Service Integration in Complex Systems: Lessons Learned and Watch Items

Bryce L. Meyer
James T. Wessel

May 2011

Software Engineering Institute | Carnegie Mellon

Overview

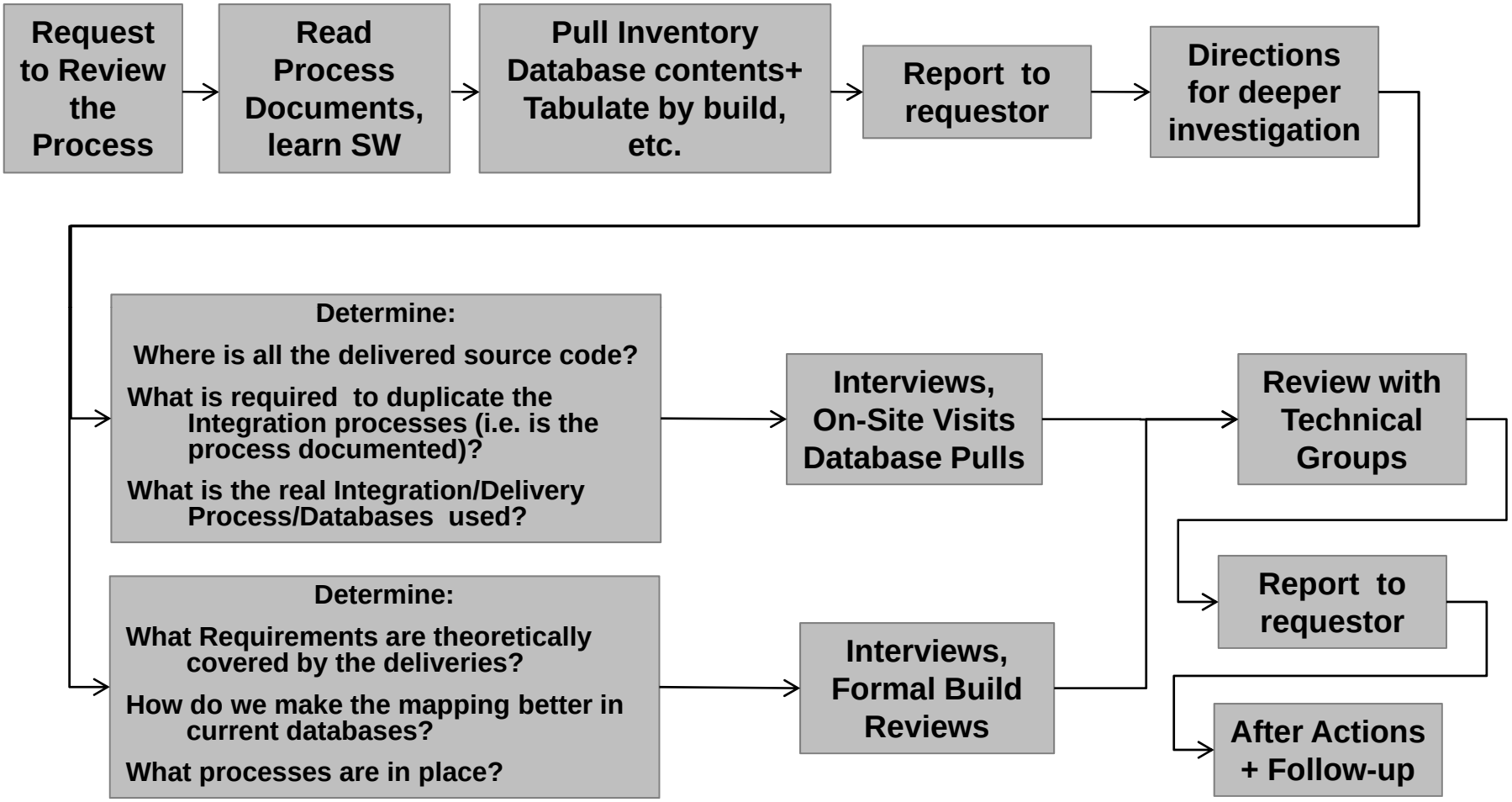
High-level troubles found in the investigation

One example of process and structure

Pitfalls we saw, questions to answer, possible solutions:

1. Roles and Responsibilities (technical and contractual)
2. Deliveries
3. Integration Process and Testing

Troubleshooting Integration: An investigation process and how we found the pitfalls



Troubles We Have Seen:

- Frustration is high for all parties involved
- Integration takes much longer than expected
- Resulting product is buggy, big, and slow; start-up is LONG

Some of these issues can be traced to integration pitfalls

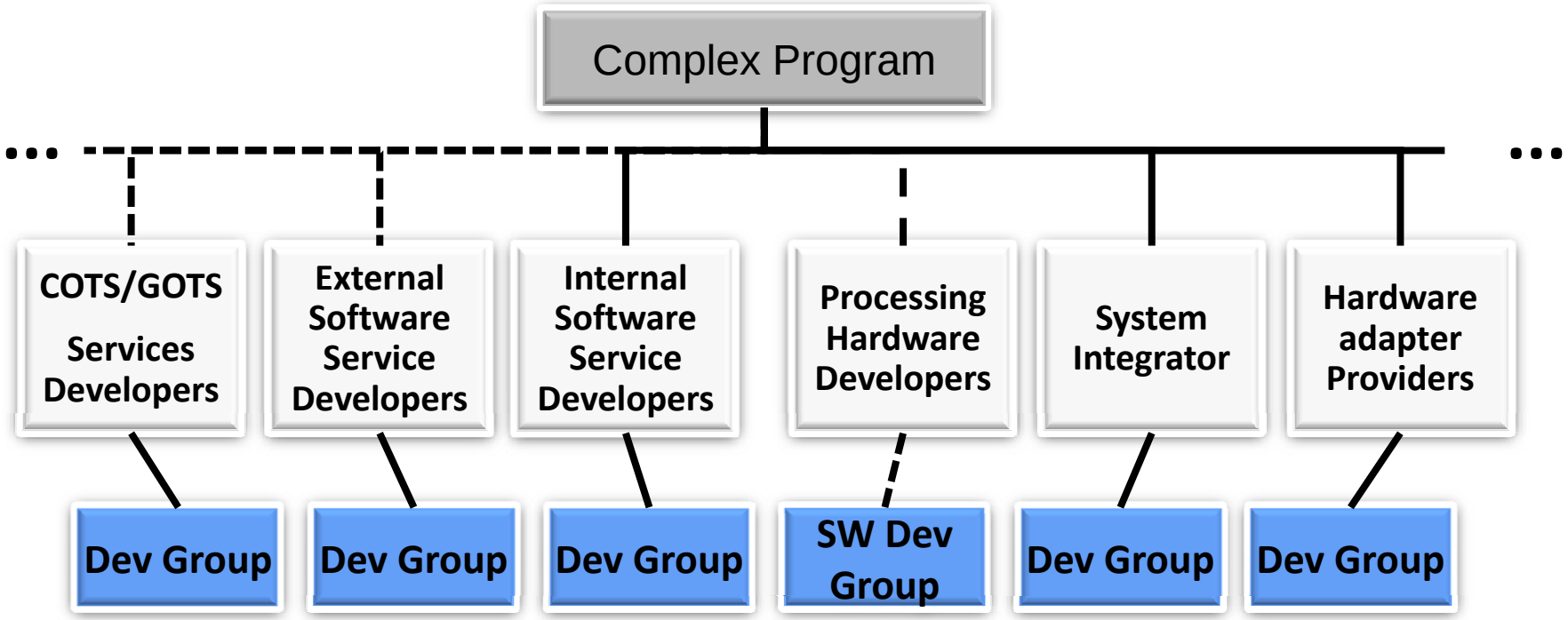
(and to classic software issues…See the SEI CMMI™ too)

Troubles We Have Seen: (continued)

Release state is uncertain

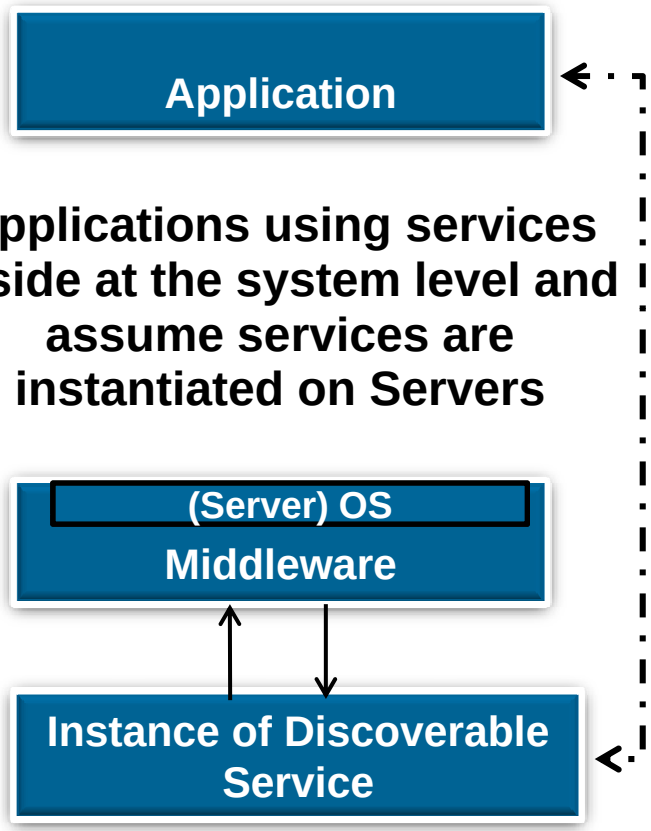
- Uncertainty if the right builds are in use
- Trouble tracing requirements satisfaction/ capabilities to actual delivered code
  - Difficulty in getting Interface Documents completed and signed off
  - Difficulty in finding source code for modules and in replicating the compile process
  - False Fixes caused by incorrect root cause (due to version conflicts between modules)

Example: Complex Multi-Provider Organization

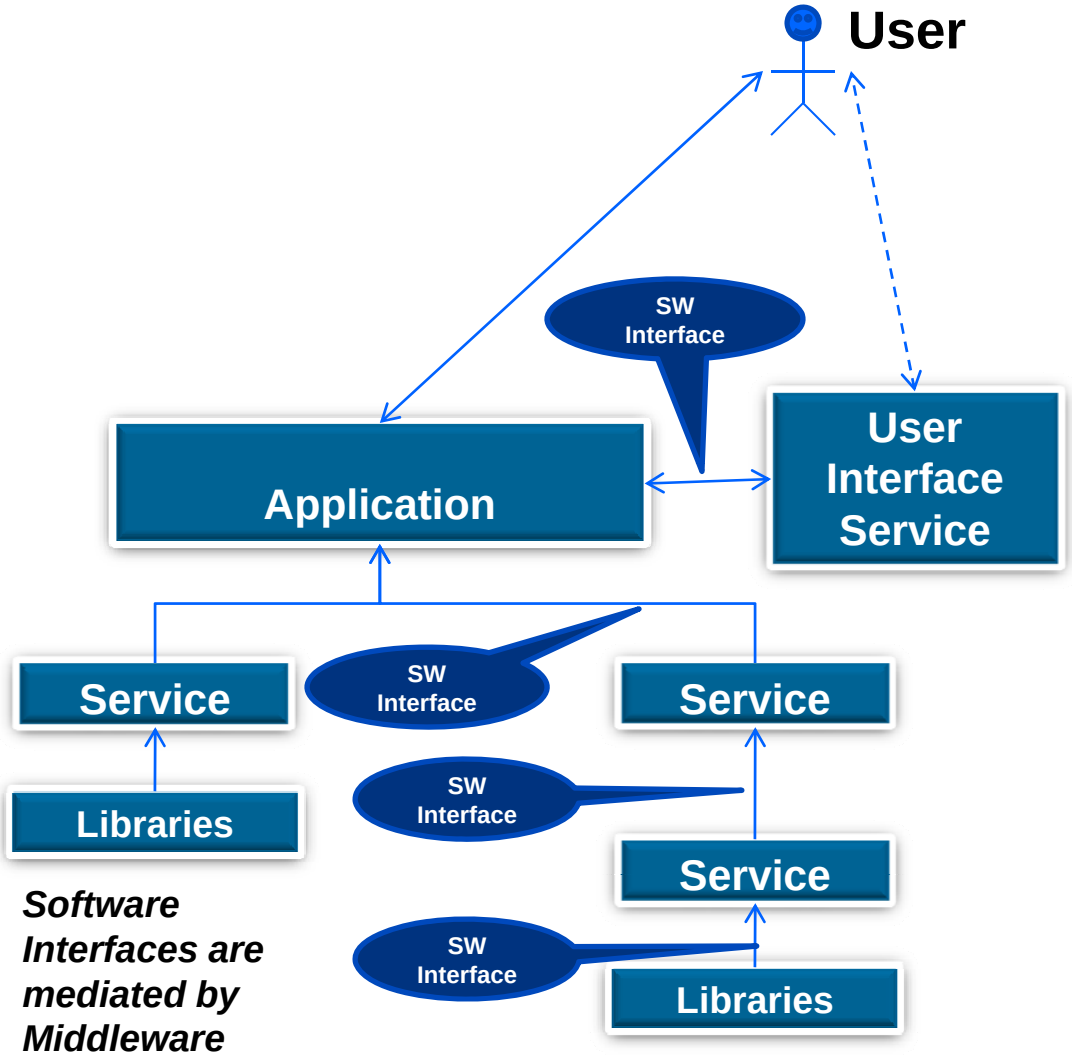


Example: Simplified Complex System

This schematic represents the context in which the example software was delivered

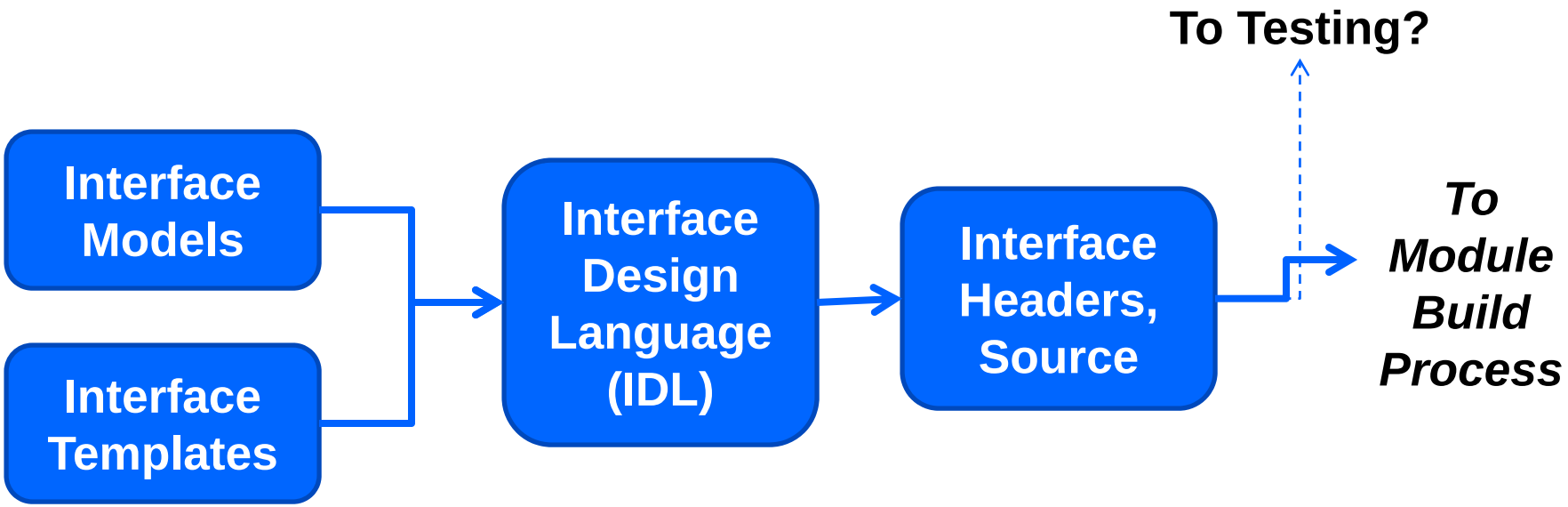


System Decomposition



Software Interfaces are mediated by Middleware

Example: Interface Build Process (integrating a large system)



Pitfall #1: What We Saw

- Contracts limited the access to various types of design documents and source code
- Code was dropped off in an immature state due to misunderstandings about pre-delivery testing
- Integrator had little support from developers, modules were in an uncertain state, and uncertain how to fix integration problems that were encountered

Pitfall #1: What We Saw (Continued)

- Providers were uncertain when to deliver, what items to deliver, and who would receive reports and data
- Uncertain of characteristics for interfaces to other providers, or if they can talk to other providers
- Uncertain of the role of the integrator
- Uncertain of reuse of middleware or other libraries

Pitfall #1: Questions to Answer and Possible Solutions

What standards must I meet for interfacing (internal to project, and to external services)? Use recognized standards and methods, define model data types and methods to be used. Systems should rely on middleware mediation, interfacing via middleware and standard, not directly.

See also [Bianco et al. 2007] for other considerations.

What do I have to provide as a supplier? (Documents? Source Code?) Will I have to let other suppliers see my source? NDAs? (See Pitfall #2) Supplier contracts and specifications must be specific. Ensure NDA signed off before first delivery. Source code access may be required if compile is by integrator or if customer will be maintaining the code. See [Morris et al. 2010] sec 3.1

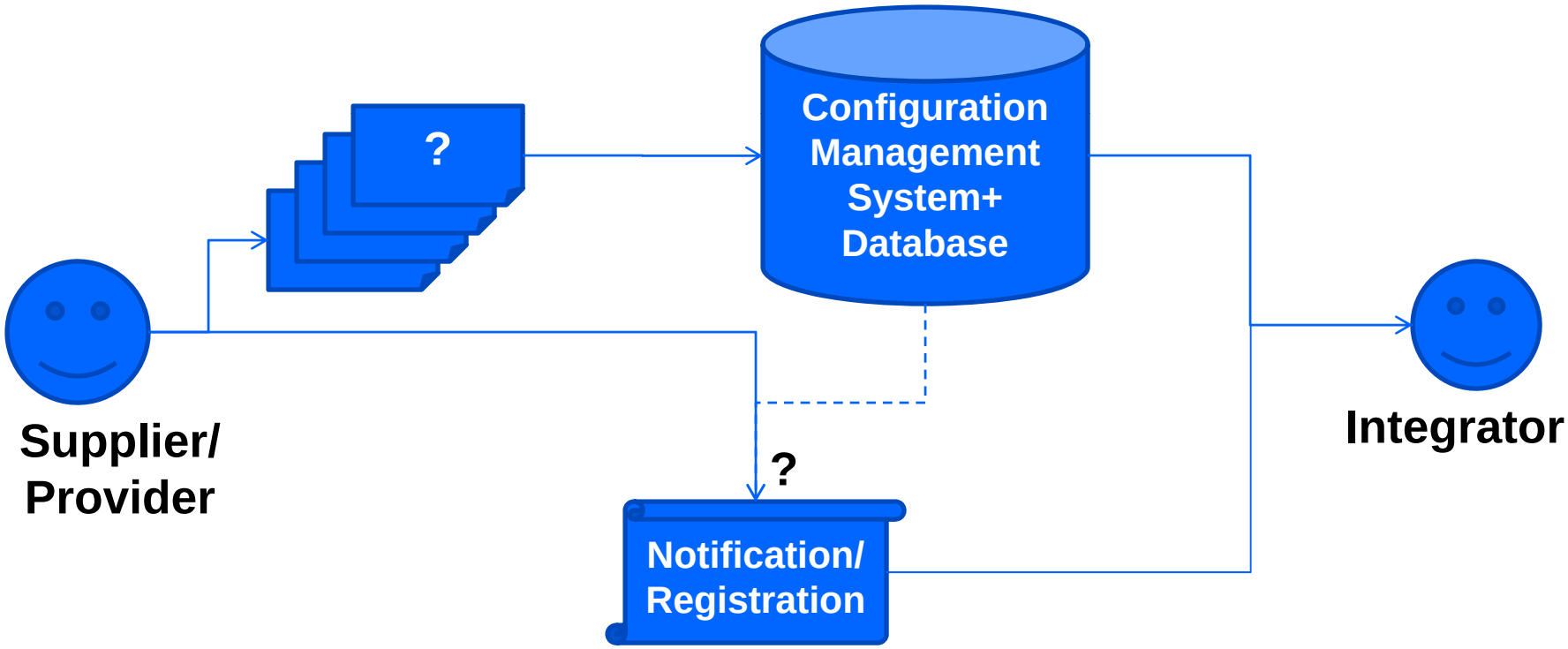
What resource restrictions do I have to meet, under what conditions, testing using what methods, and with what other software running? (See Pitfall #3) Very clear performance parameters by service must be defined before first delivery to integrator, and spell out testing required to prove fit.

Note [Lewis et al. 2008], Sec. A5 describes a set of similar questions to answer

Pitfall #2: Provider Deliveries

Where do software/service providers deliver code?

What do they deliver?



Pitfall #2: What We Saw

Procedures for delivery and expectations for delivered code varied

Symptoms:

- Confusion by customer: Where is my Code? Do the modules correspond with the delivered source?
- Confusion by Integrator: What versions am I building?

Pitfall #2: What We Saw (continued)

One delivery mechanism was specified, another in use

- Code was supposed to be registered in a delivery database, code (+documentation) and compiled modules to a Rational Clear Case instance
- Due to time pressure, delivery database was replaced with a form, Clear Case still used for compiled modules; Source Code no longer required
- Some Providers followed old process, some new

Pitfall #2: Questions to Answer

What	Why
Where do I make my deliveries? (i.e. Database, Workflow Engine, etc.) What forms, standards, process must I follow (ex: naming conventions)? Do I have access and to what?	This is very important for configuration control, and contract completion. Can also be useful for development in future releases and testing.
Do I deliver fully compiled modules?	Even if integrator does the compile, need something to check against.
Do I deliver files (beyond source) needed to do a full compile? (ex: generating scripts)	Even if integrator does not compile, useful for troubleshooting, and testing in the next increment (if contracts allow).
Do I deliver testing stubs? Test Scripts? Instrumented code?	Allows integrator to test the services and module, pre-integration.
Do I deliver documents and models? What statistics and data?	Quality Control, Process Metrics, and troubleshooting
Reuse items: COTS, GOTS?	Recommended that the integrator pull these, not the developer. Can lead to version issues and copyright concerns.
What testing/certification has to be completed before delivery?	Need to know the maturity of the module…is it mature enough to integrate, test, deliver?

Pitfall #2: Questions to Answer and Possible Solutions

Where do I make my deliveries? (i.e. Database, Workflow Engine, etc.) What forms, standards, process must I follow (ex: naming conventions)?
Agreements (and manuals) specify the process and specific locations/systems, naming standards, accounts, to deliver each type of file, form, and dataset (including models and test data). Training before first delivery for providers and integrator. Know database instance for delivery, and configuration control applied.

Do I have access and to what? Allow access to material required for developer testing and integration (compile & make). Users will need licenses in the delivery/CM system. Provide access to problem/trouble ticketing.

Pitfall #2: Questions to Answer and Possible Solutions

What do I deliver? (e.g., compiled modules, files beyond source needed to do a full compile, testing stubs, Test Scripts, Instrumented code, documents and models? Reuse items: COTS, GOTS?)

Provide code/scripts/files required for compiling/integration. Provide a successfully compiled image to confirm integration. Provide stubs and test items to check the modules function once compiled, and documents to ensure functionality.
See [Lewis et al. 2008], Tables 12, 13 may be useful.

Pitfall #3: Integration Process and (Associated) Testing

A sub-optimal Integration process can impede success

- Can be associated with Pitfall #1 & #2

The Integration Process should account for the varied levels of integration testing required. This will require access to parts of the integrated software, namely:

- Middleware
- Stubs or Modules which interface to other Provider Code (if linked)
- Common Infrastructure

A check-out step prior to formal integration can provide confidence in modules/code delivered for integration.

Pitfall #3: What We Saw

Providers services were written against older versions of services from other providers (Check-Out can catch)

Providers were reluctant to help other providers troubleshoot and uncertain of the state of the stubs used to test (MOA can assist)

Sandbox environment assembled for Provider use at Integrator was overscheduled (Schedule review can catch)

Performance allocations were exceeded, testing against performance was light and with low fidelity (Performance Team review should verify)

Pitfall #3: Questions to Answer and Possible Solutions

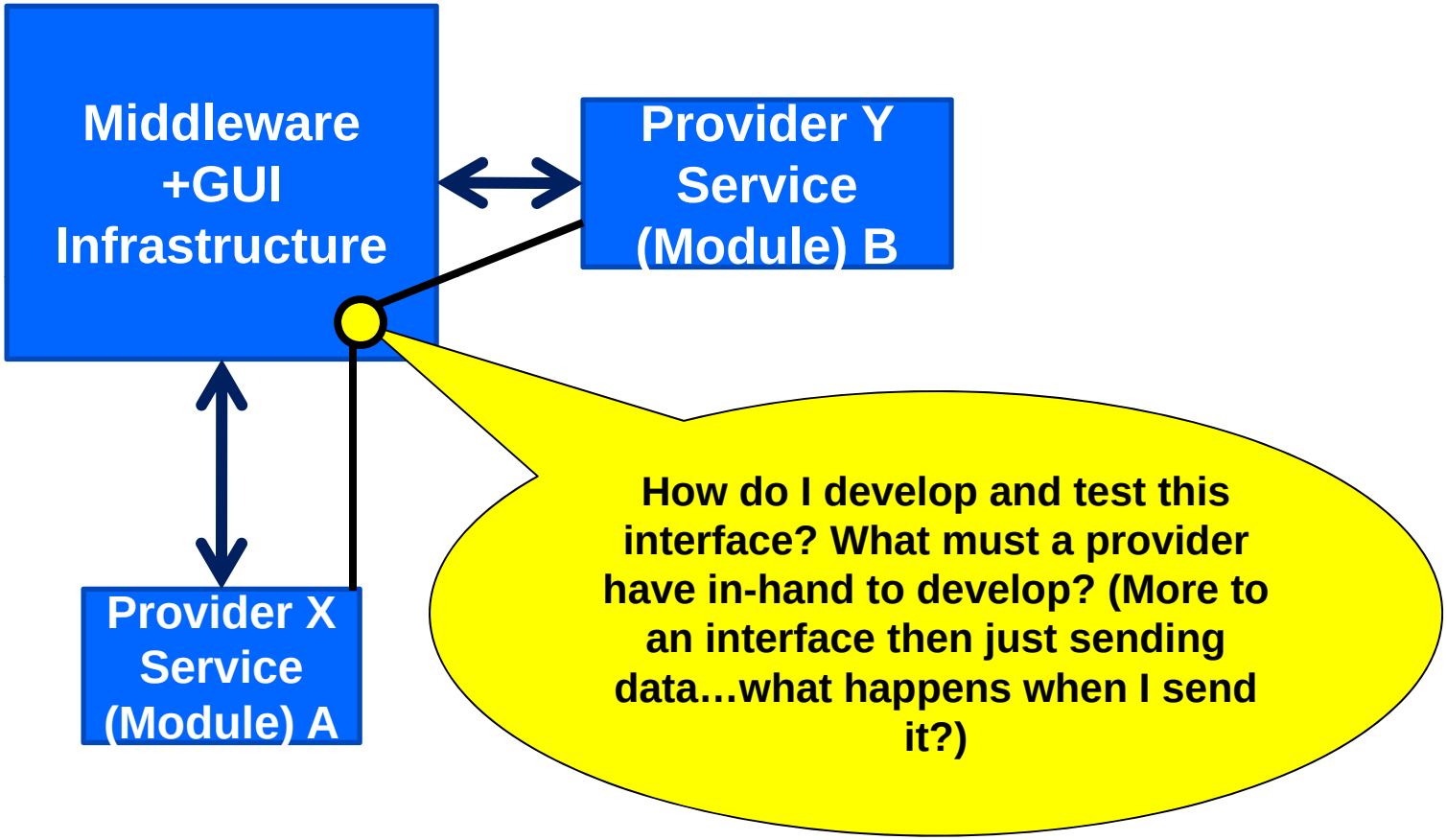
How do I integrate my complex system? Integration process is associated to the Architecture and must be defined before middleware and development by providers.

How is my middleware provided to developers? This is a critical path provider item. Provide complete releases with development kit and key test scripts, help manuals, and design documentation. Provide plenty of time.

What level of maturity must the code have? How do I know the level? What testing/certification has to be completed before delivery? What issues are resolved by this delivery? Compilation/Test history, focus on known problems. Certifications provide confidence. Mature test infrastructure provides confidence (e.g., Common test scripts and procedures, and common sandbox systems), as does Third party pre-integration. Mature Software Version Description is helpful (e.g., list of problems addressed, how addressed, and what files/databases/initial datasets are required for the module)

[see Bianco et al., 2008 for Service Level Agreement (SLA) information]

What should I deliver to providers to allow development?



Pitfall #3: Questions to Answer and Possible Solutions

How do my providers test interfaces to other providers? What test facilities do my providers have to mature their code? Provide the middleware build, stubs from other providers, and a series of planned test cases/use cases common to all providers. A sandbox environment with complete builds for middleware, GUI, and modules from other providers on a representative processor.

See also [Morris et al. 2010]

How do I allocate and test resource usage pre- and post- integration? Create key performance test cases. Performance is tied to end-user requirements, which are broken down to specifications that link to architecture and design. Provide enough environment to show coverage of expected modules on representative hardware. [Meyer et al. 2010]

What Scenarios/Use Cases with desired Quality Attributes can be used to 'bleed' issues out early on?

See [Bianco et al. 2007, Sec. 5, and Morris et al. 2010, Sec. 4.5]

Summary

Programmatic troubles can stem from Integration.

Integration can be a complex process for diverse groups.

Integration is a significant project: Plan for Integration Process, Deliveries, and Testing. Avoid the pitfalls by answering key questions about contracts, expectations, documentation, resources, etc.

References

[Chrissis et al. 2011] Chrissis, Mary Beth., Konrad, Mike., & Shrum, Sandy. *CMMI for Development: Guidelines for Process Integration and Product Improvement, 3rd Edition.* Pearson Education, Limited 2011.

[Bianco et al. 2007] Bianco, Phillip., Kotermanski, Rick., & Merson, Paulo. *Evaluating a Service-Oriented Architecture* (CMU/SEI-2007-TR-015). Software Engineering Institute, Carnegie Mellon University, 2007.

[Cohen et al. 2010] Cohen, Sholom G., & Krut Jr., Robert W. *Managing Variation in Services in a Software Product Line Context* (CMU/SEI-2010-TN-007) Software Engineering Institute, Carnegie Mellon University, 2010.

[Lewis et al. 2008] Lewis, Grace., Morris, Edwin J., Smith, Dennis B., & Simanta, Soumya. *SMART: Analyzing the Reuse Potential of Legacy Components in a Service-Oriented Architecture Environment* (CMU/SEI-2008-TN-008) Software Engineering Institute, Carnegie Mellon University, 2008.

[Bianco et al. 2008] Bianco, Phillip., Lewis, Grace., & Merson, Paulo. *Service Level Agreements in Service-Oriented Architecture Environments* (CMU/SEI-2008-TN-021) Software Engineering Institute, Carnegie Mellon University, 2008.

[Morris et al. 2010] Morris, Edwin J., Anderson, William., Bala, Sriram., Carney, David., Morley, John., Place, Patrick., & Simanta., Soumya. *Testing in Service Oriented Environments* (CMU/SEI-2010-TR-011) Software Engineering Institute, Carnegie Mellon University, 2010.

[Meyer et al. 2010] Meyer, Bryce L. & Wessel, James T. *Characterizing Technical Software Performance Within System of Systems Acquisitions: A Step-Wise Methodology* (CMU/SEI-2010-TR-007) Software Engineering Institute, Carnegie Mellon University, 2010.

Acronyms

Acronym	Use
NDA	Non- Disclosure Agreement
SEI CMMI™	Software Engineering Institute (SEI) Capability Maturity Model Integration (CMMI)
SLA	Service Level Agreement
SOA	Service Oriented Architecture
SoS	System of Systems
SW	Software

Contact Information

Bryce L. Meyer

Senior Member of the Technical Staff

Acquisition Support Program

412-268-7700

bmeyer@sei.cmu.edu

James T. Wessel

Senior Member of the Technical Staff

Acquisition Support Program

908-418-0323

jwessel@sei.cmu.edu

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN “AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this presentation is not intended in any way to infringe on the rights of the trademark holder.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

BACKUP

Abstract

Service Oriented Architecture based projects rely on a diverse team of service providers, application providers, middleware provider, and an integrator.

Often, the process for fusing the elements together to form a usable system of systems does not reflect the procedures that have evolved over the course of the program, due to the immense task of communication, due to timeline pressure, and often due to weak process enforcement.

This talk will cover:

- a series of pitfalls encountered on such large projects
- to show the audience how to avoid the pitfalls
- and what processes and tools should be considered when embarking on integration
- and what affect architecture decisions had on integration

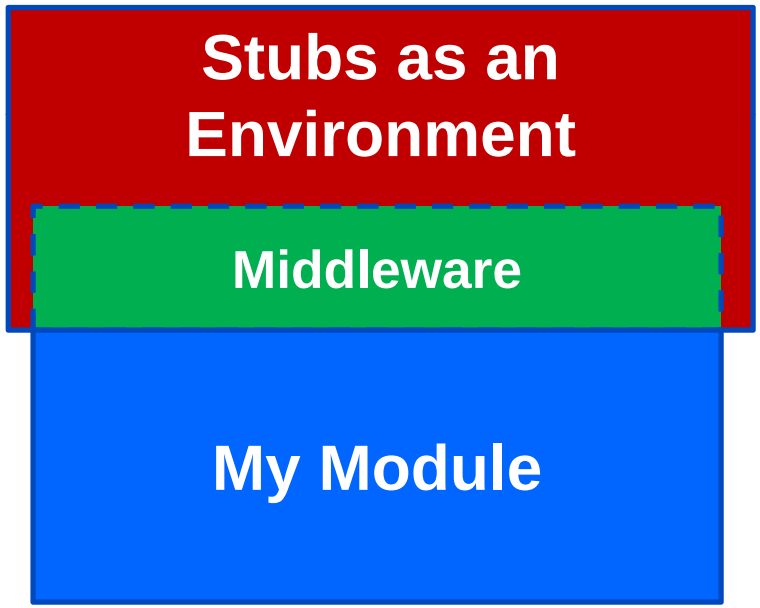
Testing Interfaces in Developing Code: Strategy - Stub provided as a unit by Integrator, Likely w/SDK.

Advantages:

- Same Single Package provided to each provider
- Integrator understands package
- Could be managed with one NDA per provider (provider to integrator)

Drawbacks:

- Models to create stubs must be current and accurate
- Long timeframe to create environment
  - May lag current provider code
- Very limited testing, environment must simulate resource consumption.



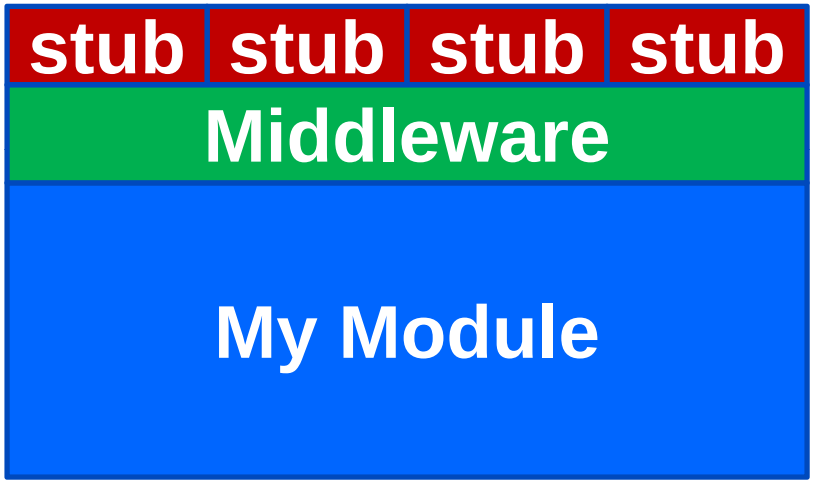
Testing Interfaces in Developing Code: Strategy - Stubs provided by Each Module/ Service Provider

Advantages:

- Short timeframe

Drawbacks:

- Lots of coordination, lots of NDAs
- Diversity in Stub currency
- Allows only limited testing of module function.



Testing Interfaces in Developing Code: Strategy: Modules Provided by Providers or by Integrator

Advantages:

- Real Code, Real interfaces: High Fidelity Testing

Drawbacks:

- Complex Planning to keep current, avoid serial development
- Strong Intellectual Property NDAs/Concerns: Why arm a competitor?