Viewpoints: Development, Deployment, and Operational

CSCE 742 - Lecture 15 - 16 - 10/30 and 11/01/2018

The Viewpoints

- The Context, Functional, Information, and Concurrency Views define *what* you are building.
    - The static and runtime structure of the system.
- The Development, Deployment, and Operational Views define *how* you will build the system.

The Viewpoints

- This class and next…
- The **Development View** defines how to implement the system.
- The **Deployment View** defines how to transition the system to live operation.
- The **Operational View** defines how to keep the system alive in the field.

Development Viewpoint

Designing Development Environments

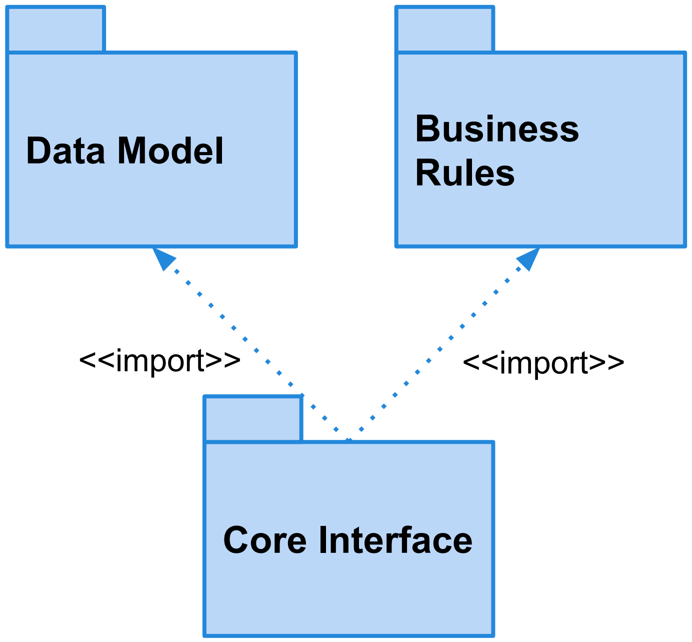
- Supporting design, development, and testing of complex systems requires the right environment.
  - Code structure, dependencies, build management, design constraints, design standards.
- The **Development View** addresses the concerns of developers and testers.
  - All software projects involve some amount of new code being written.
  - This view provides a stable environment for more detailed design work.

Concern: Structure Organization

- Software is often organized into groups of related classes or functions.
  - Some languages have built-in support for this: Packages in Java, Namespaces in C#.
  - We will refer to them generically as **packages.**
- Packages are groupings of functionality
  - Classes in Java, groups of functions in C.
  - Packages are *not* functional elements.
  - Elements may contain packages (for organizing source code).
  - Packages may depend on component interfaces.

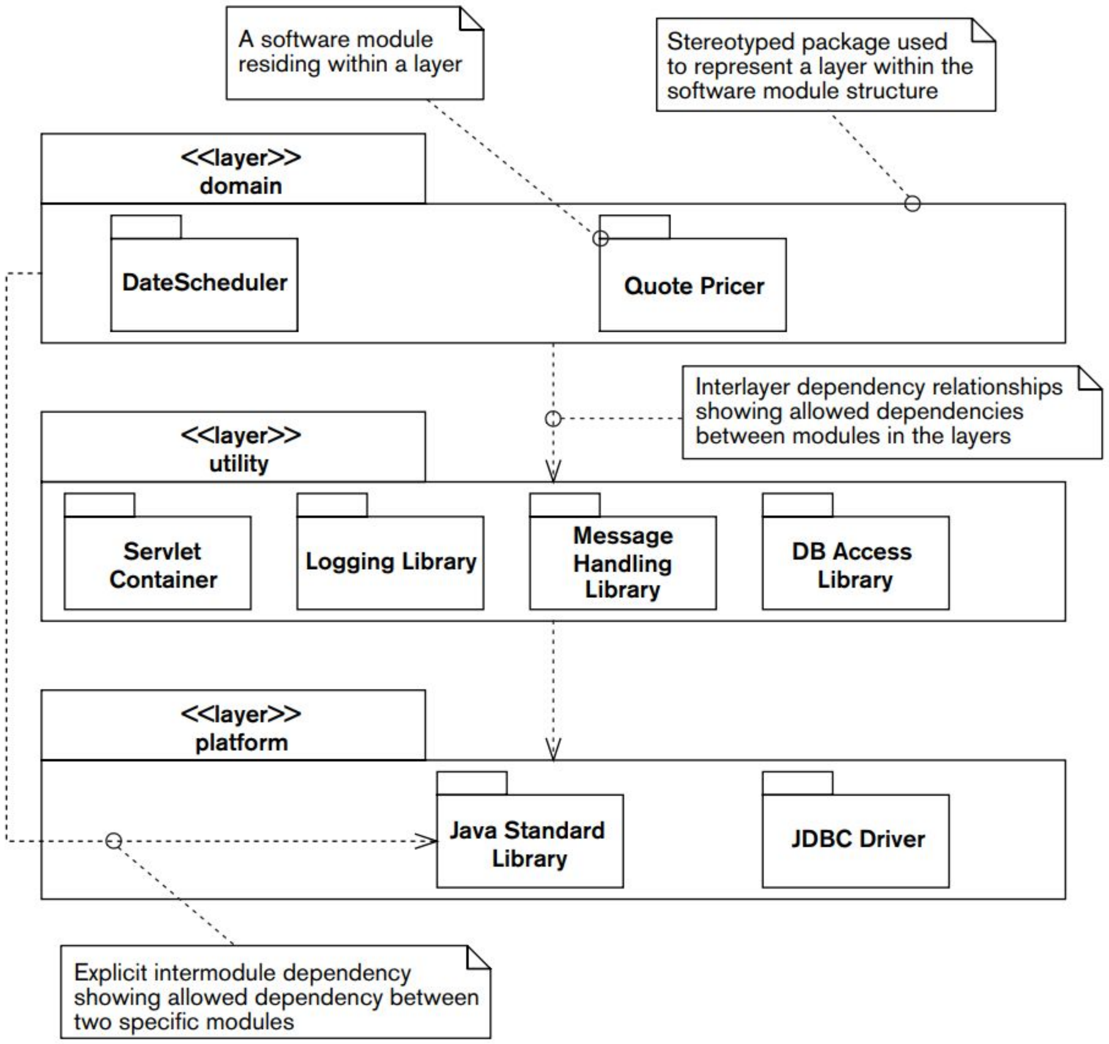
Package Diagrams

Packages can be specified in design phase using UML Package Diagrams



- Folders indicate packages.
- Classes are often listed inside the folder (omitted to save space).
- Arrows indicate dependencies.
- Can be annotated with <<import>> (one package imports from another) or <<merge>> (a package is composed of multiple subpackages).
- Unlabeled arrow can be interpreted as an <<import>>

Layered Package Diagrams


A software module residing within a layer
Stereotyped package used to represent a layer within the software module structure
<<layer>> domain
DateScheduler
Quote Pricer
Interlayer dependency relationships showing allowed dependencies between modules in the layers
<<layer>> utility
Servlet Container
Logging Library
Message Handling Library
DB Access Library
<<layer>> platform
Java Standard Library
JDBC Driver
Explicit intermodule dependency showing allowed dependency between two specific modules

Package Design Activities

- Identify and classify the packages
  - Group source code files or packages (if they exist) or logical element subdivisions into packages.
  - Group package when it makes sense.
- Identify the package dependencies
  - Identifies impact of making changes.
- Identify layering rules
  - Can packages call packages only in their layer and one above/below, or throughout structure?
  - How do you preserve performance and flexibility?

Package Cohesion Principles

- Packages are a **source code management** and a **release management** idea.
  - Source management: grouping related classes.
  - Release management: for use, packages are often distributed as libraries
    - Jar files in Java; Assemblies in C#, .lib / .dll / .a files for C code.
- Package design follows principles:
  - Reuse-release equivalence principle
  - Common reuse principle
  - Common closure principle
  - Acyclic dependencies principle

Reuse-Release Equivalence Principle

- Granule of reuse is the granule of release.
    - To reuse code, it must arrive in a complete, black-box, package that can be used but not changed.
- Packages should be tracked using change-control system.
- Package should be understood in terms of public functions / classes / interfaces.
    - No need to look at all the source code.
- Each package is treated like a product.

Common Reuse Principle

- Improper grouping of classes creates unwanted dependencies.
- Classes that tend to be reused together belong in the same package.
    - If not, then perhaps they should be in separate packages.
- The classes in a package are reused together. If you reuse one of the classes, you reuse them all.
    - All classes in the package should be reusable in the same context.

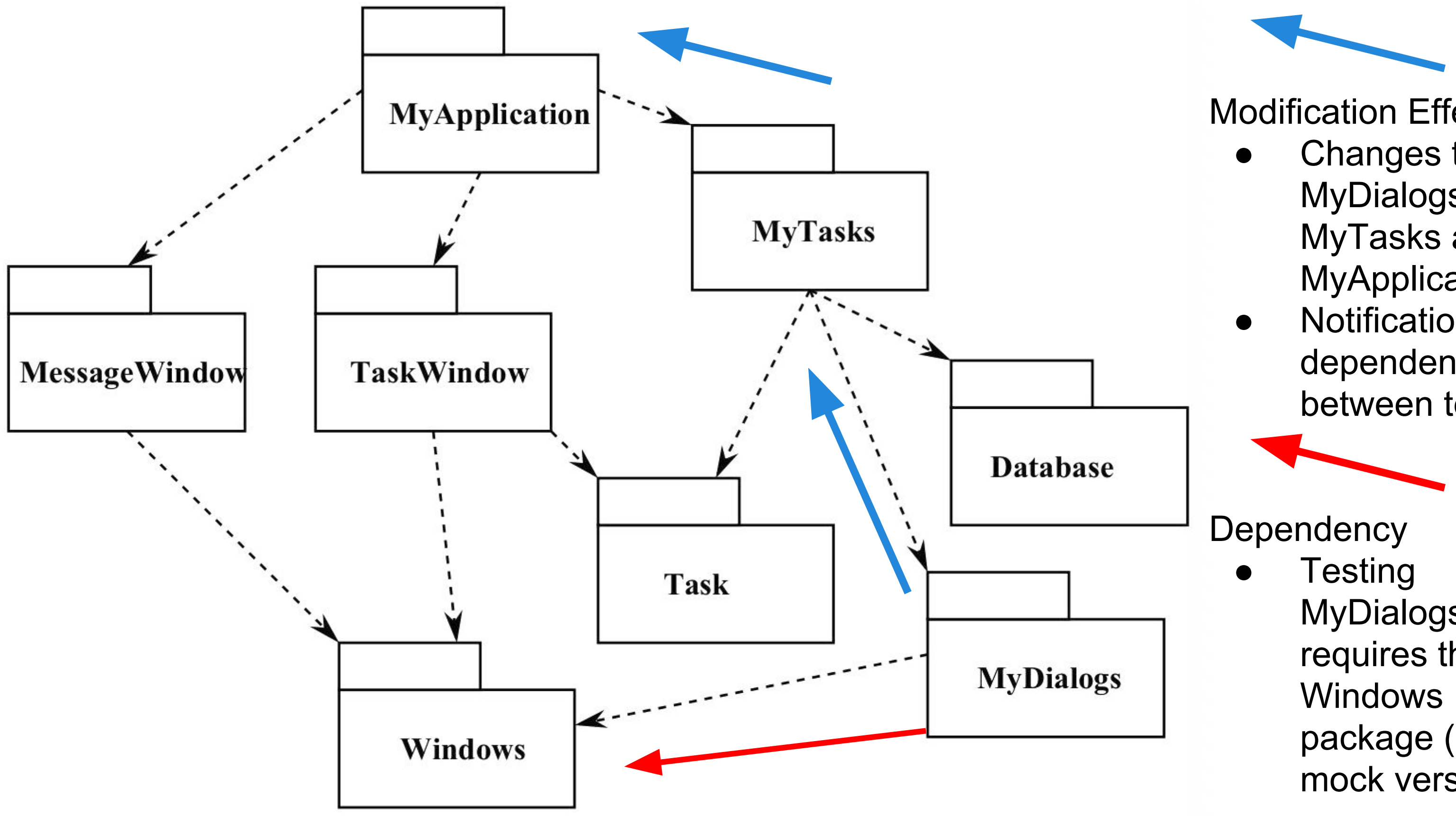
Common Closure Principle

- If one class needs to be changed, they all are likely to need to be changed.
- Conversely, all classes within a package are **closed** to the same kinds of changes.
- Helps pull tightly-coupled classes together in one package.
  - To enable easy distribution, updates, release, maintainability, localize all changes to a package.
  - A change will affect a minimal number of packages.

Acyclic Dependencies Principle

- The dependency graph between packages must be acyclic.
  - That is, if package A depends on package B, then B must not depend on A.
- Packages are units of work and reuse.
  - Versioned; clients can decide when to upgrade.
  - Changes to one package should not require an immediate update by other teams.
- Graph makes the dependencies of packages explicit.
  - Cycles in the graph would break versioning.

Acyclic Example (Good)



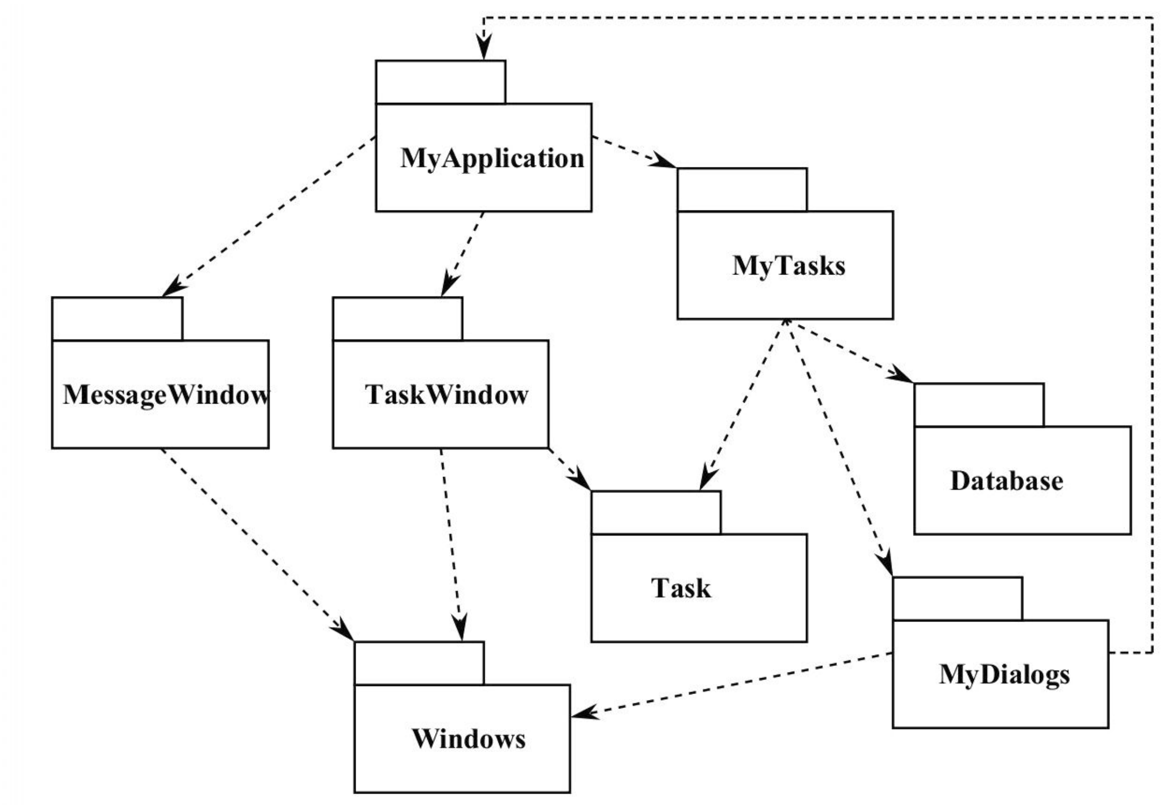
Modification Effect:

- Changes to MyDialogs affect MyTasks and MyApplication.
- Notification dependency between teams.

Dependency

- Testing MyDialogs requires the Windows package (or a mock version).

Cyclic Example (Bad)



Release:

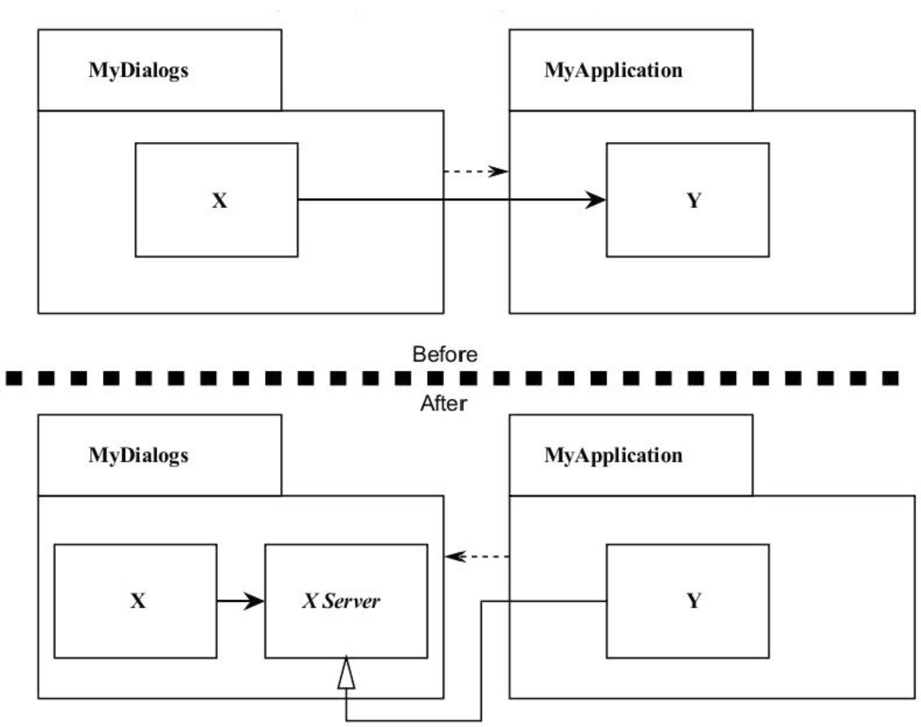
- Must be simultaneous with MyApplication.
- But this means MyTasks must also be coordinated (it is a dependency of MyApplication and depends on MyDialogs).
- This means that it must also be coordinated with Task and Database (dependencies of MyTasks).

Testing:

- MyDialogs requires MyApplication, so…
- **MyDialogs is dependent on all packages(!) for testing!**

Fixing Circular Dependencies

- Apply the Dependency Inversion Principle.
  - Create an abstract class with the interface MyDialogs needs.
  - Put the class into MyDialogs
  - Inherit into MyApplication.
  - Reverses the dependency, breaking the cycle.



Package Refactoring

- Can also create third package with class(es) that both MyApplication and MyDialog depend on.
- Package contents and dependency hierarchy must be actively managed and refactored.
    - Dependencies will change as system expands.
    - Circular dependencies must be pruned out.
    - Coordinating this movement is important job for architect.

Concern: Standardizing Common Processing

- Isolate common processing into separate code packages.
    - Logging in a single package.
    - Error Management in a single package.
    - ...
- Identify and specify areas of common processing.
- Define design guidelines, patterns, and packages for these features

Establishing Design Constraints

- Establish design principles and constraints.
    - Reduces risk and effort duplication by defining a standard approach to problem solving.
    - Commonality in system elements increases overall technical coherence and makes it easier to understand, maintain, and use.
- Requires definition of:
    - Common processing
    - Standard design approaches
    - Common external software elements

Common Processing

- How do we perform…
  - Initialization and recover
  - Termination and restart
  - Message logging and instrumentation
  - Internationalization
  - Processing configuration parameters
  - Security (authentication, encryption)
- These aspects of element design benefit greatly from using a common approach.
  - Even if we can’t just define a single package containing this type of processing.

Example: Message Logging

- All components must log human-readable messages that clearly state what has occurred and any corrective action that is expected in response.
- Messages must be logged at one of the following levels: Fatal, Error, Warning, Information, Debug.
- Elements should log messages at all five possible logging levels.
- Logging should be achieved via a standard library (as defined later) to standardize destination, format, configuration, and so on.

Design Constraints

- Define standard design approaches.
  - Look for situations where common processing is performed orr where implementation of an element has system-wide impact.
  - Define the approach, where it will be used, and why it should be used.
  - Design patterns - recipes for solving design problems.
- Define common external elements
  - Identify external software that can be reused to save development time or effort.
  - Identify where and how it will be used.

Example: Standard Design for Internationalization

- For internationalization of locale-sensitive resources, use an external resource catalog to store resources outside the source code files.
  - This means that all strings must be extracted from a message catalog before they can be used in a program (e.g., to write a log message).
- As the server software is being written entirely in Java, the internationalization implementation will use the Java Platform's native internationalization facilities: the resource bundle, the formatting classes in the java.text package, and the Locale class.
- The relationships between these different elements of the internationalization technology are as follows. [. . .]
- [You would place a definition of a design pattern for using the Java internationalization facilities here.]

Example: Standard External Elements for Logging

- All message logging must be performed using the standard CCJLog package, which is part of the standard build environment.
- The CCJLog package must be used in a standard way, which is documented as a code sample in the `src/server/sample/logging/CCJLog` source directory.

Concern:
Design and Testing Standardization

- Design
    - Software is developed by teams.
    - Try to ensure common approaches to architectural design, class design, use of patterns, and interface design.
    - Establish clear guiding principles, based on quality.

- Testing
    - Define common approaches, tool use, and testing conventions to ensure consistent testing efforts.
    - Requires reasonable tool chain and workflow, standard test data, and automation.

Concern: Codeline Organization

- Code must be stored in directories, managed in source control, built and tested regularly, and released into production.
- Managed through the **codeline structure**.
  - Source control files with a well-defined structure.
  - Stored in version management.
  - With a associated automated system to build, test, and release the system.
  - Should be defined as part of the development view.

Codeline Organization

Codeline organization captures:

- How code will be organized into source files.
- How the files will be grouped into modules.
- What directory structure will be used to hold the files.
- How the source will be automatically built and tested.
- What type of tests will be run and when they are run.
- How the binaries will be released into a test or production environment for testing and use.
- How the source will be controlled using configuration management to coordinate multiple developers.
- What automated tools will be used for the build, test, and release process and how they will work together.

For any system of substantial size, continuous integration should be the goal
Every time a commit occurs on stable or development branch
System is automatically rebuilt
Tests are run in increasing scope: unit / subsystem / system
Failures are reported to interested stakeholders
Goal: always have a shippable product.

Continuous Integration

- Development practice that requires code be frequently checked into a shared repository.
- Each check-in is then verified by an automated build.
  - The system is compiled and subjected to an automated test suite, then packaged into a new executable.
- By integrating regularly, developers can detect errors quickly, and locate them more easily.
- Goal: **always have a shippable product!**

CI Practices

- Maintain a code repository.
- Automate the build.
- Make the build self-testing.
- Every commit should be built.
- Keep the build fast.
- Test in a clone of the production environment.
- Make it easy to get the latest executable.
- Everyone can see build results.
- Automate deployment.

How Integration is Performed

- Developers check out code to their machine.
- Changes are committed to the repository.
- The CI server:
  - Monitors the repository and checks out changes when they occur.
  - Builds the system and runs unit/integration tests.
  - Releases deployable artefacts for testing.
  - Assigns a build label to the version of the code.
  - Informs the team of the successful build.

How Integration is Performed

- If the build or tests fail, the CI server alerts the team.
    - The team fixes the issue at the earliest opportunity.
    - Developers are expected not to check in code they know is broken.
    - Developers are expected to write and run tests on all code before checking it in.
    - No one is allowed to check in while a build is broken.
- Continue to continually integrate and test throughout the project.

Common Approaches

- Build environment
  - Machine configuration can be problematic
  - https://travis-ci.org/
  - Build from bare-bones linux platform
  - Ensures build works on clean install
- Deployment as Virtual Machine
  - Application is packaged up as part of an image
  - Can easily be "spun up" on many machines
- Deployment as container
  - https://www.docker.com/
  - Like a "lightweight VM"
  - Allows isolated "microcomponents" to be easily deployed.

Food for Thought

- Have you defined a clear strategy for organizing the source code packages?
- Have you defined a set of rules governing the dependencies that can exist between packages at different abstraction levels?
- Have you identified all of the aspects of element implementation that need to be standardized across the system?
- Have you clearly defined how any standard processing should be performed?

Food for Thought

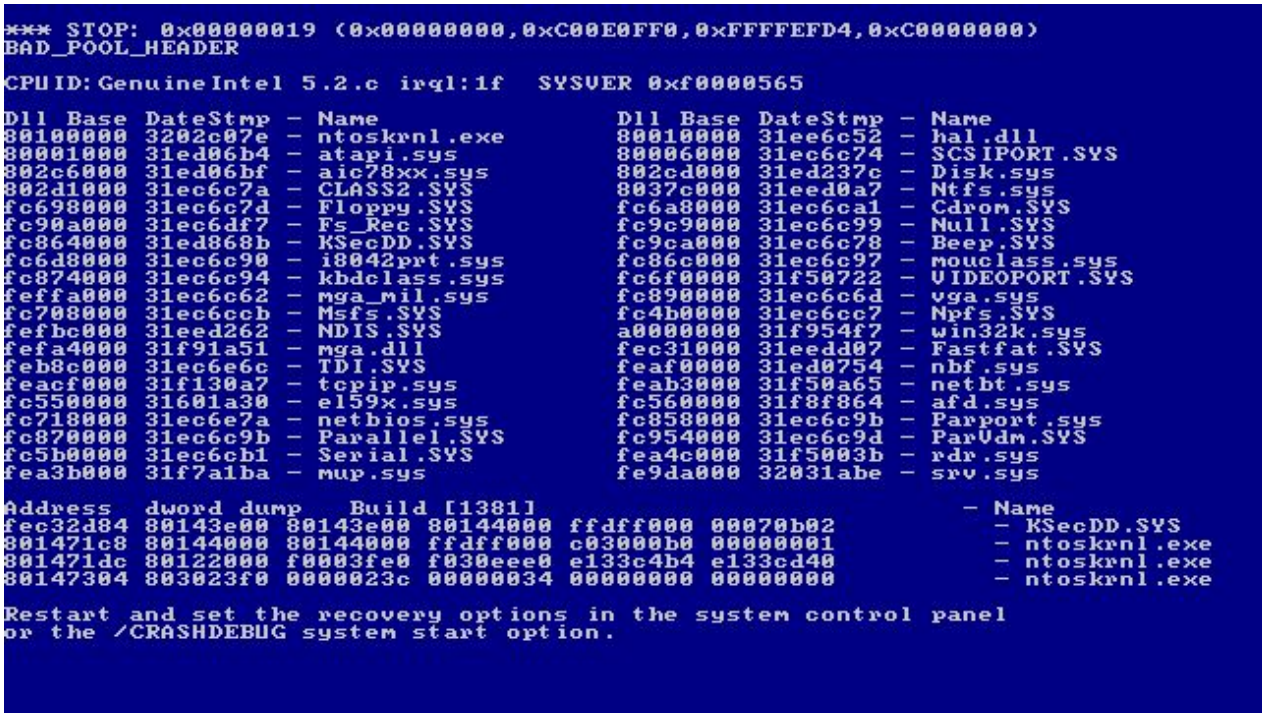
- Have you identified standard approaches to design that you need all element designers and implementers to follow?
    - If so, do your software developers accept and understand these approaches?
- Will a clear set of standard third-party software elements be used across all element implementations?
    - Have you defined the way they should be used?

Food for Thought

- Will the development and test environments work reliably and be usable and efficient for developers and testers to work in?
- Have you defined a set of tools to automate the build, integration, test, and release processes?
  - Does the set of tools include internal or third-party tools that you require to deploy to the test and production environments that you are using?

Deployment Viewpoint

… But it worked when I tested it?!?!



```
*** STOP: 0x00000019 (0x00000000,0xC00E0FF0,0xFFFFEFD4,0xC0000000)
BAD_POOL_HEADER

CPUID:GenuineIntel 5.2.c irql:1f  SYSVER 0xf0000565

Dll Base DateStmp - Name                    Dll Base DateStmp - Name
80100000 3202c07e - ntoskrnl.exe            80010000 31ee6c52 - hal.dll
80001000 31ed06b4 - atapi.sys               80006000 31ec6c74 - SCSIPORT.SYS
802c6000 31ed06bf - aic78xx.sys             802cd000 31ed237c - Disk.sys
802d1000 31ec6c7a - CLASS2.SYS              8037c000 31eed0a7 - Ntfs.sys
fc698000 31ec6c7d - Floppy.SYS              fc6a8000 31ec6ca1 - Cdrom.SYS
fc90a000 31ec6df7 - Fs_Rec.SYS              fc9c9000 31ec6c99 - Null.SYS
fc864000 31ed868b - KSecDD.SYS              fc9ca000 31ec6c78 - Beep.SYS
fc6d8000 31ec6c90 - i8042prt.sys            fc86c000 31ec6c97 - mouclass.sys
fc874000 31ec6c94 - kbdclass.sys            fc6f0000 31f50722 - VIDEOPORT.SYS
feffa000 31ec6c62 - mga_mil.sys             fc890000 31ec6c6d - vga.sys
fc708000 31ec6ccb - Msfs.SYS                fc4b0000 31ec6cc7 - Npfs.SYS
fefbc000 31eed262 - NDIS.SYS                a0000000 31f954f7 - win32k.sys
fefa4000 31f91a51 - mga.dll                 fec31000 31eedd07 - Fastfat.SYS
feb8c000 31ec6e6c - TDI.SYS                 feaf0000 31ed0754 - nbf.sys
feacf000 31f130a7 - tcpip.sys               feab3000 31f50a65 - netbt.sys
fc550000 31601a30 - el59x.sys               fc560000 31f8f864 - afd.sys
fc718000 31ec6e7a - netbios.sys             fc858000 31ec6c9b - Parport.sys
fc870000 31ec6c9b - Parallel.SYS            fc954000 31ec6c9d - ParVdm.SYS
fc5b0000 31ec6cb1 - Serial.SYS              fea4c000 31f5003b - rdr.sys
fea3b000 31f7a1ba - mup.sys                 fe9da000 32031abe - srv.sys

Address  dword dump   Build [1381]                            - Name
fec32d84 80143e00 80143e00 80144000 ffdff000 00070b02          - KSecDD.SYS
801471c8 80144000 80144000 ffdff000 c03000b0 00000001          - ntoskrnl.exe
801471dc 80122000 f0003fe0 f030eee0 e133c4b4 e133cd40          - ntoskrnl.exe
80147304 803023f0 0000023c 00000034 00000000 00000000          - ntoskrnl.exe

Restart and set the recovery options in the system control panel
or the /CRASHDEBUG system start option.
```

Planning and documenting deployment is a key (and overlooked) part of successful development.

The Deployment View

- Focuses on aspects of the system important **after** the system has been built and is **ready** to be put into live operation.
- Defines:
  - The physical environment it will run in.
    - Hardware and hosting environment (processing nodes, network interconnections, disk storage).
  - Technical environment requirements for each processing node.
  - Mapping of elements to the runtime environment that will execute them.

When do I need this?

- When the system has…
  - Complex runtime dependencies.
    - Third party libraries, network services.
  - Complex runtime environments.
    - Elements distributed across many machines.
  - Dependencies on unfamiliar HW/SW.
    - Deployed on cloud hardware.
- When the system will be deployed in…
  - Wildly varying software environments.
    - Commercial software run on a PC.
  - Wildly varying physical environments.
    - Specialist or unfamiliar hardware.

Concern: Runtime Platform Required

- Must identify the runtime platform and role of each part.
    - Compute nodes, special-purpose nodes for databases, storage, print and input devices, network services (firewalls), specialist hardware, etc.
- Must define how the platform is provided.
    - In-house physical hardware, virtual servers in the cloud from a third party, public cloud, etc.
    - Define physical location of each part of the platform.
- Define types of processing elements, dependencies between them, and mapping of functional elements to processing.

Concern: Specification and Quantity of Hardware or Hosting

- Physical model of the hardware needed.
- Can specify hardware you will purchase or for abstract virtual machines.
- Developers are interested in:
    - Intel or Sun CPUs, Linux or Windows, resources
- System admins are interested in:
    - What hardware and how much.
- What service level will you maintain?
- Do you need specific hardware, or general purpose?

Concern: 3rd Party Software

- Most systems rely on 3rd party OS, libraries, messaging, application servers, databases.
- Make clear all dependencies between your system and any 3rd party products.
- Are there hardware requirements for these?
- Tells developers what they can make use of.
- Tells sys admins what they need to install and maintain.
- Analysis shows gaps in context and functional views.

Concern: Network Environments

- Elements may be deployed on multiple machines. Some interelement interactions are actually network interactions.
    - What services are required on the network?
    - How are machines linked?
    - What communication protocols are used?
    - Do we require load balancing, firewalls, encryption?
    - What is the required capacity, latency, and reliability of the links?
        - How much traffic will be carried over each intermachine link?

Runtime Platform Elements

- Runtime model defines the set of hardware nodes, how nodes connect via interfaces, which software elements are on which hardware nodes.
- Processing nodes
    - Each computer is represented by a processing node.
    - Allows estimation of the resources needed for deploying the system.
    - Where similar machines are required (server farms), can summarize as groups of nodes.

Runtime Platform Elements

- Client Nodes
  - Also represent client hardware (in less detail).
  - Less control, but can note type and quantity.
  - Speciality hardware is considered a “client node” (printers, touch screens).
- Runtime Containers
  - Special virtual machines that provide a sandboxed runtime environment for deployed elements.
- Online Storage Hardware
  - How much storage, what type, how partitioned.
  - Required reliability and speed.
  - Where does processing take place.

Runtime Platform Elements

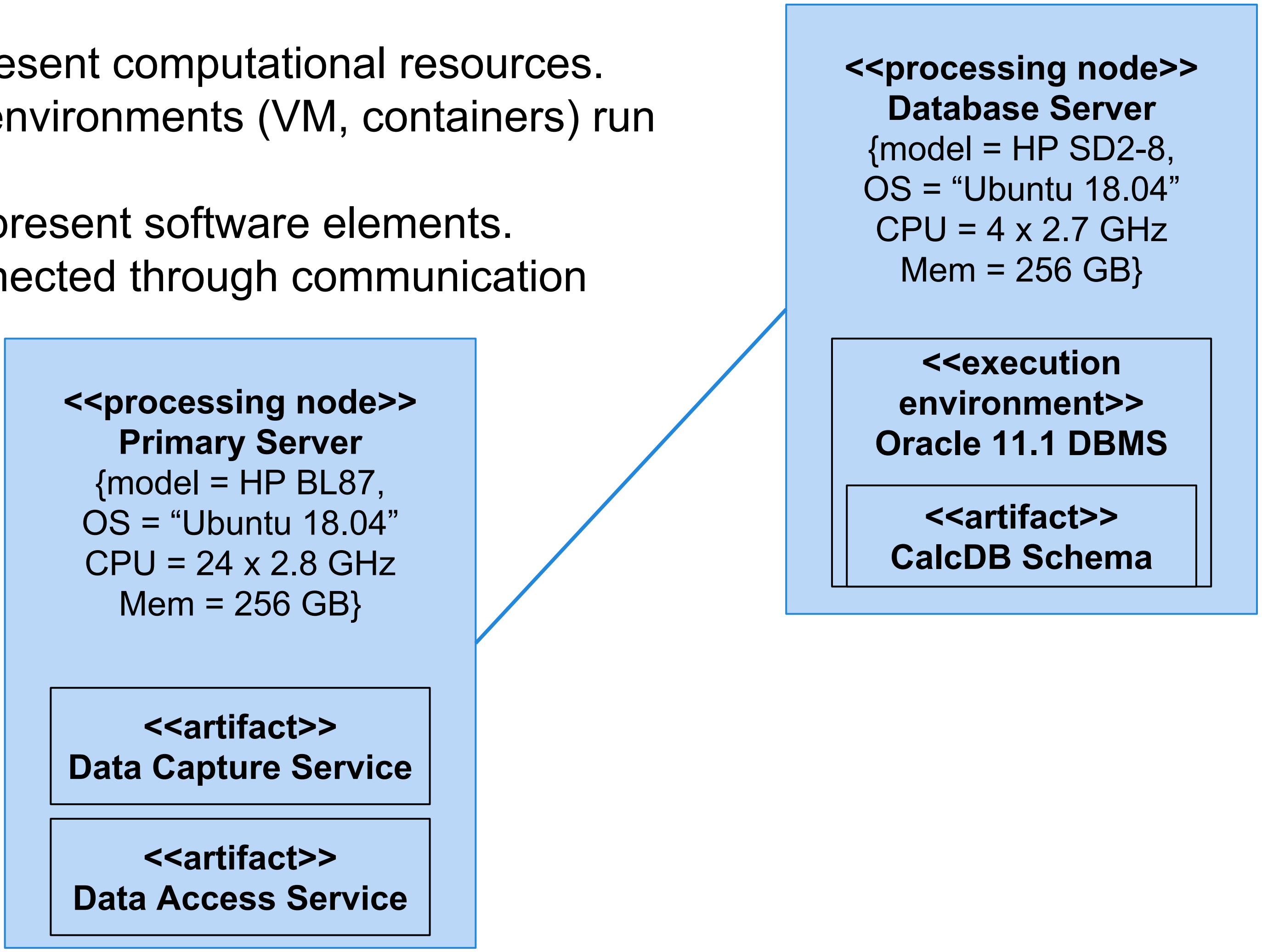
- Offline Storage Hardware
  - Archival and backup of data.
  - Ensure capacity, sufficient hardware speed, and network bandwidth.
- Network Links
  - Links between hardware nodes.
  - Network model captures more detail.
- Other Hardware Components
  - Network security, user authentication, special interfacing with systems, specialist processing.

Mapping Elements to Nodes

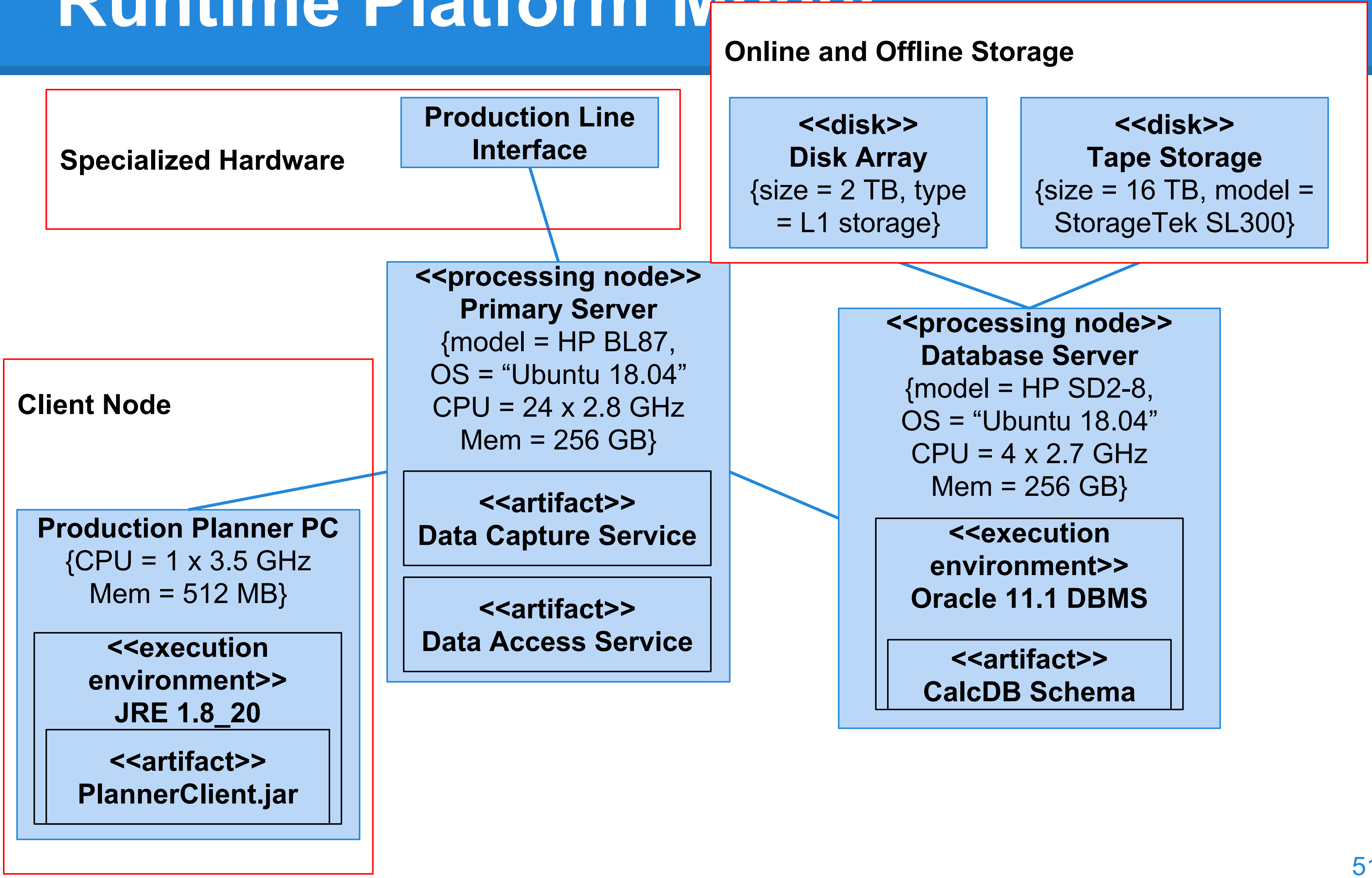
- Need to map functional elements to the processing nodes where they execute.
- Processes from Concurrency View can be mapped to processing nodes.
- If not concurrent, map elements directly to nodes.
- Captured as a UML deployment diagram showing nodes, storage, interconnections, and software elements.

Runtime Platform Model

- Nodes represent computational resources.
- Execution environments (VM, containers) run on nodes.
- Artifacts represent software elements.
- Nodes connected through communication paths.



Runtime Platform Model


Online and Offline Storage
<<disk>> Disk Array {size = 2 TB, type = L1 storage}
<<disk>> Tape Storage {size = 16 TB, model = StorageTek SL300}
Specialized Hardware
Production Line Interface
<<processing node>> Primary Server {model = HP BL87, OS = "Ubuntu 18.04" CPU = 24 x 2.8 GHz Mem = 256 GB}
<<artifact>> Data Capture Service
<<artifact>> Data Access Service
<<processing node>> Database Server {model = HP SD2-8, OS = "Ubuntu 18.04" CPU = 4 x 2.7 GHz Mem = 256 GB}
<<execution environment>> Oracle 11.1 DBMS
<<artifact>> CalcDB Schema
Client Node
Production Planner PC {CPU = 1 x 3.5 GHz Mem = 512 MB}
<<execution environment>> JRE 1.8_20
<<artifact>> PlannerClient.jar

Runtime Modeling Activities

- Design the deployment environment
  - Identify key servers
  - Identify important client hardware requirements
  - Identify network links between nodes
  - Add special-purpose hardware, specify hardware and software configurations for each node.
- Map elements to hardware
  - May suggest changes to deployment environment.
  - Manage dependencies.
  - Ensure machine capacity is available.
  - Make quality trade-offs (security vs performance)

Runtime Modeling Activities

- Estimate the Hardware Requirements
  - Perform an initial estimation, then revise as the architecture and design progress.
  - Processing power, memory, disk space, bandwidth.
- Conduct a Technical Evaluation
  - Prototype element development, perform benchmarks, perform compatibility tests.
  - Create mock systems to test integration and throughput.
- Assess Constraints
  - Formal standards, informal guidelines, assumptions.
  - Review your design to ensure all are met.

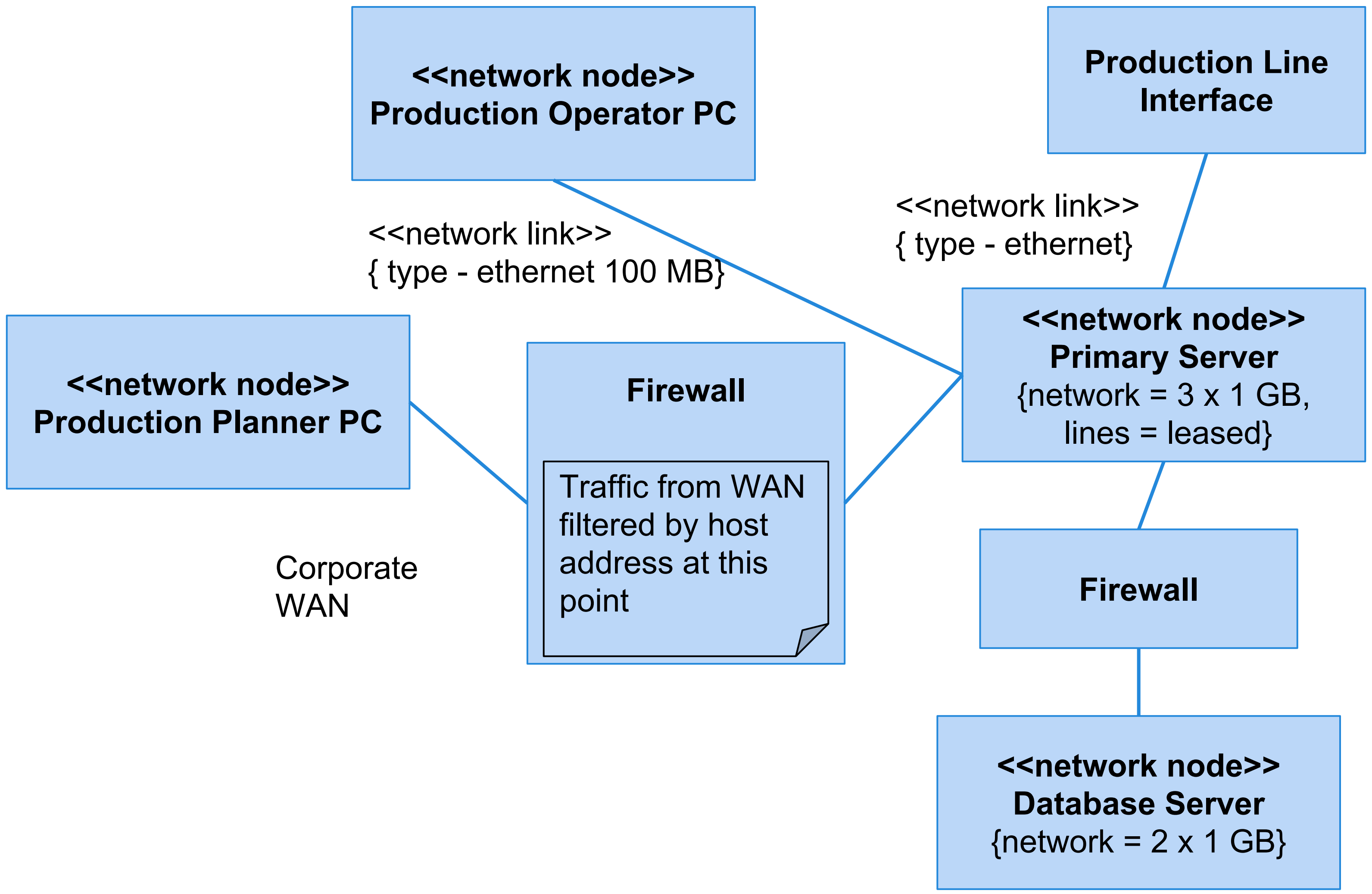
Network Models

- If networking environment is complex, you may need a network model detailing:
  - Which nodes need to be connected.
  - Specific network services required (firewalls, compression, packet encryption)
  - Bandwidth requirements and quality properties.
- Logical model, not a physical one.
  - Service-based view of the network.
- Valuable for customers planning to deploy your software in their organization.

Network Model Elements

- Processing Nodes
  - Elements that use the network to transport data.
  - Should match set from runtime platform model.
- Network Nodes
  - New nodes added that represent network services you expect to be available.
  - Firewall security, load balancing, encryption.
- Network Connections
  - Links between network and processing nodes.
  - Should describe service you expect link to provide.
  - Bandwidth, latency, quality of service, reliability, or other network qualities.

Network Models



Network Modeling Activities

- Design the Network
  - Establish connections, capacity, quality of service, and security.
  - Logical design, which is handed to specialist network designers for physical design.
- Estimate Capacity and Latency
  - Realistic estimation of the magnitude of traffic to be carried and expected round-trip time.
  - Capacity: Peak transaction throughput and a rough approximation of the size of messages required.
  - Latency: Standard metrics, distance between nodes.
  - Plan for scalability.

Technology Dependency Models

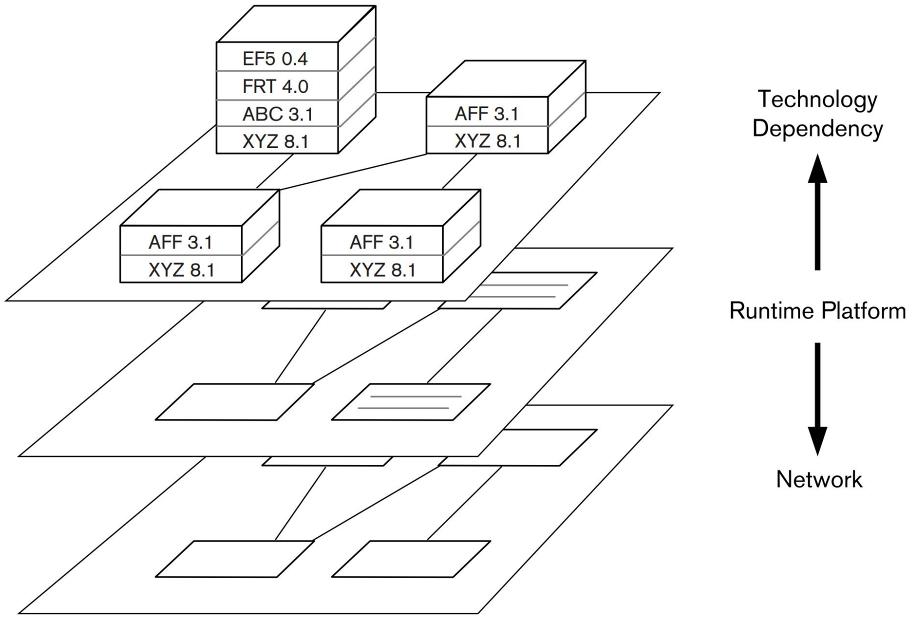
- When possible, it is ideal to bundle software and dependencies into one executable.
    - Not always possible due to efficiency, cost, licensing, flexibility.
- Deployment view should document the dependencies on a node-by-node basis.
- Can be captured in a simple table.
    - SW dependencies may have already been captured in Development View.
    - HW dependencies can be derived from test and development environment, manufacturer specs, testing you conduct.

Technology Dependence Model

Component	**Requires**
Data Access Service	HP-UX 64-bit 11.23+ patch bundle B.11.23.0703 HP aCC C++ runtime A.03.73
Data Capture Service	HP-UX 64-bit 11.23+ patch bundle B.11.23.0703 HP aCC C++ runtime A.03.73 Oracle OCI libraries 11.1.0.7
HP aCC C++ Compiler & Runtime	HP patch PHSS_35102 HP patch PHSS_35103
Oracle OCI 11.1.0.7	HP-UX optional package X11MotifDevKit.MOTIF21 HP-UX patch PHSS_37958

Intermodel Relationships

- Runtime Platform Model: Used by group responsible for deployment early in the project.
- Network Model: Used by specialist networking group
- Technology Dependency Model: Used by system administrators during installation planning.



Pitfall: Missing or Inaccurate Dependencies

- “You need Oracle and Linux”
  - Too vague to allow safe deployment.
  - Which versions? What patches? Optional updates?
- Capture clear, detailed dependencies between SW, runtime environment, HW.
- Capture dependencies between 3rd party SW and the runtime environment.
- Perform compatibility testing.
- Use existing, proven combinations of technologies with well-understood relations.

Risk: Unproven Technology

- New tech can bring great benefits...
  - Often more features, improved performance.
- Or great risk…
  - Functional shortcomings, poor performance, availability, security.
- Use existing hardware and software that you can test before committing to.
- Get advice from people who have used a technology before.
- Create prototypes and benchmarks.
- Perform compatibility testing.

Risk: Lack of Specialist Knowledge

- System design requires significant knowledge about many subjects.
  - Teams of people specializing in different technologies and aspects of the system.
- Can easily end up in a situation where you lack detailed knowledge of a technology and must rely on vendor claims.
- Bring in new specialists when needed.
- Obtain external review of your architecture.
- Obtain binding contracts from suppliers.

Risk: Late Consideration of Deployment Environment

- Problems occur when you consider system purely from a software-oriented perspective.
  - Can make a system unusable.
  - Impacts how software is designed and implemented.
  - Can be expensive to change.
    - I.e., if you need a group of small machines instead of one large machine, architecture differs.
- Design deployment as part of architecture design, not after system has been developed.
- Obtain external review to get early feedback.

Risk: Ignoring Intersite Complexity

- Many systems “live” in multiple locations.
    - Cloud computing environment, servers in different geographic locations.
- Important to consider impact early.
    - Impacts security, performance, scalability.
    - Network latency, increased security burden, synchronization between sites.
- Consider impact on system qualities.
- Work with infrastructure team.
- Test representative aspects as soon as possible “in the field”.

Risk:
Not Specifying Disaster Recovery

- How can the system be kept operational in the event of a significant failure.
  - Power loss
  - Storage failure
  - Natural disasters
- Often involves deployment of a special operational environment in a different location.
  - May have lower specification than production environment.
  - Should be considered as part of Deployment View.

Food for Thought

- Have you mapped functional elements to a type of element in your runtime platform?
  - Have you mapped them to specific hardware devices if appropriate?
- Is the role of each piece of your runtime platform fully understood?
  - Is the hardware or service suitable for the role?
- Have you established detailed specifications for hardware devices or hosted services?
  - Do you know exactly how many of each device or how much of each service is required?

Food for Thought

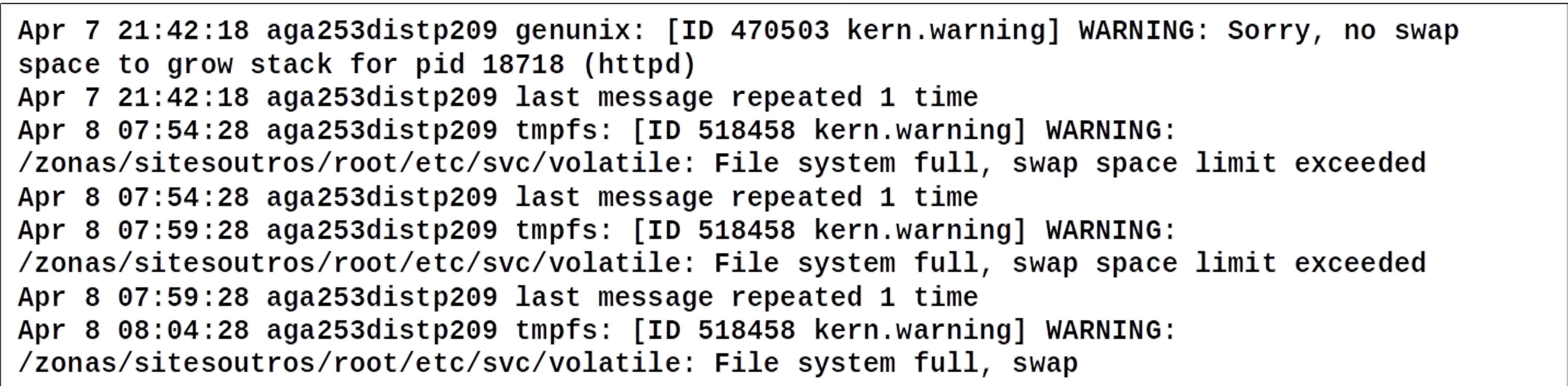
- Have you identified third-party software and documented dependencies?
- Are the network topology and services understood and documented?
    - Have you estimated and validated the required network capacity?
    - Can the proposed network topology be built to support this capacity?
- Have you performed compatibility testing to ensure that the elements can be combined as desired?

Food for Thought

- Have you used prototypes, benchmarks, and other practical tests when evaluating?
- Can you create a realistic test environment?
- Are you confident that the deployment environment will work as designed?
  - Have you obtained external review?
- Can physical constraints (floor space, power, cooling) can be met?
- Do you have a specification of a disaster recovery environment, if required?

Operational Viewpoint

What does it mean when the swap file is full?



```
Apr 7 21:42:18 aga253distp209 genunix: [ID 470503 kern.warning] WARNING: Sorry, no swap
space to grow stack for pid 18718 (httpd)
Apr 7 21:42:18 aga253distp209 last message repeated 1 time
Apr 8 07:54:28 aga253distp209 tmpfs: [ID 518458 kern.warning] WARNING:
/zonas/sitesoutros/root/etc/svc/volatile: File system full, swap space limit exceeded
Apr 8 07:54:28 aga253distp209 last message repeated 1 time
Apr 8 07:59:28 aga253distp209 tmpfs: [ID 518458 kern.warning] WARNING:
/zonas/sitesoutros/root/etc/svc/volatile: File system full, swap space limit exceeded
Apr 8 07:59:28 aga253distp209 last message repeated 1 time
Apr 8 08:04:28 aga253distp209 tmpfs: [ID 518458 kern.warning] WARNING:
/zonas/sitesoutros/root/etc/svc/volatile: File system full, swap
```

- Operational aspects of systems are often ignored during design.
- **This is a significant contributor to unexpected system down time.**

The Operational Viewpoint

- Identifies a system-wide strategy for addressing operational concerns.
  - Helps to ensure system is a reliable and effective part of its environment.
  - For packaged software, helps illustrate the types of issues that could occur once installed.
  - Documents how the system can be architected to reduce or address these concerns.
- Often least well-defined view, as many of the details are not fully-defined until construction is underway.

Concern: Installation and Upgrade

- [Insert your own installation horror story]
- How is installation performed?
  - Your team performs the install.
  - Users install and configure themselves.
  - Resources allocated to a cloud environment.
- Is this a pure installation or an upgrade?
  - Upgrades can be more complex.
  - Must respect existing data and settings, state of running elements.
  - Can you keep the system running during update?
- Ensure the system can be installed or updated successfully.

Documenting Installation and Upgrade

- Help the reader understand:
  - What needs to be installed or upgraded to move the system into production.
  - What dependencies exist between groups of items to be installed or upgraded (determines event order).
  - What constraints exist on the installation process.
  - What needs to be done to abandon and undo the installation/upgrade if there is a problem.
- Do not need a complete guide.
  - Instead, constraints the architecture imposes on installation and upgrade.

Installation Documentation Activities

- Identify the Installation Groups
  - Group related elements. For each group, define what elements it contains and how they will be installed or ugraded.
- Identify Dependencies
  - Identify dependencies between groups to identify order elements must be installed in.
- Identify Constraints
  - Do you need to start an element immediately after installation? Do you need to restart machines?
- Identify Backout Approach
  - What do you need to do to undo any tasks?

Concern: Functional Migration

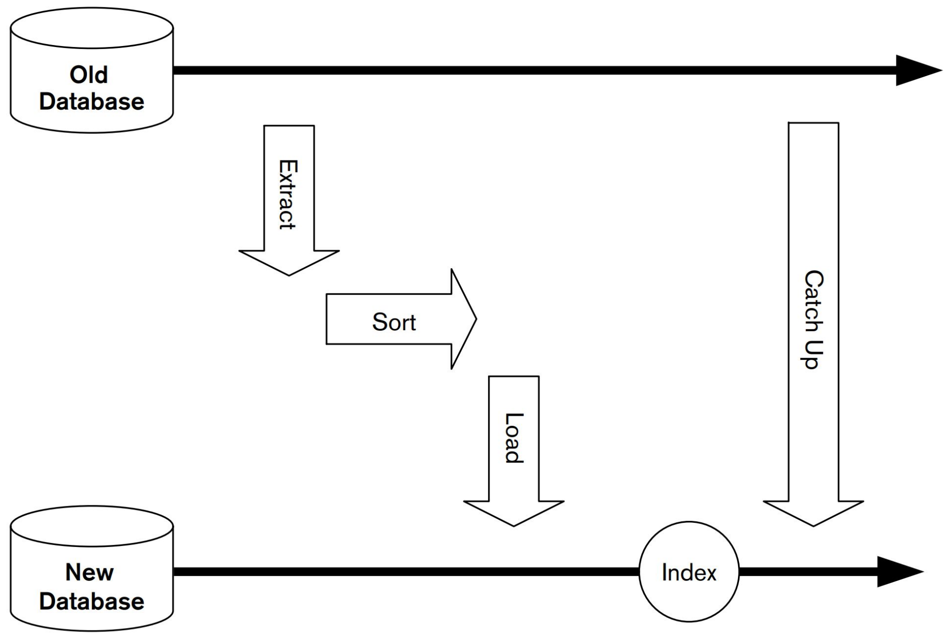
- Process of replacing existing capabilities with your new capabilities.
  - Migrating users of an older system to your system
  - Big bang: Migration at a single point in time.
  - Parallel: New and old versions of a system used side-by-side until buy-in.
  - Staged: Parts of a process of system swapped over time to manage risk.
- Problems: risk and cost
  - Big bang is cheapest, but no recovery route.
  - Others reduce risk, but more expensive.

Concern: Data Migration

- Loading data from existing systems into the new ones.
  - Should be automated as much as possible.
- May change format of GB - TB of data.
  - Data may not match new format exactly, may require conversion in parallel with live updates.
- Moving data between locations may add security and performance concerns.

Concern: Data Migration

- Migrating taxpayer database to new system.
  - Extraction - 3-5 days.
  - Sorting - 1 day.
  - Loading - 10 days.
- 100K updates expected.
  - Cannot stop tax collection for two weeks.
- Capture updates and apply in bulk once data migration is complete.



Documenting Migration

- Does not require a complete plan, but defines overall strategy and constraints.
- Allow reader to understand:
    - What strategies can be employed to migrate information and users to the system.
    - How the system will be populated with information from the existing environment.
    - How information in new and old environments can be kept synchronized.
    - How to revert to the old system if problems emerge.

Migration Documentation Activities

- Define the Migration Strategy
  - Big bang vs parallel vs staged.
  - How would this work?
  - What are the tradeoffs?
    - How long will this process takes?
    - Will it disrupt business?
    - Does this meet stakeholder needs?
- Design the Data Migration Approach
  - How to populate the system with the existing information.
  - How long will it require?
  - What resources are needed?

Migration Documentation Activities

- Design Information Synch Approach
  - Especially in parallel run.
  - Unidirectional (into new system) or bidirectional?
- Identify the Backout Strategy
  - Can you back out to the existing system?
  - Reverse data migration may not be practical.

Concern:
Operational Monitoring and Control

- Systems require routine monitoring.
- Control operations can be used to keep the system running correctly.
  - Startup, shutdown, transaction resubmission.
- How much is required depends on how many unexpected operational conditions are likely to occur.
- Balance against cost and time.
- Consider deployment environment to identify solutions.

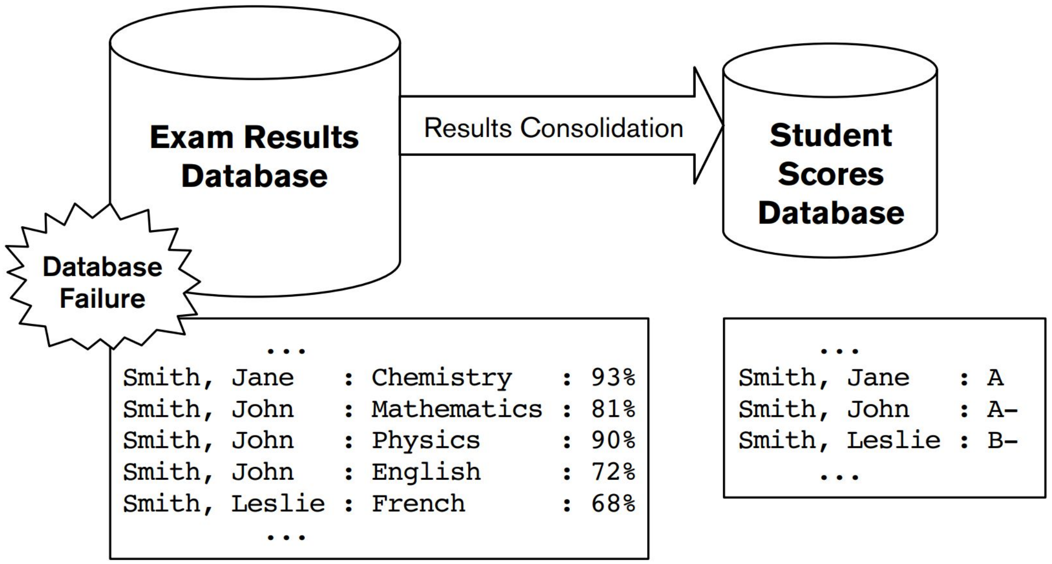
Concern: Alerting

- A system should send notifications when something bad happens.
  - Technical: Unable to connect to database.
  - Functional: Bad data on an automated input.
  - Significant non-error conditions (startup, shutdown)
- Active function of a system.
  - Sent to appropriate humans for action.
- Define which events require alerts, what information should be included, and where it should be sent.
- Avoid sending too many alerts.

Concern: Backup and Restore

- Data must be protected and insured.
  - Backup processes should be designed, built, and tested regularly.
- It must be possible to restore data from a backup in a transactionally consistent state.
  - All updates committed to the restored database or not recovered at all.
  - Consider data lost as part of restoring (at least any transactions active during failure).
- Failure in one element could corrupt system.
  - Recover or recreate lost data.
  - Revert system to older state.

Concern: Backup and Restore



```
           ...
Smith, Jane   : Chemistry   : 93%
Smith, John   : Mathematics : 81%
Smith, John   : Physics     : 90%
Smith, John   : English     : 72%
Smith, Leslie : French      : 68%
           ...
```

```
        ...
Smith, Jane   : A
Smith, John   : A-
Smith, Leslie : B-
        ...
```

- Academic records in databases.
  - Exam results database.
  - Scores database transforms data into a overall score.
- Corruption requires restoration of exam database.
  - Over three months old.
  - Results from those months will need to be reentered.
  - However, student scores already reflect that data. Must prevent reentered data from changing scores.

Documenting System Administration

- Monitoring and control facilities
  - How to monitor and adjust the system.
  - Custom utilities, existing management environments.
  - Basic message log to full-blown infrastructure.
  - Define what features you will offer, how to use them, and any limitations.
- Required routine procedures
  - What needs to be performed regularly?
  - Backup and health check procedures.
  - Define purpose of each procedure, when performed, who performs it, and the steps involved.

Documenting System Administration

- Likely error conditions
  - Error conditions may require administrative intervention (disk full, network failure).
  - What is unique about your architecture?
  - Explain error conditions, when they occur, how to recognize them, and HOW to correct them.
- Performance monitoring facilities
  - Watch the system for performance problems.
  - Extracted and analyzed routinely.
  - Explain measures taken, how they can be extracted and analyzed.

Do not use operations documentation as an excuse for bad software!

- Examples:
    - Los Angeles air traffic control: reboot system every 30 days to prevent a timer overflow or system will crash
    - Patriot missile system: reboot system every 12 hours or it won’t track incoming missles correctly
    - Therac 25: don’t press keys too quickly or use backspace key or system will give incorrect radiation dose
- Operations documents can become a CYA tool for bad software

Food for Thought

- Do you know how to install your system?
- Can you back out a failed installation?
- Can you upgrade an existing version of the system (if required)?
- Do you understand the facilities and constraints of the production environment?
    - Can you live with or mitigate these if not ideal?
- Do you know how information will be moved from the existing environment into the new system?

Food for Thought

- Do you have a clear migration strategy to move workload to the new system?
  - Can you reverse the migration if you need to?
- How will you deal with data synchronization?
- How will the system be backed up?
- Will the approach identified allow reliable restoration in an acceptable time period?
- Can the administrators monitor and control the system in production?
- Do the administrators understand the procedures they need to perform?

Food for Thought

- How will performance metrics be captured for the system’s elements?
- Can you manage configuration of all of the system’s elements?
- Is there consistency between the admin model and Development view?
- Is the data migration architecture compatible with the amount of time available to perform the data migration?
    - Are there catch-up mechanisms in place where the source data is volatile during the data migration?

Key Points

- Context/Functional/Information/Concurrency Views define *what* you are building.
- The Development, Deployment, and Operational Views define *how* you will build the system.
    - The **Development View** defines how to implement the system.
    - The **Deployment View** defines how to transition the system to live operation.
    - The **Operational View** defines how to keep the system alive in the field.

Next Time

- No class November 6th
  - Election Day!
- Perspective: Performance & Scalability
  - R&W: Ch. 26
  - Bass, Clements, Kazman: Ch. 8

- Homework:
  - Reading Assignment 2 - Tonight!
  - Project, Part 3 - Due on Nov 18
  - Assignment 3 - Nov 29