PHILIPS

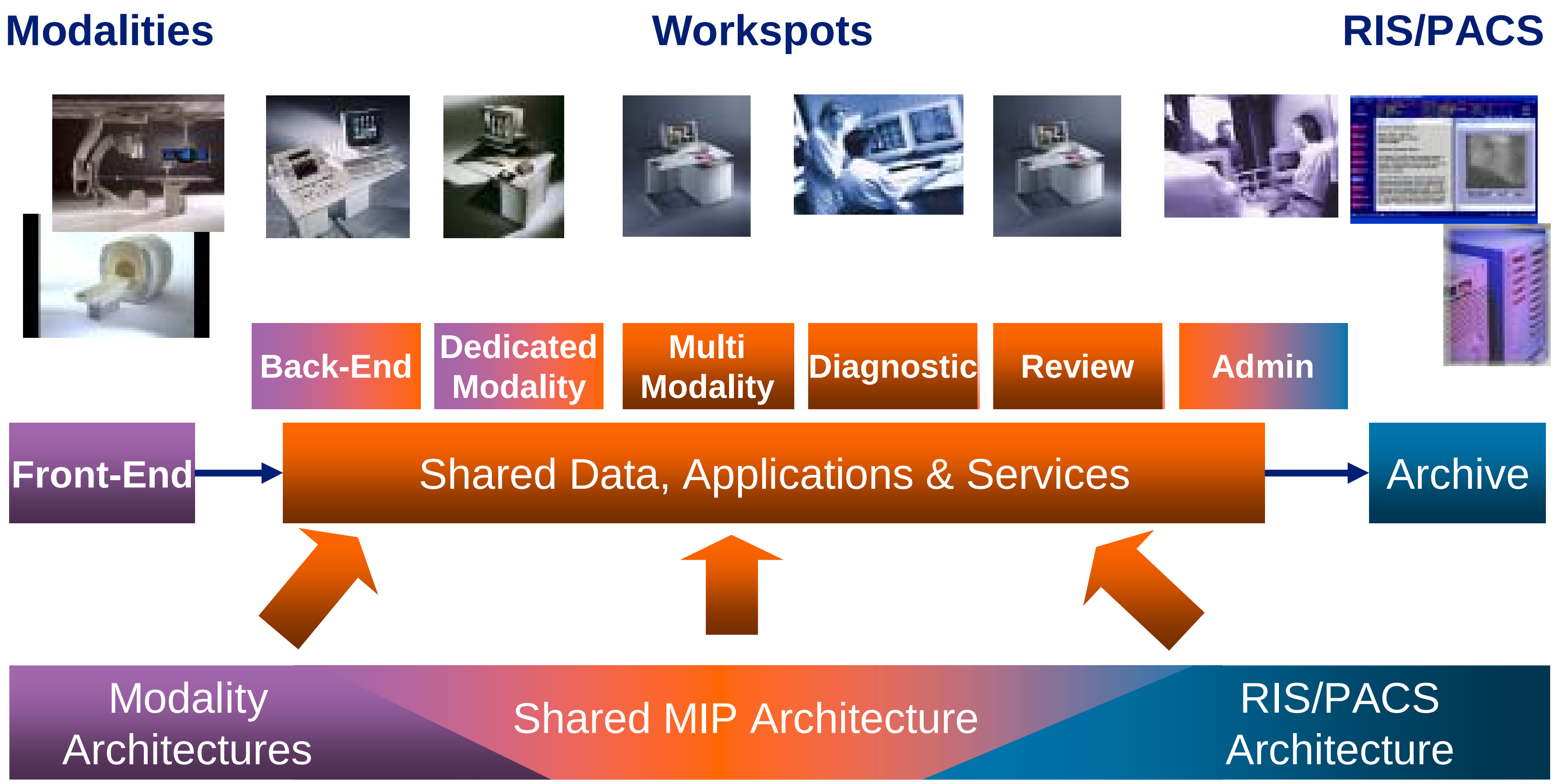
Architecture Deployment & Verification Compile- and run-time dependencies

Rick Everaerts, Philips Medical Systems, Medical IT Best

23th Systems/software Architecture Study Group Meeting

Tuesday February 1st, 2005

PHILIPS


Modalities
Workspots
RIS/PACS
Back-End
Dedicated Modality
Multi Modality
Diagnostic
Review
Admin
Front-End
Shared Data, Applications & Services
Archive
Modality Architectures
Shared MIP Architecture
RIS/PACS Architecture

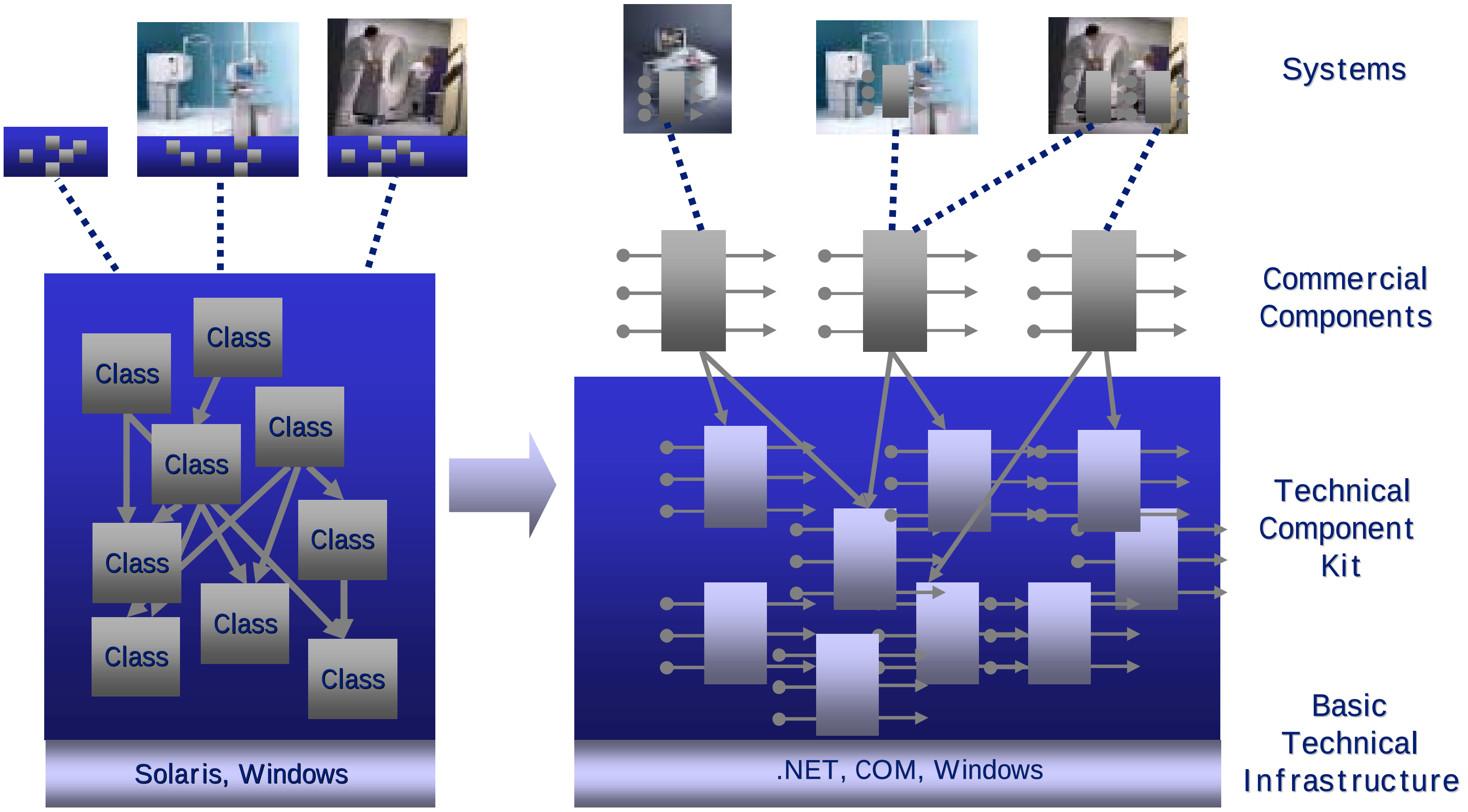
PHILIPS

Architecture Deployment & Verification
Compile- and run-time dependencies

- Introduction
  - Interfaces, Information Models, Components
- Deployment
  - Compile-time dependencies
  - Run-time dependencies
- Verification
  - Compile-time dependencies
  - Run-time dependencies
- Discussion Points

PHILIPS

Introduction: from Objects to Components



Introduction: Interfaces, Info Models & Components

- Interfaces:
  - Define the syntax
  - Generic access points to functionality
- Info Models:
  - Define the semantics
  - Allow for specialisation of the interfaces
  - Are means of variability
- Components:
  - Are the actual implementation
  - Unit of composition of interfaces
  - Define the set of capabilities

PHILIPS

Introduction: Interfaces Design

- Provides Interface:
  - Component guarantees to implement the functionality associated with the interface
- Requires Interface:
  - Component accesses functionality through this interface and relies on the functionality to be implemented *outside* the component
- Optional/Mandatory interfaces

PHILIPS

Introduction: Interfaces Design Rules

- Think in interfaces not in implementations
- Make as few assumptions as possible about the execution environment of a component

Component creator should model all pieces of functionality that have to be

‘overruled’ by the component user

through ‘requires’ interfaces

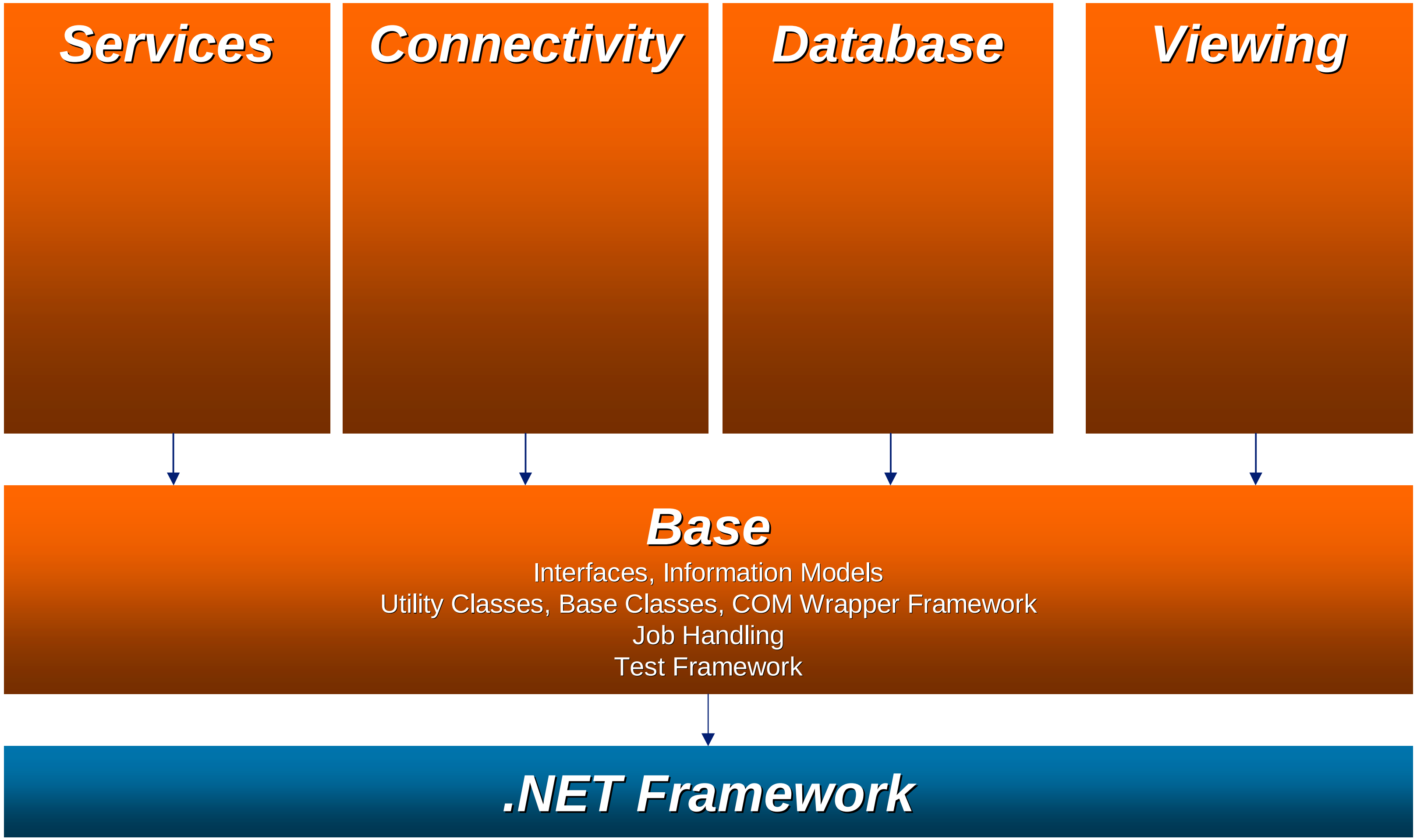
PHILIPS

Introduction: Information Models

- Explicitly managed and defined in UML
  - Team of architects with weekly meetings
- Implemented as generic Data Objects
- Described by a Data Dictionary
- Easy streamable to/from XML
- Examples:
  - PMS Imaging Information Model (PIIM)
  - Configuration Information Model (CIM)
  - Performer Information Model (PIM)

PHILIPS

Deployment: compile-time dependencies


Services
Connectivity
Database
Viewing
Base
Interfaces, Information Models
Utility Classes, Base Classes, COM Wrapper Framework
Job Handling
Test Framework
.NET Framework

PHILIPS

Deployment: compile-time dependencies

- Split the source code archive into:
  - Public and Private (external/internal)
  - Interfaces, Information Models, Utilities (allowed to be used directly)
  - Segments (ownership/functionality)
  - Delivery code and Test and Tools

- Namespace reflects the directory structure (1:1)

```
Philips.PmsMip.Public.Interfaces
                     .InformationModels
                     .Utilities.<segment>
Philips.PmsMip.Private.Interfaces
                      .InformationModels
                      .<segment>
                      .Tests.<segment>
                      .Tools.<segment>
```

PHILIPS

Deployment: compile-time dependencies

- Deliverable libraries contain namespace prefix
  - E.g. Philips.PmsMip.Public.Interfaces.dll
- Base segment owns/manages Public/Private:
  - Interfaces
  - InformationModels
  - Utilities
- Upwards compatibility of Public Interfaces/Information Models
  - Recompilation old application should be enough

PHILIPS

Deployment: run-time dependencies

- Broker instantiates classes via reflection
  - Aliases and classes mapping are defined in XML format
  - Mapping of classes to dll’s are generated as part of Build
  - Classes instantiated via Broker are cached

- Example of Broker usage:

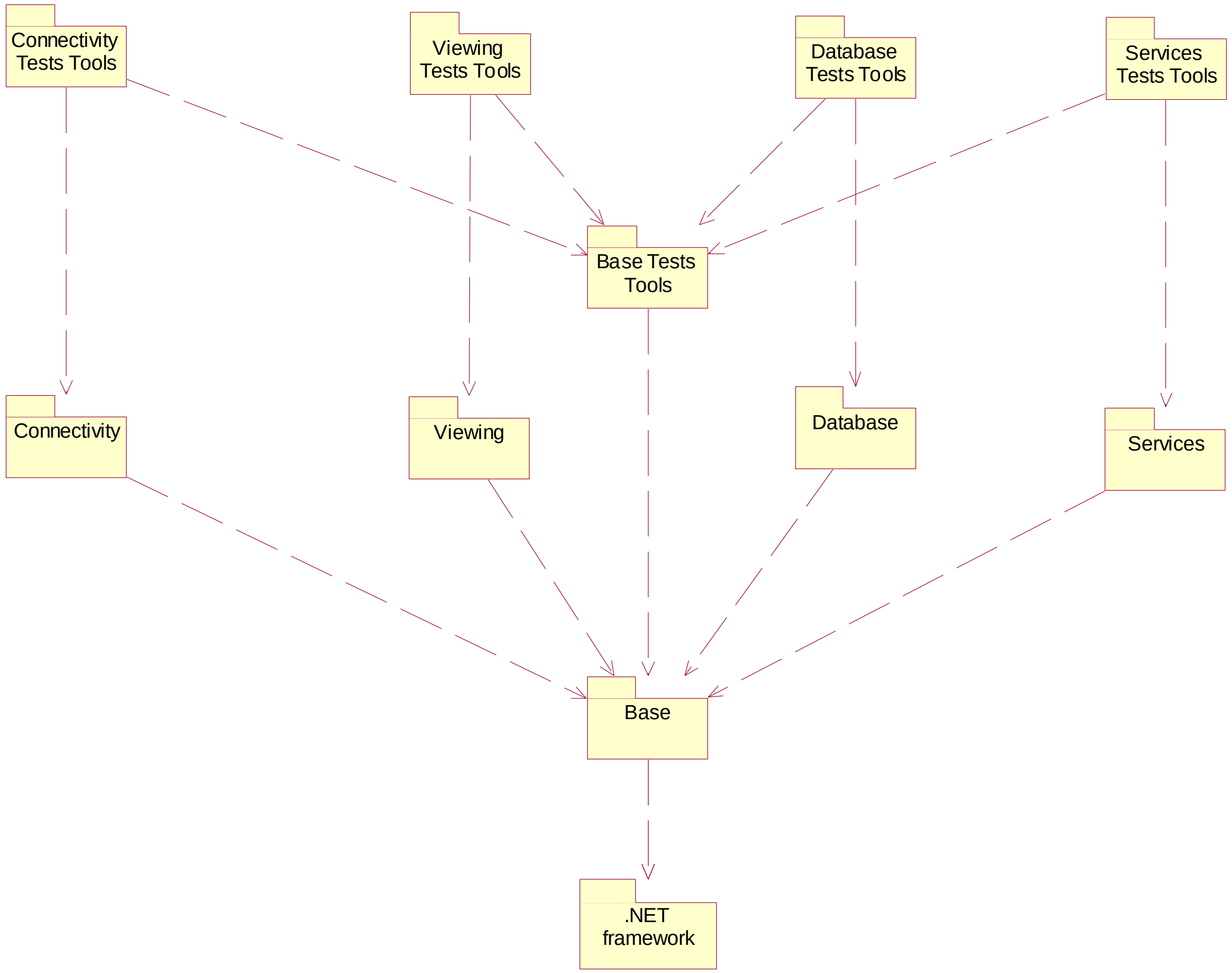
```
string context = "Public";
IBroker broker = Broker.GetBroker(context);
string alias = "ColorManager";
IColorManager colorManager = (IColorManager)
   broker.CreateInstance(alias);
```

Deployment: dependency rules

1. Program by interfaces
   - Avoids compile time dependencies
   - Makes it possible to replace an implementation by another one
   - Shields public methods not defined in interface
   - Never change an interface once released
2. No compile time dependencies between segments, except to Base segment
3. No run-time dependencies between segments, except to Base segment or specific identified via Broker
4. No compile-time dependencies from Delivery code to Test and Tools allowed
5. No runtime dependencies from Delivery code to Test and Tools allowed
6. No compile time dependencies from Public to Private packages

PHILIPS

Allowed compile-time dependencies between segments



PHILIPS

Verification: compile-time dependencies

- Build order of Visual Studio projects is fixed (list of solutions)
  1. Base delivery code
  2. Per other segment the delivery code
  3. Tools/Test code Base segment
  4. Tools/Test code per other segment
- Dependency check is done on the Visual Studio projects
  - Check the references (dll) in Visual Studio projects
  - Possibility to specify segment: namespace/dll mapping

```
# Private base
Base:Philips.PmsMip.Private.Base
Base:Philips.PmsMip.Private.InformationModels
Base:Philips.PmsMip.Private.Interfaces
Base:Philips.PmsMip.Private.Utilities
```

- Build will fail for any violation, including Coding Standard
  - Check on namespace and class file directory structure

PHILIPS

Verification: run-time dependencies
Automatic regression testing

- Use Test framework
- Set-up Test Environment as close as possible to the real situation
- Test against Mandatory and Optional interfaces
- Vary the test data
- Test all flexible aspects of components
- Always keep the software working
- Test alternative usage's of components
- Keep improving (automated) test cases over time
- Measure code coverage and improve if too low
- Make reference implementations or stubs
- Refactor the (automated) test cases when needed
- Investigate why a bug was not found and take appropriate actions
- Also test non-functional requirements (memory, CPU, etc)

PHILIPS

Discussion points

- Is namespace and directory 1:1 mapping really needed for compile-time dependency check?
- How can you make sure you do not need tools/tests at runtime, if you use automatic regression tests for testing the product code?
- Can run-time dependencies be checked statically?
- Upwards compatible without recompilation old application?

PHILIPS