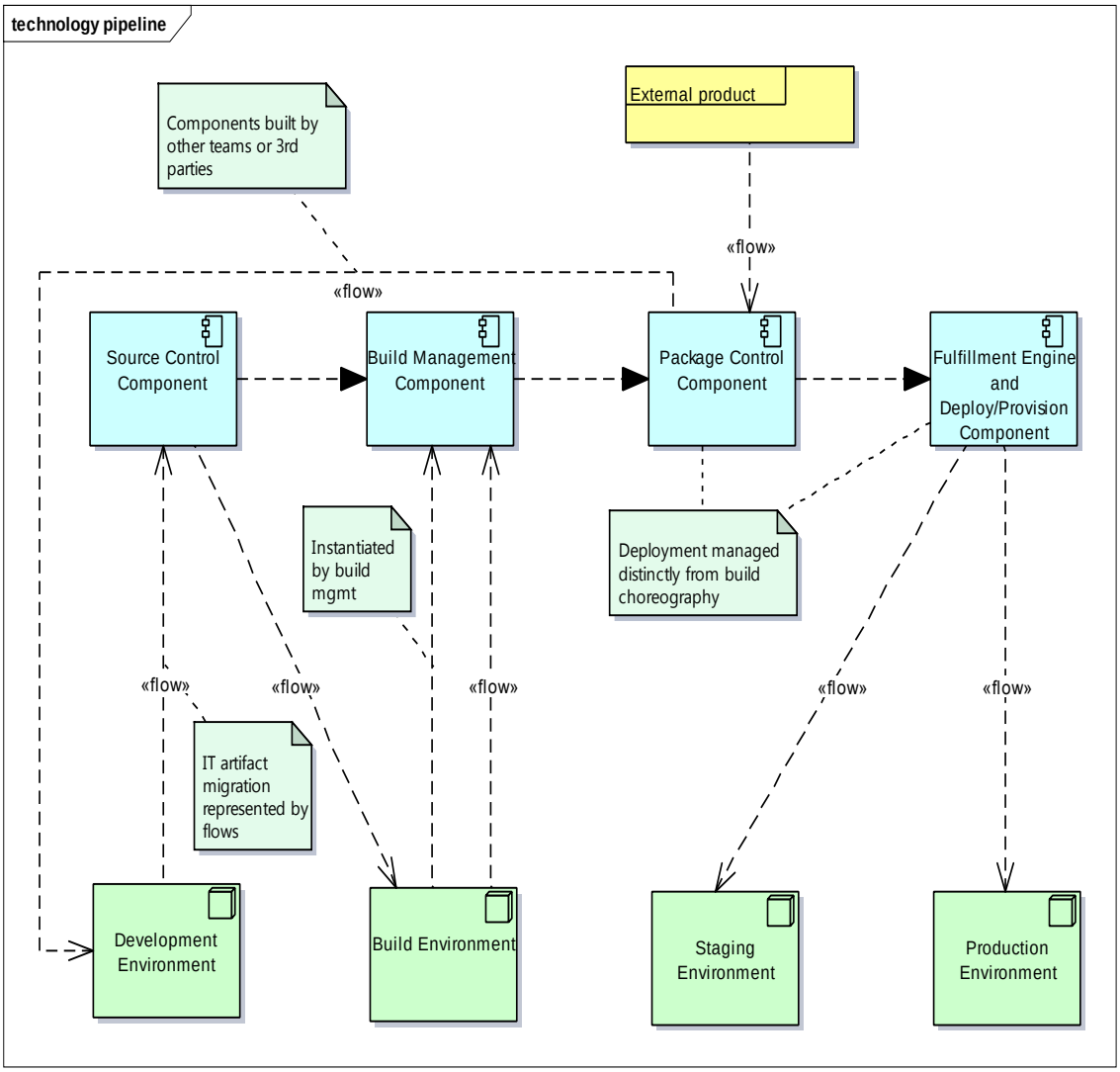


Figure 12: Development Pipeline

In Figure 12, artifacts flow through the pipline (green nodes) through the pipeline control infrastructure (blue components). Additionally, the following interactions are not shown on the diagram:

- The Source Control component may hold infrastructure definitions for the environments.
- The Fulfilment Engine may instantiate and provision the base environments as well as installing packaged artifacts on them.

Artifact and Data Object

As of early 2015, the IT4IT standard is moving towards replacing the term "artifact" with the term "data object". The insight from this reference implementation is that both terms are in fact needed. (This is also consistent with the ArchiMate® standard,[2] which views "artifact" as a Technology Layer construct, while Data Object and its analog Business Object are constructs of the Application and Business Layers, respectively.)

The concept of an "artifact" is represented in the Calavera simulation by the following files:

- *.java
- *.class
- *.jar
- *.xml

While such artifacts may follow well-defined syntax, they are arbitrarily complex. Therefore, a metadata layer emerges to manage them. Metadata is built from concise textual fragments, made consistent and suitable for structured data management at scale. Examples include:

- File name and extension
- File location
- Commit identifier (from version control system)
- Author/developer
- Build # and date
- Version (see Maven Coordinates and Semantic Versioning, below)
- Associated project or release

These attributes are "meta" in the sense that they are *not directly required for the runtime functioning of the system, but rather assist in human understanding of its characteristics*. Such basic attributes, when aggregated and normalized into schemas, are the foundation of structured IT management and therefore the IT4IT standard.

Many other examples exist, and specific artifacts may have extensive metadata (especially those artifacts having to do with structured data management; e.g., data models and data definition schemas).

[2] ArchiMate® 2.1 Specification, Open Group Standard (C13L), December 2013, published by The Open Group; refer to: www.opengroup.org/bookstore/catalog/c13l.htm.

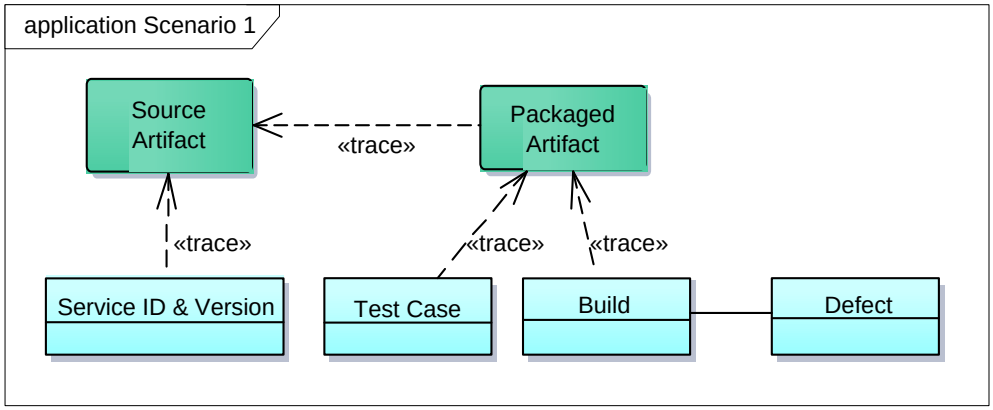


Figure 13: Relationship of IT4IT Data to Artifacts

Figure 13 graphically shows the relationship between artifacts and their metadata (*aka* IT4IT data). Maintaining the traceability between artifact and data object is a critical requirement across the IT4IT Reference Architecture. The originating point for this critical metadata is often the application manifest, although there is no industry standard. Build tools are also important, as they retain the record of when and how a given release package was constructed.

Maven Coordinates and Semantic Versioning

Two important developments in metadata are Maven coordinates and semantic versioning.[3]

Maven coordinates consist of the following:

- groupID; e.g., org.apache.maven
- artifactID; e.g., a project or application identifier (not a filename)
- version
- packaging

A fifth attribute, "classifier," is sometimes used.

Semantic versioning can be used as the basis for the version ID; according to semver.org, it consists of the following rules:

Given a version number MAJOR.MINOR.PATCH, increment the:

- MAJOR version when you make incompatible API changes
- MINOR version when you add functionality in a backwards-compatible manner

[3] More information about Maven is available here: https://maven.apache.org. Maven coordinates are documented here: https://maven.apache.org/pom.html.

- PATCH version when you make backwards-compatible bug fixes

12 Factor Principles

12-factor architecture principles[4] also provide insight into the conceptual data structures used by DevOps [Wiggins 2015]. The combination of these influences can be represented as a conceptual metamodel:

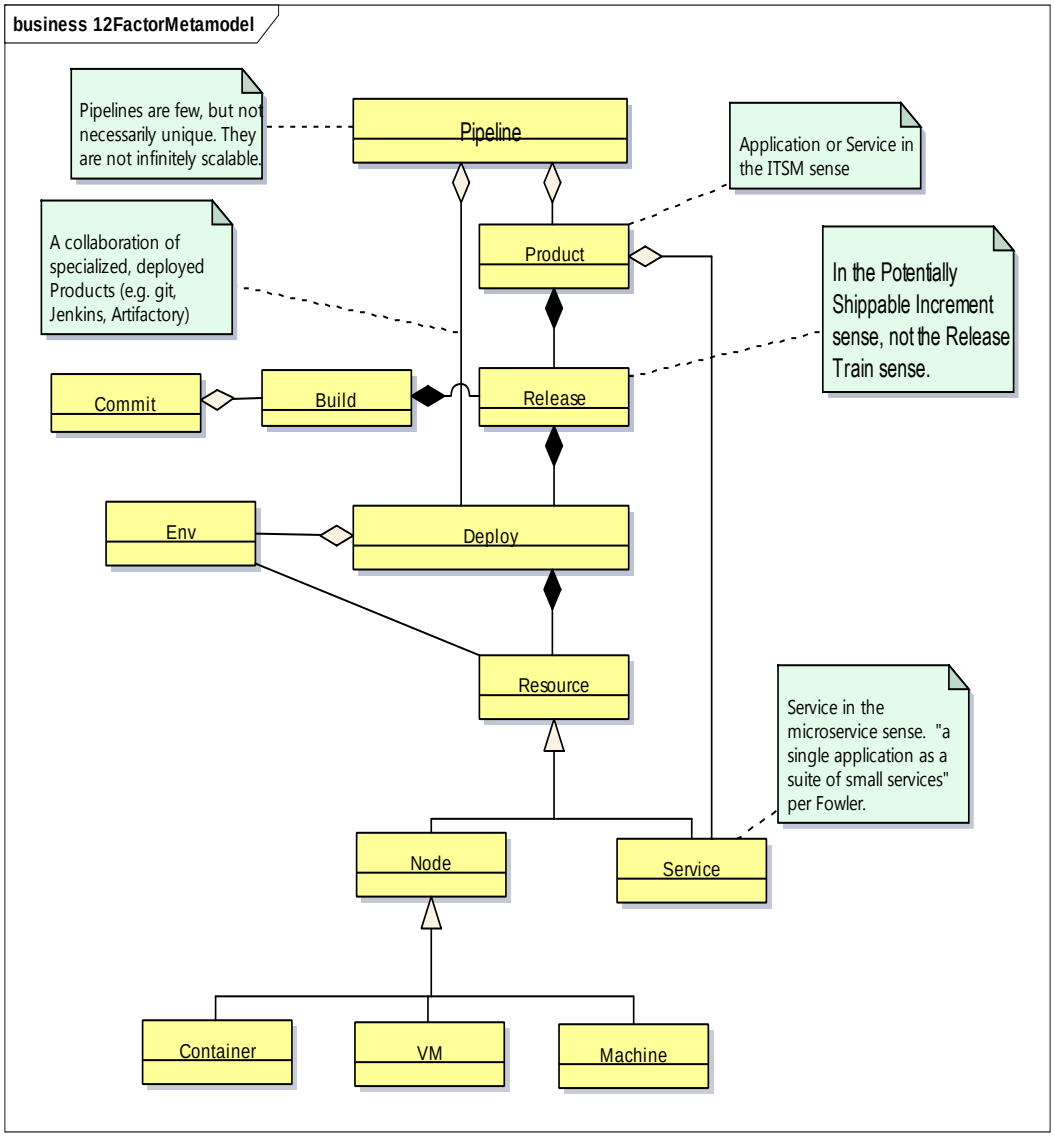


Figure 14: DevOps/12Factor Metamodel

[4] See http://12factor.net/.

The combination of basic file metadata, commit/build/release metadata, Maven coordinates, and semantic versioning provides a foundation for IT4IT metadata. Most major IT management constructs can be understood as derivative and/or dependent:

- Service and application IDs
- Release and change IDs
- Work tickets of various types: requirements, scenarios, stories, issues, risks, action items, incidents, service requests

Source and Package Repositories

The reference implementation clarified the relationship between source and package repositories. In years past, it has been common practice for compiled binaries to be stored in the same repositories as source code. However, this is no longer deemed best practice. Source repositories should be reserved for symbolic artifacts that can be meaningfully versioned, with differences represented in a human-understandable way. Package repositories in contrast are for binary artifacts that can be deployed.

Package repositories have a further additional role as proxy to the external world of downloadable software. For example, developers are directed to download the approved Java version from the local package repository, rather than going to the Oracle or OpenJDK site and getting the latest version, which may not be suitable for the environment.

Package repositories are also used (as indicated in Figure 12) to enable collaboration between teams working on larger applications. Teams check built components into the package repository for other teams to download, rather than everyone always building all parts of the application from source.

In these uses, package repositories serve as an important control point in reducing variability in the IT pipeline; reducing such variability is one of the overall goals (G0.6.6) for the Agile work stream.

Testing Practices versus Test Management Systems

Finally, the IT4IT Reference Architecture presumes a test management system, but it is important to distinguish between tests that are written and executed internally to the application *versus* tests that are external.

For example, a suite of tests based on JUnit does not require an external "test management system" to run. JUnit is incorporated into the development and build environments as a library include. Tests are written as Java source code in a particular manner to be interpreted by JUnit.

On the other hand, an external "black box" testing capability can be seen as its own functional component or system. Examples would include Apache Jmeter, SOASTA, or Stormrunner (*aka* Loadrunner).

Yet other testing products can be run as either a locally included capability, or an external service. Examples include Selenium.

Finally, some Agile testing philosophies (test-driven design, behavior-driven design) start to converge test definition with functional requirements. In this sense, the Test Management component overlaps with the Requirements Management component.

Future Directions for Simulation

The following features were deferred to future versions:

- Source code QA (e.g., static analysis)
- Code review collaboration enablement
- Test-driven infrastructure as code
- Defect and issue tracking
- Integration testing
- Monitoring and automated rollback
- Full ITSM suite integration

Continuous Deployment Scenario

Overview

In the previous section, scenarios were discussed that had been proven using a reference implementation. This section takes a broader view of some of the use-cases described in current industry practice and represents them in terms of the IT4IT Reference Architecture; a full reference implementation is pending.

The core continuous deployment scenario is described thus:

Develop and deploy software functionality from development to operations in a maximally automated model, with sustainable velocity and demonstrating effective feedback.

Eric Minick (formerly of UrbanCode, now of IBM) suggests a set of overlapping, cumulative DevOps use-cases [Minick 2012] including:

- Build software
- Deploy software
- System testing on deployment
- Continuous delivery with monitoring and rollback

We will adopt this set to start as our process requirements.

Note that this set does not include infrastructure as code.

It is essential in reading these scenarios to understand that they are intended to be run frequently and iteratively. Without this, the core Agile goals of fast feedback and maximizing information are not realized.

Process Requirements

CD.REQ.01: Build software.

In this scenario software is constructed, tested, incorporated into a mainline trunk, and built into releasable packages on an ongoing basis. This is the basic Continuous Integration use-case.

CD.REQ.02: Deploy software.

In this scenario software is pulled from the package repository and applied to target environments in a repeatable fashion.

CD.REQ.03: System testing on deployment.

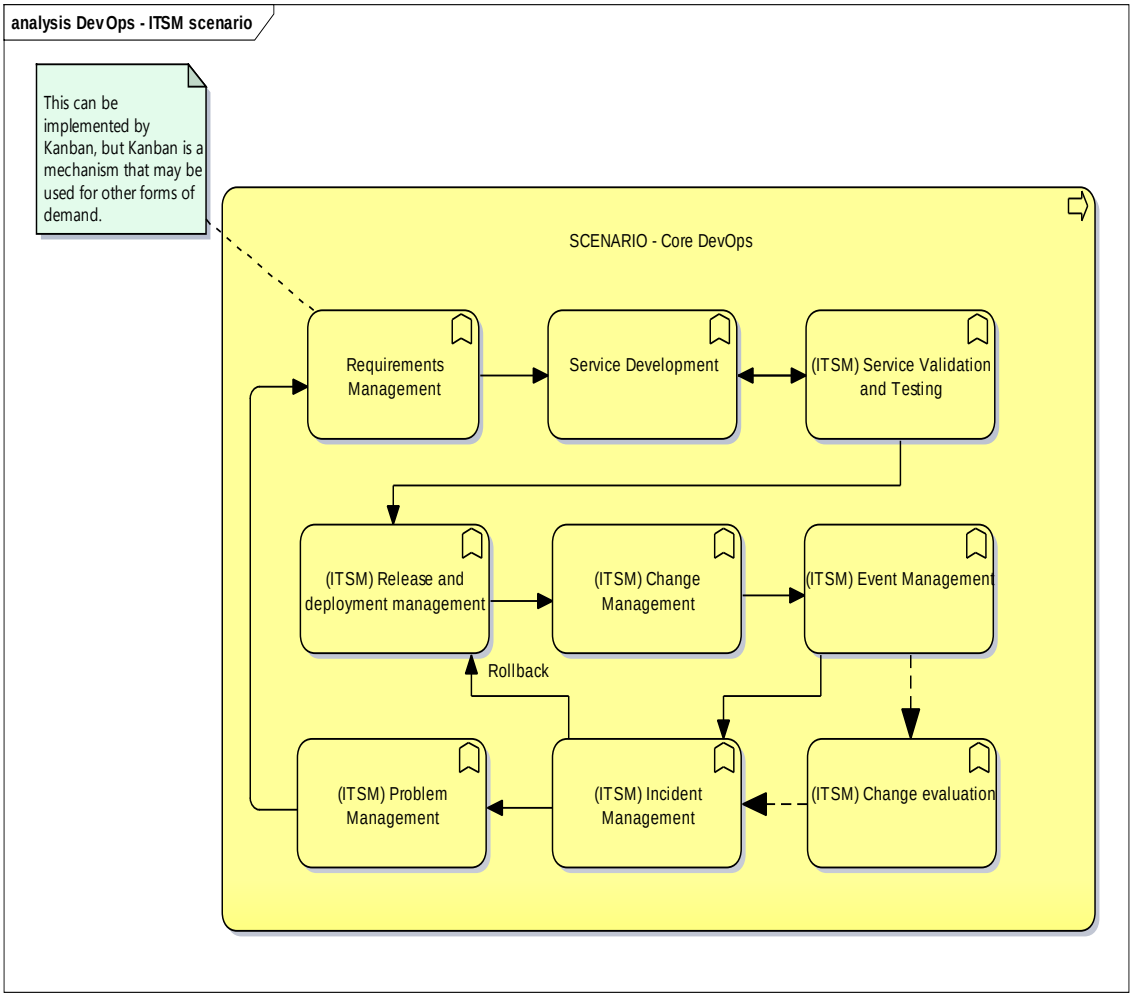
In this scenario system-level tests are applied to the deployed packages.

CD.REQ.04: Continuous delivery with monitoring and rollback.

This scenario applies monitoring, event management, and automated rollback to the previous.

As an ITSM Flow

This is a view of the end-to-end flow, interpreted in terms of well-known ITSM processes.



IT4IT™ Agile Scenario

Automation Specification

A variety of sources [Edwards 2012], [Minick 2012], [Thompson 2014], [Shortland 2012], [Betz 2011] suggest some common architectural elements, named here as encountered and mapped to the suggested IT4IT component:

DevOps Component	Example	Recommended IT4IT Mapping
Source Repository Software Configuration Management	Git, Mercurial	Source Control Component
Software Configuration Management	”“	”“
Continuous Integration and Build Management	Jenkins, Travis, Bamboo	Build Component
Test Management	Cucumber, RSpec, JUnit	Test Component, *except* in the case of embedded testing (e.g., JUnit)
Package Repository	Nexus, Artifactory	Package Component (new)
Infrastructure Manager	Chef, Puppet	Fulfillment Execution Component
Deployment Engine	”“	”“
Deployment Console	”“	”“
Configuration Management Database	BMC Atrium	Configuration Management Component
Event Management	Netcool	Service Monitoring & Event Components
Element Management		Combination of Fulfillment Execution, Service Monitoring, and Event Components

IT4IT™ Agile Scenario

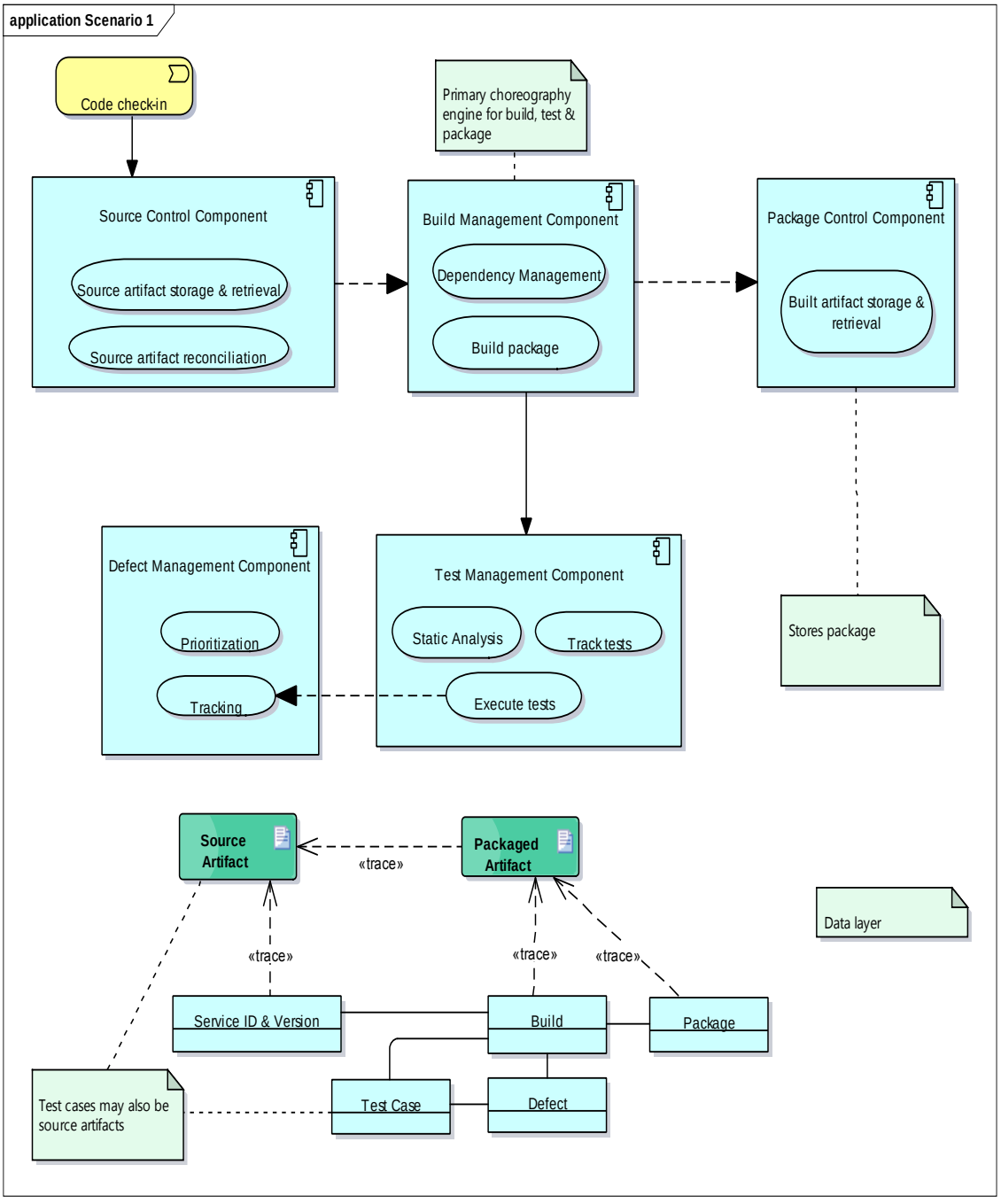


Figure 15: Build Scenario

Reading the diagram from left to right:

1. First, code is checked into the Source Control component. (This has been renamed from Service Development Component, Q1 2015.)
2. This triggers workflow to start the new build. This may be done immediately, or on a batch basis (i.e., the nightly build).
3. The Build Management component, performing Continuous Integration, may run tests (invoking the Test Management component), including functional tests as well as external static analysis if called for. Test execution may result in defects being logged. However, builds that fail because embedded unit tests fail may not result in defects. They may simply be "failed builds" and investigated and remediated on that basis. See note in previous section on testing practices *versus* testing systems.
4. If the build succeeds, the built package is stored in the Package Control component (added to IT4IT Reference Architecture early 2015).

Data Layer

As covered in the simulation discussion, the concepts of artifact and data object must be kept distinct, with traceability between the two. The Source Control component must associate the source code with the service ID and version, and the build process develops further information including tests applied, build metrics, and any defects detected.

The package record is distinct from the actual packaged artifact, consisting again of metadata such as publication status, date, URL, traceability back to the build record, and so on.

IT4IT™ Agile Scenario

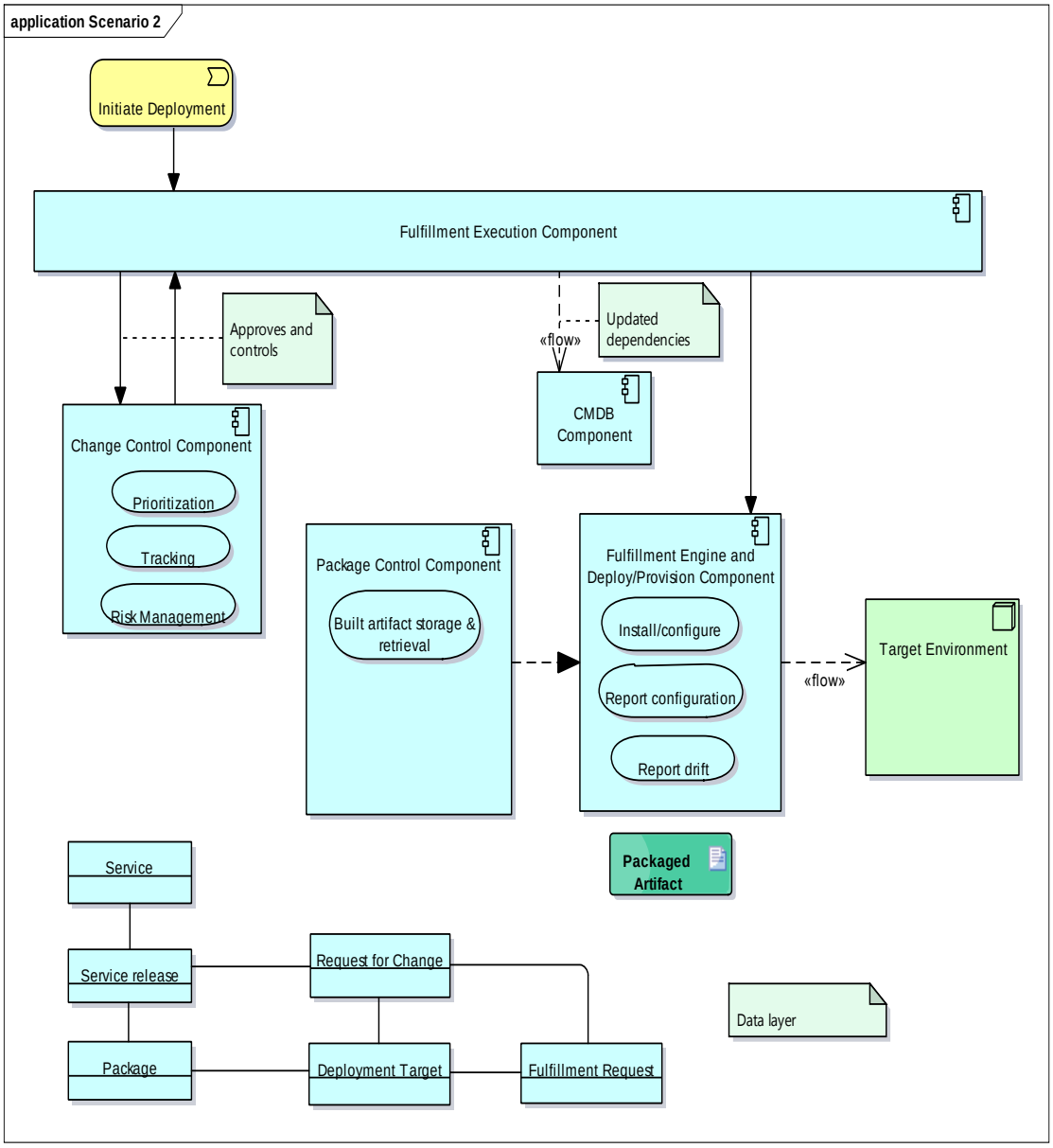


Figure 16: Deployment to Target

1. First, some event (automated; e.g., from a successful build, or human initiated) has initiated the deployment process.
2. A proposed change has been submitted to and approved by the Change Control component (this may be a standard change and approved in an automated manner).
3. The actual artifacts/package are pulled from the Release Composition component into the deployment management system(s) and applied to the target system.

4. Monitoring is established or updated.
5. The CMDB component is updated as required with any new dependencies or other information originating from the package manifest.

Data Layer

The service may be decomposed into service release and package. The RFC minimally must be associated with the service and optionally the release and/or package. Fulfillment Request represents work tracking additional to/driven by the RFC. The target must be specified and upon successful deployment the CMDB must be updated with the service/target dependency and any other new items or dependencies as appropriate.

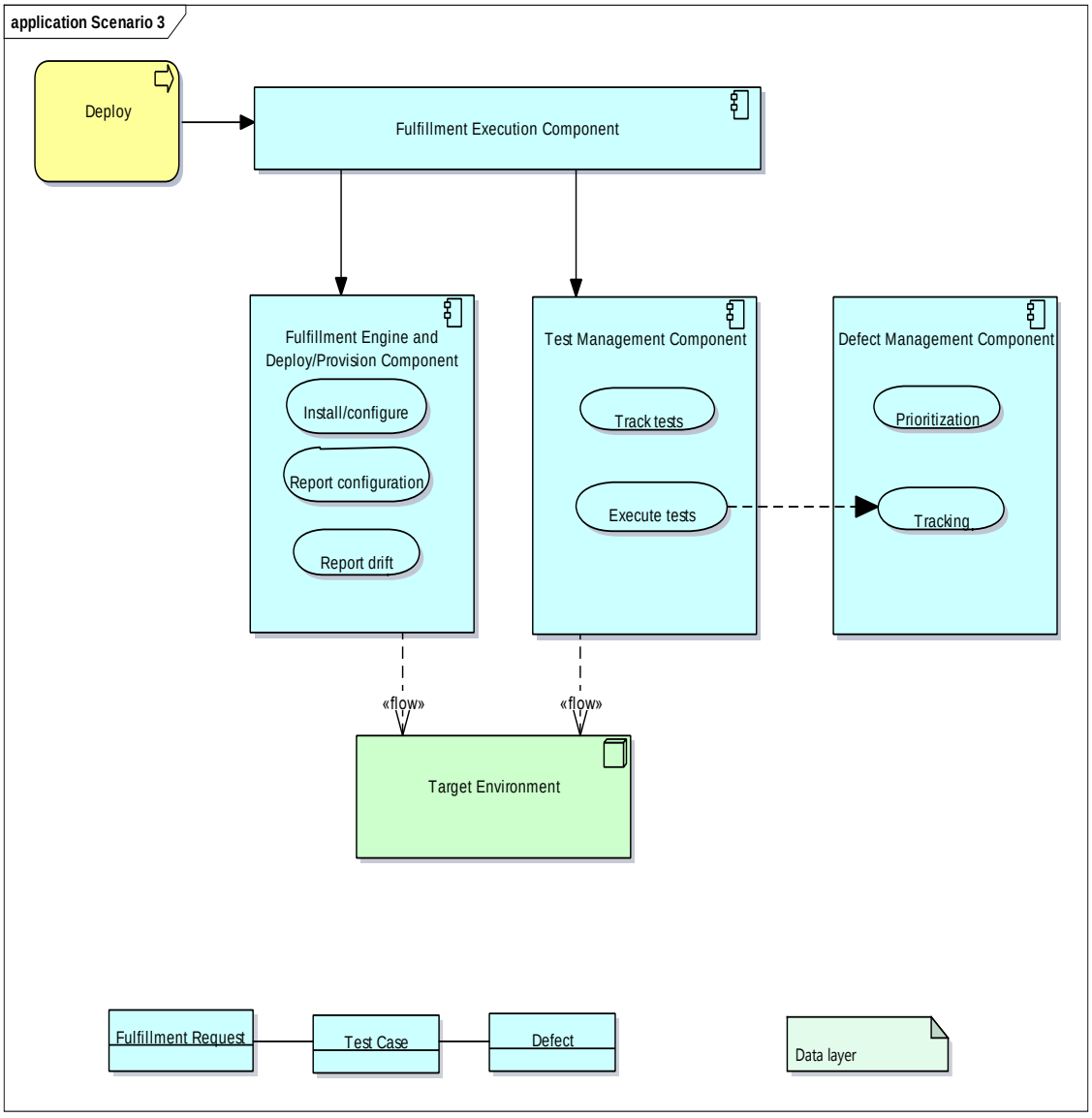


Figure 17: Test on Deploy

6. Per the last scenario, some event has resulted in the deployment of software to a target system.
7. The Test Management component executes system-level, integration, and/or performance tests against the target system (which may be any environment).
8. Results (as in the first scenario) are recorded in the Defect Management component.

The reader may note that Incident Management is not included in this scenario; it is in the next one.

Data Layer

The Fulfillment Request (deriving from the Request for Change) drives the application of one or more test case(s). These would be typically systems or integration tests, such as performance testing suites. Issues resulting from the test cases would be registered in the defect system.

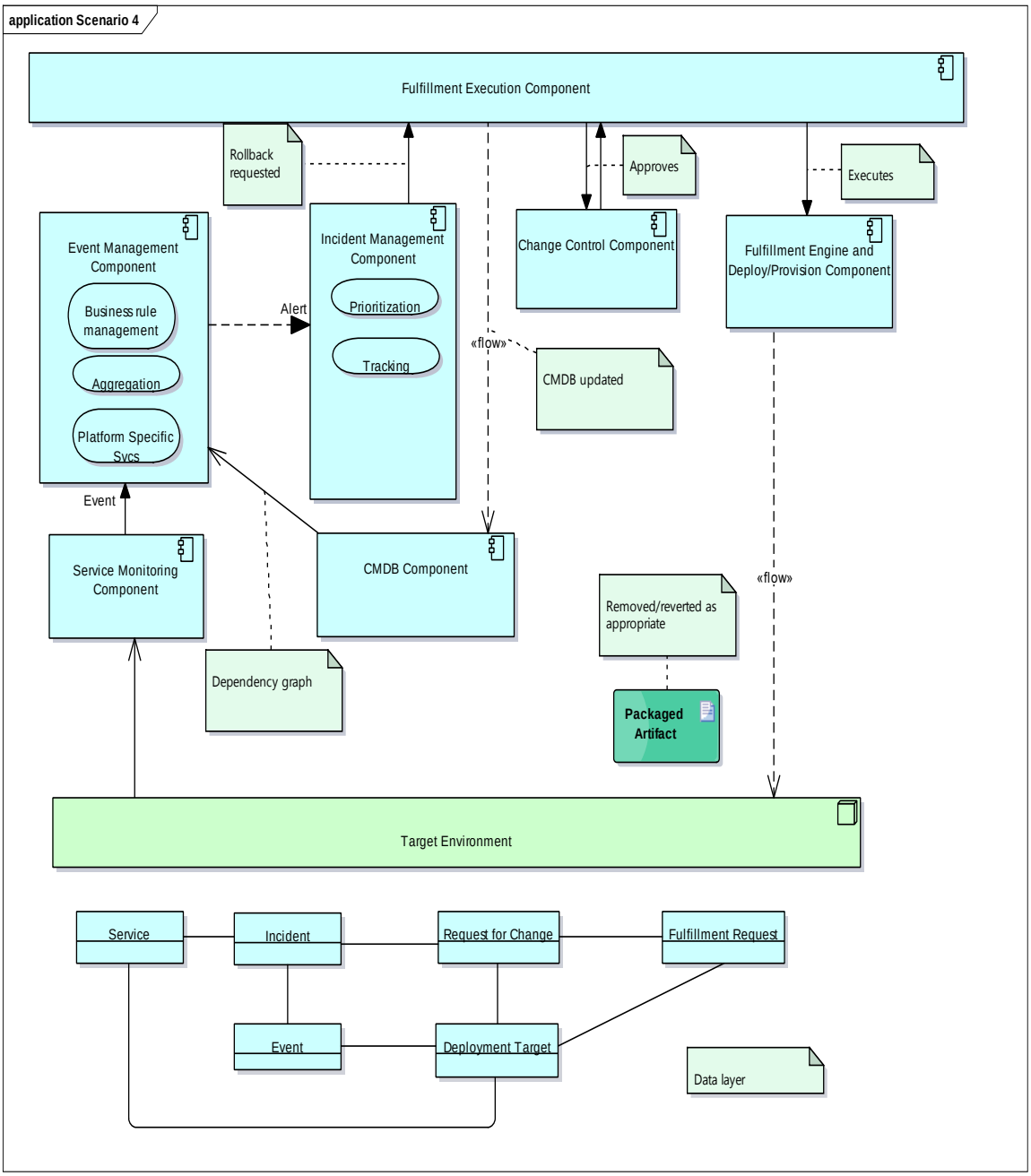


Figure 18: Automated Incident Detection and Rollback

1. The software is now deployed into an environment with some level of operational monitoring, starting at the lowest service monitoring level; e.g., probes and log monitors that detect the basic operational status of the managed element.

2. The Service Monitoring component then raises events into an Event Management component (sometimes called a manager of managers).

3. The Event Management component draws upon CMDB dependencies and its own business rules to determine whether an actual Incident should be declared.
4. The Incident is declared and, if severe enough, may warrant rollback of the change (via a new Change Request).
5. The proposed rollback of the Change is approved via the Change Control component.
6. The Deployment Management system then reverses the change (e.g., through pulling the previous build and re-deploying it).
7. (Not shown) Causal analysis and remediation then are presumed to occur.

Data Layer

One or more Events are recognized against a known pattern resulting in the registration of an Incident against the Service. If rollback is decided, a new Change (RFC) is created (and usually expedited). This then creates a work order in the fulfillment engine, resulting in the removal/reversion of the installed package.

Notes:

- There are various patterns for the roles that element monitors, event managers, and incident systems play, but a three-tier pattern is often seen.
- The term Service Monitoring component is a bit of a misnomer, as true service-awareness is not understood at the element or system level, but rather require the dependencies from the CMDB.

Data Architecture

Data Object	Attributes	Additional Information
Build	Build ID	Unique identifier for a Build
Deployment Target (Configuration Item)	CI ID	Configuration Item unique identifier, representing a CI that is a valid deployment target
Defect	Defect ID	Unique identifier for a Defect
Event	Event ID	Unique identifier for an Event
Fulfillment Request	Fulfillment Request ID	Unique identifier for a Fulfillment Request
Incident	Incident ID	Unique identifier for an Incident
”“	Incident severity	Relative priority/impact of Incident (perhaps derived from service management); used to determine if rollback is called for
Package	Package ID	Unique identifier for a Package
Release	Release ID	Unique identifier for a Release
Request for Change	Change ID	Unique identifier for an RFC

Data Object	Attributes	Additional Information
Product/Service	Product/Service ID	Unique identifier for a Product/Service
”“	Product/Service Version	Identifies the version of a service
Test Case	Test Case ID	Unique identifier for a Test Case

Proposed Changes to the Reference Architecture for the DevOps Scenario

Two changes have been proposed by the Agile work stream and accepted:

- The creation of a Package component
- The renaming of the Service Development component to Source Control component

Kanban and Queueing

Kanban Scenario

Overview

A critical concern for Agile product development is identifying where queues are prescribed, and therefore where work-in-process may accumulate and where demand is expressed for the end user. As Don Reinertsen notes [Reinertsen 2009]:

"Queues matter because they are economically important, they are poorly managed, and they have the potential to be much better managed. Queues profoundly affect the economics of product development (IT services are a form of product). They cause valuable work products to sit idle, waiting to access busy resources. This idle time increases inventory, which is the root cause of many other economic problems.

Queues hurt cycle time, quality, and efficiency. Despite their economic importance, queues are not managed in today's development processes. Few product developers are aware of the causal links between high capacity utilization, queues, and poor economic performance. Instead, developers assume that their cycle times will be faster when resources are fully utilized. In reality, high levels of capacity utilization are actually a primary cause of long cycle time.

Manufacturing companies use operations theory to identify bottlenecks and eliminate them – they have identified the value chain is only as good as the weakest link (or weakest capacity). Little's Law and associated rules can be adopted for modeling the service value chain and eliminate both bottlenecks and waste."

It is a very interesting question, but beyond the scope of this document, as to how queues emerge in a cooperative environment. There is some indication that queues proliferate as environments become more specialized. At a certain point, this seems to become dysfunctional. See [Leffingwell 2010], [Larman 2009], and [Reinertsen 2009].

The Kanban movement is responding to this by essentially centralizing heterogeneous demand into simplified, common queuing mechanisms. A given team's Kanban board may encompass Requirements, Changes, Service Requests, Work Orders, and even Incidents and Problems – anything that results in someone spending some time on it. Which of these the Kanban board includes depends on the context and function(s) of the team in question.

IT4IT™ Agile Scenario

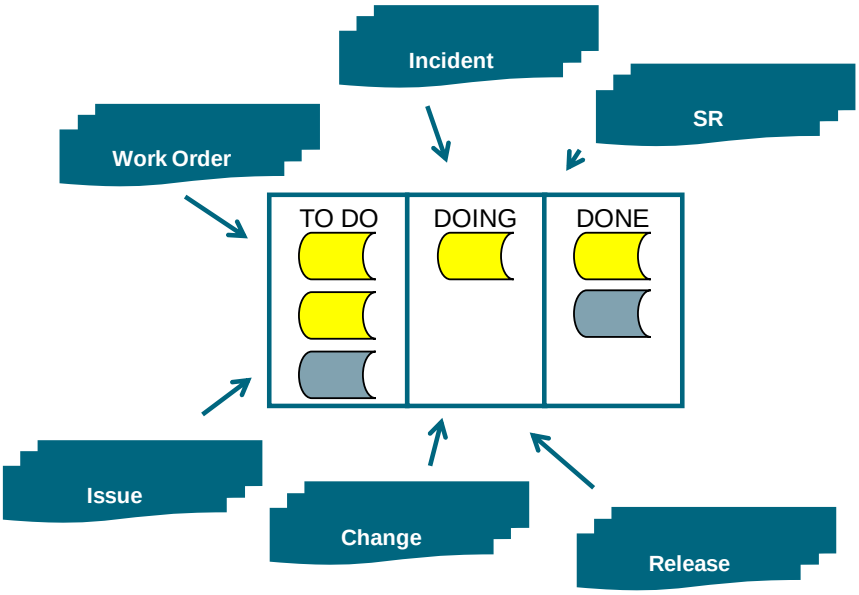


Figure 19: Heterogeneous Kanban Board

This industry direction, while clear, remains largely undocumented. The reader is referred to Ian Carroll's website IT Ops Kanban [Caroll 2013] and the 2014 presentation by ING Bank: ITIL and DevOps at War in the Enterprise [Bouwman 2014]. In that presentation, one of the presenters specifically discourages the use of a problem ticket in a service management tool if the problem is also a user story. Similar themes can be found in Kim's "The Phoenix Project" [Kim 2013] and in Limoncelli's "The Practice of Cloud System Administration" [Limoncelli 2014], p.184:

"Development and operations can best speak the same language by using the same tools wherever possible. This can be as simple as using the same bug-tracking system for both development and operations/deployment issues."

This seemingly simple advice is contradictory to operating models based on differentiated process architectures, such as those based on ITSM implementations.

The IT4IT Reference Architecture therefore explicitly identifies the functional components that imply queues and is working towards a logically unified queuing interface so that demand may be understood globally across the IT Value Chain. Some initial requirements for this follow.

Requirements

Here is a high-level representation of IT4IT functional components possibly containing queues:

Some initial requirements for this might be:

Process	Requirements	
Unified demand	KQ.REQ.01	A view of unified demand should be possible via integrating all queues

IT4IT™ Agile Scenario

Process	Requirements	
Universal queue identification	KQ.REQ.02	All functional components that contain queues should be identified
Common backlog	KQ.REQ.03	Support a common backlog for development and infrastructure tasks
Cumulative flow	KQ.REQ.04	Cumulative flow metrics should be available on all queues
Integrate queues	KQ.REQ.05	Smaller silo queues need to be integrated into longer, cross-functional, higher value queues
Queue data management	KQ.REQ.06	Queue data should have common aggregation into a business intelligence environment (consider process mining)
Queue definition	KQ.REQ.07	Accommodate the need of self-organizing teams to define their own queues, especially lifecycle states (e.g., “definition of done”)

Process Flow

The process questions here are meta-questions:

- How can queues be recognized, managed, and rationalized?

Continuous improvement approaches are often used for this.

Automation Specification

Here is a high-level representation of IT4IT functional components possibly containing queues:

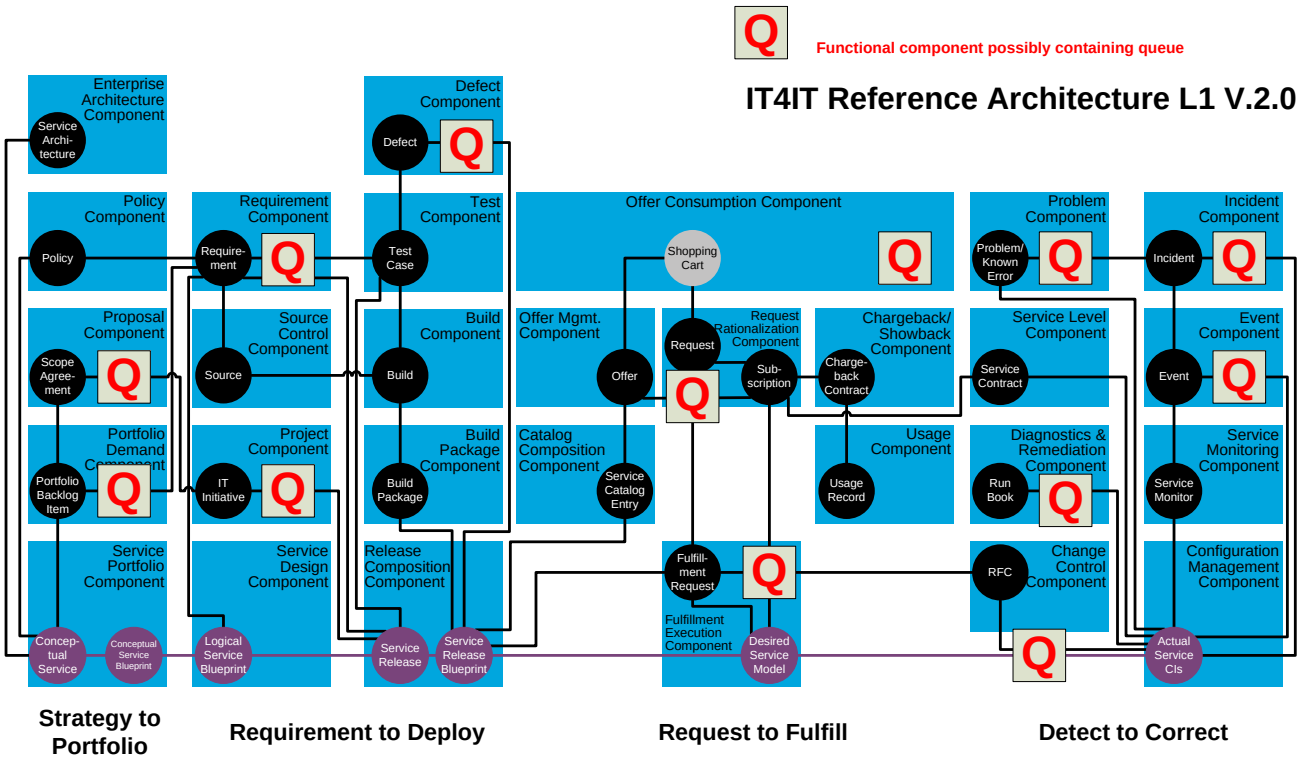


Figure 20: Components Potentially Containing Queues in the IT4IT Reference Architecture

For clarity, here is a list of the identified components:

- Proposal component
- Portfolio Demand component
- Requirement component
- Project component
- Defect component
- Request Rationalization component
- Offer Consumption component
- Fulfillment Execution component
- Diagnostics & Remediation component
- Change Control component
- Event component
- Incident component

One possible enabling solution is to define a simple abstract queuing interface, specifying which components implement it.

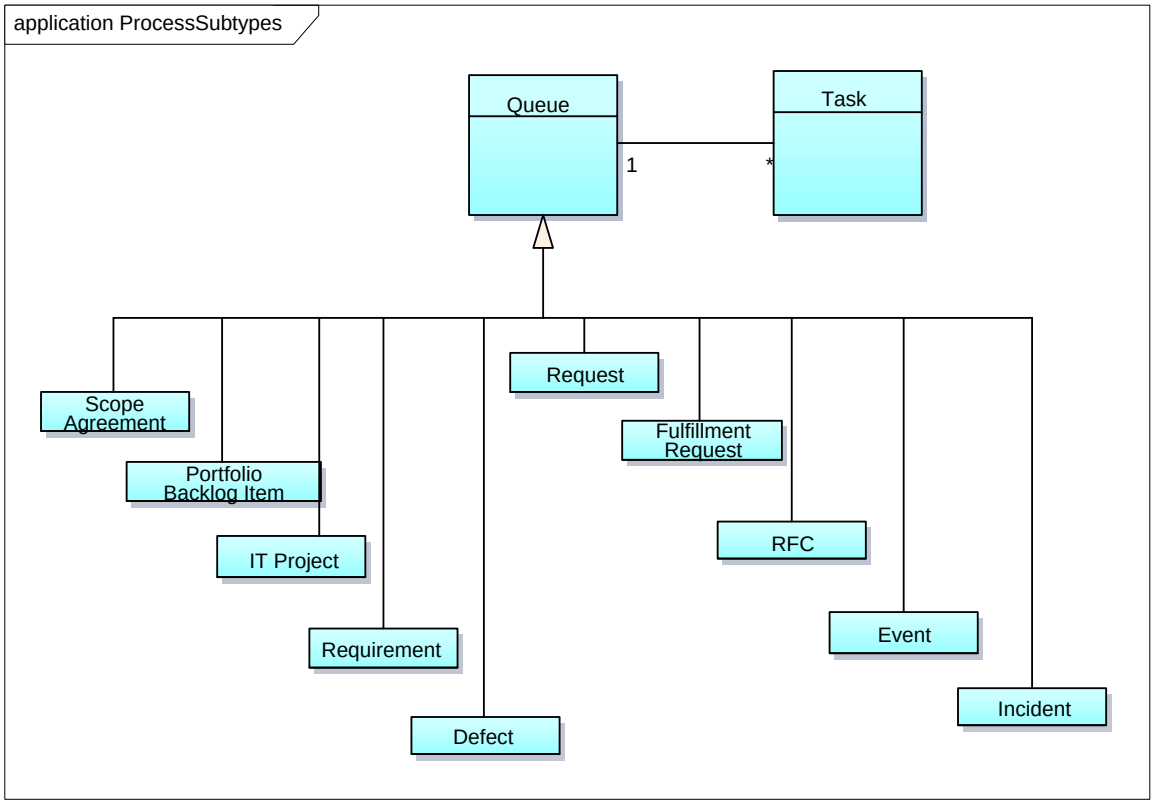


Figure 21: Common Process Class with Subtypes

A similar approach is seen in [Betz 2011a], p.102, where the following entities are defined as subtypes of a master process entity:

- Demand Request
- Project
- Release
- Change
- Service Request
- Transaction
- Incident
- Improvement Opportunity
- Risk
- Problem

Other queues seen in the development world include:

- User story
- Requirement
- Defect
- Requirement
- Epic
- Issue
- Risk
- Action item

Conceptual Architecture

There are two conceptual classes: *Queue* and *Task*.

It is not necessary to define a complex behavioral interface; common data attributes across the queues would add a great deal of value. If all process entities derive from a common type, what does that type require that can apply to all processes?

Any Queue should be able to provide the following information:

- Basic identity information; e.g., "Incident"
- A collection of its Tasks, preferably filtered
- Any given Task, if provided its ID.
- Cumulative Flow Diagram aggregate metrics:
- Arrivals
- Time in Queue
- Quantity in Queue
- Departures

Any Task should be able to provide the following information:

- Identify its Queue; e.g., "Incidents"
- Identity information; e.g., "Oracle outage"
- Process (e.g., what is its current parent Queue?)
- Priority
- Performers (who is it currently assigned to and what is the referral history?)

- Partners (pre and post-dependent Tasks)
- Parents (overarching/aggregate Tasks)
- Estimated completion
- Current status

The following attributes are proposed for the two classes, considered as entities:

Artifact	Attributes	Additional Information
Queue	ID	Globally unique identifier for the queue
”“	QueueName	The process name; e.g., Story, Change, Request
”“	LifecycleStates	Enumeration of valid lifecycle states
Task	ID	Enumeration of valid lifecycle states
”“	QueueID	The foreign key for the parent queue; e.g., Change or Request
”“	Performer	The party currently responsible for it
”“	Precondition	What needs to occur previously?
”“	Postcondition	What must occur afterwards?
”“	Parent	Is there a containing task?
”“	Priority	Within the scope of the process type, what is the priority?
”“	AnticipatedCompletion	When is completion anticipated?

Further work is needed regarding:

- Mutability and historical tracking (audit trails)
- Cross-process prioritization (global prioritization)
- Task estimation

References

(Please note that the links below are good at the time of writing but cannot be guaranteed for the future.)

- [Alliance 2001] A. Alliance: Agile Manifesto and Principles, 2001; refer to: http://agilemanifesto.org/principles.html.
- [Allspaw 2009] J. Allspaw, P. Hammond: 10 Deploys per Day: Dev & Ops Cooperation at Flickr, O'Reilly Publications, 2009; refer to: www.slideshare.net/jallspaw/10-deploys-per-day-dev-and-ops-cooperation-at-flickr.
- [Anderson 2010] D.J. Anderson: Kanban: Successful Evolutionary Change for your Technology Business, Blue Hole Press, Sequim, WA, 2010.
- [Bell 2010] S.C. Bell, M.A. Orzen: Lean IT, CRC Press, Boca Raton, Florida, 2010.
- [Betz 2011] C.T. Betz: Release Management Integration Pattern – Seeking DevOps Comments, 2011; refer to: www.lean4it.com/2011/01/release-management-integration-pattern-seeking-devops-comments.html.
- [Betz 2011a] C.T. Betz: Architecture and Patterns for IT: Service and Portfolio Management and Governance (Making Shoes for the Cobbler's Children), 2nd Edition, Elsevier/Morgan Kaufman, Amsterdam, 2011.
- [Betz 2015] C.T. Betz: Calavera Project, Github, 2015; refer to: https://github.com/dm-academy/Calavera.
- [Bouwman 2014] J-J. Bouwman, M. Heistek: ITIL and DevOps at War in the Enterprise, 2014; refer to: www.youtube.com/watch?v=_dDsdbkSgOc, DevOpsDays.
- [Burrows 2015] M. Burrows: Kanban from the Inside: Understand the Kanban Method, Connect it to what you Already Know, Introduce it with Impact, Blue Hole Press, 2015.
- [Buschmann 1996] F. Buschmann: Pattern-Oriented Software Architecture: A System of Patterns, Wiley, Chichester; New York, 1996.
- [Carroll 2013] L. Carroll: IT Ops Kanban – Kanban Case Study for IT Operations, 2013; refer to: http://itopskanban.wordpress.com/before/.
- [Duvall 2007] P.M. Duvall, S. Matyas, A. Glover: Continuous Integration: Improving Software Quality and Reducing Risk, Addison-Wesley, Upper Saddle River, NJ, 2007.
- [Edwards 2012] D. Edwards: Integrating DevOps Tools into a Service Delivery Platform, 2010; refer to: http://dev2ops.org/2012/07/integrating-devops-tools-into-a-service-delivery-platform-video/.
- [Fowler 1997] M. Fowler: Analysis Patterns: Reusable Object Models, Addison-Wesley, Menlo Park, CA, 1997.
- [Fowler 2003] M. Fowler: Patterns of Enterprise Application Architecture, Addison-Wesley, Boston, MA, 2003.

- [Gamma 1995] E. Gamma: Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA, 1995.
- [Gartner 2014] Gartner: Seven Steps to Start Your DevOps Initiative, 2014; refer to: www.gartner.com/doc/2847717/seven-steps-start-devops-initiative.
- [Goldratt 1997] E.M. Goldratt: Critical Chain, North River, Great Barrington, MA, 1997.
- [Goldratt 2004] E.M. Goldratt, J. Cox: The Goal: A Process of Ongoing Improvement, North River Press, Great Barrington, MA, 2004.
- [Hay 1996] D.C. Hay: Data Model Patterns: Conventions of Thought, Dorset House Pub., New York, 1996.
- [Hay 2006] D.C. Hay: Data Model Patterns: A Metadata Map, Morgan Kaufmann; Oxford: Elsevier Science [distributor], San Francisco, CA, 2006.
- [Hohpe2003] Hohpe, G. & Woolf, B. (2003), Enterprise integration patterns : designing, building, and deploying messaging solutions, Addison-Wesley, Boston
- [Hubbard 2010] D. Hubbard: How to Measure Anything: Finding the Value of Intangibles in Business, Wiley, Boston, MA, 2010.
- [Humble 2011] J. Humble, D. Farley: Continuous Delivery, Addison-Wesley, Boston, 2011.
- [Kaplan 2005] J.D. Kaplan: Strategic IT Portfolio Management: Governing Enterprise Transformation, Pittiglio Rabin Todd & McGrath Inc., United States, 2005.
- [Kim 2013] G. Kim, K. Behr, G. Spafford: The Phoenix Project: A Novel about IT, DevOps, and Helping your Business Win, IT Revolution Press, 2013.
- [Kniberg 2011] H. Kniberg, K. Beck, K. Keppler: Lean from the Trenches: Managing Large-Scale Projects with Kanban, Pragmatic Bookshelf, Dallas, TX, 2011.
- [Krafcik 1988] J. Krafcik: Triumph of the Lean Production System, Sloan Management Review 30(1), 41-52, 1988.
- [Larman 2002] C. Larman: Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process, Prentice Hall PTR, Upper Saddle River, NJ, 2002.
- [Larman 2009] C. Larman, V. Bodde: Scaling Lean & Agile Developments: Thinking and Organizational Tools for Large-Scale Scrum, Addison-Wesley, Upper Saddle River, NJ, 2009.
- [Leffingwell 2010] D. Leffingwell: Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise, Pearson Education, 2010.
- [Liker 2004] J.K. Liker: The Toyota Way: 14 Management Principles from the World's Greatest Manufacturer, McGraw-Hill, New York, 2004.
- [Limoncelli 2014] T.A. Limoncelli, S.R. Chalup, C.J. Hogan: The Practice of Cloud System Administration: Designing and Operating Large Distributed Systems, Vol. 2, Pearson Education, 2014.
- [Maizlish 2005] B. Maizlish, R. Handler: IT Portfolio Management Step-By-Step: Unlocking the

Business Value of Technology, John Wiley & Sons, Hoboken, NJ, 2005.

- [Minick 2012] E. Minick: A DevOps Toolchain: There and Back Again, Slideshare.net, 2012; refer to: www.slideshare.net/Urbancode/building-devops-toolchain.
- [OASIS 2013] OASIS: Topology and Orchestration Specification for Cloud Applications (TOSCA), Version 1.0, 2013; refer to: http://docs.oasis-open.org/tosca/TOSCA/v1.0/os/TOSCA-v1.0-os.html.
- [Ohno 1988] T. Ohno: Toyota Production System: Beyond Large-Scale Production, Productivity Press, Cambridge, MA, 1988.
- [Poppendieck 2003] M. Poppendieck, T.D. Poppendieck: Lean Software Development: An Agile Toolkit, Addison Wesley, Boston, 2003.
- [Poppendieck 2007] M. Poppendieck, T.D. Poppendieck: Implementing Lean Software Development: From Concept to Cash, Addison-Wesley, London, 2007.
- [Reinertsen1997] D.G. Reinertsen: Managing the Design Factory: A Product Developer's Toolkit, Free Press, New York; London, 1997.
- [Reinertsen 2009] D.G. Reinertsen: The Principles of Product Development Flow: Second Generation Lean Product Development, Celeritas, Redondo Beach, CA, 2009.
- [Ries 2011] E. Ries: The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses, Crown Business, New York, 2011.
- [SAFE 2015] The Scaled Agile Framework (SAFe), Scaled Agile, Inc., 2015; refer to: www.scaledagile.com.
- [Scotland 2010] K. Scotland: Defining the Last Responsible Moment, 2010; refer to: http://availagility.co.uk/2010/04/06/defining-the-last-responsible-moment.
- [Shortland 2012] A. Shortland, M. Lei: Using Rundeck and Chef to Build DevOps Toolchains, 2012; refer to: http://dev2ops.org/2012/05/using-rundeck-and-chef-to-build-devops-toolchains-at-chefcon/.
- [Silverston 2008] L. Silverston: The Data Model Resource Book, Volume 3: Universal Patterns for Data Modeling, Wiley, Indianapolis, Ind., 2008.
- [Thompson 2014] L. Thompson: Hitchhikers Guide to OpenStack Toolchains, 2014; refer to: www.openstack.org/assets/presentation-media/Hitchhikers-Guide-to-OpenStack-Toolchains.pdf.
- [TSO 2011] ITIL Service Strategy: 2011 Edition, The Stationery Office, Norwich, UK, 2011.
- [Wiggins 2015] A. Wiggins: The Twelve-Factor App., 2015; refer to: http://12factor.net/.
- [Womack 1990] J.P. Womack, D.T. Jones, D. Roos: The Machine that Changed the World, based on the Massachusetts Institute of Technology 5-million dollar 5-year study on the Future of the Automobile, Rawson Associates, New York, 1990.
- [Womack 2003] J.P. Womack, D.T. Jones: Lean Thinking: Banish Waste and Create Wealth in your Corporation, Free Press, New York, 2003.

Acronyms and Abbreviations

CMDB	Configuration Management Database
CMMI	Capability Maturity Model Integration
DevOps	Development and Operations
DIA	DevOps Implementation Approach
DIM	DevOps Implementation Model
DMM	DevOps Maturity Model
IT	Information Technology
ITIL	Information Technology Infrastructure Library
ITSM	IT Service Management
KPI	Key Performance Indicator
MTBF	Mean Time Between Failures
MTTR	Mean Time to Recovery
RUP	Rational Unified Process
SAFe	Scaled Agile Framework
SIT	System Integration Testing
TOSCA	Topology and Orchestration Specification for Cloud Applications (OASIS)
UAT	User Acceptance Testing
WIP	Work-in-Process

Acknowledgements

The Open Group gratefully acknowledges the contribution of the following people in the development of this document:

- Charles Betz
- Eran Cohen
- Sue Desiderio
- Phillippe Geneste
- Gunnar Menzel
- Lars Rossen
- Vasu Sasikanth Sankhavaram
- Karel van Zeeland

About the IT4IT™ Forum

The IT4IT Reference Architecture refers to the capability or capabilities required to manage the business of IT, covering IT end-to-end from plan, through build and operate. It assumes the principle that the business of running IT is industry-agnostic and that IT leaders share the same problems and opportunities in managing the service lifecycle effectively. At the core, these problems are rooted in IT structure, competencies, and capabilities and the missing link has been the lack of an IT operating model. The IT4IT Reference Architecture proposes that it is possible to establish an IT operating model standard mapped to the existing IT landscape yet flexible enough to support the volatility inherent in the IT industry and accommodate changing IT paradigms (composite apps, Agile Development, mobile technology, multi-sourcing, etc.).

The IT4IT Forum was created when its predecessor, the IT4IT Consortium, transferred its activities to The Open Group. The IT4IT Consortium came into being in 2011 as a practitioner-driven initiative. The Consortium was comprised of IT professionals from multiple industry segments and several IT vendors who agreed to share their experiences for the purpose of developing and publishing future-safe prescriptive guidance for implementing end-to-end an IT4IT architecture with full insight. Past and present members include Enterprise Architects and IT department leaders or industry consultants from: Royal Dutch Shell, Achmea, Munich RE, PwC, Accenture, AT&T, HP IT, ING Bank, and University of South Florida.

"Cloud services and multi-provider outsourcing are adding new degrees of complexity to IT service management. The Consortium will use its real-life, cross-industry expertise to define a new operating model for IT."

Dr. Dirk Heiss, Global Infrastructure Services Officer, Munich RE

For more information on the IT4IT Forum, please visit www.opengroup.org/it4it.

About The Open Group

The Open Group is a global consortium that enables the achievement of business objectives through IT standards. With more than 500 member organizations, The Open Group has a diverse membership that spans all sectors of the IT community – customers, systems and solutions suppliers, tool vendors, integrators, and consultants, as well as academics and researchers – to:

- Capture, understand, and address current and emerging requirements, and establish policies and share best practices
- Facilitate interoperability, develop consensus, and evolve and integrate specifications and open source technologies
- Offer a comprehensive set of services to enhance the operational efficiency of consortia
- Operate the industry's premier certification service

Further information on The Open Group is available at www.opengroup.org.