SAIC
From Science to Solutions™

Software Supportability: A Software Engineering Perspective

Stephany Bellomo
SAIC, Project Manager

My Background

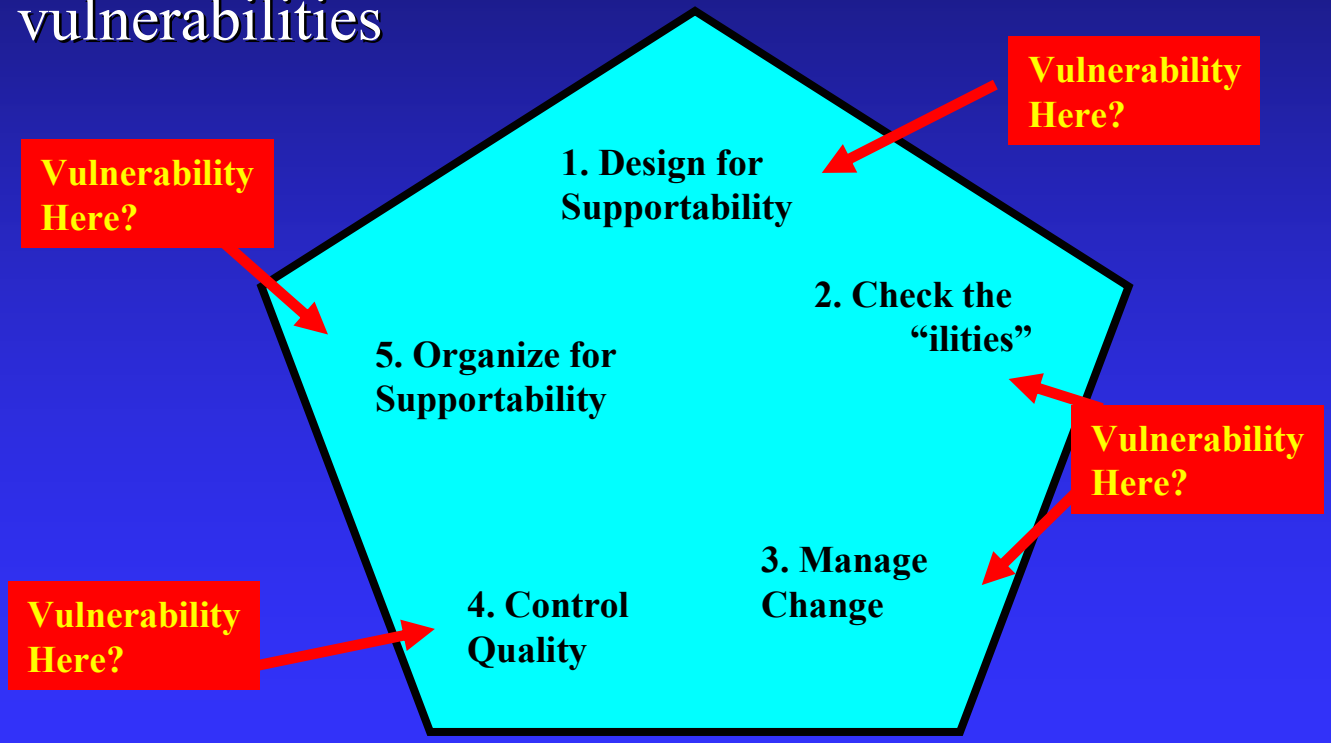
- MS, Software Engineering
- Lockheed Martin, Satellite System Programmer (C++ Developer, DBA)
- Intuit, Software Project Manager, (C++, Java, CORBA, Architecture)
- Verisign, IT Project Manager
- SAIC, Software Project Manager for CDC Select Agent Program

Overview

- 5 Supportability Principles
- Lesson's Learned
- Key Phase Recap
- Conclusion
- Contact Information

Supportability Principles Introduction

- Methodical approach to protecting system against vulnerabilities



Look for Vulnerabilities

Supportability Principle Overview

- 5 Supportability Principles
  1. Design for Supportability
  2. Check the “ilities”
  3. Manage Change
  4. Control Quality
  5. Organize for Supportability

Look for Vulnerabilities

Design Suggestions for Managers

1. **Design for Supportability**

   - Designing for supportability requires diligence on the part of both managers and engineers

   - What can managers do to identify vulnerabilities?
     - "What if" scenarios
     - Ask your technical team what the Achilles heel is – They will tell you!

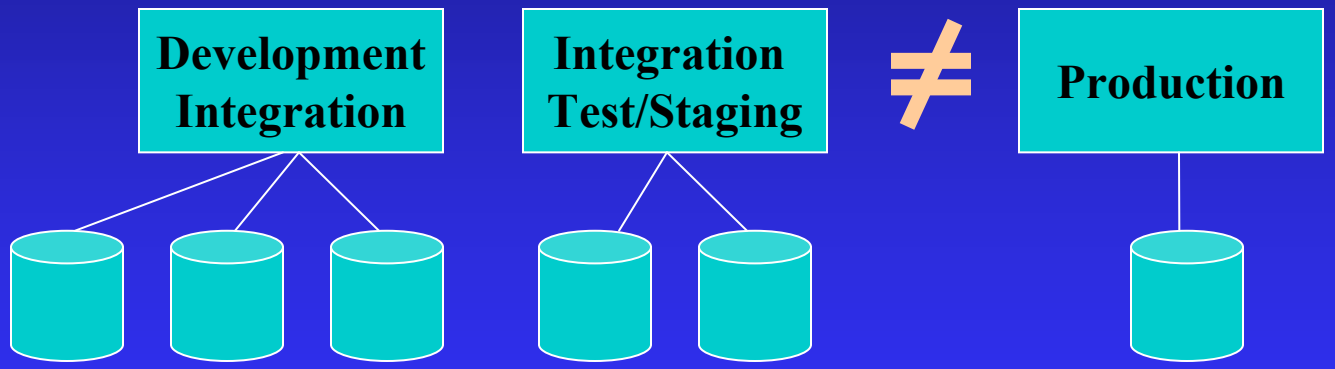
Look for Vulnerabilities

Design Suggestions for Engineers

- What can engineers to improve supportability through design?
  - Use a fully replicated production environment for pre-release testing
    - Don’t skimp
  - Parameterize using configuration files
  - Use frameworks to control design
  - Carefully evaluate COTS products before incorporating into the design
  - Incorporate distributed component design up front

Staging Example

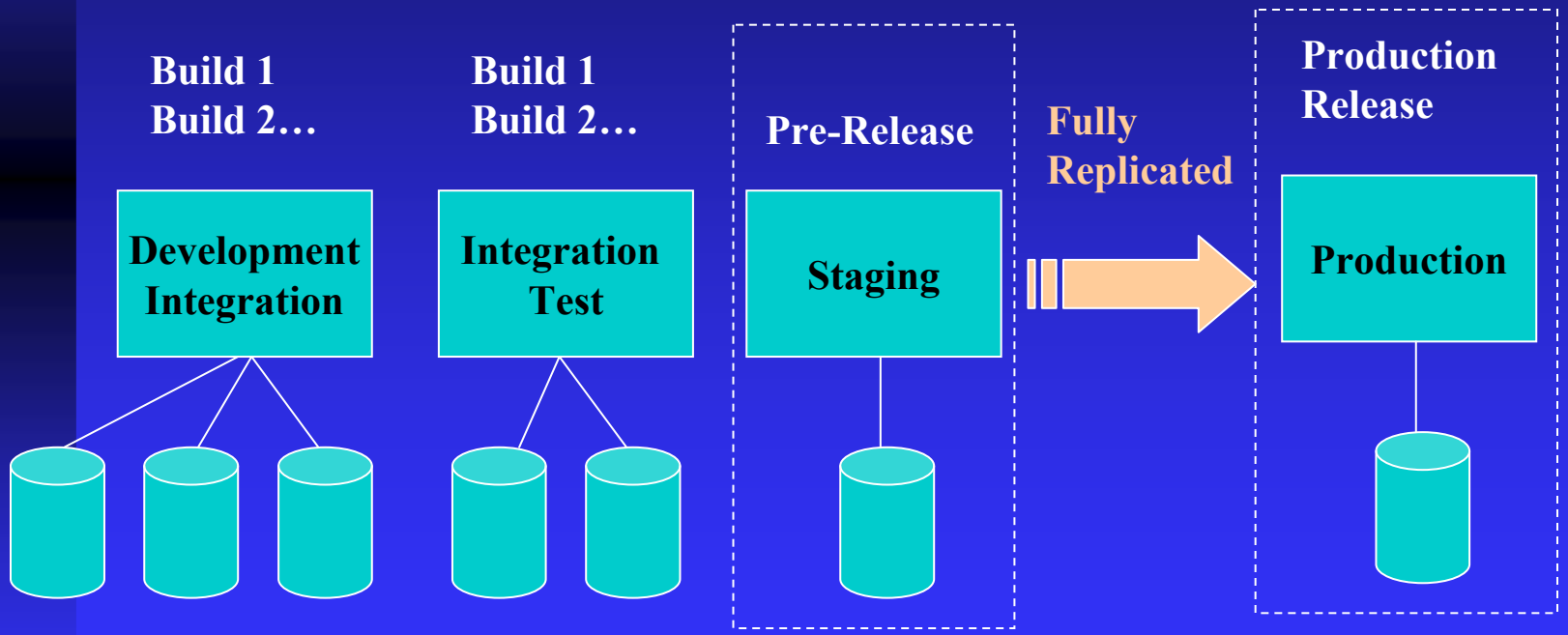
- Projects often double-use Integration Test and Staging
  - "It's not exactly the same environment as production, but theoretically it should work"



Stage = Production

Staging Example Cont.

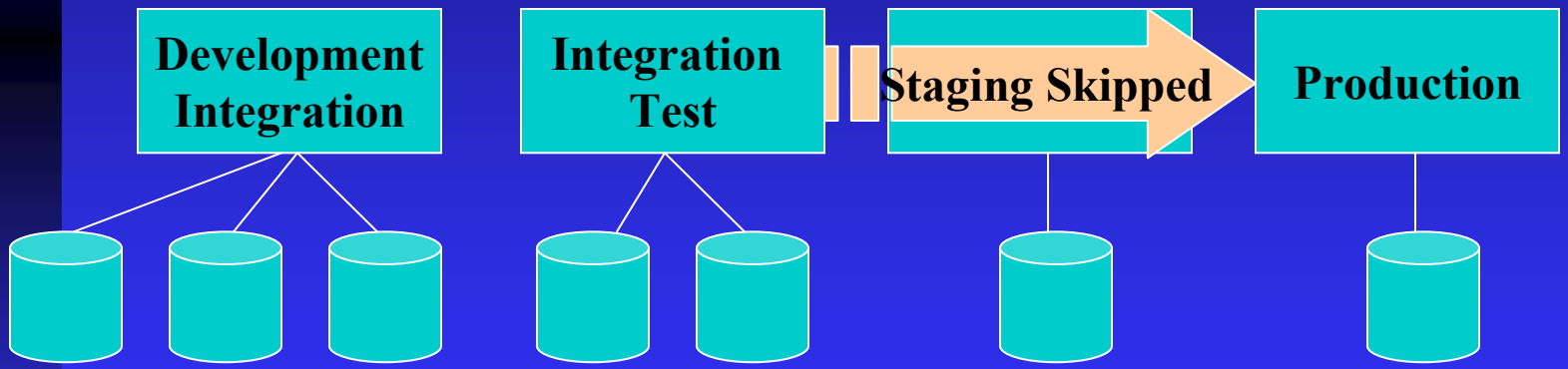
- Fully Replicate the Pre-Release Staging Environment



Stage = Production

Staging Lesson Learned

- Example: Recently technical lead skipped the staging for a small, non-production build
  - 8 hrs later still working deployment issues



  - Issue identified ½ hour after pushing to Stage

Stage = Production

Configuration File Example

- Design Tips for Engineers
  - Use Configuration Files
    - Avoid hard-coding variables (I.e, IPs, hostnames, DB names, etc.)
  - Benefit – Supports dynamic changes to hardware setup

Config File

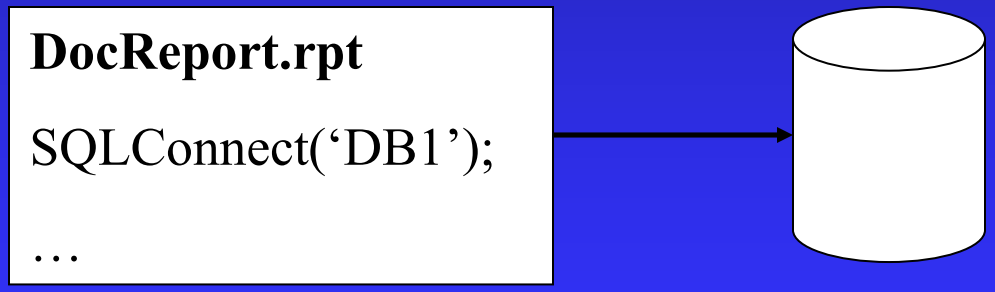
Setenv IP 122.11.333

Setenv DBNAME DB1…

Use Config Files

Configuration File Lesson Learned

- Recently migrated a legacy system to another HW configuration for high-availability (clustering)
  - Spent 2 weeks removing hard coded values
  - Host names and IPs were embedded throughout the code and reports



Use Config Files

Frameworks and Design Patterns

- Encourage developers to consider frameworks and design patterns during design phase
  - Frameworks
    - Data Entry Frameworks, Business Rules Frameworks, etc.
  - Design Patterns: Elements of Reuseable Object-Oriented Software
    - By Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides
  - COTS Best Practice
    - I.e, Documentum, Crystal Enterprise, Oracle Security, SQL Server, etc.

Focus on Frameworks

Framework Definition

- A **Framework** is a set of cooperating classes that make up a reusable design for a specific class of software
  - L. Peter Deutsch. Design reuse and frameworks in the Smalltalk-80 system
  - Quoted in Design Patterns: Elements of Reuseable Object-Oriented Software by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (gang of four)

Focus on Frameworks

Frameworks Lesson Learned

- Indicators that you need a framework
  - Frequently making the same types of code changes
  - Frequently adding fields to the schema
    - Example: Document Tracking Table

DocumentTracking

DocID	DocName	Reviewed DT	Approved DT
D1	Doc1	10-01-2005	10-15-2005
D2	Doc2	10-03-2005	10-17-2005

…adding tracking tables and date fields to DB for each new Event

Focus on Frameworks

Frameworks Lesson Learned Cont.

- Framework-Driven Event Model
  - Event additions are data driven
  - No schema changes needed to add an Event

EventType

EventID	Event Type
E1	Review
E2	Approve
E3	New Event

Add new event here

DocEvent

DocID	EventID	EventDT
D1	E1	10-01-2005
D2	E1	10-15-2005
D1	E3	10-15-2005

Event Framework Adds Date here

Focus on Frameworks

COTS Lessons Learned

- COTS are generally a good thing, but can drive bad design decisions
  - This is an ever increasing problem as the government encourages use of COTS

- Two Real Life Examples of COTS abuse
  1. Cold Fusion Dot Com experience
  2. Business rule scripting in UI or PDFs

Use COTS Carefully

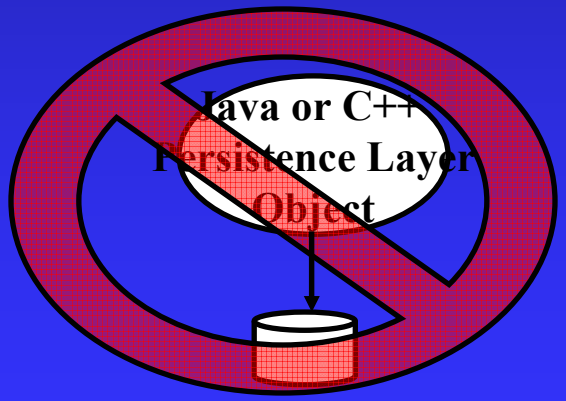
Distributed Design Intro

- Enforce Distributed Component Design through physically distributed methods, not coding standards
  - Software distributed component architecture can be enforced by RMI (I.e, Web services, COM, etc.)
  - Node distribution severs ties to object libraries
- What happens if you try to “fake it”?
  - Library dependencies aren’t discovered until production release testing
    - Result – Last minute scrambling…

Distribute Early and Often

Distributed Design Don'ts

- Plan for Unforeseen System Interface Requirements to other systems
  - Build Internal System Interfaces
- Don't rely on coded frameworks (COTS or homegrown) to encapsulate layers



Distribute Early and Often

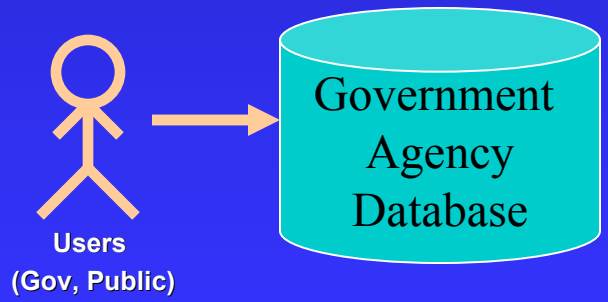
Distributed Design Lesson Learned - 1

- Example: In 1993 first job out of VA Tech, worked on a DoD satellite simulation system
  - Tasked to resolve this error for 6 months
    - ERROR: File not found!
  - Why?
    - Distributed design enforced by coding standard
    - No physical separation of software components
      - Months to untie code dependencies after physical distribution

Distribute Early and Often

Distributed Design Lesson Learned - 2

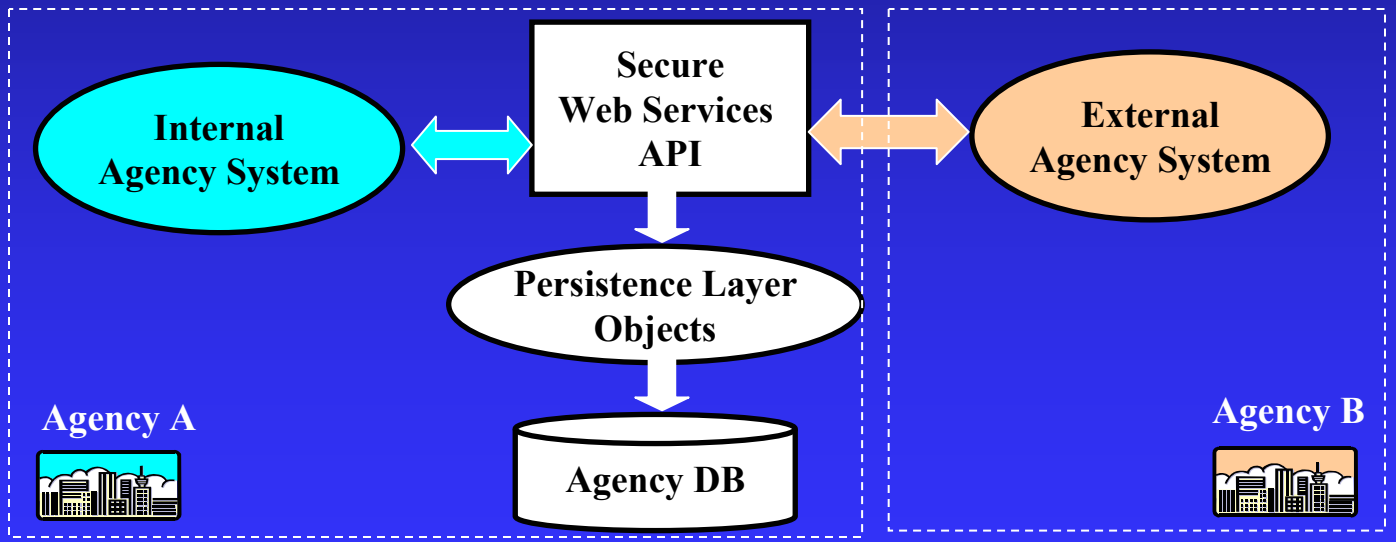
- Original SOW requirement
  - Mile-high view - Build Single Government Agency Database
- Requirements change
  - Allow another Government Agency to securely view data in database
- Good news
  - System is framework-based and extendible
  - However, still significant work to put persistence layer behind web services interface



Distribute Early and Often

Distributed Design Do's

- Do use distributed component interfaces to separate software layers (I.e., Web Services API)
  - Provides extendible data access through a secure interface



Distribute Early and Often

Check the “ilities”

2. **Check the “ilities”**
   - ✓ Security
   - ✓ Reliability
   - ✓ Flexibility
   - ✓ Maintainability
   - ✓ Scalability
   - ✓ Availability

Check the “ilities”

Configuration Management

3. **Manage Change**
   - Don't attempt too much change at once
   - Evaluate system impacts with changing requirements
     - Use the CCB*
   - Resist the temptation to "just add it in this time"

CCB = Configuration Control Board

Change a little. Test a lot…

Database Configuration Management

- Worst configuration management issues consistently revolve around Database CM
  - I.e., Stored procedures, Schema versioning, Scripts, Hand-data entry
- Reasons for poor database CM
  - In my experience, DBAs often don't have formal Software training
    - SW Developers trained to use CM tools at entry level, but DBAs often not included in CM training
  - DBAs often don't have to integrate with others
    - Work independently
      - Don't need to update baselines to test code

Enforce Database CM

Database CM Lesson Learned

- Database Management Fundamentals
  - Creating and enforcing Database change procedures must be part of DBA Responsibility
  - Stored procedures must be and scripts stored under configuration control
    - Example – "Lost stored procedure story"
  - All databases should be made through scripts AND TESTED!!!

Enforce Database CM

Quality Control

4. **Quality Control**

- Monitor to maintain quality and identify new risks
  - Keep CMMI inspections technical
  - Develop processes and follow them
- Enforce Independent Verification and Validation
  - At a minimum, developers should not test their own code
- QA person should report to Program Manager

Anytime is good time for a Technical Question

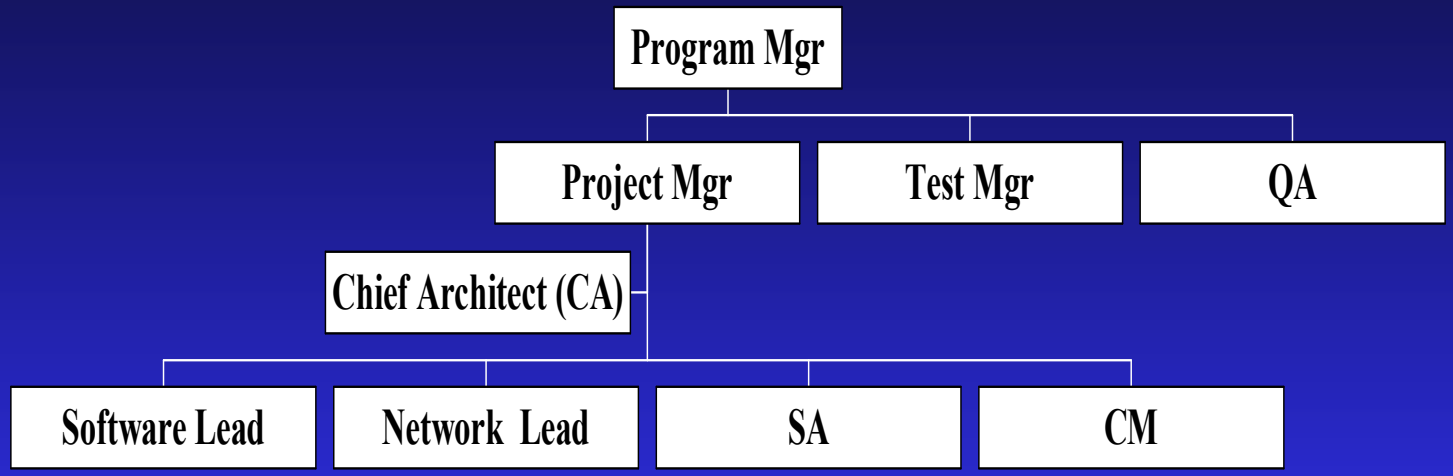
Organize for Supportability

5. Organize for Project for Supportability

- Supportability failures often occur between teams or areas of expertise
  - I.e., software team, network team, SA, Security, etc.
- Mitigation strategy
  - Assign someone the specific role of enforcing cross-disciple technical quality

Architect: The Tie that binds

Organize for Supportability Cont.



- Chief Architect leads cross-discipline teams
  - Qualified Tech Leads start as Software, Network or System Engrs
- Challenge: Finding architects that can manage outside their "Comfort Zone"

Architect: The Tie that binds

Key Phrases

- ***Look for Vulnerabilities***
- ***Stage = Production***
- ***Use Config Files***
- ***Focus on Frameworks***
- ***Use COTS Carefully***
- ***Distribute Early and Often***
- ***Change a Little. Test a lot…***
- ***Enforce Database CM***
- ***Anytime is a Good Time for a Technical Question***
- ***Architect: The Tie that Binds***

Conclusion

- In all project activities, ask yourself these questions:
  1. Does this Design Decision promote Supportability?
  2. Have we considered all the “ilities”?
  3. How well are we Managing Change?
  4. Are we adequately Controlling Quality?
  5. Are we organized for Supportability?

Contact Information

- My contact information:
  - Stephany.a.bellomo@saic.com
- Feel free to send me questions and/or comments