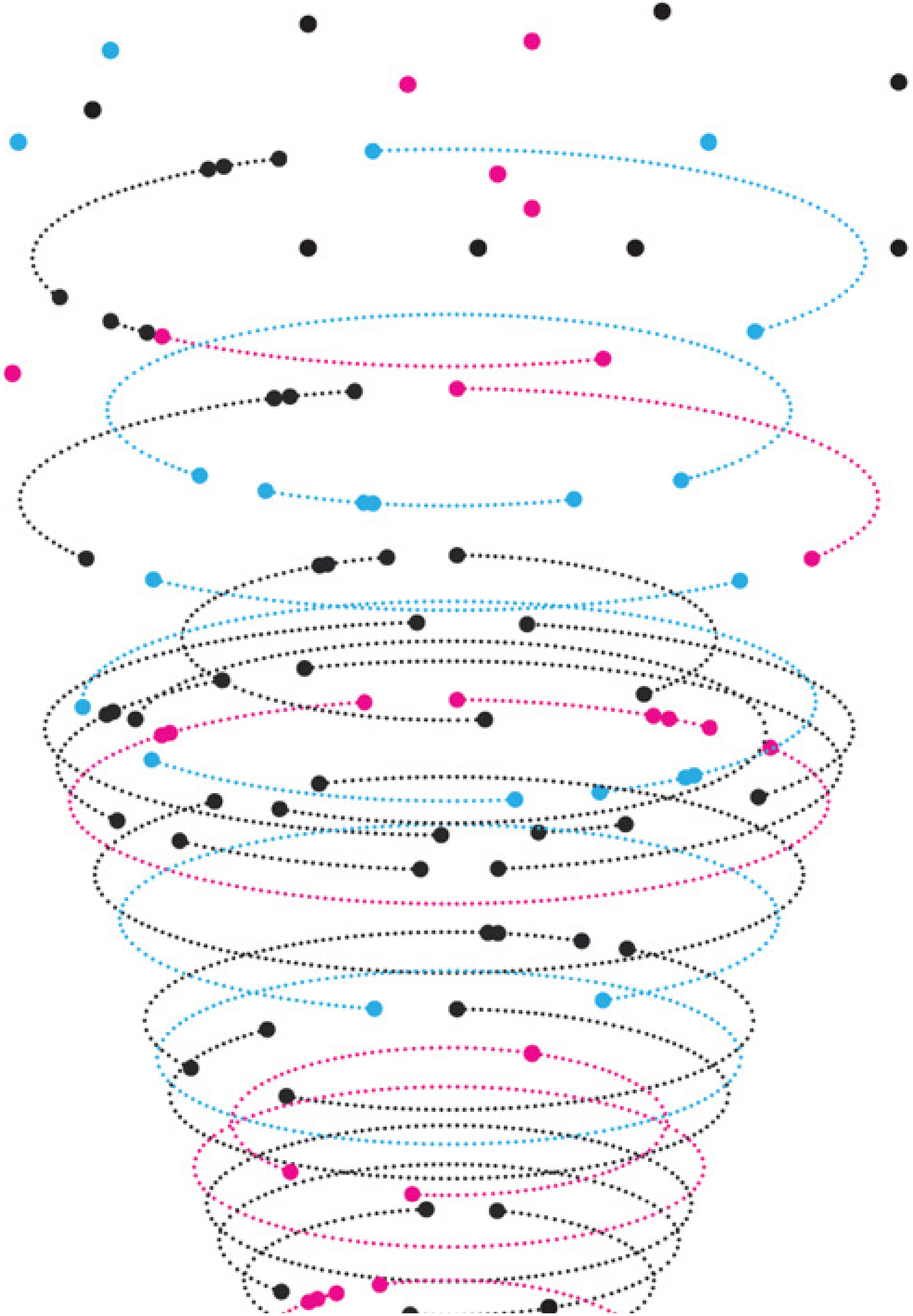


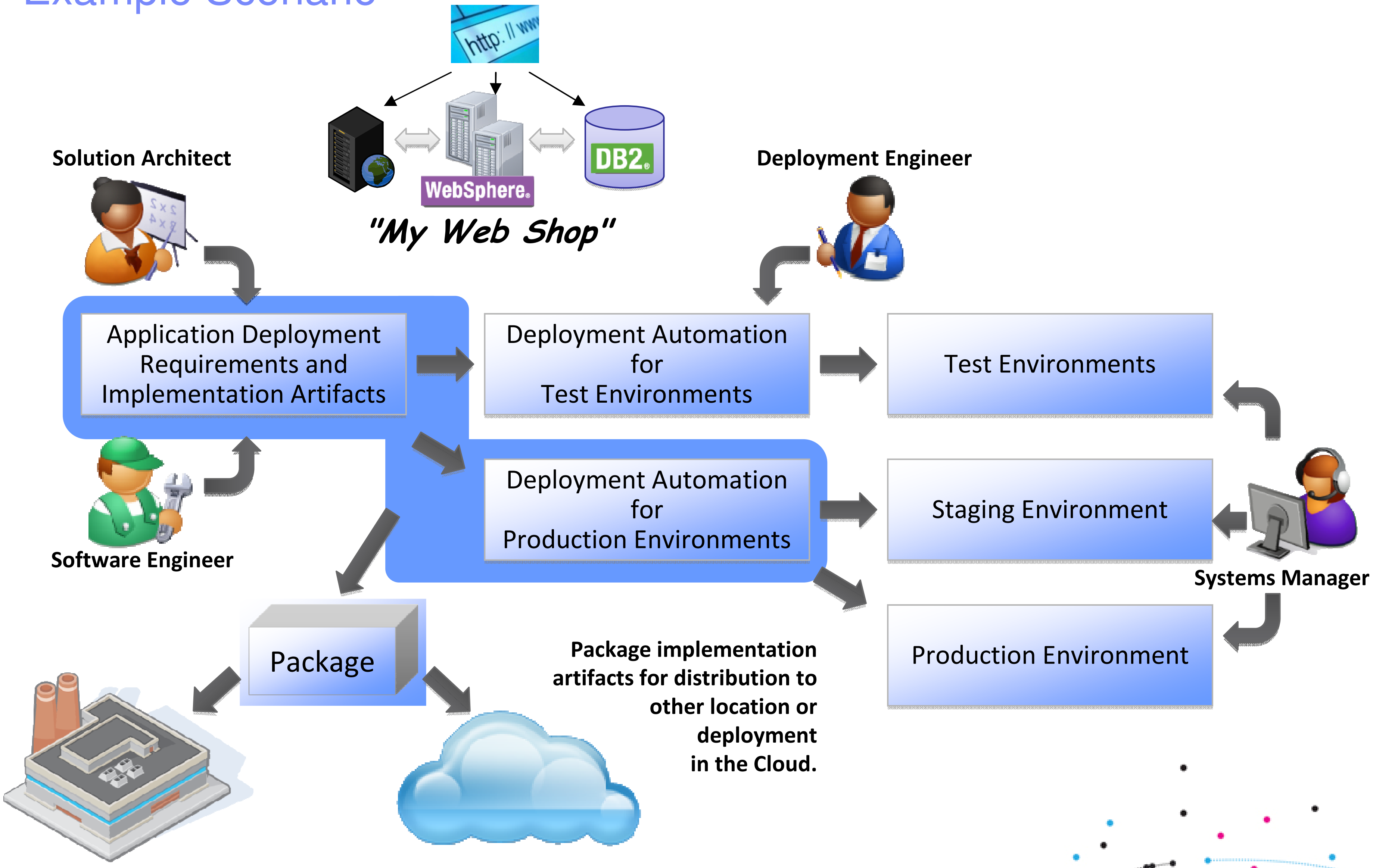
IBM Next Generation ALM Seminar

Davyd Norris – IBM Rational
Tony Altamura – Odecee

Challenges in Today’s IT Environment

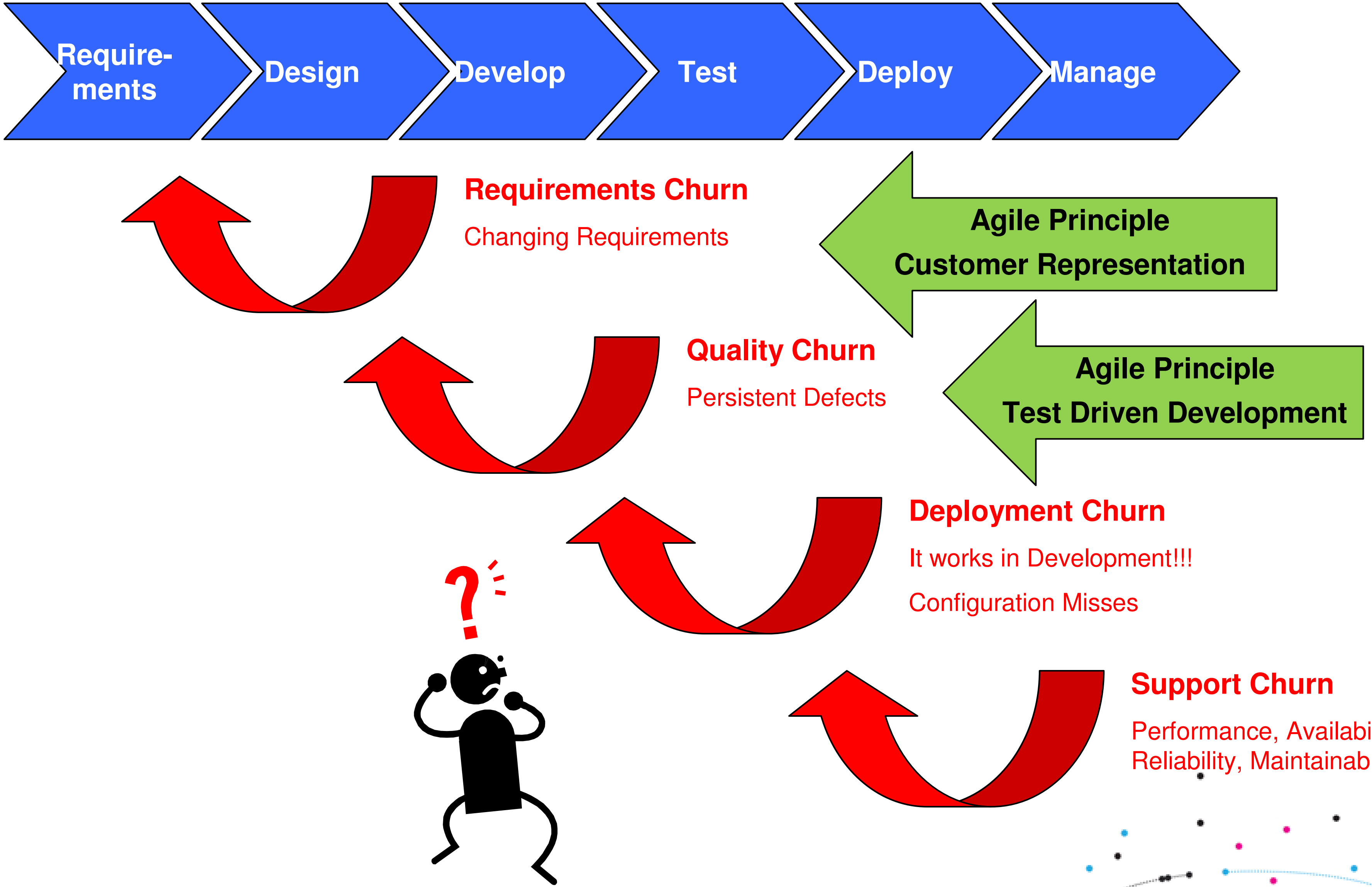
- Integration between Development and Operations teams is fractured due to:
  - different reward systems: Business Functionality Improvements vs. Runtime Stability
  - multiple management chains with inherent politics
  - fragile or non-existent trust between groups: “You broke my app!” vs. “You brought down my environment!”
  - conflicting standards and unclear ownership: “You did it wrong” and “Its not my job”
- Complicated by:
  - do more with less: organizational shrinkage, retirement, staff reductions, etc;
  - brain drain or proprietary expertise: knowledge is captured in someone's head rather than an accessible, repeatable format
  - unclear delivery pipelines for new/enhanced technologies and functionality
- Contributes to:
  - lack of representative environments for spin up and test
  - manually intensive deployments due to lack of automation, informal coordination amongst deployment specialists
  - poor value realization from limited adoption of new tools, technologies and methods for enhanced delivery and operations

Example Scenario



Package implementation
artifacts for distribution to
other location or
deployment
in the Cloud.

Need to Integrate Software Delivery and Operations



Requirements | Design | Develop | Test | Deploy | Manage

Requirements Churn

Changing Requirements

Quality Churn

Persistent Defects

Deployment Churn

It works in Development!!!

Configuration Misses

Support Churn

Performance, Availability, Reliability, Maintainability

Agile Principle
Customer Representation

Agile Principle
Test Driven Development

Deployment is a Complex Problem

- Development and Operations teams collaboration challenges
  - Hand-off from development teams is inconsistent and manual
  - Application component requirements do not match IT infrastructure
- Deployment requirements are difficult to validate
  - Enterprise, Software & IT architects all use different formats
  - No standardization or templates for reuse
- Complex series of steps
  - Deployment engineers often execute manual steps
  - Not repeatable, prone to error
  - Automations are hard to build, maintain and reuse
  - Hard to tell what if the right things were installed

India dev
China dev
US Dev
Operations Manager
Deployment Architect
Operations

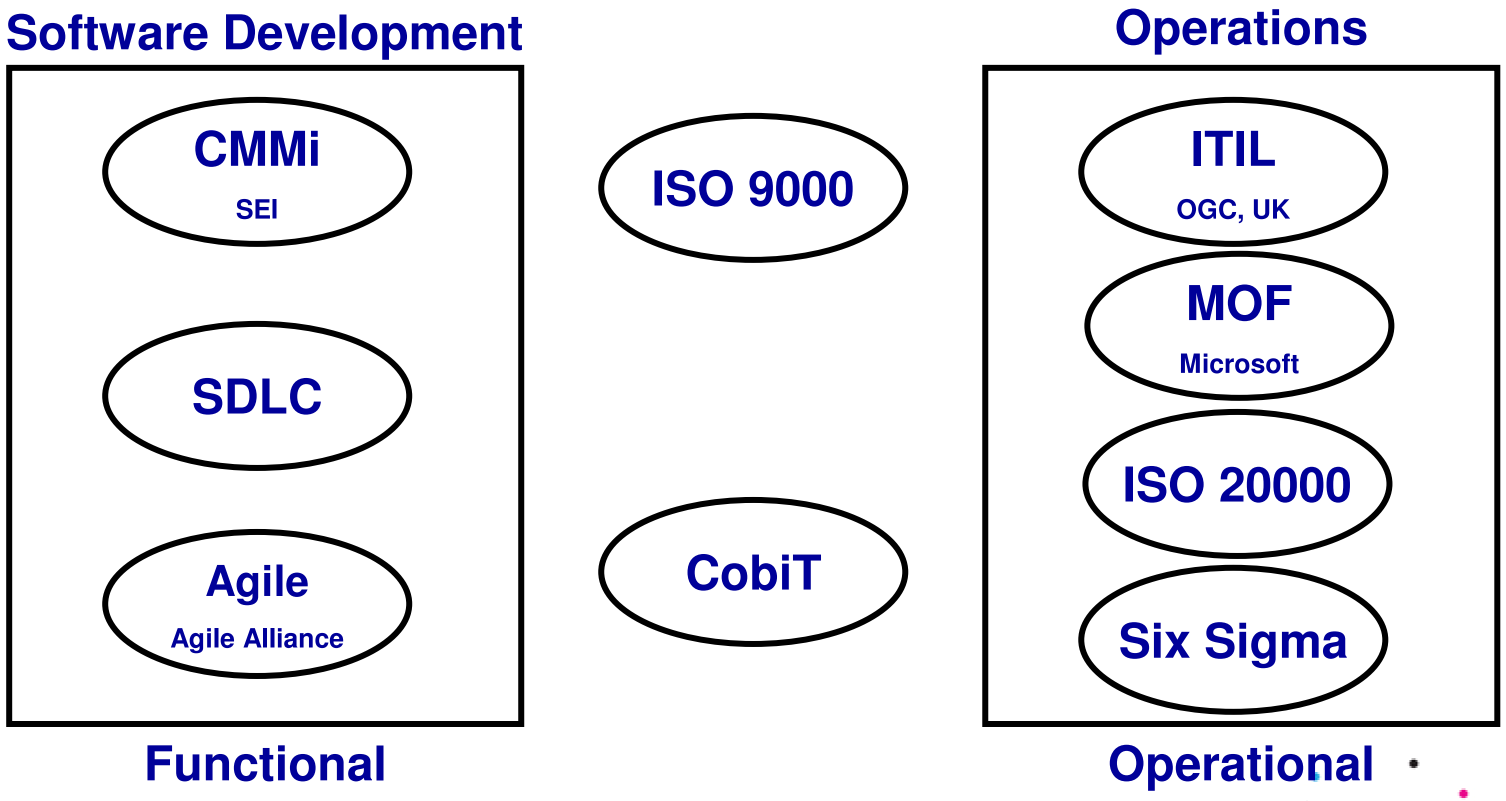
✓ 50% of applications put into production are later rolled back
(Gartner)

✓ 60% - 80% of an average company's IT budget is spent on maintaining existing applications
(Intelligent Enterprise.com)

✓ Software related downtime cost industries almost $300 billion annually
(CENTS - Comparative Economic Normalization Technology Study)

Delivery and Operations Use Separate Process Control Frameworks

- Accentuates Enterprise IT Integration Challenges!



Software Development

CMMi

SEI

SDLC

Agile

Agile Alliance

Functional

ISO 9000

CobiT

Operations

ITIL

OGC, UK

MOF

Microsoft

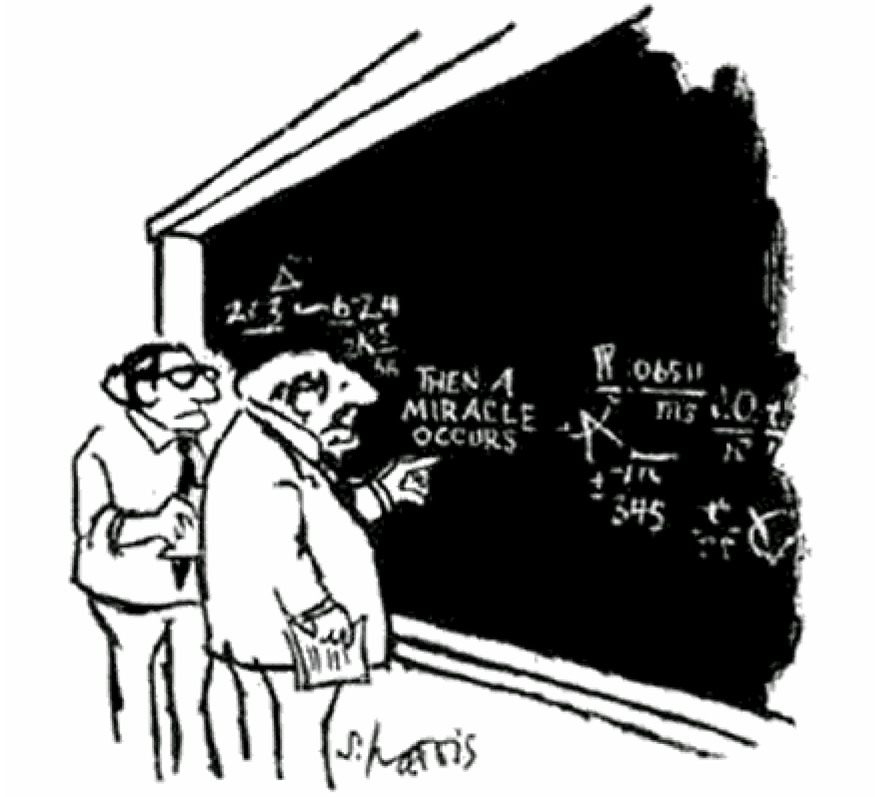
ISO 20000

Six Sigma

Operational

The Result

- Software not designed for Operability and Supportability
- Operations Processes not geared for Service Management

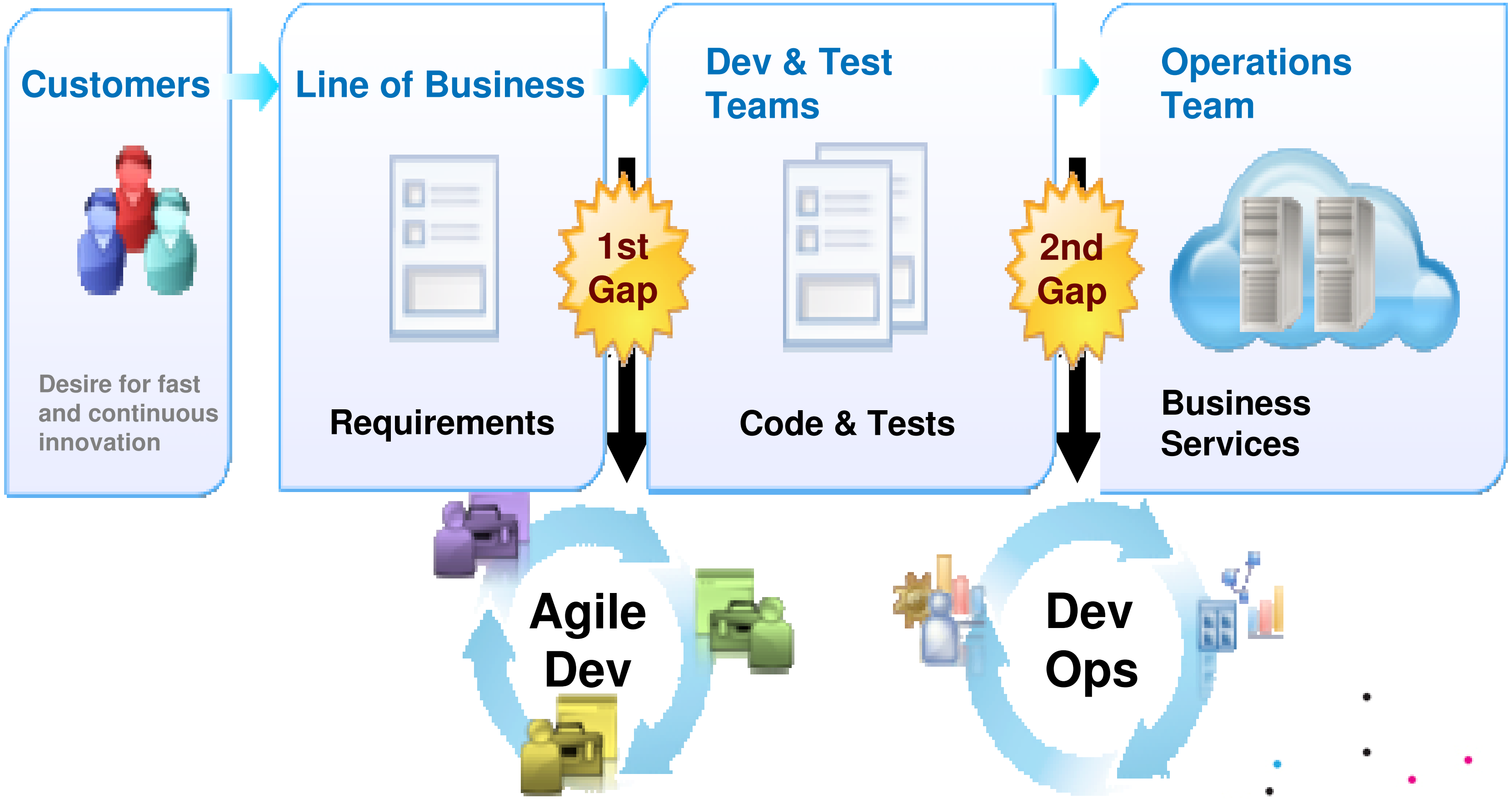


"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

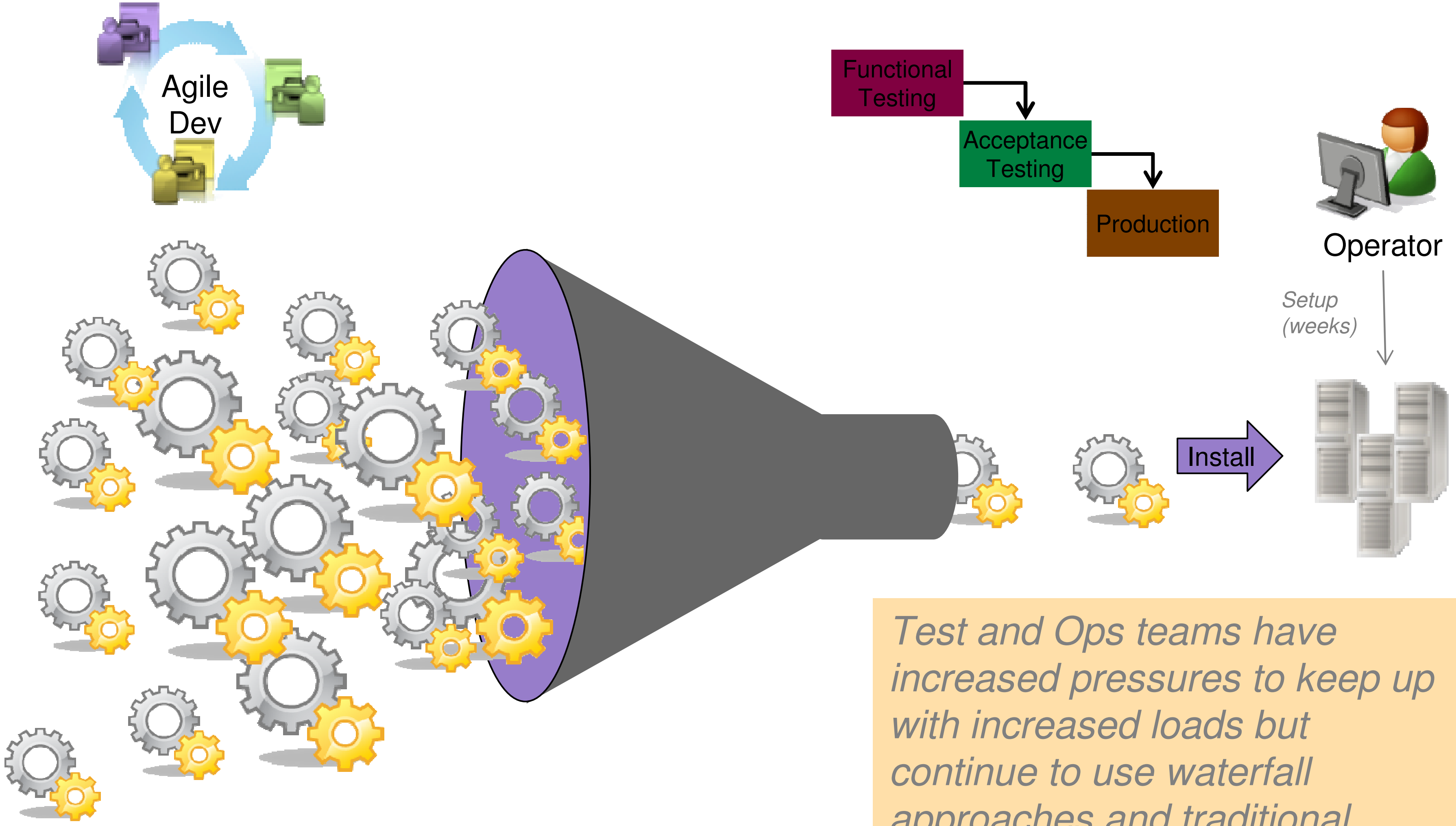
NextGen ALM has Emerged in Response to Multiple Challenges at All Levels of the Business

- People Challenges
  - – Multiple skills versus silo skills
  - – Automated information repository versus “human repository”
  - – Massive and ongoing change
- Technology Challenges
  - – “Next big thing” driving business innovation
  - – Industry innovating faster than it can absorb change
  - – “State of the art” is a sliding scale
- Organizational Management Challenges
  - – IT is driving revenue and business differentiation
  - – Impact of failure is massive
  - – Lines between IT and the business have blurred
- Viewed as a way to surmount People, Technology, and Organizational challenges

Addressing Application Lifecycle Management gaps



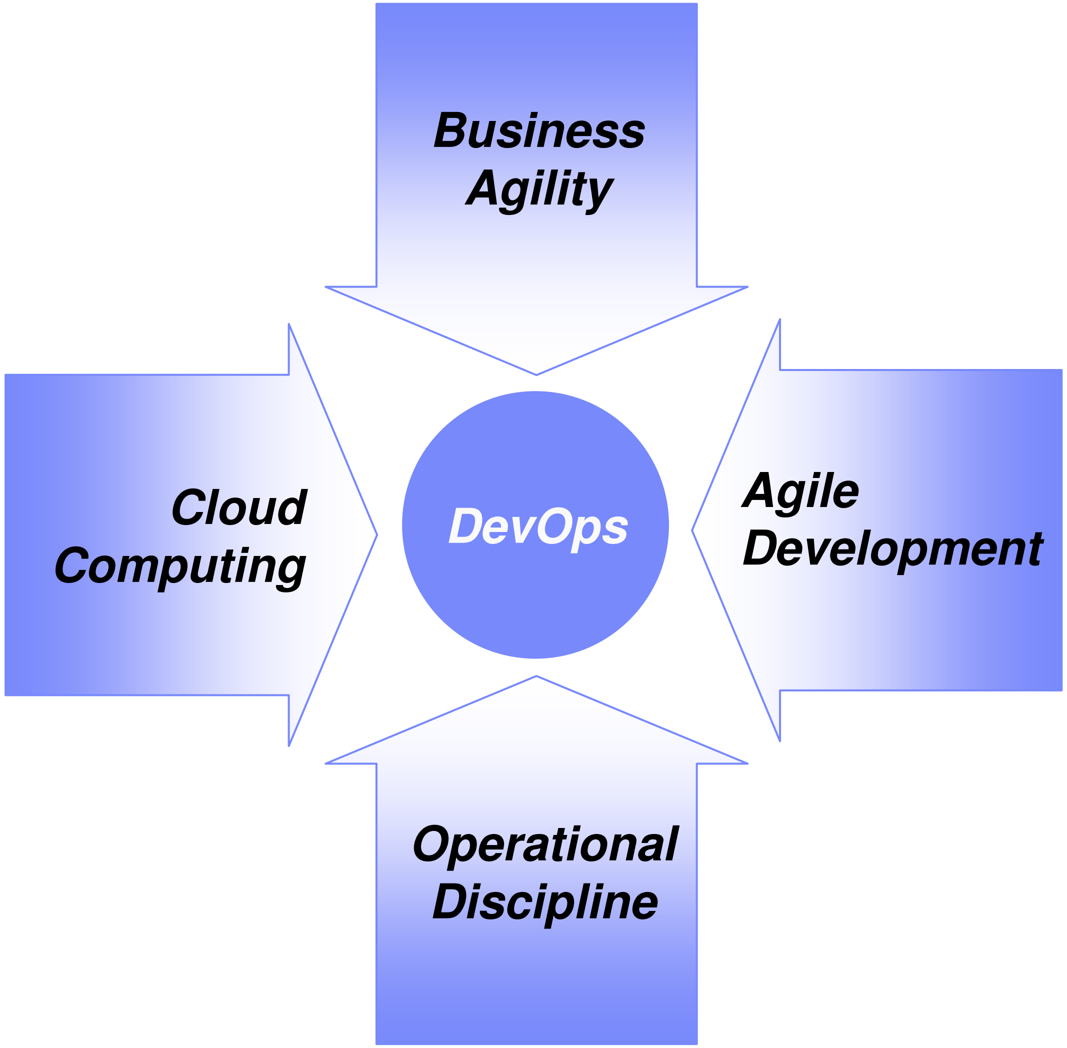
With only Agile Development improvements…



Test and Ops teams have increased pressures to keep up with increased loads but continue to use waterfall approaches and traditional tools.

CI builds are piling up

Four key drivers are putting DevOps on ‘must do’ list!



Business Agility

Cloud Computing

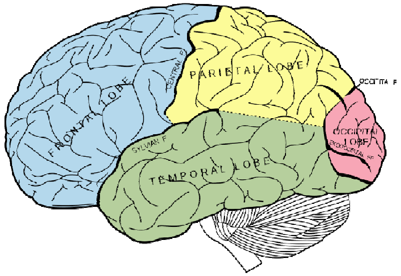
DevOps

Agile Development

Operational Discipline

DevOps Culture & Techniques

- What does "DevOps" actually mean to an Enterprise?
  - “increased application velocity with managed risk”.
  - not simply a statement of rapid provisioning, but more like changing the mindset of Operations to artifacts, rather than administration
- Challenges
  - Agile development implies dedicated, autonomous, empowered teams
  - Operations teams organized as a shared resource
  - how do you deliver rapidly while utilizing shared, non-dedicated resources and maintain environmental integrity
- How do we get there?
  - need to understand processes & standards
  - tools do not “give” you DevOps, but promote, enable and automate best practices



What We Know vs. What We Don’t Know

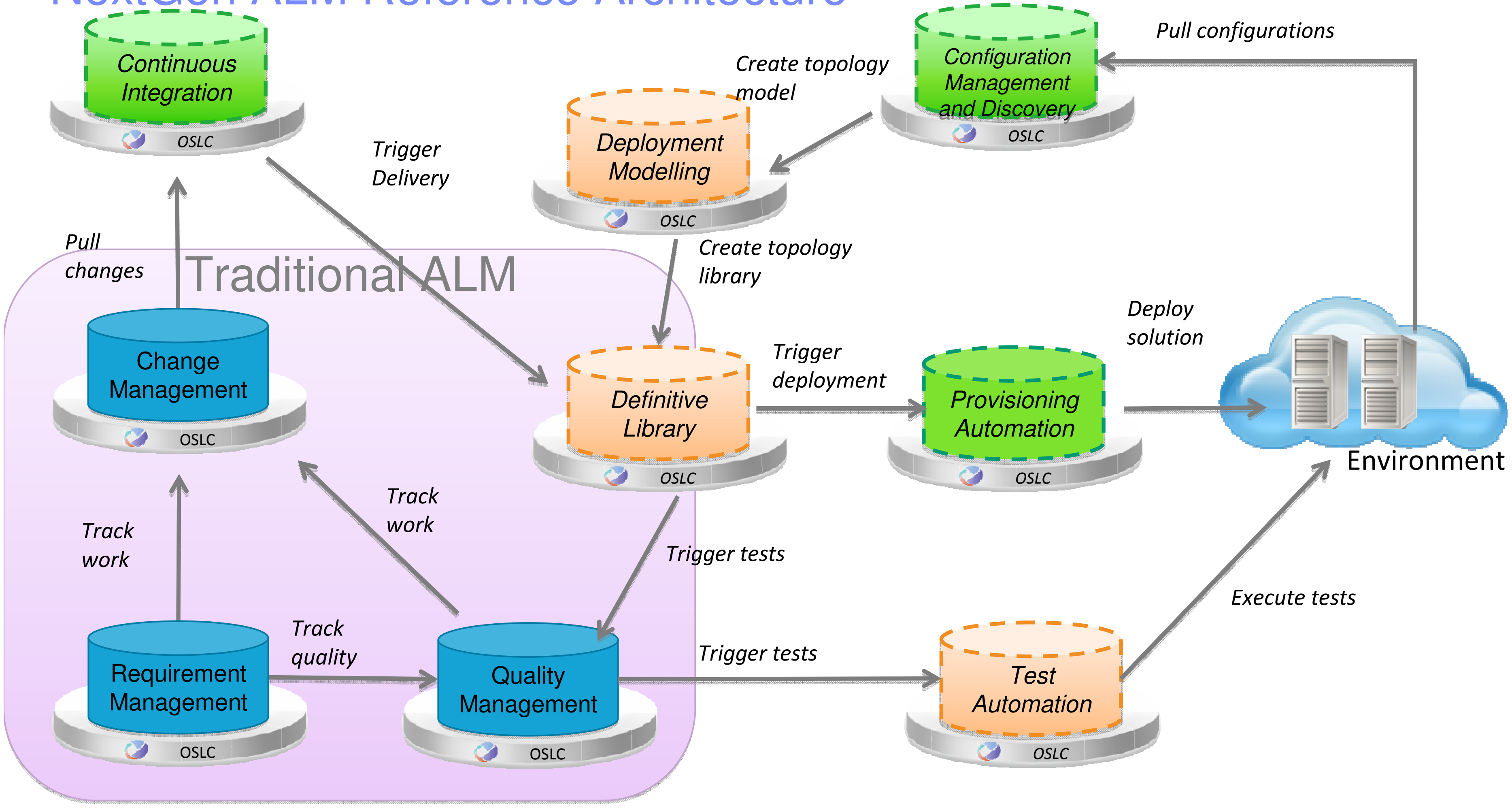
- Companies have processes that work (and some that don’t)
  - But are those existing processes providing the full set of functionality required to run today’s business?
  - Is there even an awareness that things can be done differently?
- Best practices from 1990 and 2000 likely have new, more efficient ways to do it today
  - How can someone even learn about it?
- There are many moving parts!
  - Before we even start to implement NextGen ALM in an organisation there are already a vast array of platforms, products, technologies, integrations, methods and tools
- We propose the following
  - “12 Steps to Better DevOps” to update existing processes. Kind of a “DevOps 12 step recovery program”
  - A NextGen ALM Reference Architecture, defining key elements in the solution
  - A set of NextGen ALM best practices or patterns, which can be implemented separately or layered on top of each other, providing greater value than the sum of the parts

12 Steps to Better DevOps*

- Do you use source control for your build and configuration artifacts?
- Can you deploy a system in one step?
- Do you deploy your applications daily and verify them?
- Do you have an issue tracking system for operations, linked to a bug database used for development?
- Do you validate platform software against expected KPIs, before deploying your application?
- Do you have well defined delivery pipeline?
- Do you have agreed upon patterns for applications and platforms?
- Can your developers launch, use, and destroy representative environments on demand?
- Do you provide Infrastructure and Platform as a Service for your development teams?
- Do you have automated tests to validate your application function and security?
- Do your new operation engineers understand how to automate system administration?
- Do your operations and development teams collaborate regularly?

*Based on “The Joel Test: 12 Steps to Better Code”
http://www.joelonsoftware.com/articles/fog0000000043.html

NextGen ALM Reference Architecture

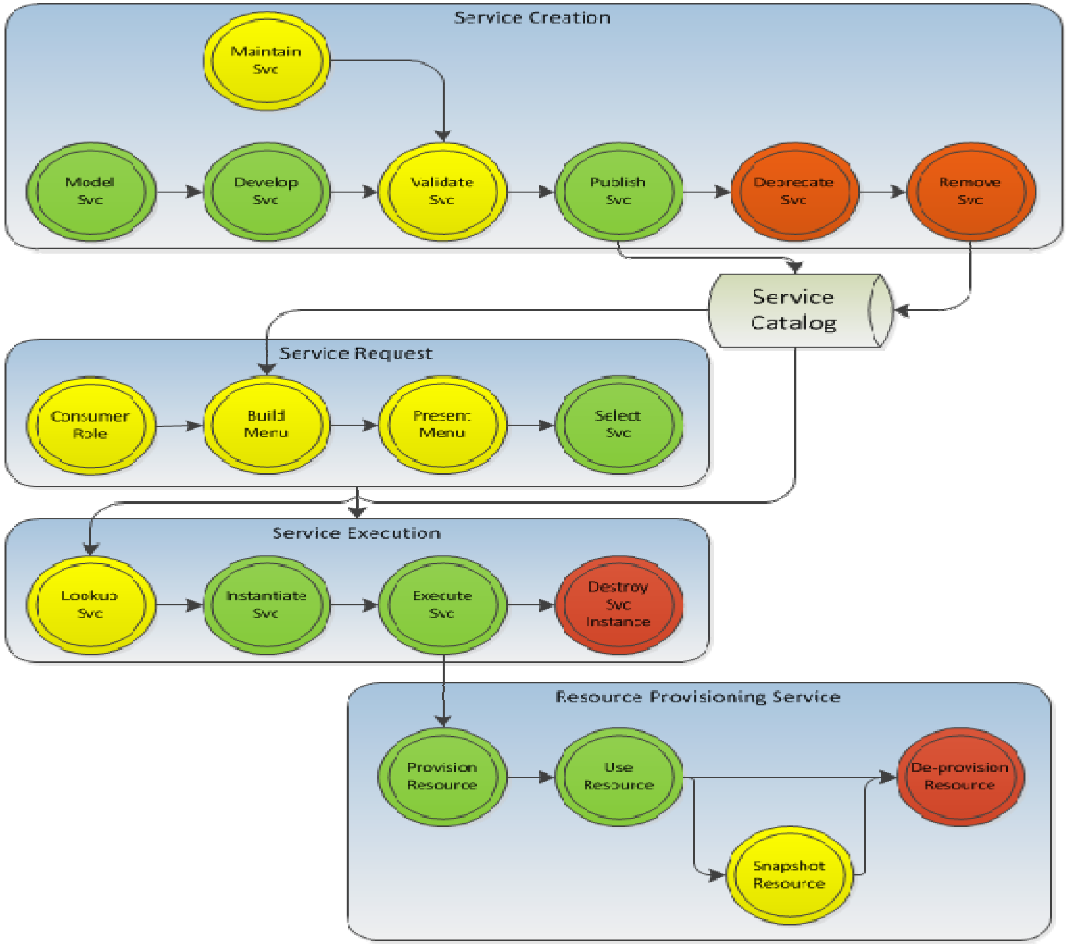


Adopt in any order, at any time

Best Practice: Define a Solution Lifecycle

- A Solution Lifecycle should define how new Solutions are created, deployed, maintained, and retired. A Solution should incorporate aspects of the platform, middleware, and application. New Solutions should adhere to the established architectures.
- Implement it by:
  - Impose a standard architectural pattern for applications to follow before they are integrated into the shared environment.
  - Define a consistent logical architecture for each application
  - Define a physical architecture for each environment which supports the logical architecture as part of an established pipeline
  - Map application onto the infrastructure
- Avoid Anti-patterns:
  - Lack of logical consistency among environments along the pipeline
  - Lack of consistent conceptual framework for describing architecture between Development & Operations
  - Not considering the lifecycle of individual applications and how the change a single application impacts others; may require compliance for all other apps before one app's dependencies may change.

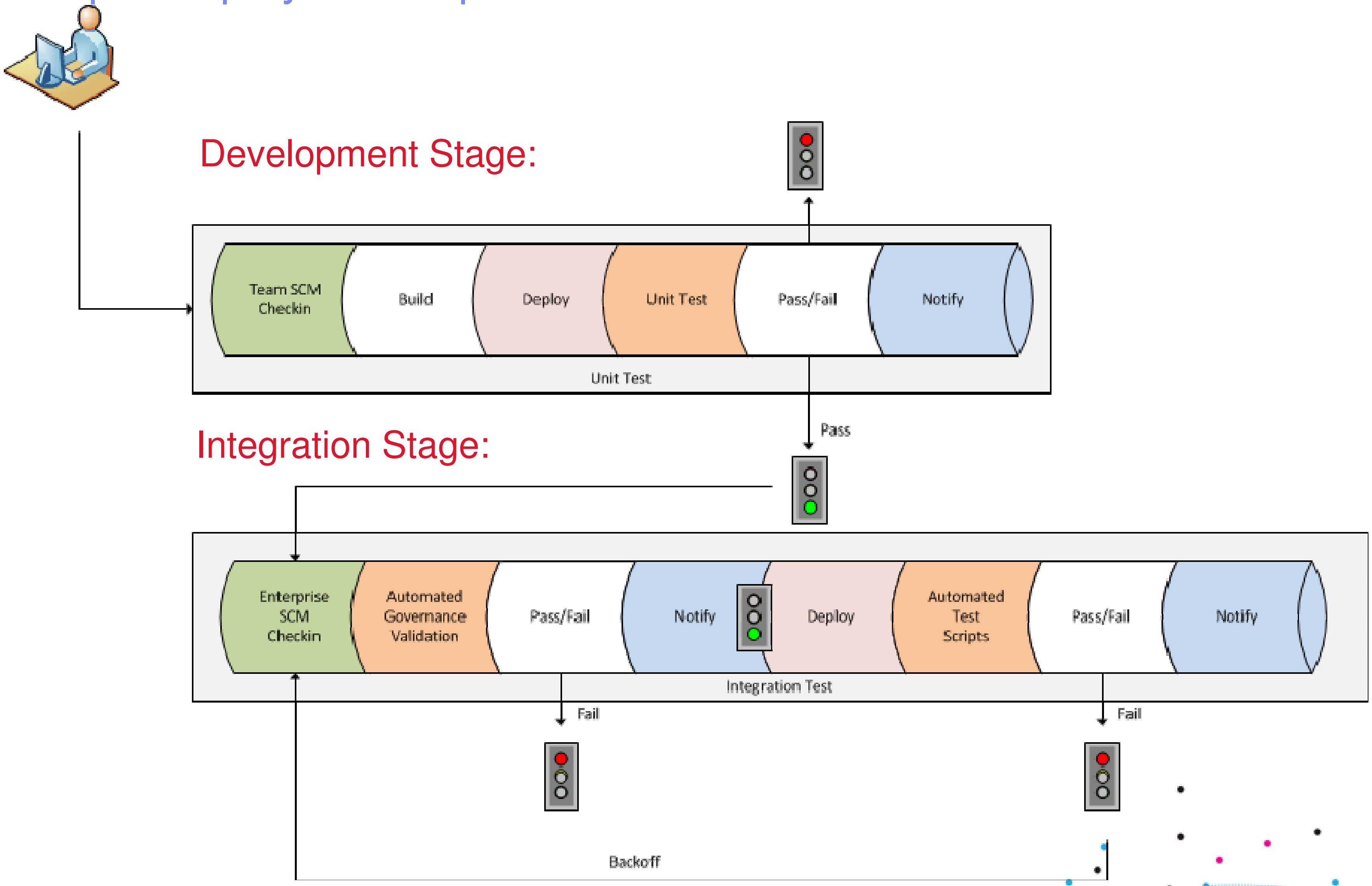
Example Solution Lifecycle



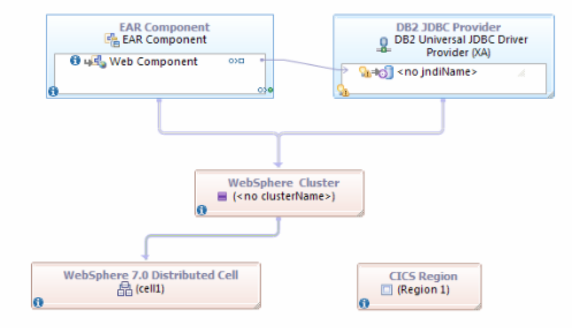
Best Practice: Define a Delivery Pipeline

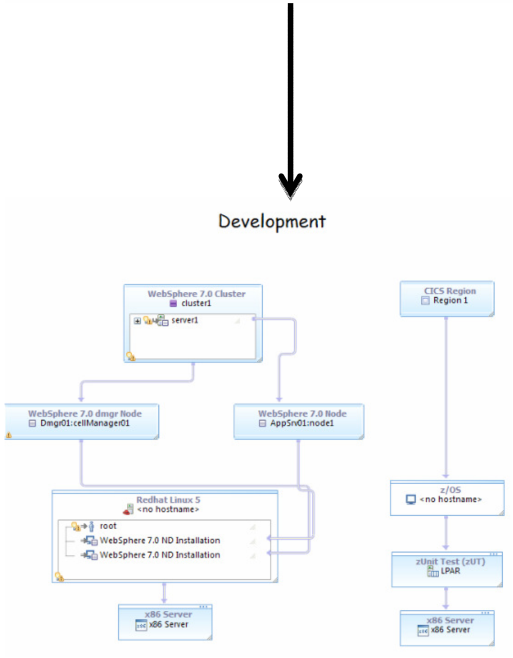
- Define a standard pipeline for delivery of the solution into each environment
  - there are many ways to deliver development artifacts through the various testing stages and into a production environment
  - left alone, each team and organization will create delivery solutions for their own particular purposes, none of which will allow for reuse and integration
- Establish streamlined governance that comes from using standard patterns rather than manual governance that requires detailed knowledge to guarantee compliance
  - delivery pipelines formalize an end-to-end process to provide common and consistent mechanisms to manage asset migrations
- Implement by:
  - well defined processes and hand-offs
  - common, automated mechanisms to ensure consistent build, test and promotion.
  - complete flow from Unit Test through Production
  - well defined interfaces for interaction and integration
  - standardized reporting mechanisms for pipeline activity health
- Avoid Anti-patterns:
  - roll-your-own methods: Just because you can, doesn't mean you should

Example Deployment Pipeline

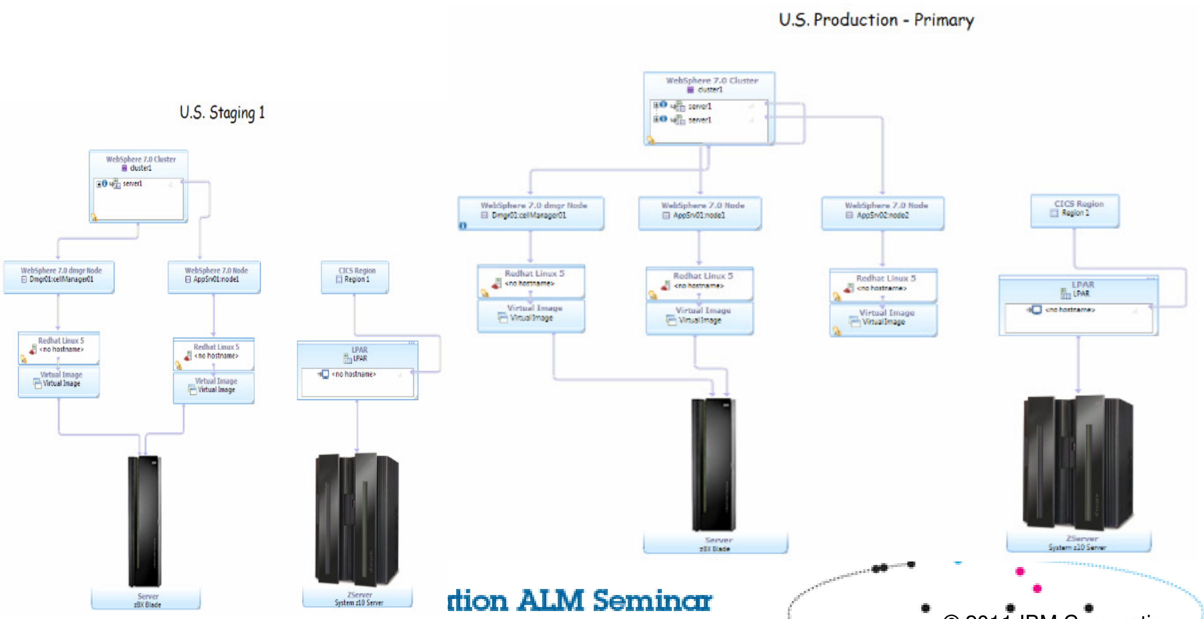


Pipeline Example – Development Stage



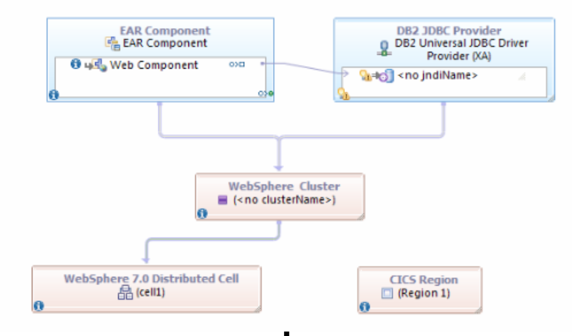


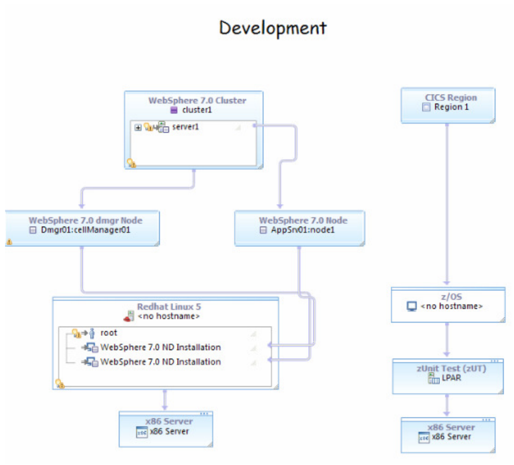
- Deploy/Debug/Test/ Validate Development stage
- Perhaps drive via Continuous Delivery for short feedback loops
- Define automation for deployment and automation for verification/acceptance tests

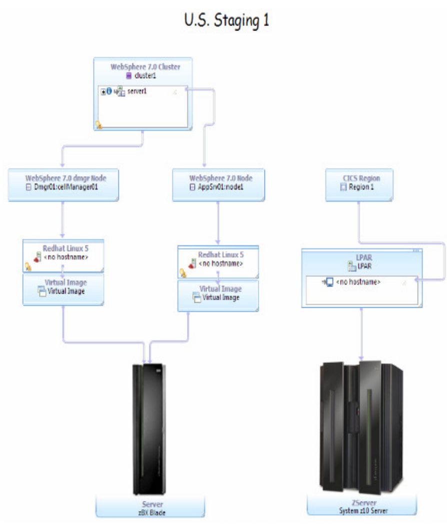


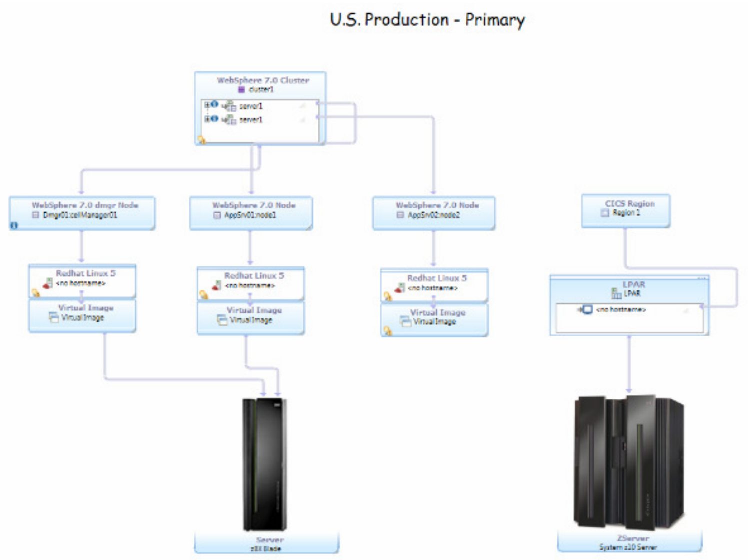
Pipeline Example – Integration Stage

- Promote to next staging area
- Verify deployment automation prepared in Development also works in Staging



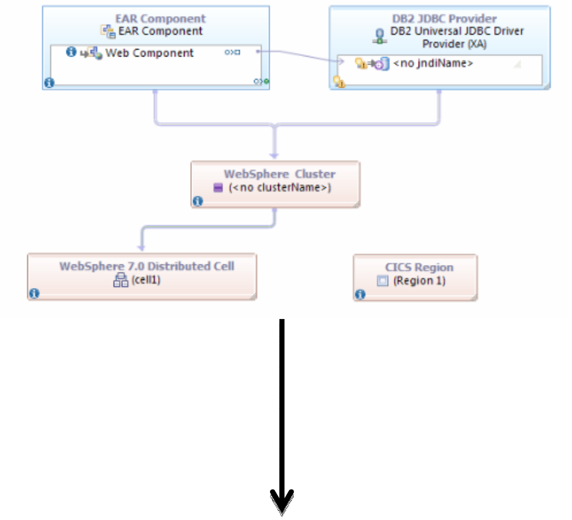


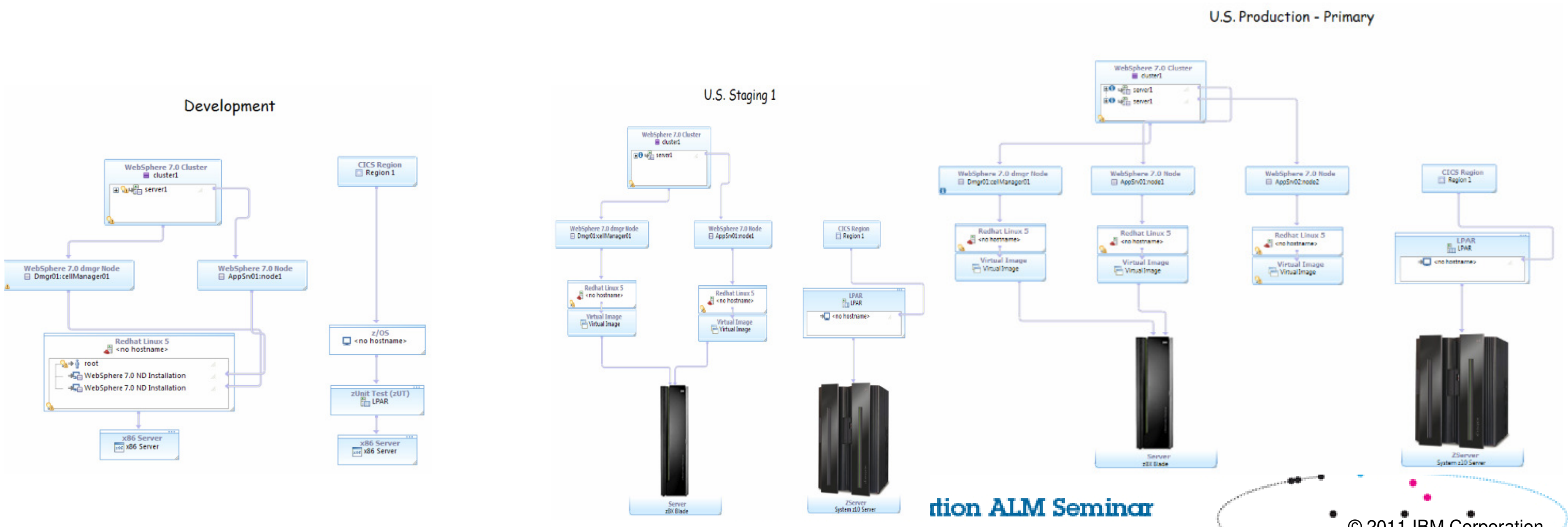




Pipeline Example – Production Stage

- Promote to production
- Re-use same deployment automation from prior environments
- Use same automatic validation leveraged in prior milestones





tion ALM Seminar

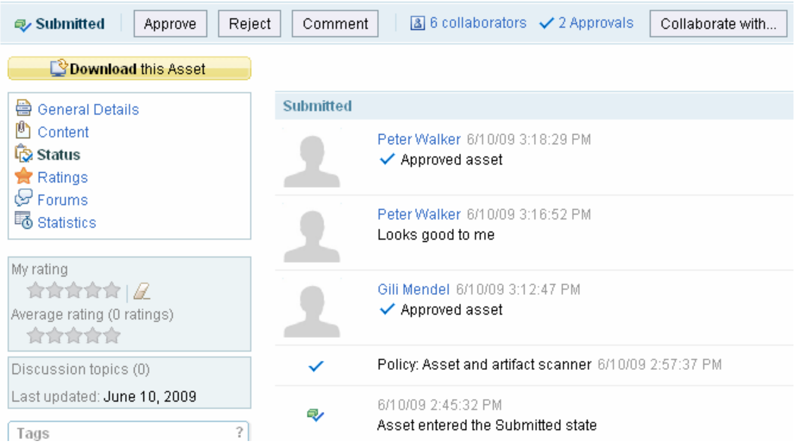
Best Practice: Establish a Definitive Library of Deployment Assets

- ITIL recommends establishing a ”Definitive Media Library” – a place where all the master or ‘gold’ versions of software assets are stored and maintained
  - we take this further to include master configuration data, installation scripts, topology patterns, release notes and other ‘non-build’ items essential in provisioning an environment
- Many organisations take this to mean a basic file system
  - some extend the concept and use a Version Control system
  - this is a good start but it it means many important details about component interdependencies, maturity levels, and existing deployments
- Using a formal Asset Management Repository gives you control over
  - Releases consisting of a baseline through many components, libraries, provisioning data, topology assets and development data (known defects, requirements and changes)
  - Formal asset lifecycles
  - Approvals and review
  - Policies for automated validation, review, retirement, compliance, provisioning

Best Practice: Establish a Definitive 'Software' Library

Gain control over the:

- **Components** to ensure only certified releases are deployed
- **People** who are stakeholders in the decision making
- **Workflow** to manage sharing
- **Policies** to enforce rules
- **Access permissions** to control access
- **Traceability and auditing** for plans and automations



Submitted | Approve | Reject | Comment | 6 collaborators | ✓ 2 Approvals | Collaborate with...

Download this Asset

- General Details
- Content
- Status
- Ratings
- Forums
- Statistics

My rating

Average rating (0 ratings)

Discussion topics (0)

Last updated: June 10, 2009

Tags ?

Submitted

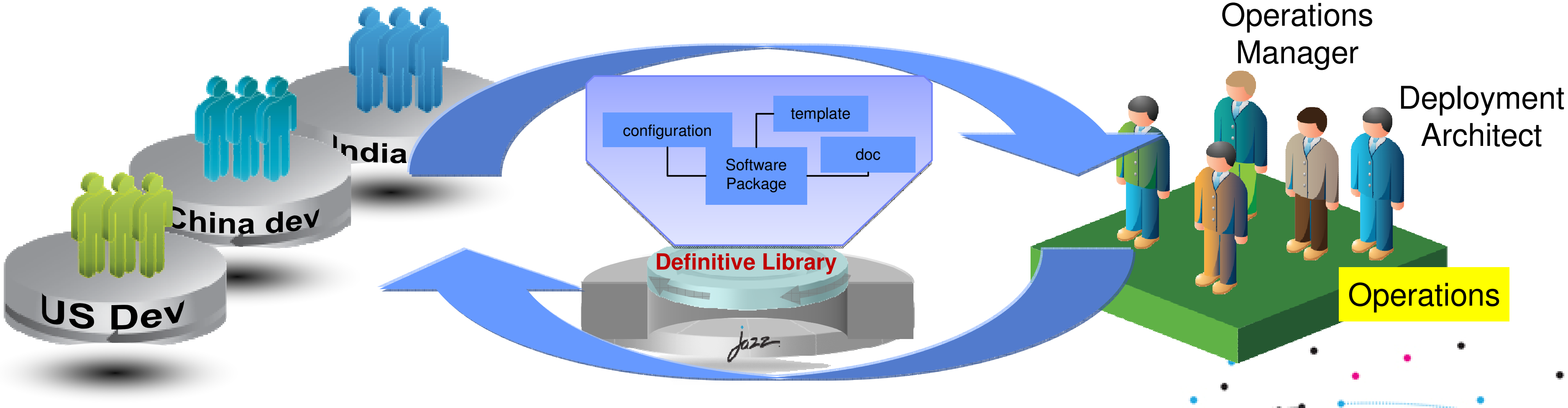
Peter Walker 6/10/09 3:18:29 PM
✓ Approved asset

Peter Walker 6/10/09 3:16:52 PM
Looks good to me

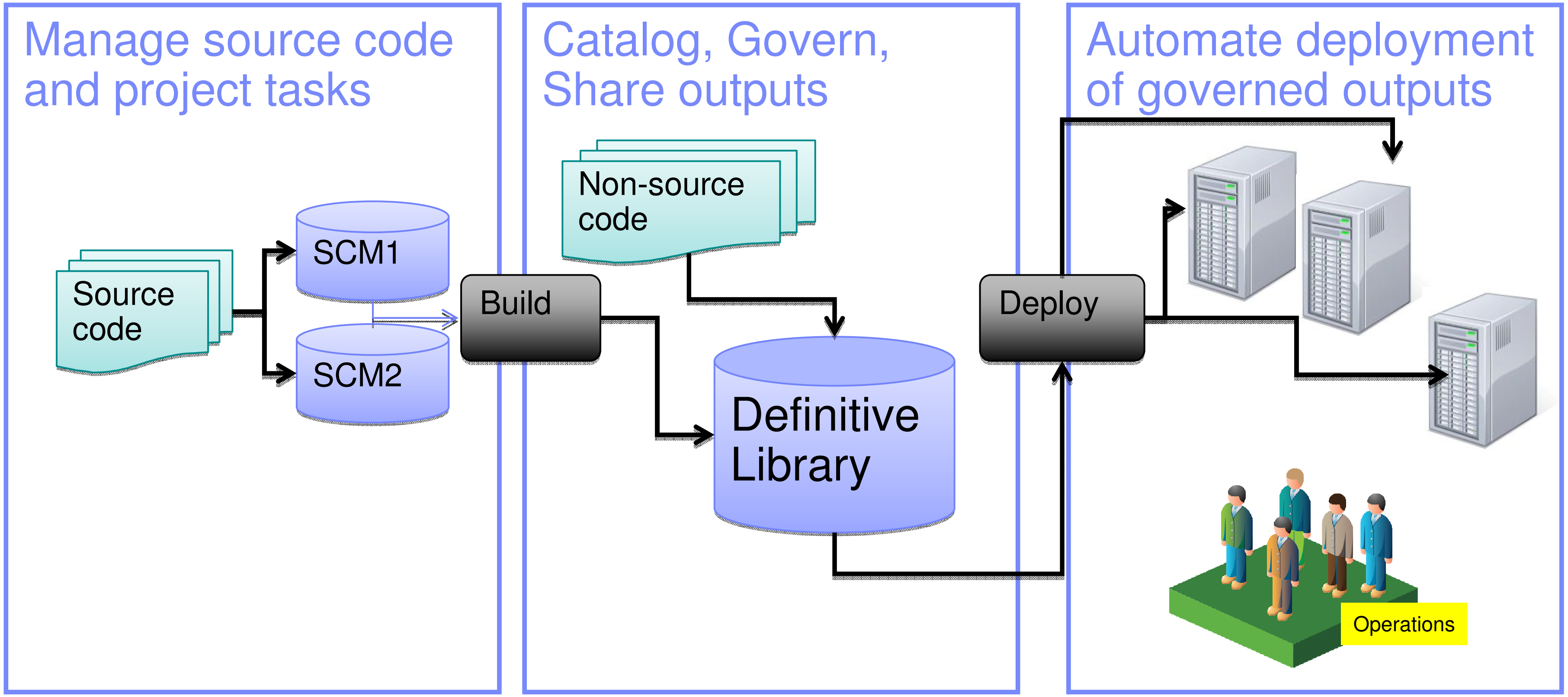
Gili Mendel 6/10/09 3:12:47 PM
✓ Approved asset

✓ Policy: Asset and artifact scanner 6/10/09 2:57:37 PM

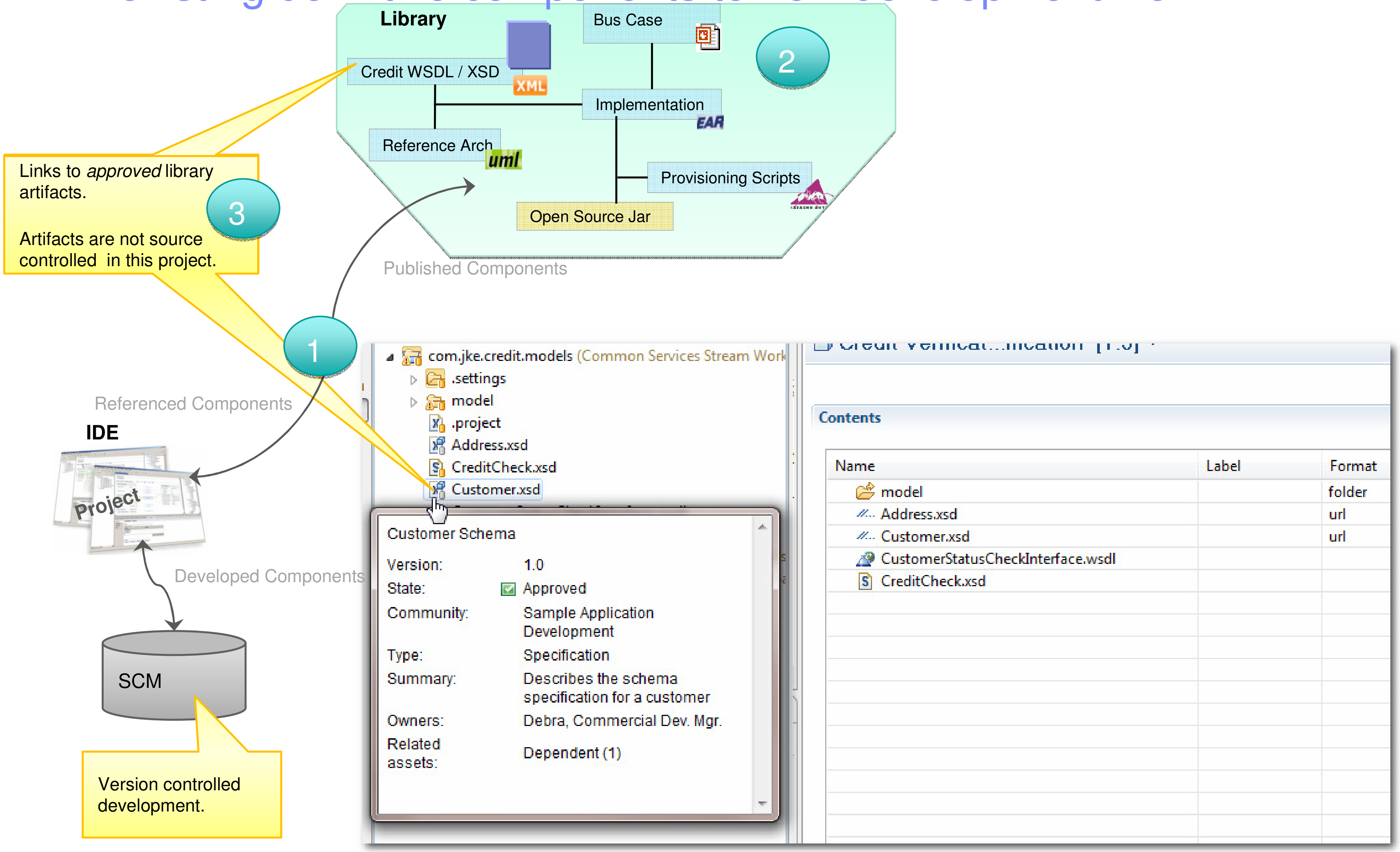
6/10/09 2:45:32 PM
Asset entered the Submitted state



DevOps using a Definitive Library


Manage source code and project tasks
Source code
SCM1
SCM2
Build
Catalog, Govern, Share outputs
Non-source code
Definitive Library
Deploy
Automate deployment of governed outputs
Operations

Link existing definitive components to new development work



Links to *approved* library artifacts.

Artifacts are not source controlled in this project.

Version controlled development.

- com.jke.credit.models (Common Services Stream Work
  - .settings
  - model
  - .project
  - Address.xsd
  - CreditCheck.xsd
  - Customer.xsd

Customer Schema

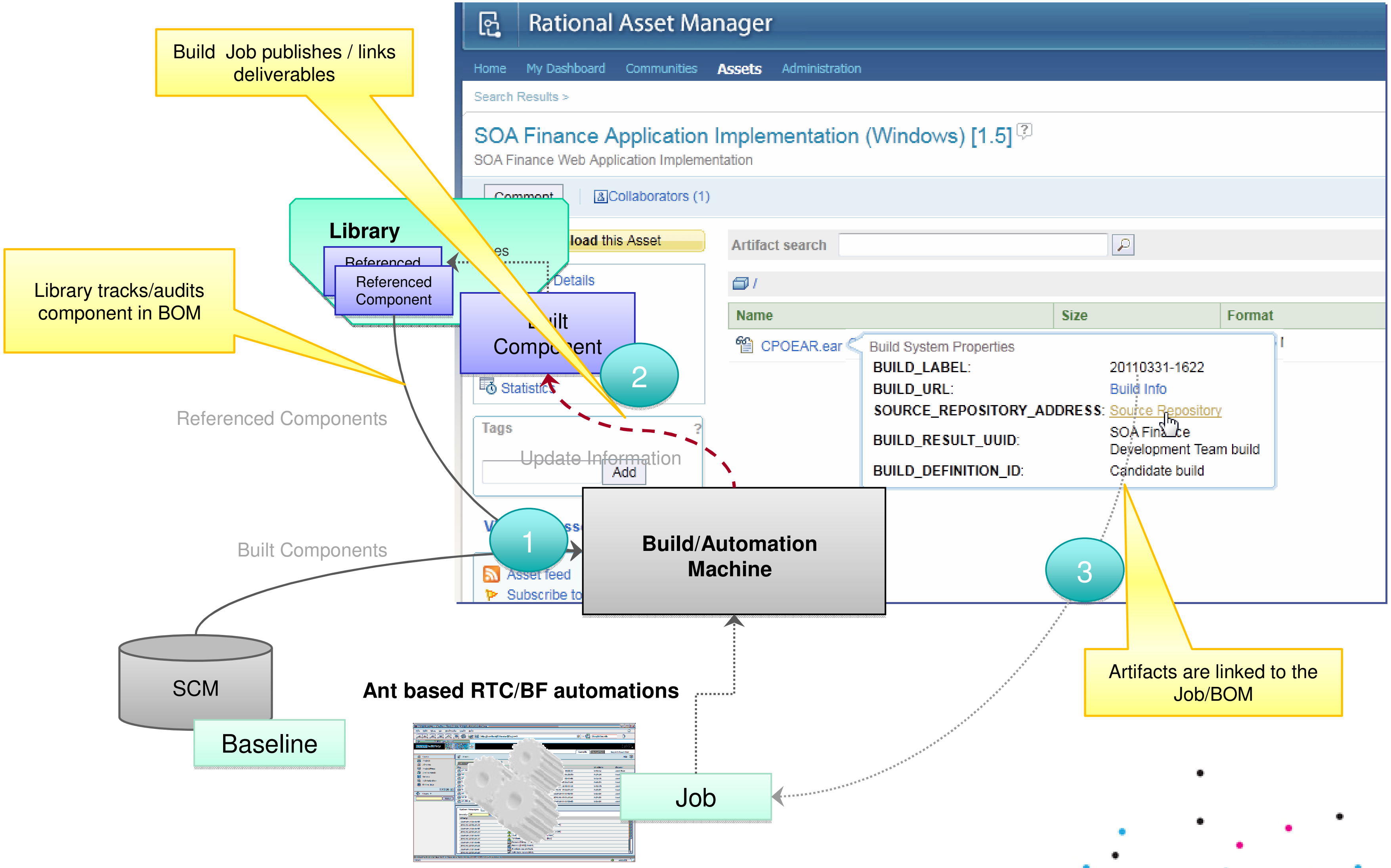
Version: 1.0
State: Approved
Community: Sample Application Development
Type: Specification
Summary: Describes the schema specification for a customer
Owners: Debra, Commercial Dev. Mgr.
Related assets: Dependent (1)

Credit Verificat...ification [1.0] *

Contents

Name	Label	Format
model		folder
Address.xsd		url
Customer.xsd		url
CustomerStatusCheckInterface.wsdl		
CreditCheck.xsd		

Build using linked and vetted outputs from the library



Build Job publishes / links deliverables

Library tracks/audits component in BOM

Artifacts are linked to the Job/BOM

Ant based RTC/BF automations

Rational Asset Manager

SOA Finance Application Implementation (Windows) [1.5]

SOA Finance Web Application Implementation

Build System Properties

BUILD_LABEL: 20110331-1622

BUILD_URL: Build Info

SOURCE_REPOSITORY_ADDRESS: Source Repository

BUILD_RESULT_UUID: SOA Finance Development Team build

BUILD_DEFINITION_ID: Candidate build

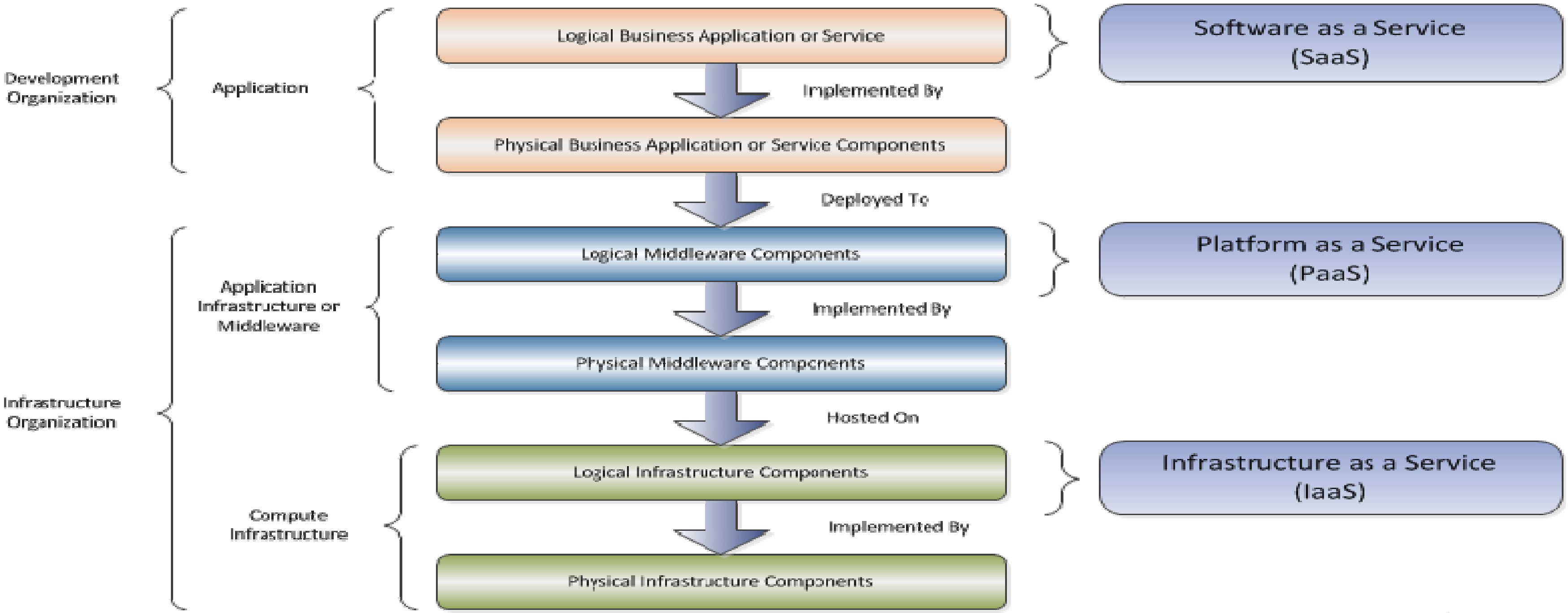
Best Practice: Establish a DevOps Pattern Library

- Ensure that Development Architects & Operations Architects agree on standardized platforms and what architectural patterns will be supported
  - Setup pattern workgroup to develop, collaborate and refine patterns
  - Establish a catalog of standard, support patterns in the DSL for consumption by both Development and Operations
- DevOps is a collaborative effort to align and optimize solution delivery between development and operations
  - Standardization reduces cost with consistent administration, consistent problem determination, consistent maintenance
  - Establish and follow exception process for infrequent cases outside of mainstream application development
- Communicate early in the process using unambiguous topology specifications
  - Traditionally, very little intersection between development and operations terminology
  - Need to find common ground where worlds meet
- Avoid Anti-patterns:
  - Lack of consensus on architectural patterns
  - One size fits all (likely doesn't fit anyone well)
  - Custom Everything (increased cost to management and maintain)

Establishing the Pattern Library - Figure out what you have!

- Discovering existing systems as a basis for new patterns
  - leverage your CMDB and Operational Discovery Agents to capture, quantify and refine the patterns that currently exist
  - Document the patterns in a form that can be used to build new systems, and create the delivery pipeline automatically
- Pattern definition is key to DevOps
  - You will not see the value if every solution is a one-off
- Companies have many patterns (and anti-patterns) that exist

Layers



Development Organization

Application

Logical Business Application or Service

Implemented By

Physical Business Application or Service Components

Deployed To

Software as a Service (SaaS)

Infrastructure Organization

Application Infrastructure or Middleware

Logical Middleware Components

Implemented By

Physical Middleware Components

Hosted On

Platform as a Service (PaaS)

Compute Infrastructure

Logical Infrastructure Components

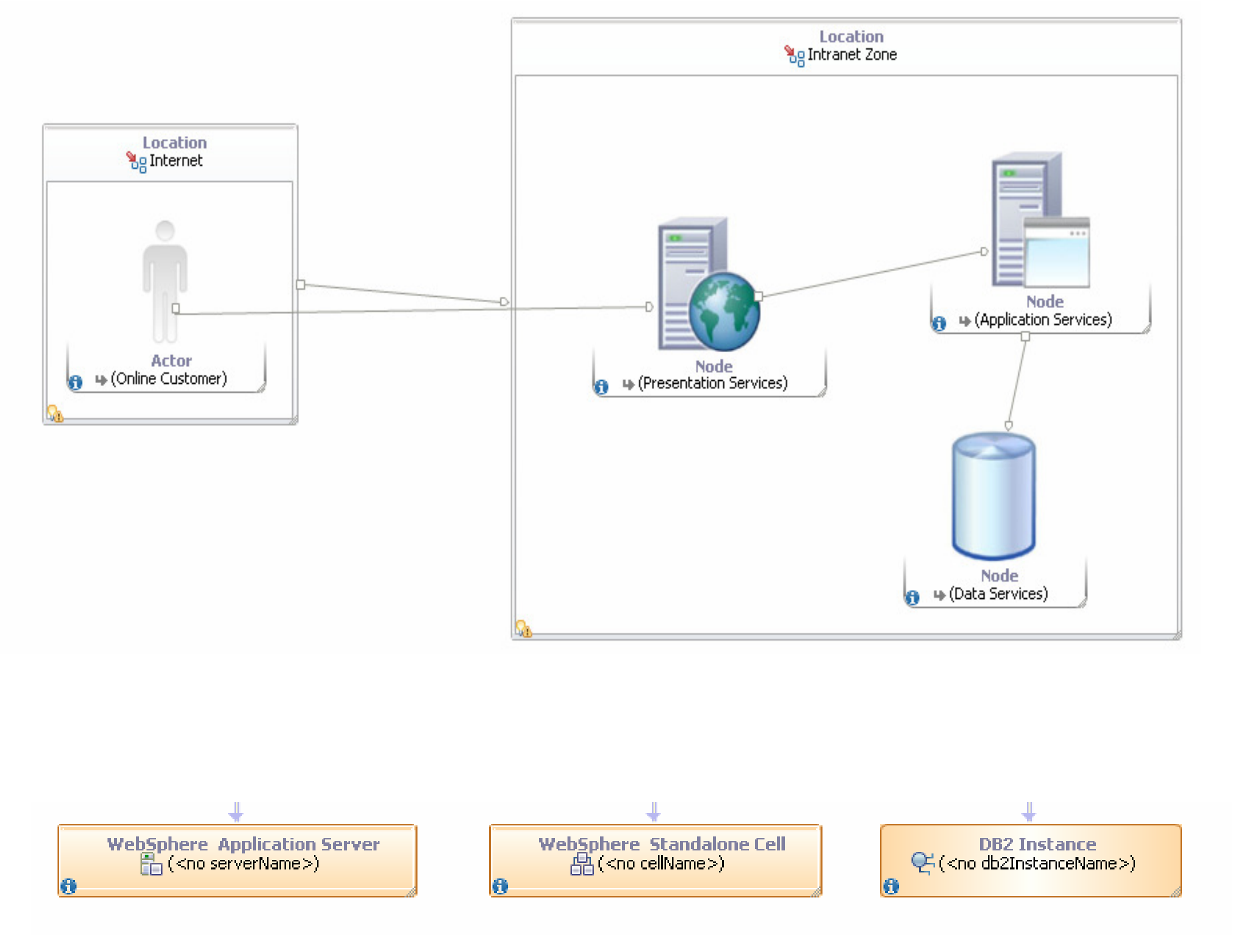
Implemented By

Physical Infrastructure Components

Infrastructure as a Service (IaaS)

Logical Pattern

- Describe a skeleton and assumptions to build an application upon; two flavors: abstract design pattern or conceptual pattern of platform
- Created by: **IT Architects** or an **Architecture Board**
- Consumed by: Starting point for **Application Architect**

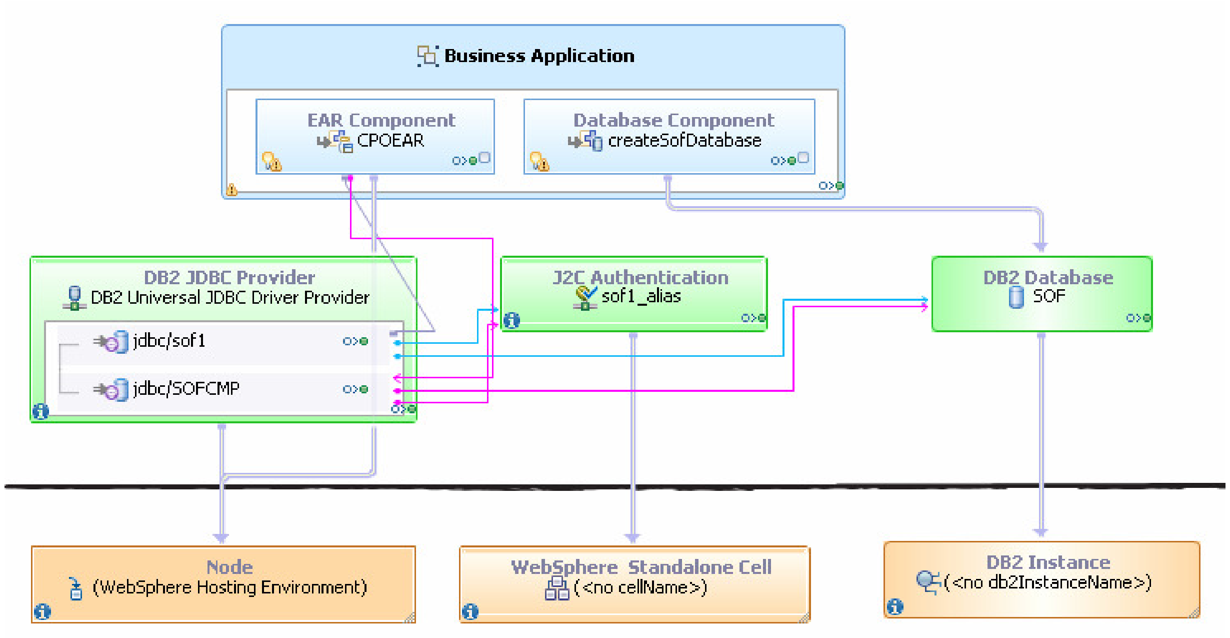


Application Pattern

- Describes application architecture within the bounds of the logical pattern. Captures required enterprise dependencies independent of each stage

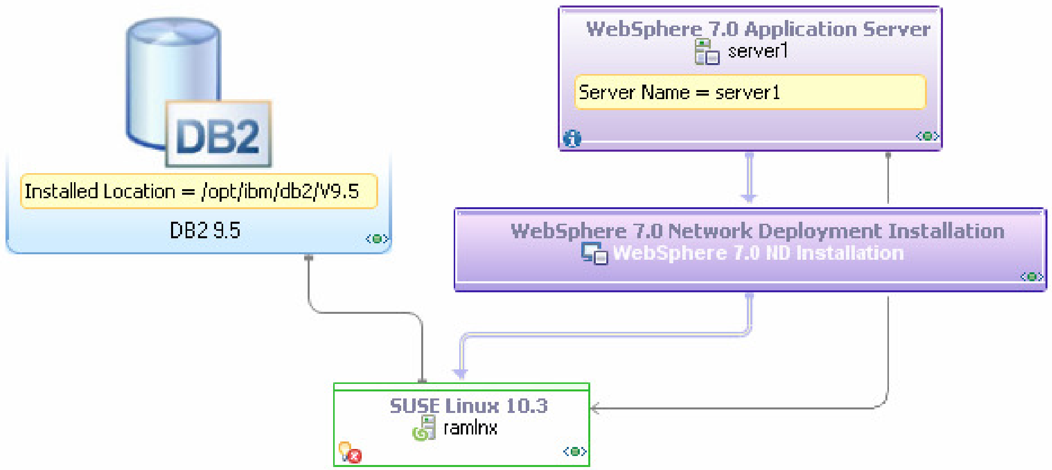
- Created By Application Architects

- Consumed By Deployment Architects or Specialists

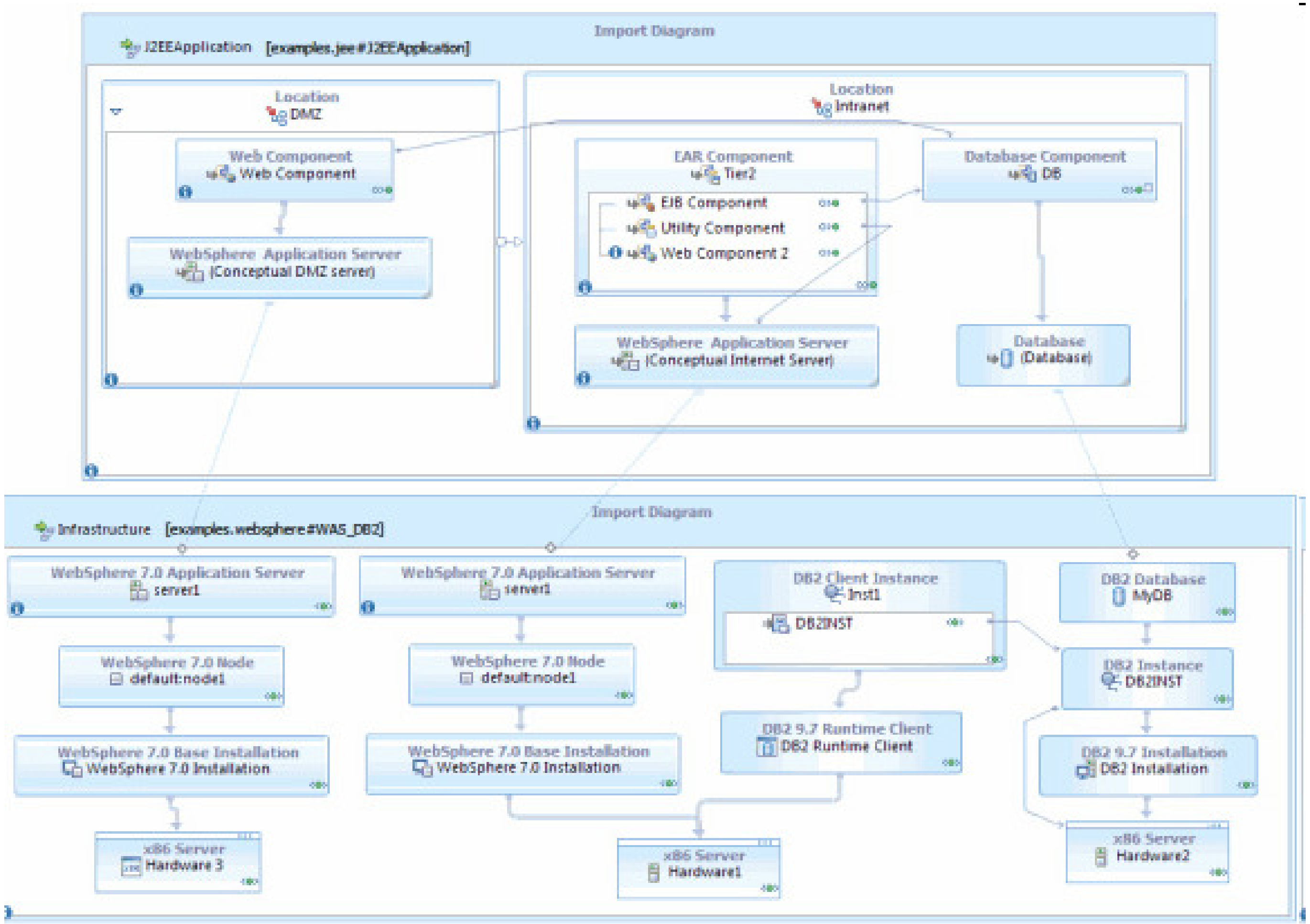


Platform Pattern

- Defines standard, supported configurations of middleware, operating systems, and infrastructure. By standardizing and limiting configurations, provide greater re-use, lower cost of ownership.
- Created by Subject Matter Experts; either individually defining their own areas or coming together with IT Architects to define supported compositions
- Consumed by Deployment Architects or Specialists


DB2
Installed Location = /opt/ibm/db2/V9.5
DB2 9.5
WebSphere 7.0 Application Server
server1
Server Name = server1
WebSphere 7.0 Network Deployment Installation
WebSphere 7.0 ND Installation
SUSE Linux 10.3
ramlnx

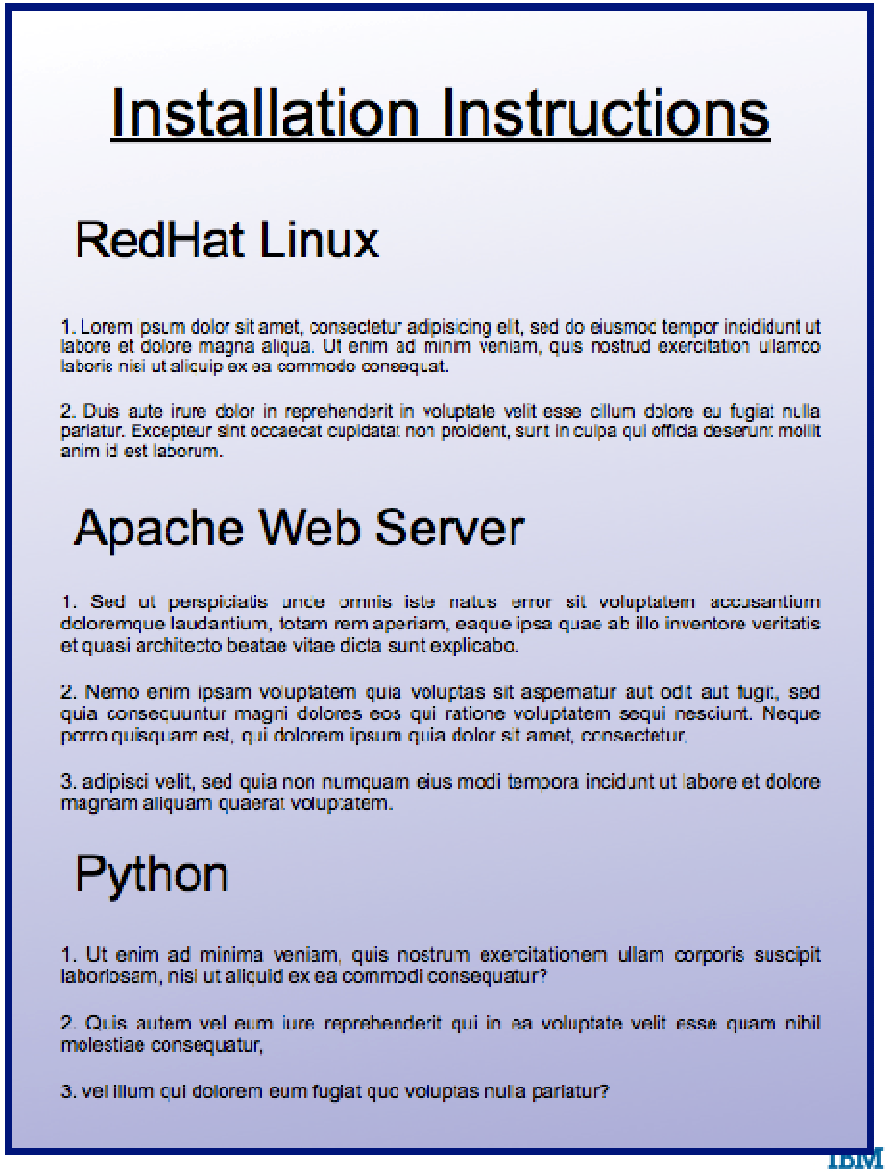
Assembling the Patterns to Build a Solution



Best Practice: Treat Infrastructure Artifacts as Code

- Source Control Management can’t be limited to business applications
  - Automation routines and scripts are fundamental to Operations
- Managing Operations routines like source code offers several benefits:
  - Central point of truth as routines and environments change
  - Backup in case of loss
  - Identify possible regressions by comparing with prior versions
- Example Managed Assets:
  - Perl, Jython, WSADMIN, ANT scripts
  - Service orchestration routines (opsware, buildforge, etc)
  - Infrastructure Gold copies components

Installation Today



Installation Instructions

RedHat Linux

1. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut alicuip ex ea commodo consequat.

2. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Apache Web Server

1. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo.

2. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur,

3. adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem.

Python

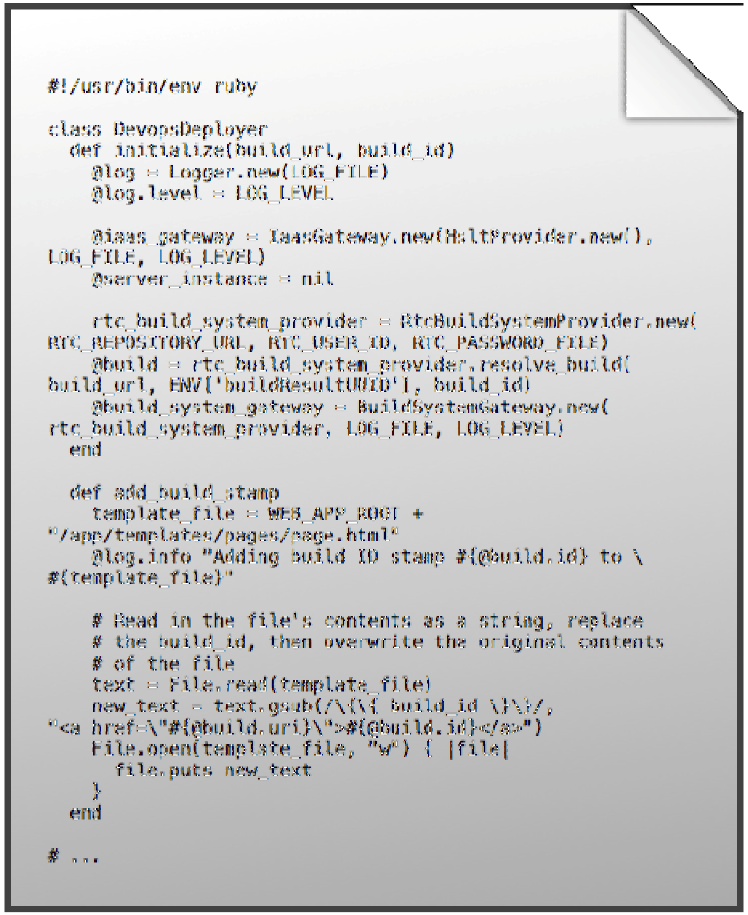
1. Ut enim ad minima veniam, quis nostrum exercitationem ullam corporis suscipit laboriosam, nisi ut aliquid ex ea commodi consequatur?

2. Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur,

3. vel illum qui dolorem eum fugiat quo voluptas nulla pariatur?

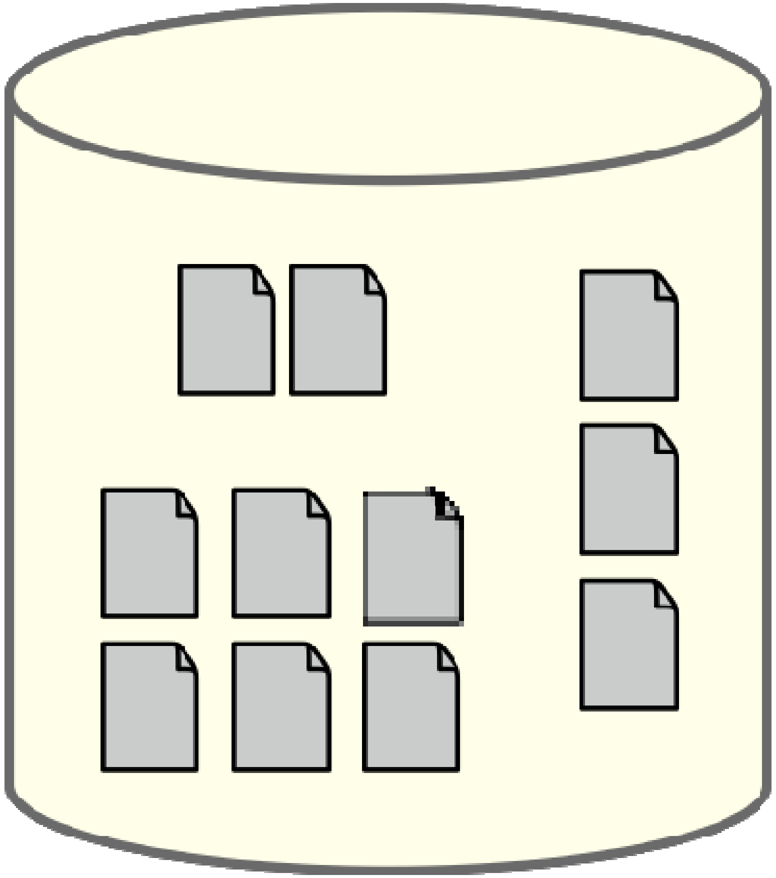
Infrastructure as Code

- As more routines are developed within the infrastructure areas to perform automated provisioning, resource management and administrative activities, they become more important as an enterprise asset.
- The loss of these routines can be catastrophic to the operations of the business and need to be managed like other code assets
- Well defined process for check-in/checkout, testing and migration of assets is required

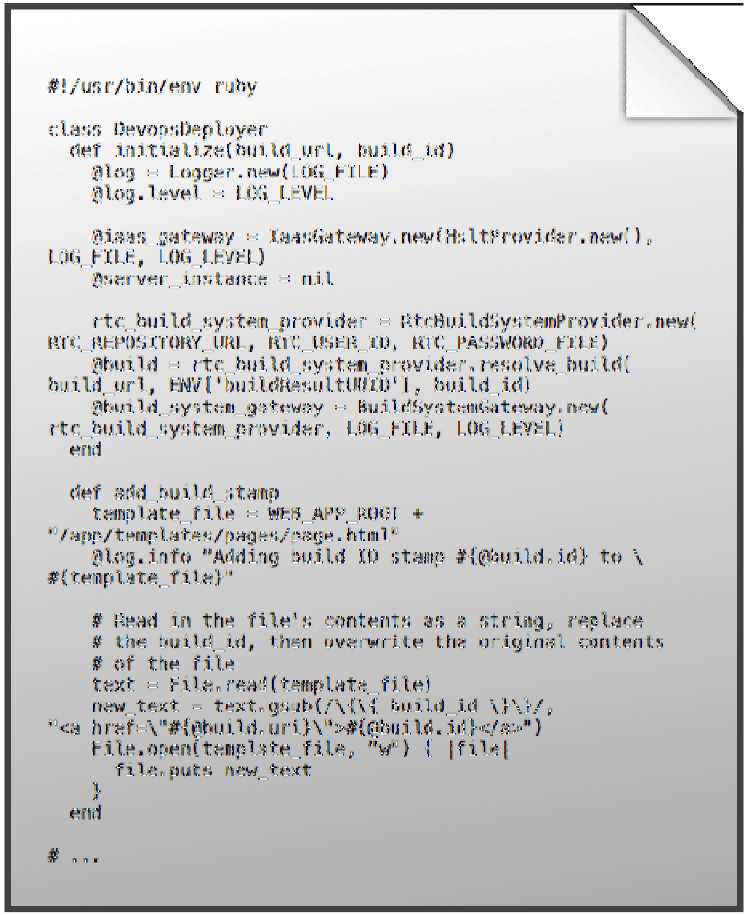


```
#!/usr/bin/env ruby

class DevopsDeployer
  def initialize(build_url, build_id)
    @log = Logger.new(LOG_FILE)
    @log.level = LOG_LEVEL

    @iaas_gateway = IaasGateway.new(HsltProvider.new(),
LOG_FILE, LOG_LEVEL)
    @server_instance = nil

    rtc_build_system_provider = RtcBuildSystemProvider.new(
RTC_REPOSITORY_URL, RTC_USER_ID, RTC_PASSWORD_FILE)
    @build = rtc_build_system_provider.resolve_build(
build_url, ENV['buildResultUUID'], build_id)
    @build_system_gateway = BuildSystemGateway.new(
rtc_build_system_provider, LOG_FILE, LOG_LEVEL)
  end

  def add_build_stamp
    template_file = WEB_APP_ROOT +
"/app/templates/pages/page.html"
    @log.info "Adding build ID stamp #{@build.id} to \
#{template_file}"

    # Read in the file's contents as a string, replace
    # the build_id, then overwrite the original contents
    # of the file
    text = File.read(template_file)
    new_text = text.gsub(/\{\{ build_id \}\}/,
"<a href=\"#{@build.uri}\">#{@build.id}</a>")
    File.open(template_file, "w") { |file|
      file.puts new_text
    }
  end

# ...
```

Infrastructure As Code



SCM



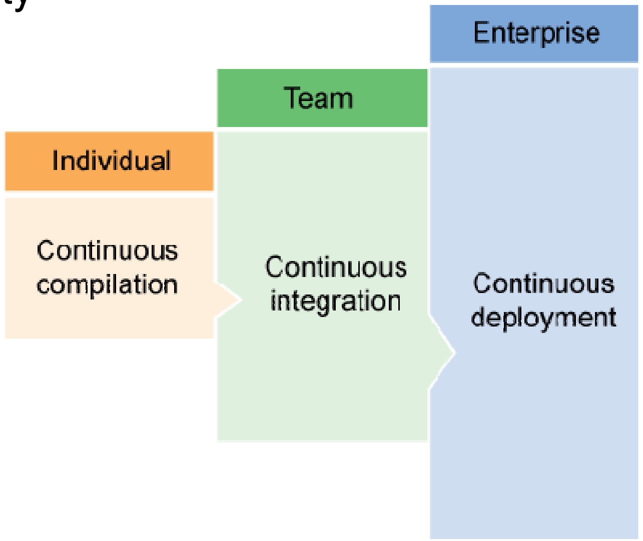
```ruby
#!/usr/bin/env ruby

class DevopsDeployer
  def initialize(build_url, build_id)
    @log = Logger.new(LOG_FILE)
    @log.level = LOG_LEVEL

    @iaas_gateway = IaasGateway.new(HsltProvider.new(),
LOG_FILE, LOG_LEVEL)
    @server_instance = nil

    rtc_build_system_provider = RtcBuildSystemProvider.new(
RTC_REPOSITORY_URL, RTC_USER_ID, RTC_PASSWORD_FILE)
    @build = rtc_build_system_provider.resolve_build(
build_url, ENV['buildResultUUID'], build_id)
    @build_system_gateway = BuildSystemGateway.new(
rtc_build_system_provider, LOG_FILE, LOG_LEVEL)
  end

  def add_build_stamp
    template_file = WEB_APP_ROOT +
"/app/templates/pages/page.html"
    @log.info "Adding build ID stamp #{@build.id} to \
#{template_file}"

    # Read in the file's contents as a string, replace
    # the build_id, then overwrite the original contents
    # of the file
    text = File.read(template_file)
    new_text = text.gsub(/\{\{ build_id \}\}/,
"<a href=\"#{@build.uri}\">#{@build.id}</a>")
    File.open(template_file, "w") { |file|
      file.puts new_text
    }
  end

# ...
```

Best Practice: Automate the Delivery Pipeline

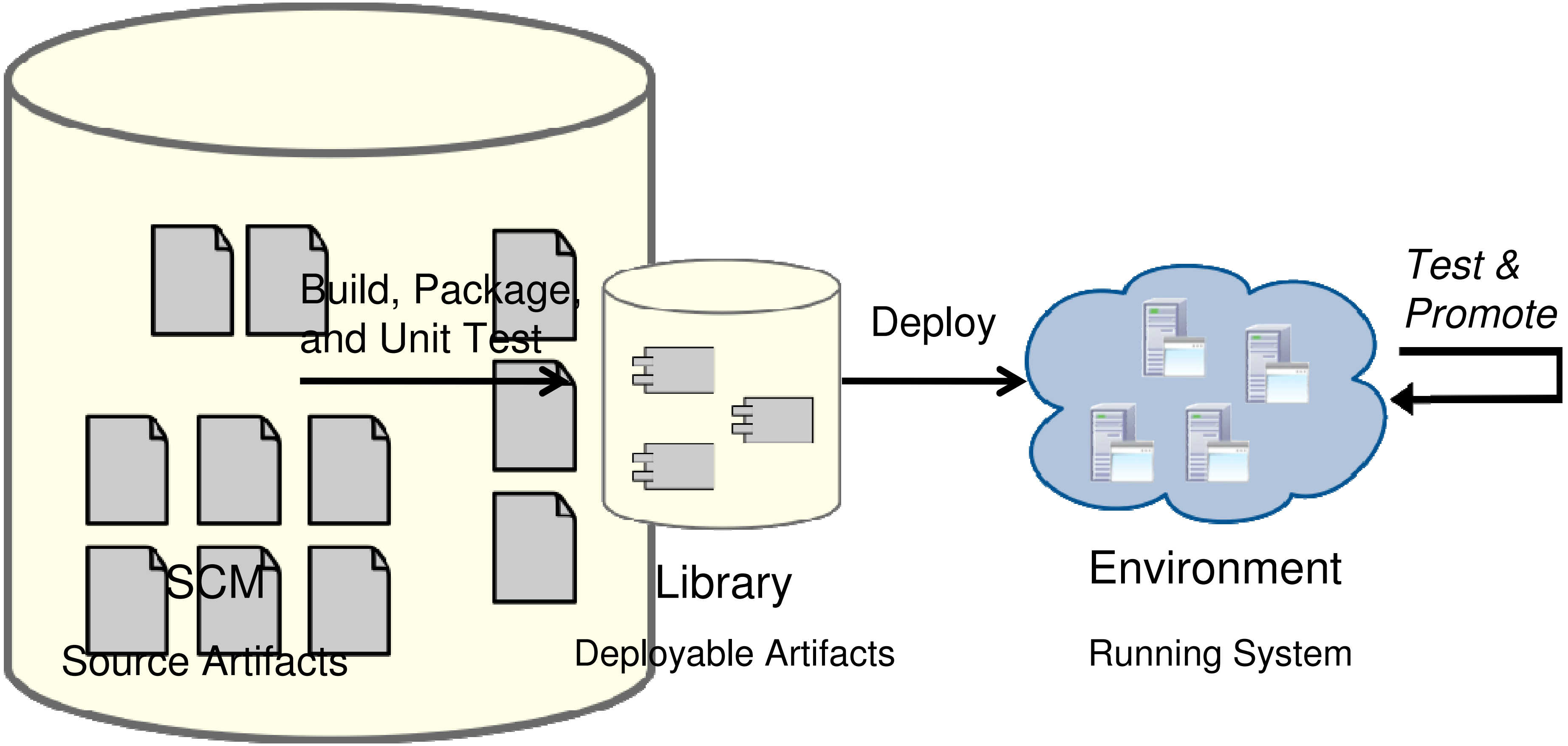
- Don’t fix problems through administrative consoles once per problem, per environment – Fix once in the automation logic
  - Automation becomes part of the system under test
  - Results in automation logic undergoing testing O(100)s times prior to deployment into production (as frequently as once per check-in)
- Developer access to representative Environments for their target application architectures.
  - Without representative Environments, Developers can't validate their code early and Operators have no validation of whether the Application will run in its planned production environment
  - Provide standard Environments for re-use with push button automation to stand-up
  - Use virtual or Cloud-based representative Environments that provide progressively more realistic configurations towards production
- Deployment and validation tests are automated against representative environments.
  - use the same automated pipline to roll application out into representative environment
  - Perform automatic verification at time of build and deployment
    - automated tests against the deployed environment;
    - pro-active validation to look for potential problems with future versions (e.g. using migration toolkits)

Continuous Delivery - A Logical Progression of Existing Approaches

- In the early 2000s, modern IDEs introduced the idea of continuous compilation
  - the code is compiled when a file is saved
- This was soon followed by continuous integration
  - the code is built, and a set of unit tests are run when it is checked into source control
- Just as continuous compilation improves individual productivity, and continuous integration improves a development team's productivity, continuous deployment improves an organization's productivity

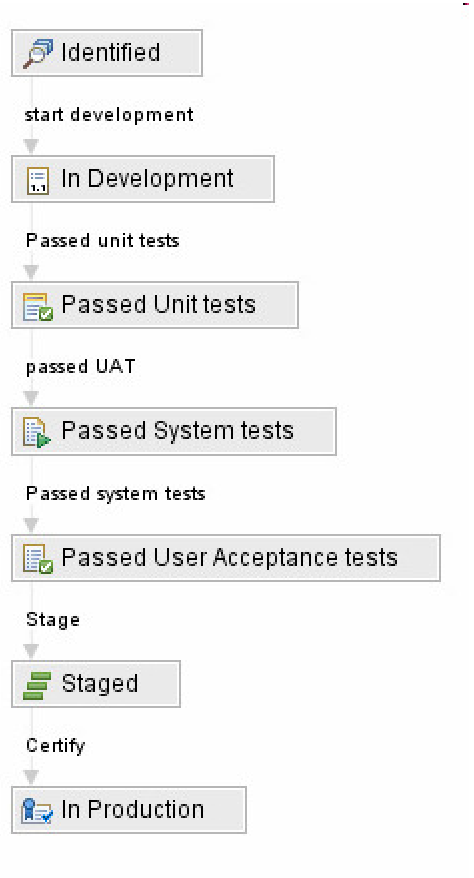


Continuous Delivery



Link Automation Gates to Component Lifecycles in DSL

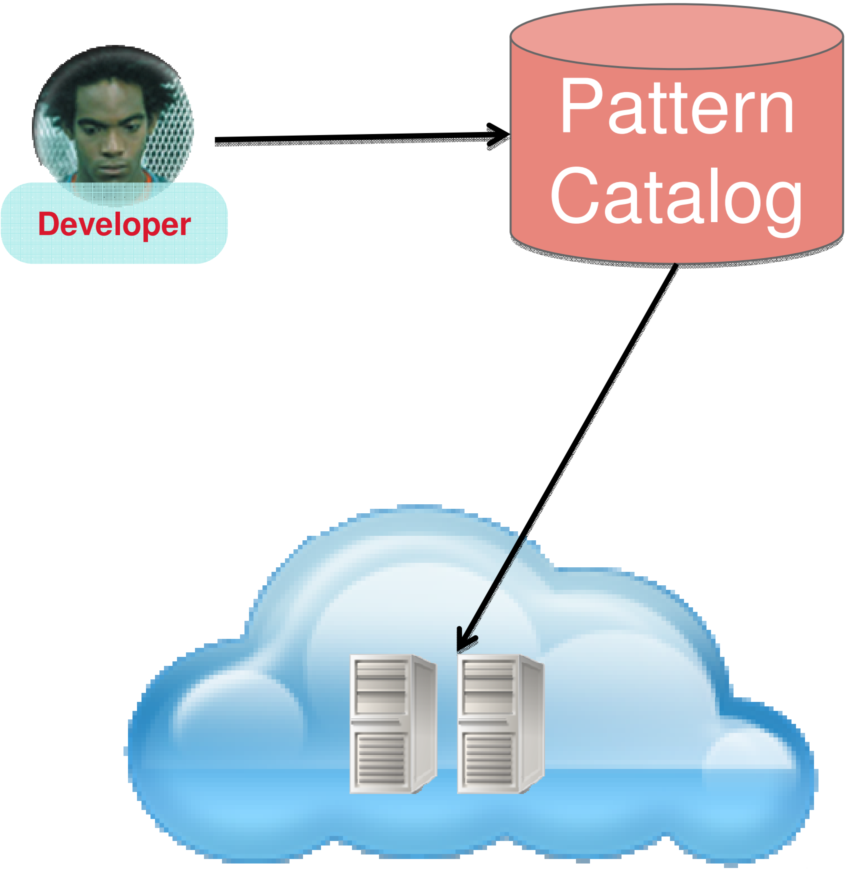
- Defining asset lifecycles in your Definitive Library provides a perfect way to automate the deployment
  - Tie automated deployment to the Release lifecycle state
  - Store 'gold' versions of deployment scripts as an asset, baselined with the release
- In early stages of the release maturity lifecycle, trigger continuous deployment system to provision and deploy development environments automatically on every change
- In later stages, trigger notification to testers that the system is ready to be deployed to test environments, and let test team decide when to kick off provisioning
- In final stages, lock the production deployment processes out until all reviews and approvals are complete and signed off



Automated Provisioning of Environments

- Developers request environment for a lease period
  - Work with a representative environment

- In concert with broader lifecycle – ensuring its good and making it available to the next stage in the lifecycle



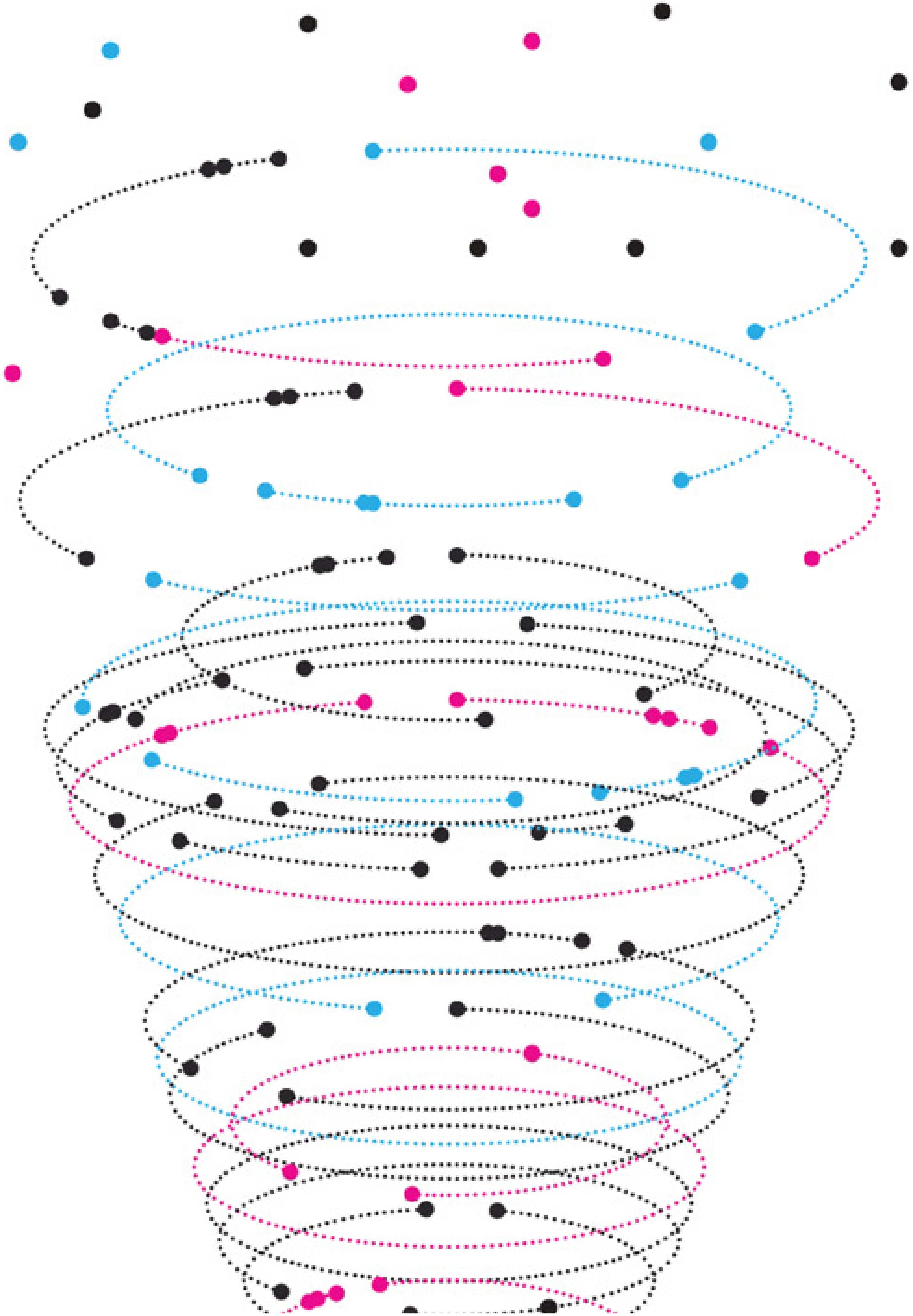
Compliance Validation

- Various checks throughout the lifecycle and pipelines to ensure compliance with enterprise concerns
  - Best practices
  - Code coverage
  - Licensing

- Automated validation hooks throughout to capture and report
  - Unit test against the build code
  - Integration tests against integration environment
  - Functional tests against the running system
  - Acceptance and Compliance tests

Best Practice Summary

- Define a Solution Lifecycle
- Define a Delivery Pipeline
- Establish a Definitive Library of Deployment Assets
- Establish a DevOps Pattern Library
- Treat Infrastructure Artifacts as Code
- Automate the Delivery Pipeline



IBM Next Generation ALM Seminar

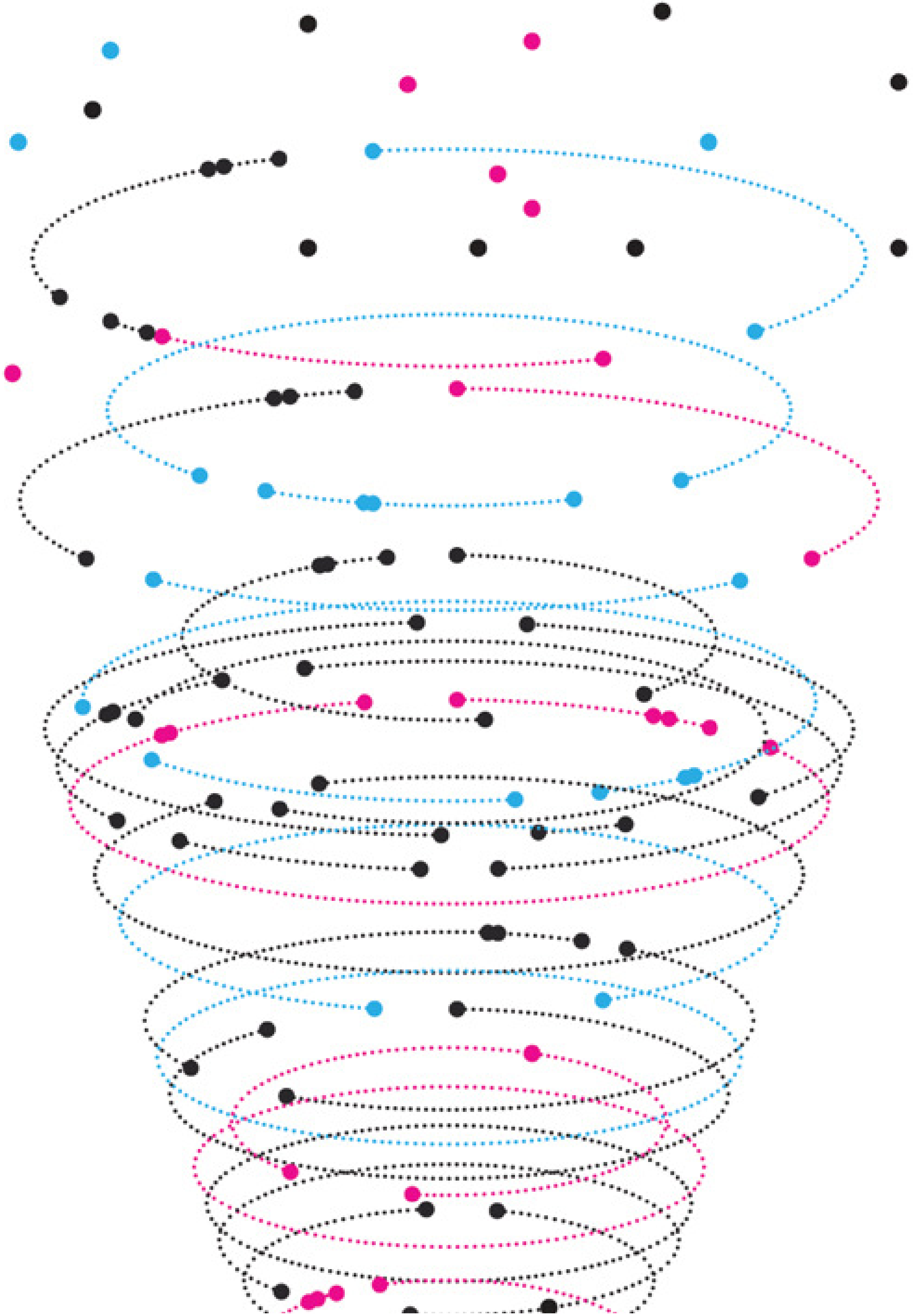
Real Life Examples

Example – An Insurance Organisation

- Situation
- Solution
- Benefits

Example – A Federal Government Department

- Situation
- Solution
- Benefits



IBM Next Generation ALM Seminar

ANZ Bank Case Study

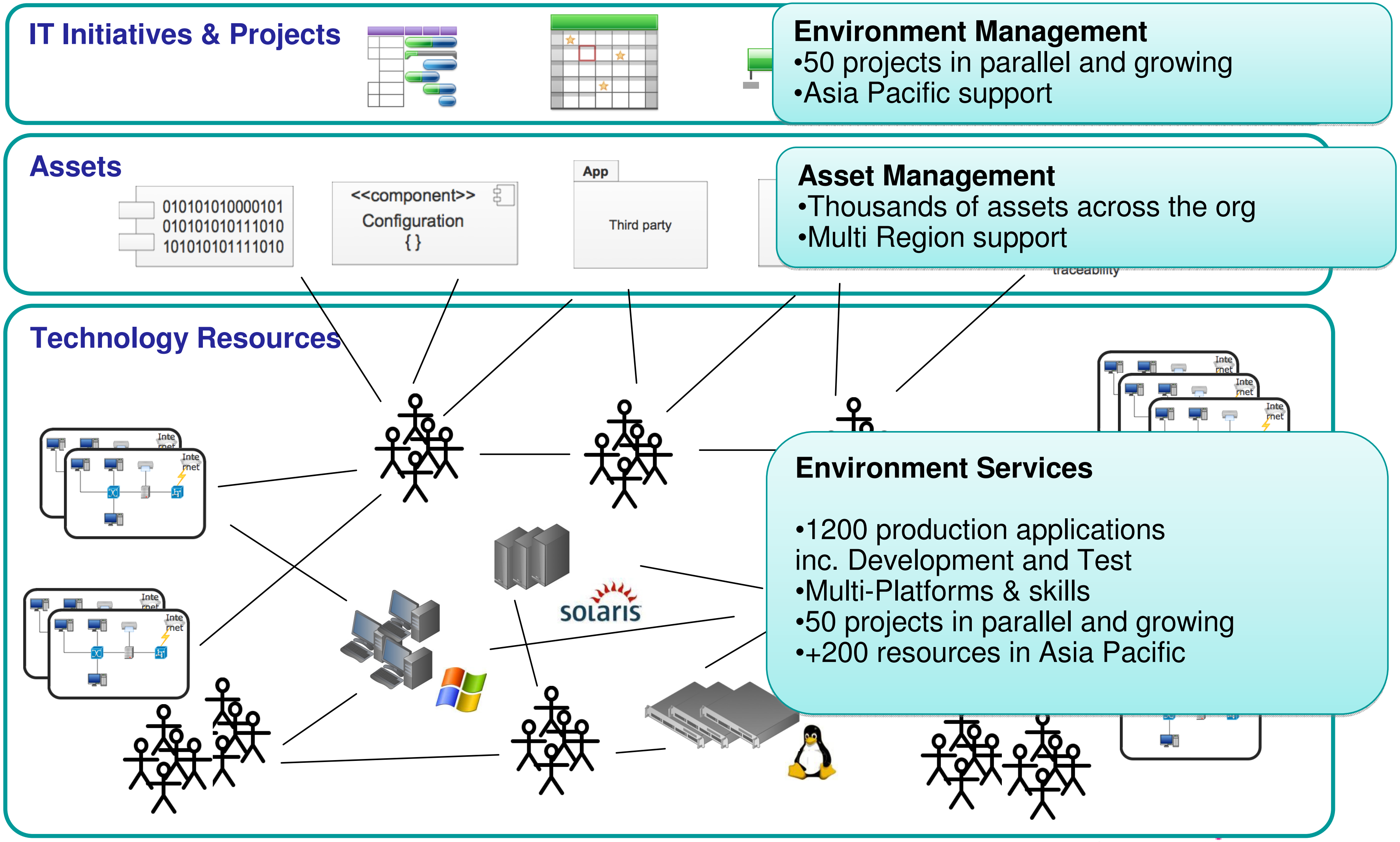
Agenda

- Situation at ANZ Bank
- Challenges at ANZ
- Solution
- Benefits

About ANZ Bank

- Opened its first office in Sydney in 1835
- Global headquarters is located in Melbourne, Australia
- Top ten listed company in Australia, and the number one bank in New Zealand
- Operates in more than 32 countries across Australasia, Europe, Pacific and America (Regional Bank)
- Over 5.7 million customers worldwide
- Employs more than 48,000 people around the world
- Provides a range of banking and financial products and services to around eight million customers

About ANZ Technology



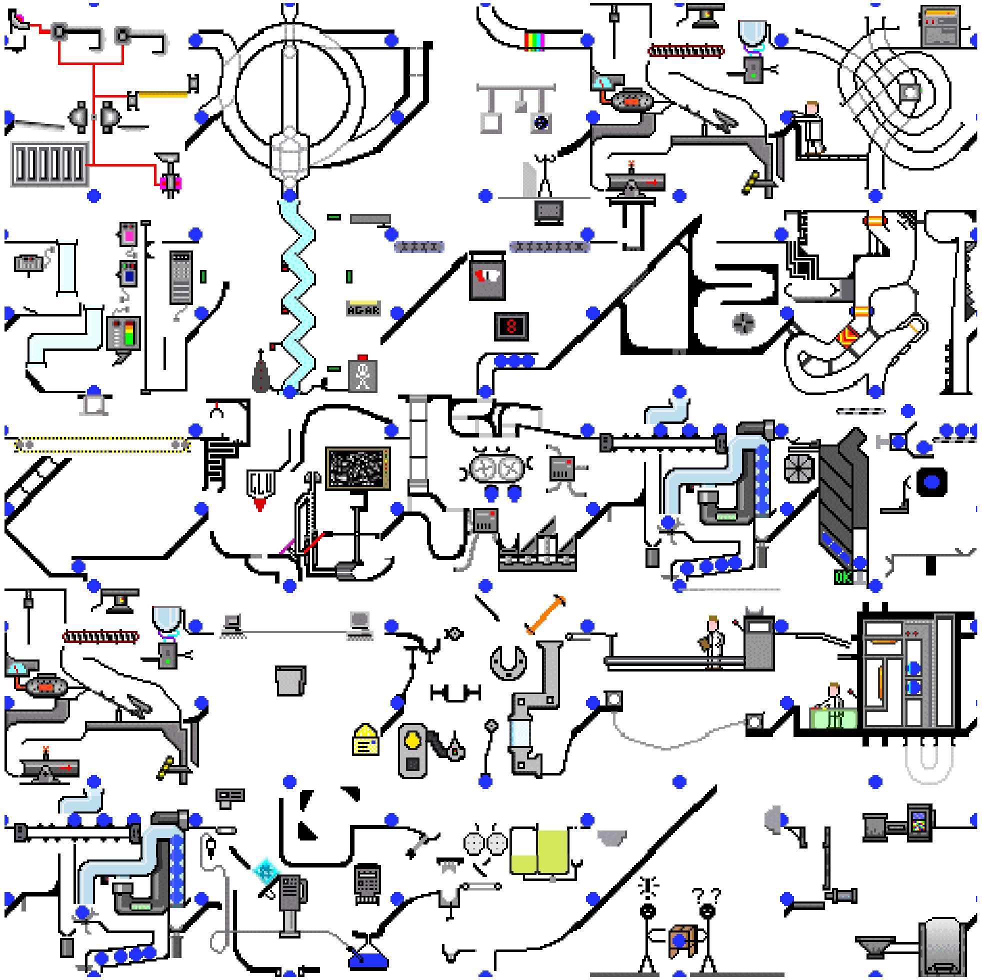
Environment Management
- 50 projects in parallel and growing
- Asia Pacific support

Asset Management
- Thousands of assets across the org
- Multi Region support

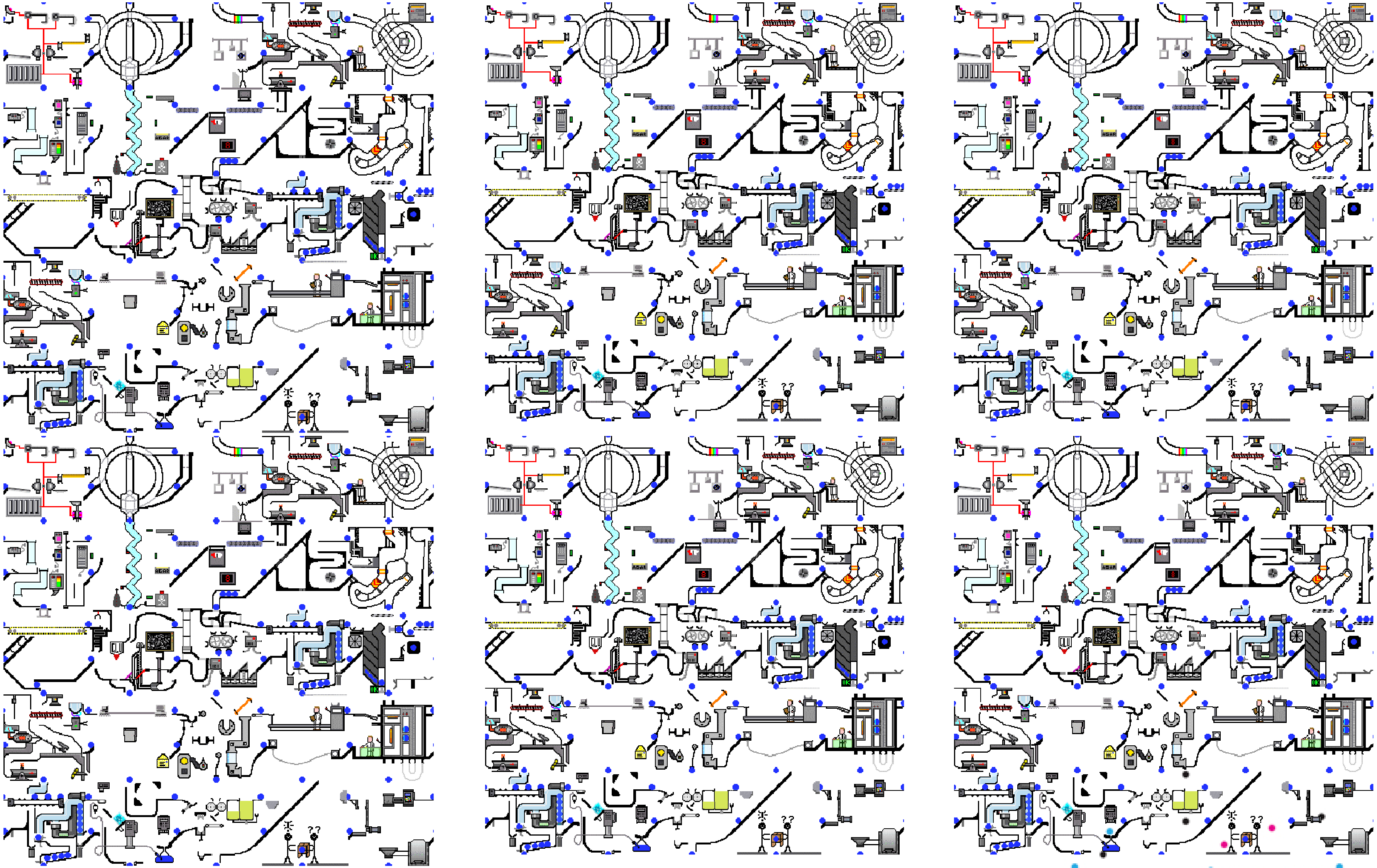
Environment Services

- 1200 production applications inc. Development and Test
- Multi-Platforms & skills
- 50 projects in parallel and growing
- +200 resources in Asia Pacific

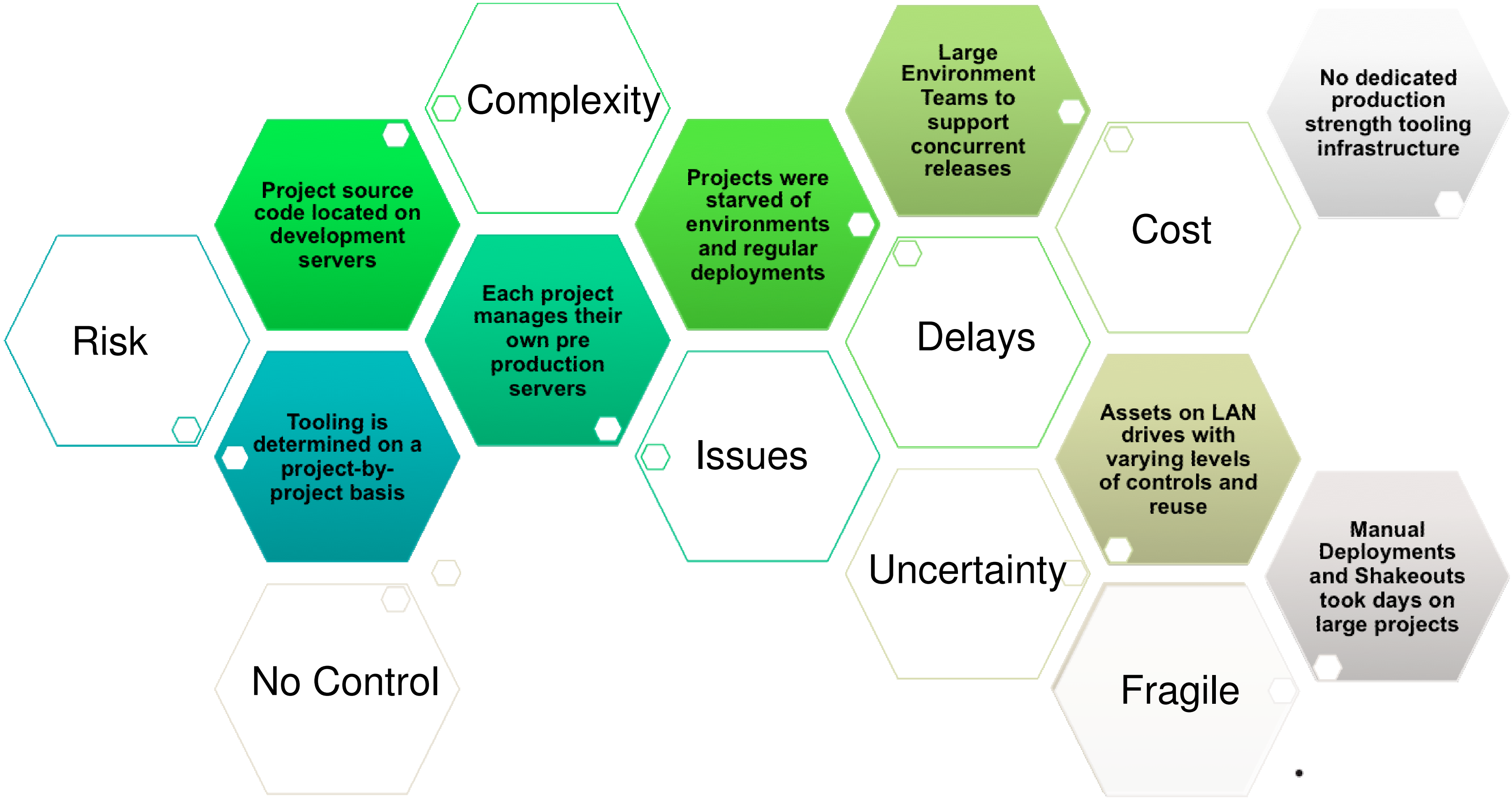
Model x 1



Model x n



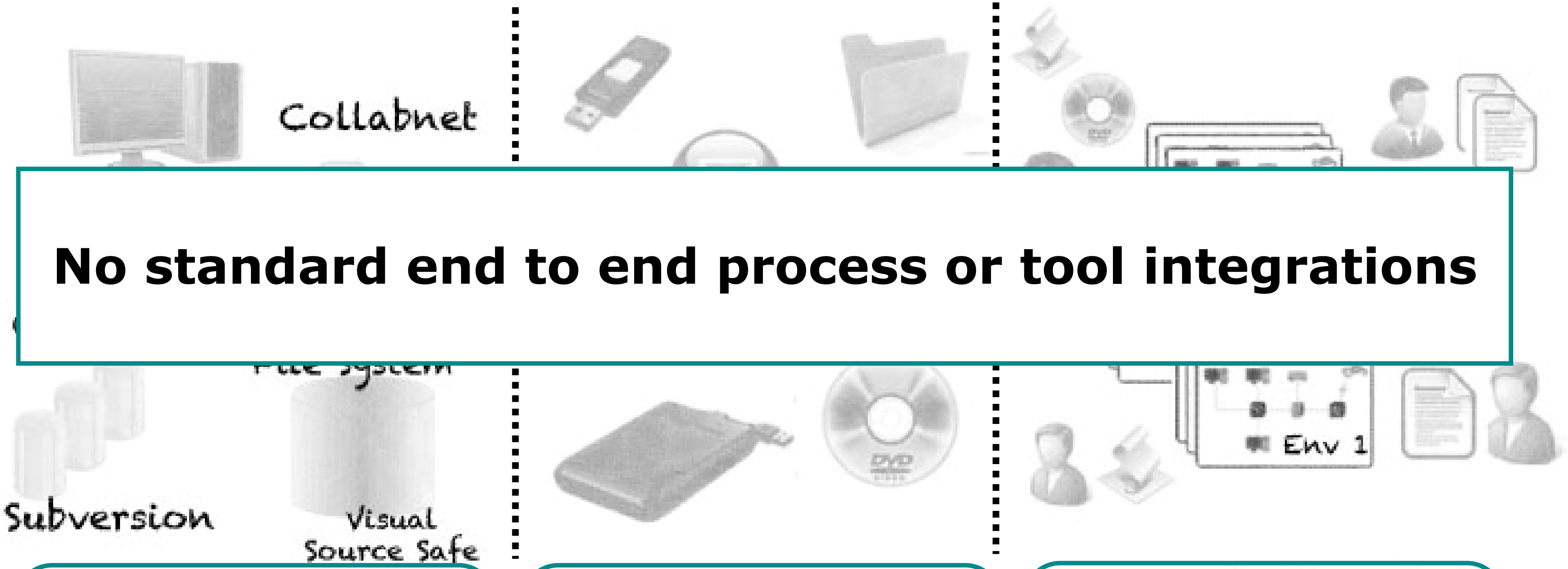
The situation at ANZ Bank



Complexity
Project source code located on development servers
Projects were starved of environments and regular deployments
Large Environment Teams to support concurrent releases
No dedicated production strength tooling infrastructure
Cost
Risk
Each project manages their own pre production servers
Delays
Tooling is determined on a project-by-project basis
Issues
Assets on LAN drives with varying levels of controls and reuse
Uncertainty
Manual Deployments and Shakeouts took days on large projects
No Control
Fragile

The situation at ANZ Bank

Source Control | Artefact Management | Deployment



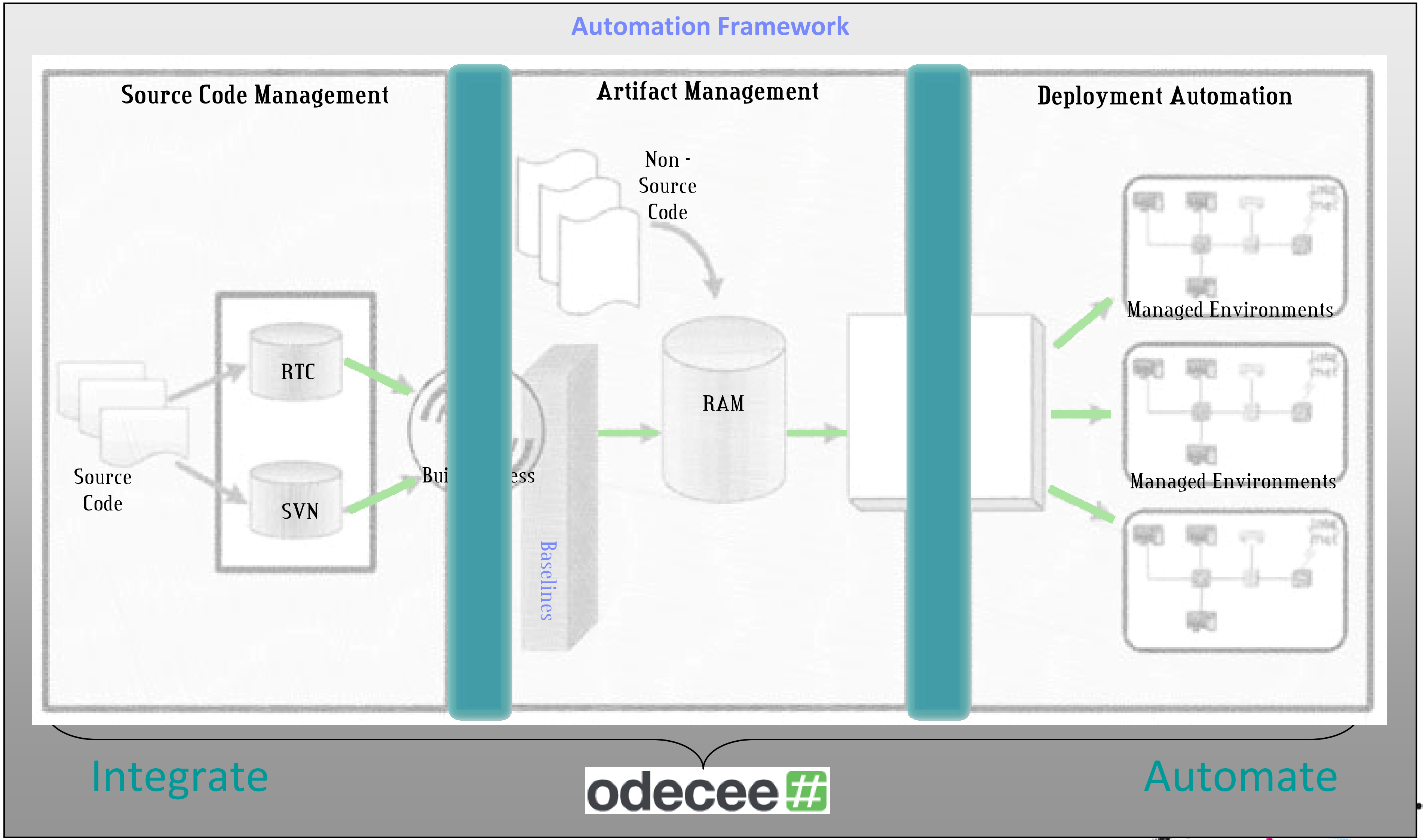
No standard end to end process or tool integrations

Many instances
Many tools
Not managed

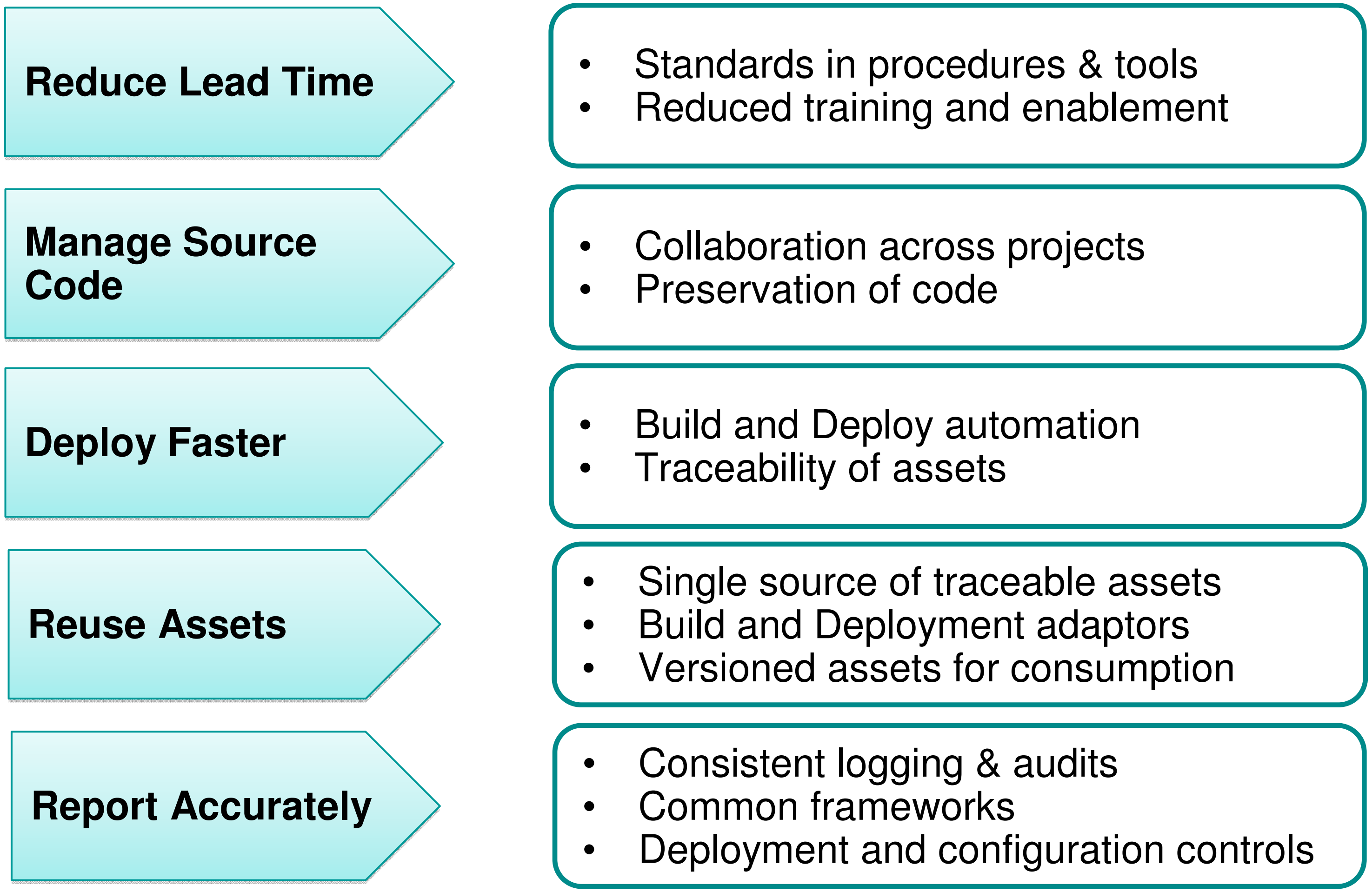
Many methods
Many repositories
Not managed

No standard method
No standard tool
Not managed

Solution - Software Delivery lifecycle Integration



What this means for ANZ Bank



Reduce Lead Time

- Standards in procedures & tools
- Reduced training and enablement

Manage Source Code

- Collaboration across projects
- Preservation of code

Deploy Faster

- Build and Deploy automation
- Traceability of assets

Reuse Assets

- Single source of traceable assets
- Build and Deployment adaptors
- Versioned assets for consumption

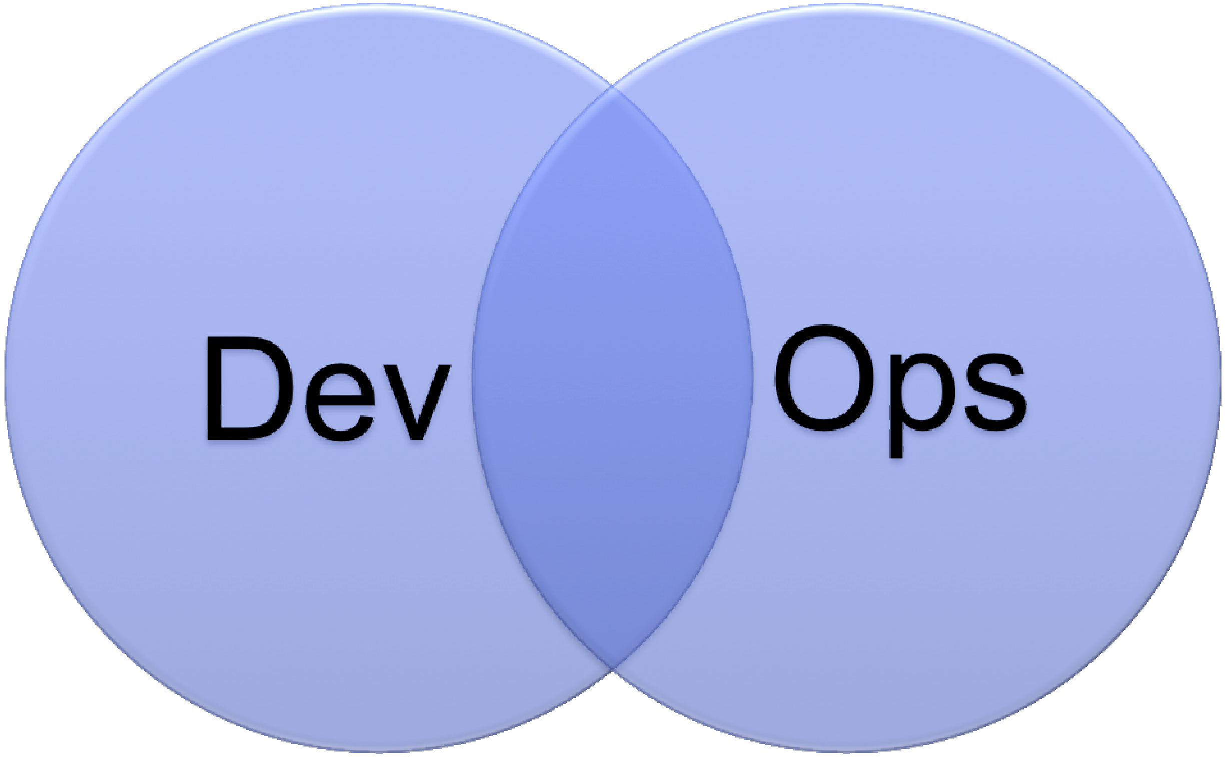
Report Accurately

- Consistent logging & audits
- Common frameworks
- Deployment and configuration controls

The ROI benefit to ANZ

- Improved scalability of solution delivery
- Ability to assume an increased workload of 30-35 percent without adding resources
- Greater efficiency in responding to regulators’ audits of process
- Staff can now move easily between projects as environments have the same set-ups and tools
- Test environment deployment reduced from three days to 15 minutes
- Deployment cycle improvements by at least 50%
- Test cycles reduced by at least 30%
- AU$12m expected to be saved in the first year – representing an ROI of AU$4.27m – with increased savings in future years

Improvements in ANZ



Customer quotes & Questions

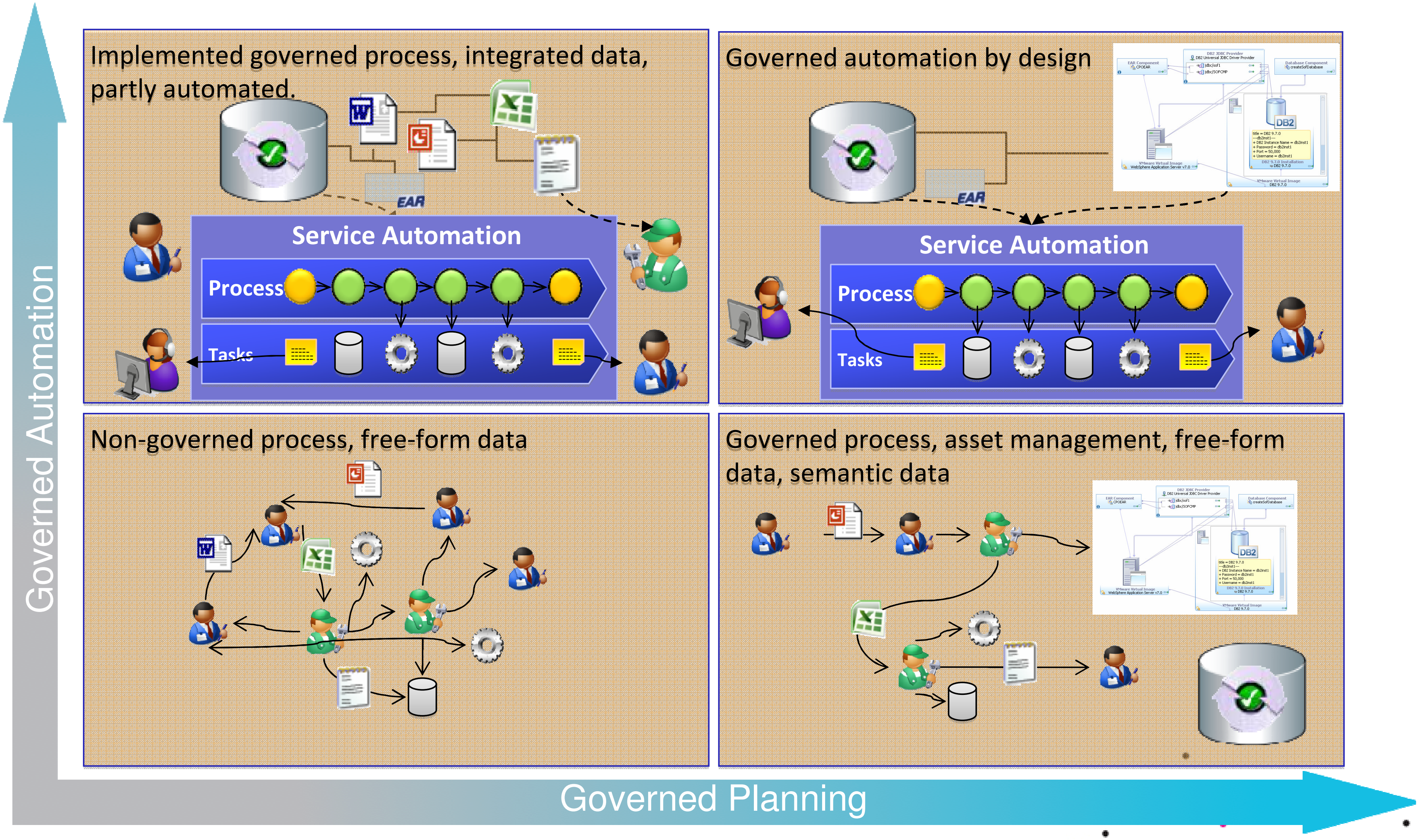
- "The project went incredibly smoothly, driven by good project management and backed by Odecee's expertise and strong track record in environment management."

—Frank Fabian, Head of Testing Environments, Delivery Services, ANZ Technology, ANZ

- "We wanted a solution that was flexible enough to handle changes in the environment and application space, and Odecee delivered."

—Frank Fabian, Head of Testing Environments, Delivery Services, ANZ Technology, ANZ

Introduction of NextGen ALM is an Evolutionary Process



Implemented governed process, integrated data, partly automated.

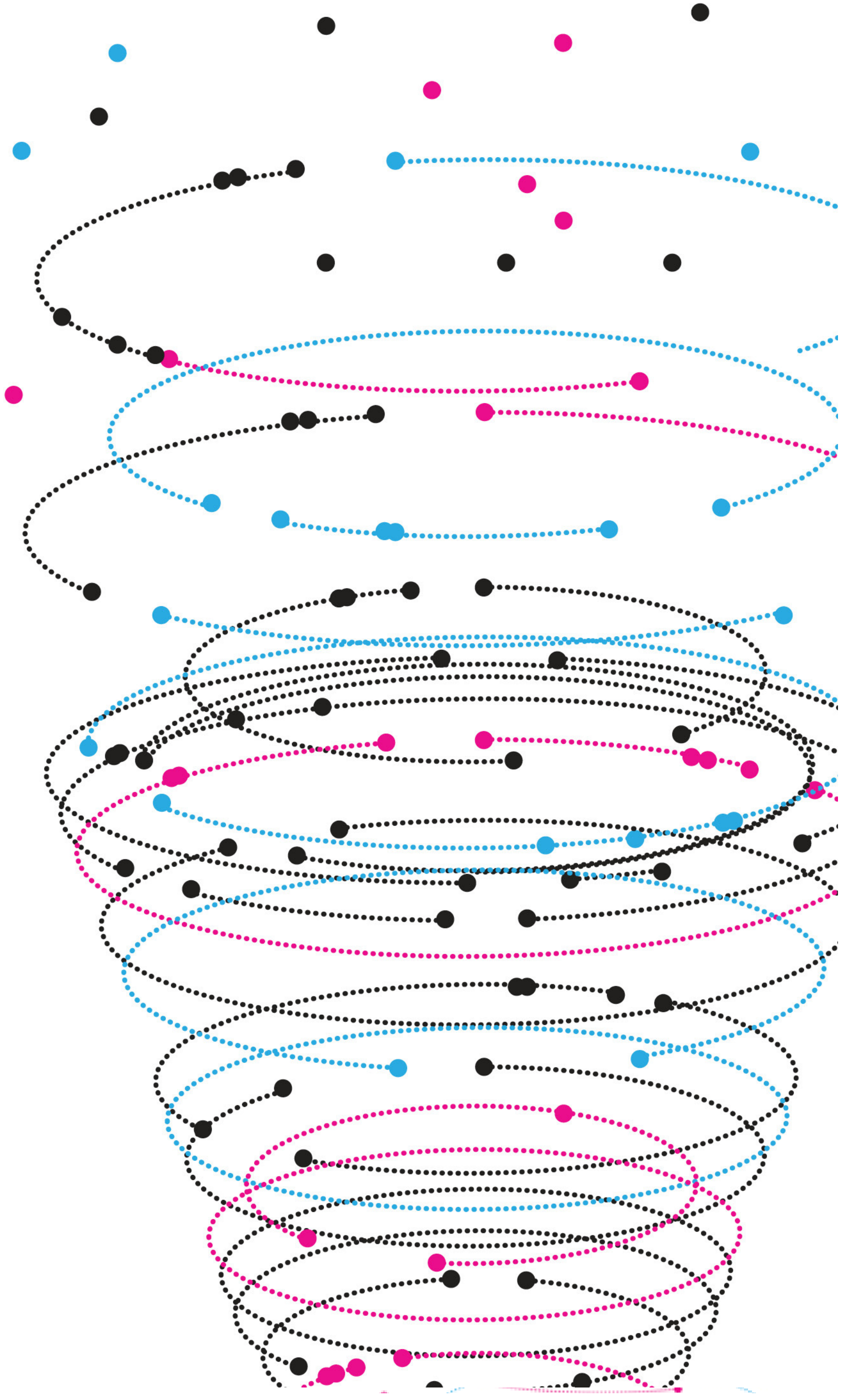
Governed automation by design

Non-governed process, free-form data

Governed process, asset management, free-form data, semantic data

IBM Next Generation ALM Seminar

Please take a few minutes to complete the evaluation form and return it at the registration table to receive your **complimentary gift**

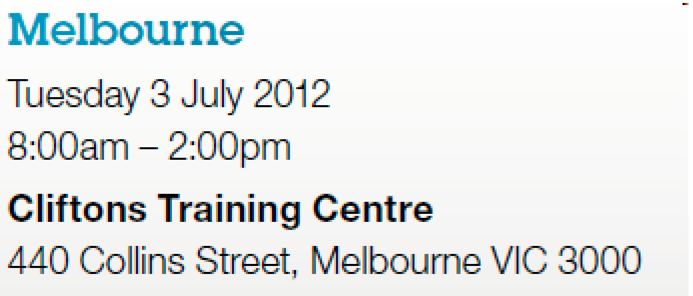


Smarter software for a smarter planet:

Take the fast track to better software delivery

Join us for a hands-on ALM workshop led by IBM's expert technical team

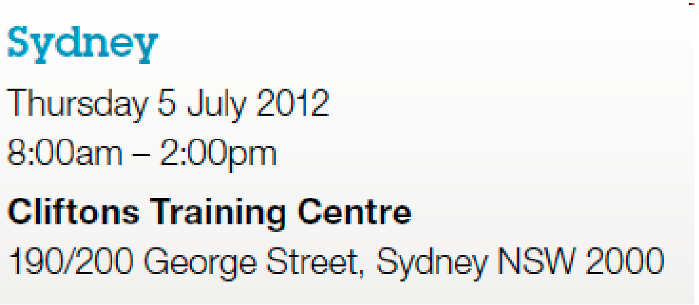
Your opportunity to test drive IBM's next generation ALM solutions.
Please express your interest on the evaluation form or register directly on the website. More information is available on the workshop flyer.



Melbourne

Tuesday 3 July 2012
8:00am – 2:00pm

Cliftons Training Centre
440 Collins Street, Melbourne VIC 3000



Sydney

Thursday 5 July 2012
8:00am – 2:00pm

Cliftons Training Centre
190/200 George Street, Sydney NSW 2000