

Slide 1




Wyzzk, Inc.

Product Family and Program of Projects Architectures

Copyright © Wyzzk, Inc. 2004
Version 5.0

Slide 2



Wyzzk, Inc.


Lesson Goal & Objectives

- Understand the purpose and nature of product family and program architectures.
- Upon completion of the lesson, the participant will be able to:
 - Describe the issues and constraints of a product family architecture
 - Describe the issues and constraints of a program of projects architecture

Copyright © Wyzzk, Inc. 2004
Version 5.0

Enterprise Architecture - 2

Slide 3




Lesson Outline

- Product Family (Line-of-Business, Product Line) Architecture
 - What is product family architecture?
 - Variability
 - Commonality
 - Versioning & Bug Fixes
 - Use Cases and Other Requirements
 - Documentation
- Program (programme) of Projects Architecture
 - What is program of projects architecture?
 - Patterns
- Constraints and issues
- Impact on Project Architecture

Copyright © Wyzzk, Inc. 2004
Version 5.0 Enterprise Architecture - 3

Slide 4

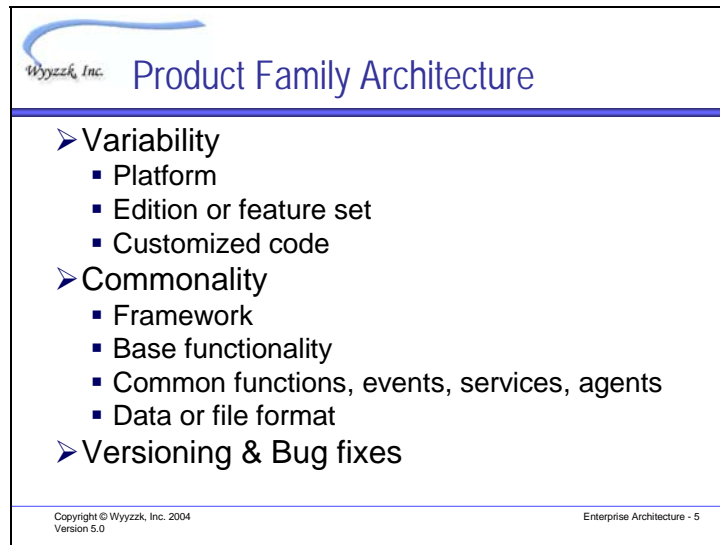


Product Family Architecture

- This concerns the architecture of a set of applications that are related because they are variations on the same product or set of products
- This is sometimes called line-of-business architecture or product line architecture

Copyright © Wyzzk, Inc. 2004
Version 5.0 Enterprise Architecture - 4

These used to be called line-of-business architectures, but at some point in time, line-of-business came to refer to business units, such as accounting, human resources, IT, marketing, sales, and so on. The current term appears to be product family, though a recent book by Hassan Gomaa calls this product line architecture.



The slide features a blue logo for Wyzzik, Inc. in the top left corner. The title "Product Family Architecture" is centered at the top. The main content is a bulleted list with three primary categories: Variability, Commonality, and Versioning & Bug fixes. Each category has sub-bullets. At the bottom, there is a thin blue line with copyright information on the left and "Enterprise Architecture - 5" on the right.

Wyzzik, Inc. Product Family Architecture

- Variability
 - Platform
 - Edition or feature set
 - Customized code
- Commonality
 - Framework
 - Base functionality
 - Common functions, events, services, agents
 - Data or file format
- Versioning & Bug fixes

Copyright © Wyzzik, Inc. 2004
Version 5.0 Enterprise Architecture - 5

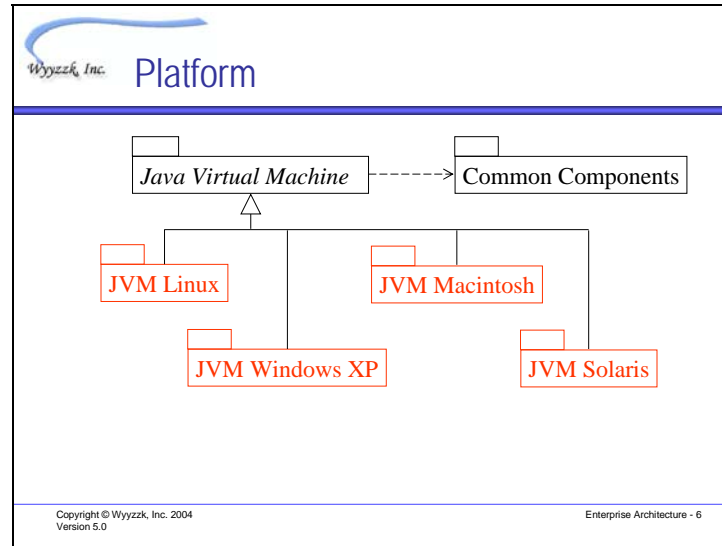
A major theme of software architecture is to identify the most likely kinds of changes to occur, then isolate or encapsulate the part of the application(s) that will need to be modified because of that kind of change. This is particularly apparent at the Product Family level of architecture. Kinds of changes may include platform, edition, or customized code.

You also have to consider methods for handling the common parts of the product families. This could be done using frameworks, base functionality in layers or tiers, libraries of things such as functions, components, services, event handlers, or agents, and defining common data or file formats.

Framework could be something like a backplane or plug and play technology, or it could be a communication framework. Think of things such as Eclipse or CORBA.

Base functionality can be the lower layers of a layered architecture, or some of the tiers of a tiered architecture. At a minimum, the products in the family can share a common function library, agents, services, or event managers. Typically all products in the product family will share a common data and/or file format. Another major issue for product families is how to handle versioning and bug fixes.

Slide 6



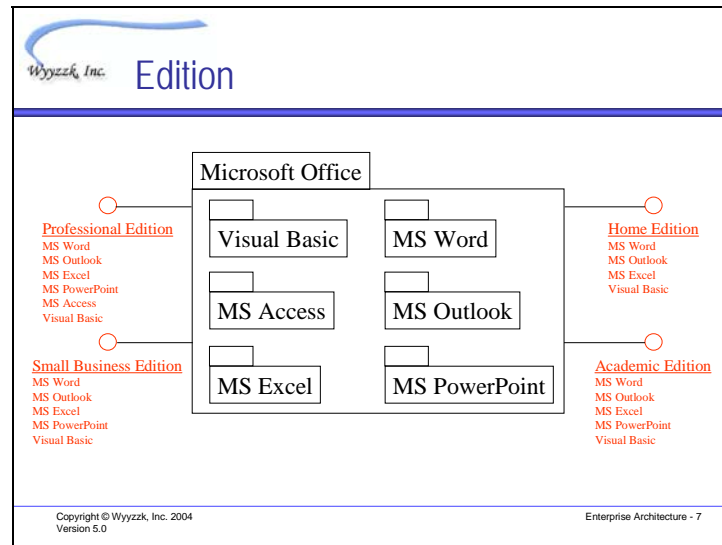
This example shows a possible architecture for the Java Virtual Machine (JVM) family of products. Some parts of the JVM are exactly the same, no matter what platform it runs on. In addition, each version of the JVM includes code for the specific platform on which it runs. This is a product family where the variation point is the platform on which the software runs. Notice that the platform specific subsystems are completely independent. The only shared code is in the Java Virtual Machine subsystem.

What you may find in this architecture is that the Java Virtual Machine may all be abstract or pure virtual. In other words, there is no real code for the Java Virtual Machine. Then each platform specific subsystem will contain the actual code to implement those features written for a specific platform. That is what we are showing in this diagram. The italicized name for Java Virtual Machine indicates that this subsystem is abstract – in other words, there is no actual code in this subsystem. Each platform specific subsystem provides code to implement the Java Virtual Machine. There may also be some concrete code in Java Virtual Machine, possibly a function library or web services that run on any platform.

The nice thing about this kind of architecture is that it is relatively simple to maintain. Any changes to the platform specific parts of the application only happen in the platform specific subsystem. Care has to be taken with changes to the Java Virtual Machine. Adding something new is easy, because nothing depends on it. You can choose when to expose the new functionality (if at all) in your specific platform subsystem. Making a change to an existing feature in Java Virtual Machine is strongly discouraged, because that change could immediately impact code in all of the platform specific subsystems.

This is a very good architecture because the primary kind of change to this product family is the addition of new platforms and that kind of change is isolated in platform specific subsystems. In this kind of architecture, if the most frequent kind of change were changes to the Java Virtual Machine, this would be a bad architecture, because those kind of changes have a ripple effect throughout the whole architecture.

Slide 7



This example shows a possible architecture for the Microsoft Office family of products. Microsoft Office is composed of a set of applications. Each specific edition of Microsoft Office is composed of a subset of the total set of applications. In addition, the specific features available in a particular application may change from one edition to another. This is a product family where the variation point is the feature set of the software.

The nice thing about this kind of architecture is that it is relatively simple to maintain. There is really only one set of code, but a subset of features is exposed for a particular edition. The issues with this kind of product family are those of versioning and backward compatibility. You don't have to worry about compatibility between editions, because all the editions are actually using exactly the same underlying code and file structures. Various companies sell licensing software that makes it relatively easy to create the different editions by turning on or off specific applications or features of applications.

This is a very good architecture because the primary kind of change to this product family is updates to the core applications and features. Any change is immediately made to all editions, because they are really all sharing the same software. Adding a new edition is also easy because it just means creating a new interface, which probably means

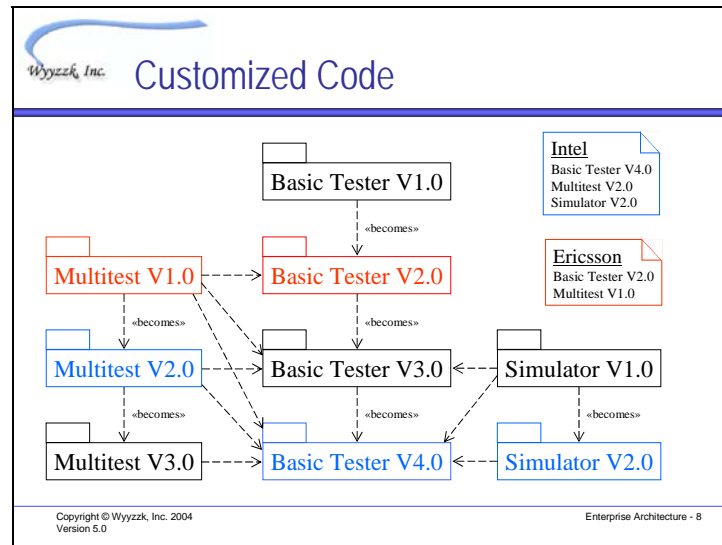
creating a new edition in the licensing software. This kind of architecture does not support different versions of software for the different editions. There is one large code base (Microsoft Office) and the editions are created by turning on and off features.

There are several ways to implement turning on and off features.

One way is to use licensing software to select the allowed features. Using this software, you define a set of features (interfaces) that you will expose in the software. Then you define a type of license (enterprise, small business, etc.) and create a list of features that are legal for this license. When a user installs the software, the license key identifies to the licensing software which kind of license has been purchased. The licensing software then locks or unlocks the indicated features for that type of license. The advantage to this scheme is that there is one release of the software with everything in it. You don't have to manufacture different CD's for different versions of the product. The disadvantage is that the licensing software could be cracked, allowing the customer access to all the software features, even the ones they did not purchase.

Another approach would be to have different releases of the product stored in a version control system. In the version control system, you create a different release for each version of the product (professional, academic, etc.), then identify the code files that will be linked together to create that version of the product. You create different CD's in manufacturing, one CD for each different version of the product. In creating the package, you have to be careful that the correct CD is put into each package. The advantage is that a customer only gets delivered exactly what they purchased, and there is no way to crack licensing software to get access to additional features or software – the additional features or software are not on the CD.

Slide 8



The typical way you get customized code is that you have a standard code base, but different versions customized for different customers. With the edition version, we are assuming packaged software, where there are multiple standard editions for purchased. For customized code, we are assuming that this is a business-to-business relationship. Now our customer is one particular company, and they may want our product customized for their particular purpose. Another of our customers wants the same software, but customized somewhat differently.

The big issue here is what is the nature of the customization? Is it something simple like a change to the look and feel of the user interface, or something more complex, such as addition or subtraction of features, or specialized data formats? Here we assume that the customization involves specific features for specific customers. Some of these features may only be used by one customer, some will be used by multiple customers. For a variety of reasons, different customers may be on different versions of the underlying basic software.

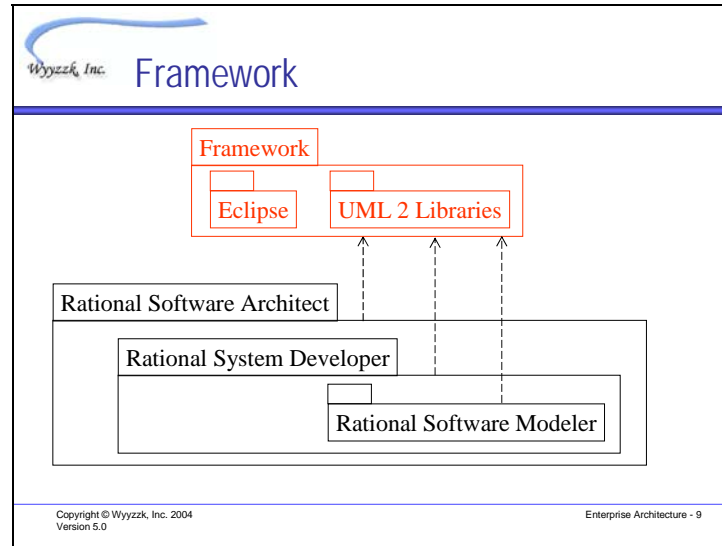
Assume the basic software controls a microchip testing facility. We sell a basic version of the software that provides basic control functions for one type of microchip using one

type of test equipment (write a test script, run the test script to test batches of chips, put rejected chips in one bin, good chips in another bin). We sell additional features: test multiple types of microchips, run multiple tests on the same batch of microchips, control multiple types of test equipment, perform error analysis, define multiple bins for categories of chips (pass completely, fail completely, partial failure, ambiguous-test again), simulator for developing test scripts. Some of these features are only available on particular versions of the base software (or later versions).

In general, maintaining customized code is a nightmare over time. It seems like a good idea up front, but can quickly grow unmanageable. The multiple editions idea from the previous slide is a much more manageable version of this idea. But it may not be possible to come up with a few standard editions of the software.

Notice the direction of the arrows. Basic Tester must be stand alone, so the add-on products have to work by making calls to Basic Tester. We cannot expect Basic Tester to know anything about the add-ons.

Slide 9



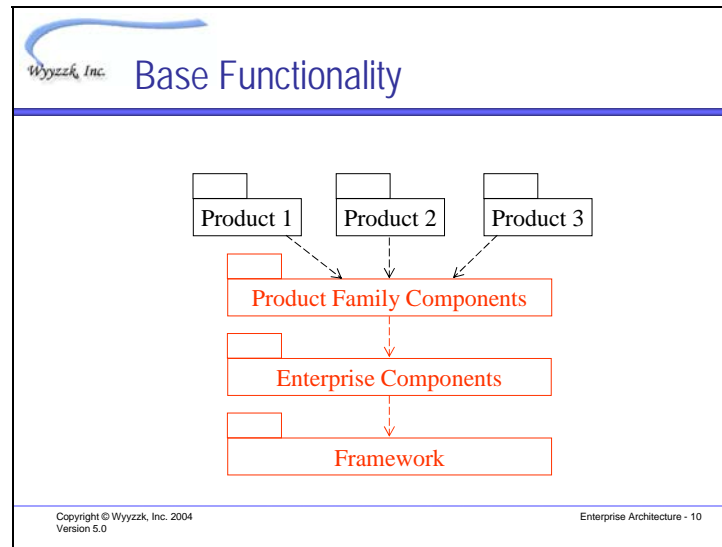
One way to create a product family architecture is to start with a basic framework on top of which everything else works. This framework can be written by the product team, something purchased, or both.

In this example, we consider the IBM RSx product family. All of the products are built on an Eclipse framework with UML 2 libraries. This provides some basic functionality. In addition, each product in the family is built on top of one of the other products.

When is something a framework versus a component library? A framework is a set of components that work together to provide a package of functionality. The framework itself is generally an executable or set of executables. In addition, the framework provides places where you can add in more functionality, sometimes called “hooks”. What you build on top of the framework enhances the functionality of the framework. So in this current example, Eclipse is a development environment of its own. The IBM RSx products use that basic development environment and adds on to it. In this particular example, Eclipse and the UML 2 libraries are open source.

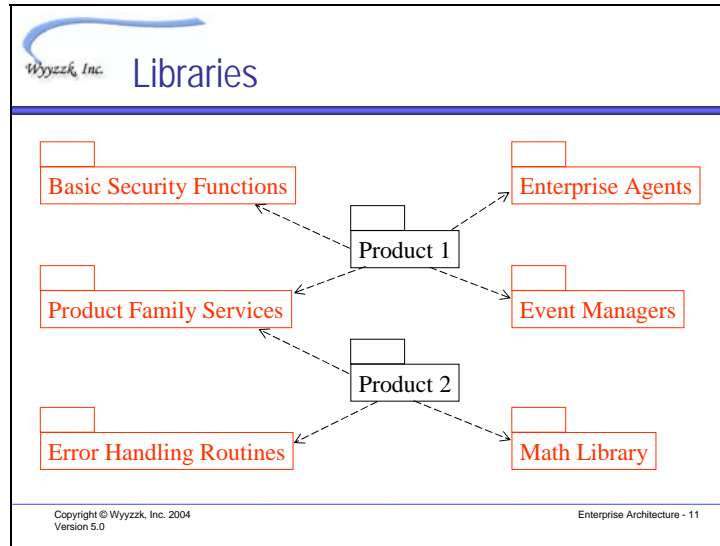
Another kind of framework that goes in and out of popularity is a backplane. A backplane provides basic services so that any applications you plug into it are automatically interconnected.

Slide 10

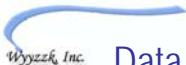


Similar to the idea of a framework, here we are using multiple layers as the basis for our product line. In this case, a framework is part of the base functionality, as well as a library of components (services, agents, event managers, functions) at the enterprise level and product family levels. We are taking advantage of large amounts of commonality in order to write as little new code as possible and to make maintenance of existing code even easier.

Slide 11



Libraries are collections of things. They can be defined for any reason. The basic idea is to write a bunch of code that is generally useful and bundle it together as a library. The only real difference between this slide and the previous slide is that the subsystems (libraries) on this slide have no relationships to each other, whereas on the previous slide, the subsystems are built on top of each other.



Data & File Format

- Shared data model
- Same file format
- Ease of maintenance
- Ease of upgrades

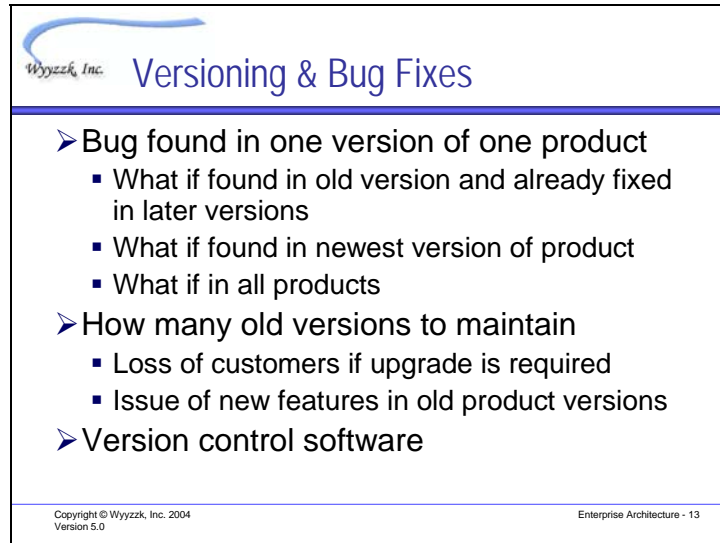
Copyright © Wyzzyk, Inc. 2004
Version 5.0

Enterprise Architecture - 12

As much as possible, a product family should share the same data. The data model should reflect the combined data models for all the products in the family. This will often mean that some fields in the data model are not used by some members of the product family, but the trade-off is ease of maintenance of one model rather than one per product.

By the same argument, if files are used by the products, they should all share the same file format.

Think about how easy it is to upgrade to a product with more features if the data model and file formats match. As a customer, you don't have to change or convert any of your data when you upgrade your product to get more features. This is a reasonable expectation from the customer, so the product should be architected to work that way.



Wyyzzk, Inc. Versioning & Bug Fixes

- Bug found in one version of one product
 - What if found in old version and already fixed in later versions
 - What if found in newest version of product
 - What if in all products
- How many old versions to maintain
 - Loss of customers if upgrade is required
 - Issue of new features in old product versions
- Version control software

Copyright © Wyyzzk, Inc. 2004
Version 5.0 Enterprise Architecture - 13

When maintaining a product family, there need to be overall policies guiding bug fixes and versioning.

If a customer finds a bug in an old version of a product, and that bug is already fixed in later versions, do you fix the old version of the product, or do you require the customer to upgrade?

If a customer finds a bug in the current version of the product and you know it exists in all previous versions, do you fix it in all versions, do you send a patch to old customers, or do you only fix the bug in the current version of the software?

What about other products in the same product family? Do you look for the bug there and fix it in all versions of all products in the family?

Other issues arise around versioning of software. Most companies will only maintain the last few versions of software, if they maintain old versions at all. Microsoft still releases patches for Windows 2000, but I believe they no longer support Windows 95. There reaches a point where a company cannot afford to keep maintaining old versions of software.

Some customers may not be able to upgrade due to running on old hardware that will not support the new software, or perhaps they are using old software that requires a certain version of your software to interact with. Sometimes pricing is the issue. So requiring upgrades may cause you to lose customers.


One issue that will often come up is that customers who cannot or will not upgrade want you to provide them with the newest features. This is often infeasible due to lack of functionality in the old versions to support the new features.

All of these things should be kept in mind when architecting your product family. The software will be maintained, fixed, features added, new products created for a very long time, so maintenance is a big issue.

Especially if you have many products or many versions in your product family, you will need a good version control system, and probably also a good build engineer, to keep track of all the different versions and to make maintaining the product family possible.

A big mistake is not tracking the different versions. One company we know created a huge problem for themselves because they did not keep track of which software units were compiled into each version of the product. They had many versions, all customized for their (very large) customers. When a bug was reported, they had no way to recreate the problem, because they had no way to create the same version of the software that the customer had. This problem only grew worse over time.

Slide 14




Use Cases & other Requirements

- Shared between products in the family
- Specialized for different products
- Can be used to find common areas and variable areas of the software

Copyright © Wyzzyk, Inc. 2004
Version 5.0 Enterprise Architecture - 14

When you think of commonality and variability, consider more than just the software. There are often many use cases and requirements in common among products in a product family. You can also use the requirements to find or explore what parts of the software are common between products and which parts need to be specialized.

Slide 15




Documentation

- UML Package diagram
 - Show subsystems, packages, libraries
 - Show common and specialized elements
- UML Class diagram
 - As needed to show important elements of the different subsystems
- UML Use Case Diagram
 - Show use cases for product family and specialized use cases
- ER Diagrams
 - Show the data model
- MS Word
 - Use Case Specifications
 - Constraints, Regulatory Requirements, Guidelines

Copyright © Wyzzyk, Inc. 2004
Version 5.0 Enterprise Architecture - 15

Slide 16




Break

- As a class, review the given Product Family Architecture.

Copyright © Wyzzk, Inc. 2004
Version 5.0 Enterprise Architecture - 16

Slide 17



Program of Projects Architecture

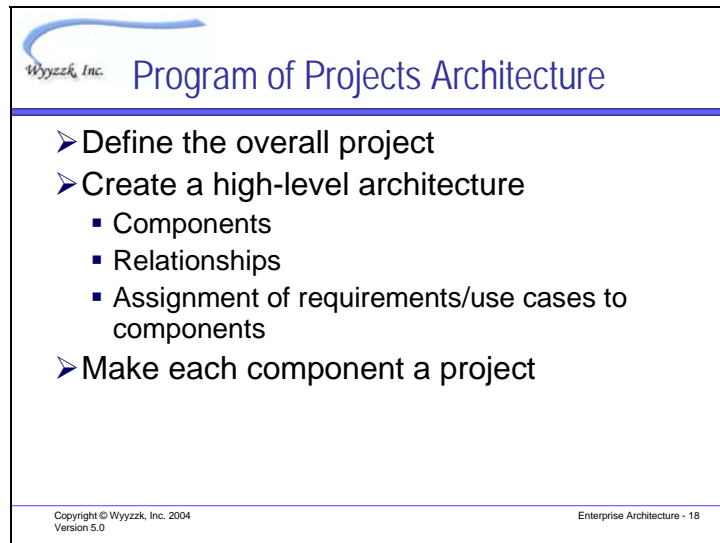
- This concerns the architecture of a set of applications that are related because they are parts of a much larger application
- This is as much an architecture of the management of the projects as it is software architecture
- Common in defense industry, becoming necessary in IT

Copyright © Wyzzk, Inc. 2004
Version 5.0 Enterprise Architecture - 17

I have talked to a number of people who know what a program of projects is, and we all agree on the definitions. I have yet to find any literature or books written on the subject. You often find these discussed by people from a defense industry background, since a program of projects is typically found in quite large applications.

The program I worked on at Lockheed Missiles and Space was a satellite communications project. The total project involved several hundred engineers and took 17 years to complete. We are starting to see very large projects (not quite this big, but still large) in IT, and few people with an IT background have any idea how to manage such a thing, either from the project management point of view or from the software architecture point of view.

Slide 18



Wyyzzk, Inc. Program of Projects Architecture

- Define the overall project
- Create a high-level architecture
 - Components
 - Relationships
 - Assignment of requirements/use cases to components
- Make each component a project

Copyright © Wyyzzk, Inc. 2004
Version 5.0 Enterprise Architecture - 18

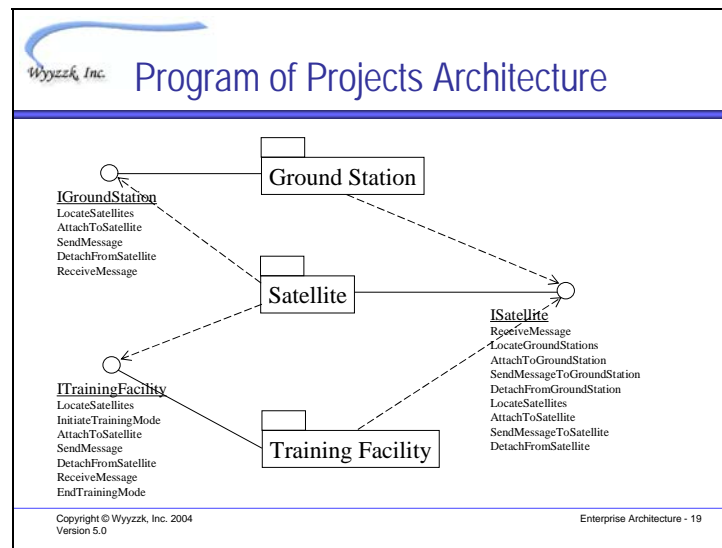
What you are doing is creating an architecture for something that is so big, it requires multiple project teams to implement. Imagine a project large enough to require 100 engineers. That is too many people for one project manager to manage. By dividing the project into multiple smaller projects, then each project may have 10-15 engineers under one project manager. There is still a program manager (or more likely a team of leaders) over all of the project managers.

Initially, the big project is treated as a single project, with a program manager, architect, business analyst, and test manager as the minimum staff. You may also have a deployment manager and some designer/developers on staff as well. All roles except the

program manager may actually be 2 or 3 people (i.e. 2 or 3 architects, 2 or 3 business analysts, etc.). The business analysts write the requirements for the whole program. These may not be very detailed at this time, but there has to be enough detail for an initial architecture to be described. The architects have to create the initial architecture, including the components, their relationships, and any constraints, regulations, guidelines, or patterns that are to be followed in the program overall. This is a similar effort to creating an enterprise architecture as was described in last week's lesson. The requirements are divided up and assigned to the different components of the system. This is to show which components implement which requirements.

Finally, each component becomes a project, with a project manager, business analyst, architect, test manager, and the rest of the project team. The project architecture, project requirements, and project schedule are parts of the overall program architecture, program requirements, and program schedule.

Slide 19

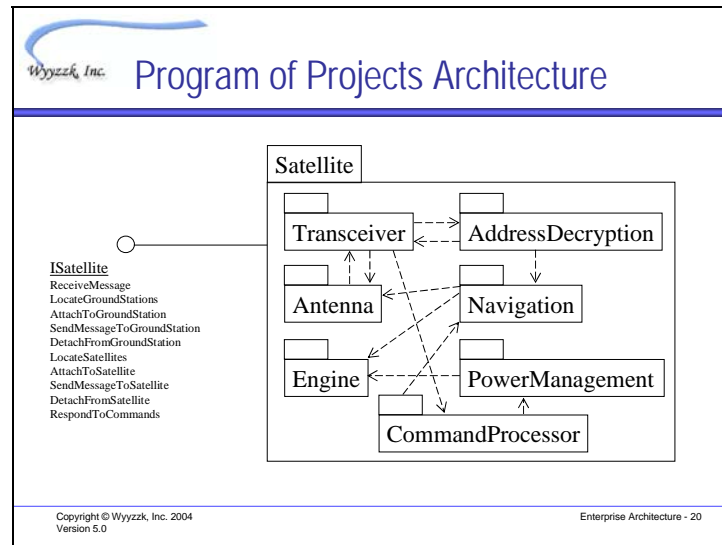


Send a Message:

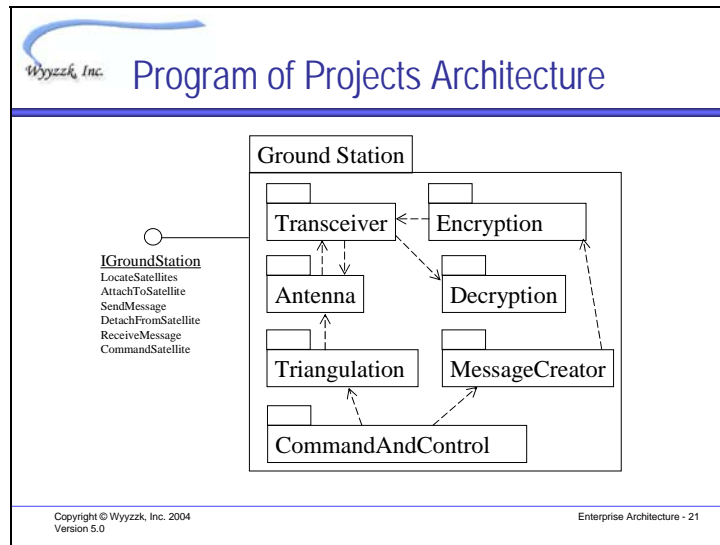
1. Ground station locates visible satellites
2. Ground station attaches to one of the satellites
3. Ground station sends a message to the satellite
4. Ground station detaches from the satellite
5. Satellite which received the message locates ground stations
6. If message is for one of those ground stations, satellite attaches to the ground station, sends the message, and detaches from the ground station.
7. If the message is not for one of those ground stations, satellite locates another satellite, attaches to the satellite, sends the message to the satellite, and detaches from the satellite. Repeat from step 5 until message is delivered to correct ground station or message returns to originating ground station.

Training Exercise:

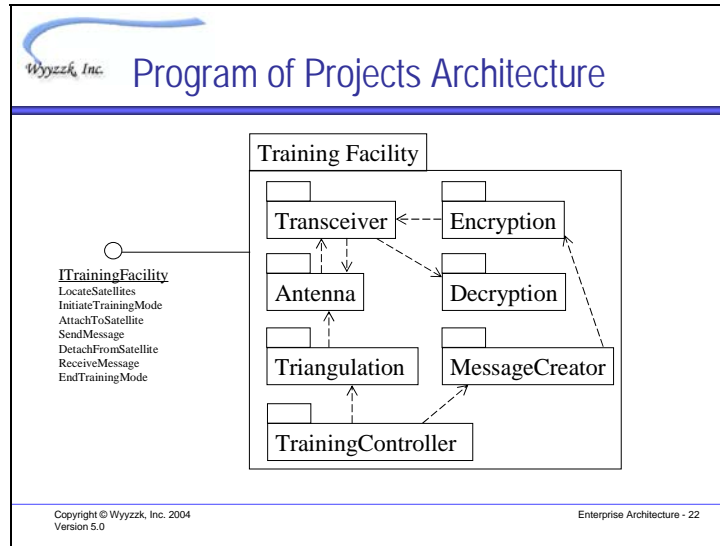
1. Training facility locates visible satellites
2. Training facility attaches to one of the satellites
3. Training facility initiates training mode
4. Training facility sends a message to the satellite
5. Training facility detaches from the satellite
6. Satellite attaches to the training facility.
7. Satellite sends the message to the training facility.
8. Satellite detaches from the training facility.
9. Training facility attaches to the satellite.
10. Training facility ends training mode.
11. Training facility detaches from the satellite.




The next couple of slides show a possible breakdown of each major component into sub-components. Each of these subcomponents (for example Transceiver or Navigation) is assigned to a whole project team, along with the requirements for that component. Also, the different teams will likely have to coordinate their architectures and schedules. For example, if the satellite has to do address decryption, then that team has to know the decryption process and the format of addresses. These are probably defined by the teams in the ground station segment, because in general the satellite is just passing on information and the ground stations do the real processing. So the ground station components get to define the formats of things like addresses, messages, and encryption/decryption algorithms. The program team are the people who determine which team is responsible for defining the various formats and algorithms.



Notice that encryption and decryption are different teams. Due to program security requirements, the people who know the encryption algorithm are not allowed to know the decryption algorithm. Also, no one on the encryption and decryption teams are allowed to work on or know about any other part of the system. (On the clearance side, this is implemented using compartmentalized clearances.)



This is deliberately similar to the ground station, since this is the training facility for the people who will be running the ground stations. The only component here that actually has a team is the training controller. The rest are the exact same components as the ground station, just re-used in the training facility component.




Data & File Format

- Coordinate shared information between components
- Typically one component (project team) is the “owner” of the format of information

Copyright © Wyzask, Inc. 2004
Version 5.0

Enterprise Architecture - 23

Within a program of projects, you may want to have shared data formats and file formats for the whole program. More commonly, some project team responsible for a component defines the data format or file format, and other project teams that use that data or file construct their code to use the defined format. The owner of the format of the information is typically assigned by the program team.



Shared libraries


- Shared libraries can be a good thing on a program
- Must be developed early, thoroughly tested, then NOT CHANGED
 - Changes propagate throughout the whole program

Copyright © Wyzzik, Inc. 2004
Version 5.0

Enterprise Architecture - 24

Shared libraries, same as in product families, can be a very good thing. After all, how many copies of a leap year function or date converter do you need? On the other hand, since these libraries are used by all project teams, they must be stabilized quite early in the program lifecycle. Additions to the libraries cause no harm, but changes to existing library elements will impact every project team on the program, so should be avoided.

Slide 25




Use Cases & other Requirements

- Divided among components (project teams)
- Requirement changes may need to be negotiated between project teams

Copyright © Wyzzyk, Inc. 2004
Version 5.0 Enterprise Architecture - 25

Slide 26




Documentation

- UML Package diagram
 - Show subsystems, packages, libraries
 - Show common and specialized elements
- UML Class diagram
 - As needed to show important elements of the different subsystems
- UML Use Case Diagram
 - Show use cases for program and for each component
- ER Diagrams
 - Show the data model
- MS Word
 - Use Case Specifications
 - Constraints, Regulatory Requirements, Guidelines

Copyright © Wyzzyk, Inc. 2004
Version 5.0 Enterprise Architecture - 26

Slide 27




Wyzzyk, Inc. Break

➤ As a class, review the given Program Architecture.

Copyright © Wyzzyk, Inc. 2004
Version 5.0 Enterprise Architecture - 27

Slide 28




Wyzzyk, Inc. Constraints and Issues

- Regulatory
 - Sarbanes/Oxley
 - HIPPA
 - Privacy laws
- Security
 - Limited access – physical, electronic
 - Encryption – data, communications
- Policy
 - Required performance
 - Required uptime
 - Fault tolerance
 - Budget for new hardware or software
 - Technical support of systems
 - Centralized vs distributed
 - Black-out periods for new releases

Copyright © Wyzzyk, Inc. 2004
Version 5.0 Enterprise Architecture - 28

These look just like the enterprise architecture! No surprise. The same kinds of things we define at the enterprise level, might be instead or additionally defined at the product family or program level.

Slide 29




Constraints and Issues

- Licensing
 - Servers
 - Usage compliance
- Maintenance
 - Status monitoring
 - Mirroring & backups
 - Scheduled down-time for hardware/software updates
 - Run-time updates
 - Allocation of maintenance costs
- Technical Support
 - User queries
 - Problem resolution (possibly over many systems)

Copyright © Wyzzk, Inc. 2004
Version 5.0 Enterprise Architecture - 29

Slide 30




Program or Product Family Group

- Responsible for setting, documenting, and enforcing all program or product family level policies for projects within the program or product family
- Oversee project teams to verify that project architecture does not violate program or product family architecture
- Determine when program or product family architecture needs to change and how to change it
- Oversee all changes to program or product family architecture

Copyright © Wyzzk, Inc. 2004
Version 5.0 Enterprise Architecture - 30

Slide 31




Impact on project architecture

- Requirements of program or product family architecture are also project requirements
- Need to negotiate problem areas with other project teams within a product family or program, in addition to negotiating problems with Enterprise level groups
 - Corporate Security
 - Technical support (for users)
 - Maintenance group
 - Accounting
 - Regulatory groups
 - Enterprise architecture group
 - Database architecture group

Copyright © Wyzzk, Inc. 2004
Version 5.0 Enterprise Architecture - 31

Some of the impact of product family and program architectures is on the architecture of your project. Your project has to technically fit into the overall architecture defined for the product family or program. Other impacts will be on scheduling of work. Your project team may have to complete some work by a particular point in time for another project team to be able to use it. Project teams in this kind of environment are less independent than they may be in companies without product families or programs of projects.

Slide 32




Impact on Project Architecture

- Patterns selected at the program or product family level have to be followed at the project level
 - For example, if the program uses SOA, then your project will be designed around services
- Constraints and regulatory requirements set at the program or product family level have to be followed at the project level
 - For example, privacy laws require personal information to be encrypted. If your project in any way uses personal information, you will have to deal with decryption and encryption, and possibly only certain people on your team will have access to that data.

Copyright © Wyzzk, Inc. 2004
Version 5.0 Enterprise Architecture - 32

Slide 33




Impact on Project Architecture

- Because of the need to have more people involved with your project, your schedule will be longer
 - Other project teams, program or product family groups, Enterprise architects, security people, regulatory agencies, and so on
- In the defense industry, you can add the need for clearances to the process
 - It is possible that team members have different clearances and need to know, and that you will have to put processes in place to ensure everyone has the appropriate access

Copyright © Wyzzyk, Inc. 2004
Version 5.0 Enterprise Architecture - 33

Slide 34



Consequences

- Perception of too much overhead
- Lack of these structures generally leads to project failure
 - Too many meetings or too few
 - Too many people making decisions or too few
 - Lack of understanding of complete project

Copyright © Wyzzyk, Inc. 2004
Version 5.0 Enterprise Architecture - 34


Many companies will not consider using a product family or program architecture because they believe it is too much overhead. Too many bosses, too much work on models and requirements, not enough work on code. These arguments come from IT, from people without really large project experience. You will not hear these arguments

from large defense projects, because they have a lot of experience with really large projects and know that without this structure, the project is most likely to fail.

Without the structure of a product line or program architecture (and the related project management changes), too much time is spent by teams working at cross-purposes, or to prevent that, too much time is spent in meetings. Too many people are involved in decisions (rather than just the program team), so decisions take a lot more time to make. Or not enough people are involved in the decisions, and global policies are decided at a project level, which may not be the best for the overall project.

Because no one is responsible at a program or product line level, there is generally a lack of understanding of the overall project, vision, or direction. Work is repeated across multiple projects, and contradictory work is performed that has to be fixed later when the individual projects try to integrate.

Slide 35



Summary

- A program or product family architecture concerns applications that are related because they are part of a program or because they are variations on the same product
- In both architectures, consider what is common among the various projects and what is different when creating the architecture
- Base the architecture on the kind of anticipated changes in the program or product family
- Constraints, issues, and policies set at the program or product family level will effect your project architecture
- You may find that many groups will be involved in your project. For example, enterprise architecture, corporate security, regulatory agencies, technical support, maintenance, and database architecture, as well as program or product family group and other project teams within the same program or product family.

Copyright © Wyzzyk, Inc. 2004
Version 5.0

Enterprise Architecture - 35