SWE40006
Software Deployment and Evolution

Lecture 3

Deployment -Update, Retire

Deployment - Story so far..

- Releasing a software involves:
  - Packaging and Advertising
- Contents of the package may change depending on target environment - Dev./ Test / Staging / Live
- Different packaging approaches and containers exist
- Advertising - Licensing, Support and Roadmap
  - Packaging and Advertising choices impact on product design -- developer should aim to capture these in the non-functionol requirements early
- Some licenses are viral
- Activation must be fail-safe

Related Learning objectives

1. Describe the activities in software deployment and apply these in a problem context.
2. Apply knowledge of software environments to plan development and deployment.

Lecture Overview

- In this lecture we will cover,
  - **Un-installing software**
  - **Activating and De-activating**
  - **Updating and Adapting**
  - Deployment Practices and Tools
  - Multi-tier and Web Applications

Software Deployment - Deactivate

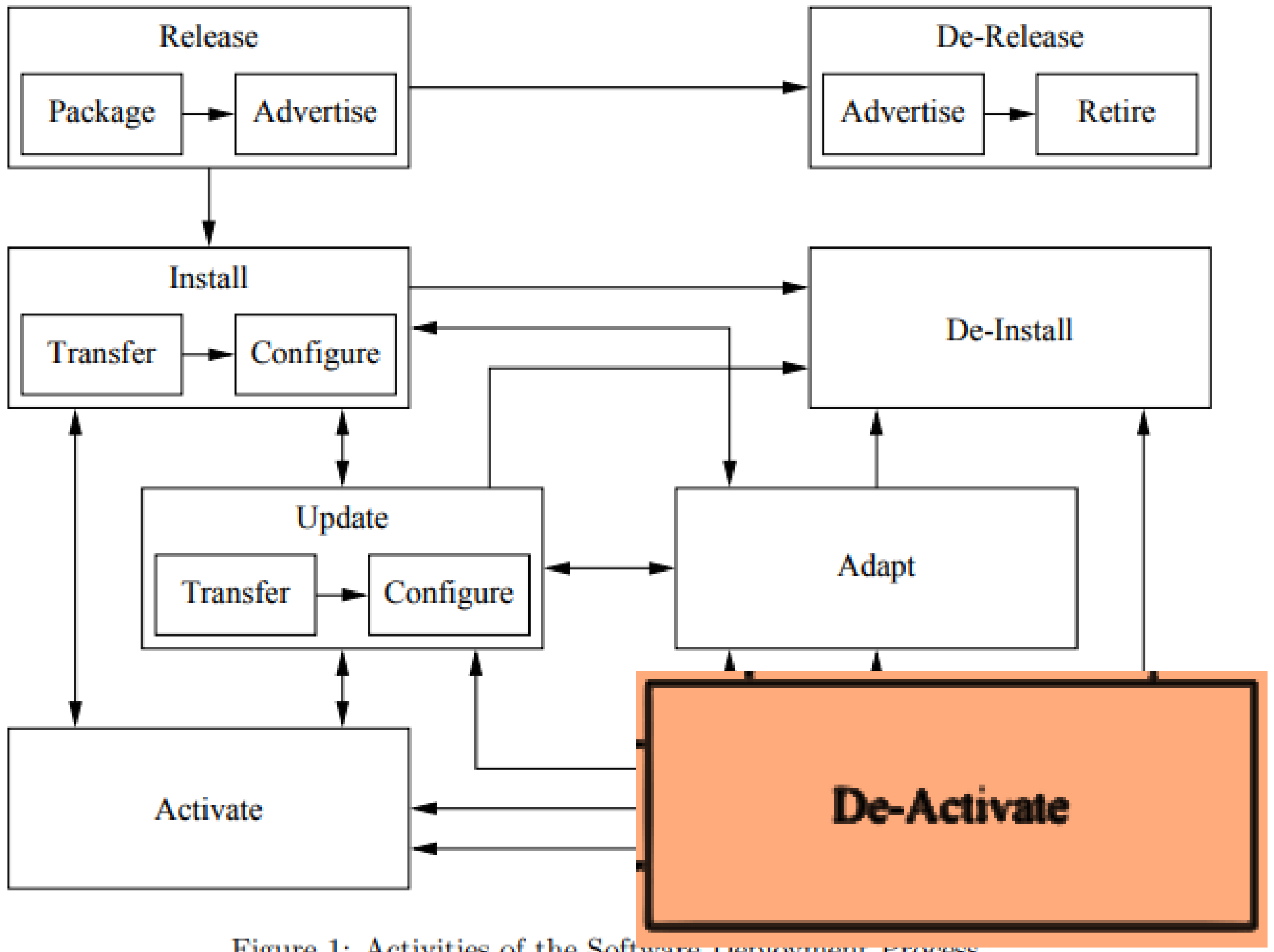


Figure 1: Activities of the Software Deployment Process.

https://www.ics.uci.edu/~andre/papers/T3.pdf

Deactivate

- Deactivating is shutting down a software system
  - Deactivating a software system should result in returning all locks (on devices) and resources properly
- Issues:
  - Software crashes during use -- should this trigger a clean ups
  - What should be cleaned!
  - What if one part of the application needs to be stopped (i.e. database server is disconnected)
- Partial de-activation support may be needed for many systems

Software Deployment – De-Install

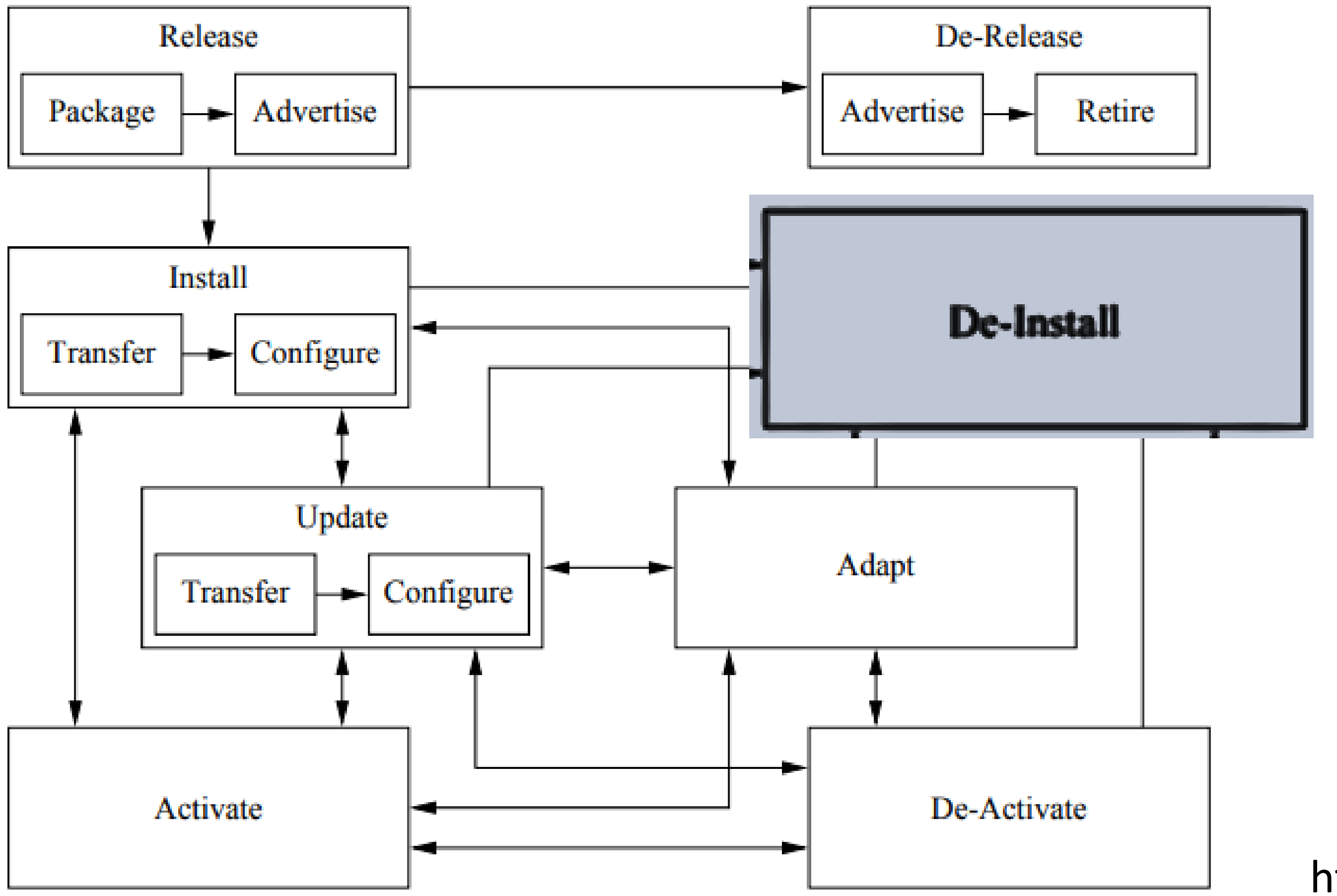


Figure 1: Activities of the Software Deployment Process.

https://www.ics.uci.edu/~andre/papers/T3.pdf

De-Install

- An ideal removal process will:
  - Remove all software components, including potentially custom drivers from the file system
  - Un-register any libraries (esp. COM components in Windows)
  - Create an archive of data files (potentially for use later or to backup) -- rarely done well
- Multi software deployments will often require deactivation of each part followed by a removal of software components

De-Install

- Removing software should be considered as an ACID* transaction
  - Should work even if power fails during the process (i.e. should recover from last state)
  - Should guarantee that no remnants are left behind (Registry keys, empty directories, hidden data directories/files etc.)
  - Currently the ACID concept is not well supported by most package managers/installers
- The entire process must be logged

*Atomicity, Consistency, Isolation, Durability
https://www.ibm.com/support/knowledgecenter/en/SSGMCP_5.3.0/com.ibm.cics.ts.productoverview.doc/concepts/acid.html

Software Deployment – Update

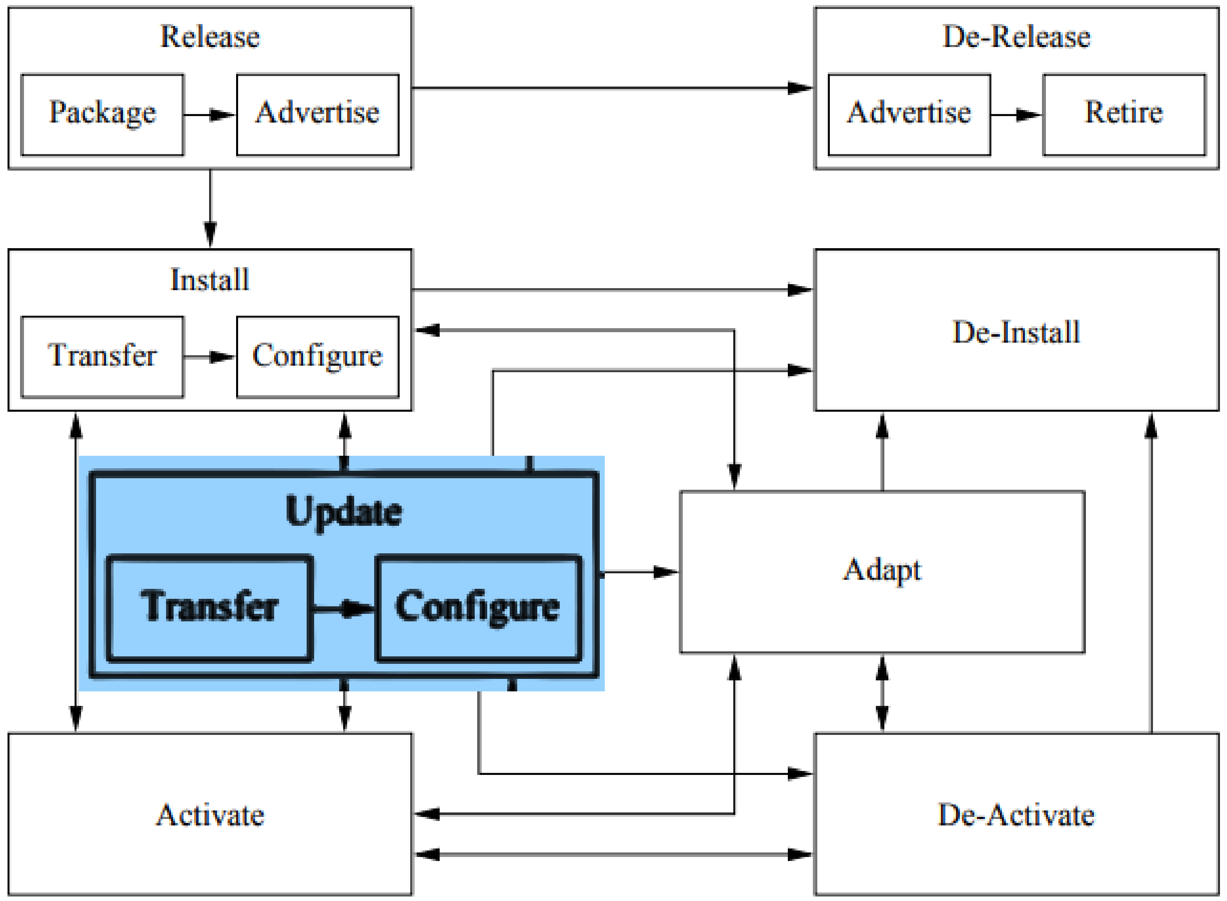


Figure 1: Activities of the Software Deployment Process.

https://www.ics.uci.edu/~andre/papers/T3.pdf

Update

- Update activity involves ’transferring and configuring’ a set of components
  - Entire software can be updated as part of this as well
- Typically this process involves replacing a few components with new ones
- Update is generally performed by a separate application that is part of the installer
- Configuring a new update is similar to configuring after an install (same issues need to be considered)

Update - Transfer

- Transfer issues:
  - Is the update possible? That is, there are no conflicts with any existing components
- Configuration issues:
  - Data migration may be needed as part of the configuration step
- Other aspects:
  - Some issues and problems may not be visible until activation (e.g. If a database server was updated)

Error Handling

- You have been taught this stuff (it is all good):
  - You can and should use error codes, and also exception handling mechanisms
  - This works in most cases
- But..... here is where the real pain comes from:
  - The exception handling (or error code based recovery) mechanisms assume that you did not change the state of the software
  - What if you did? Can you undo a committed database transaction?

Error Handling - Issues in Real World

- Consider a situation where an error happens today, but it requires you to ’undo’ transactions that took place over the last 2 weeks
  - This happens more often that you think when you update large software systems.
- Any application that has long lived work-flows and complex state can have this type of problem.
  - Things get even more messy since the data is often not reliable (or) unavailable.
  - Example: Update to a student pre-requisite checking system where rules change, but older rules were incorrectly recorded.

Software Deployment - Adapt

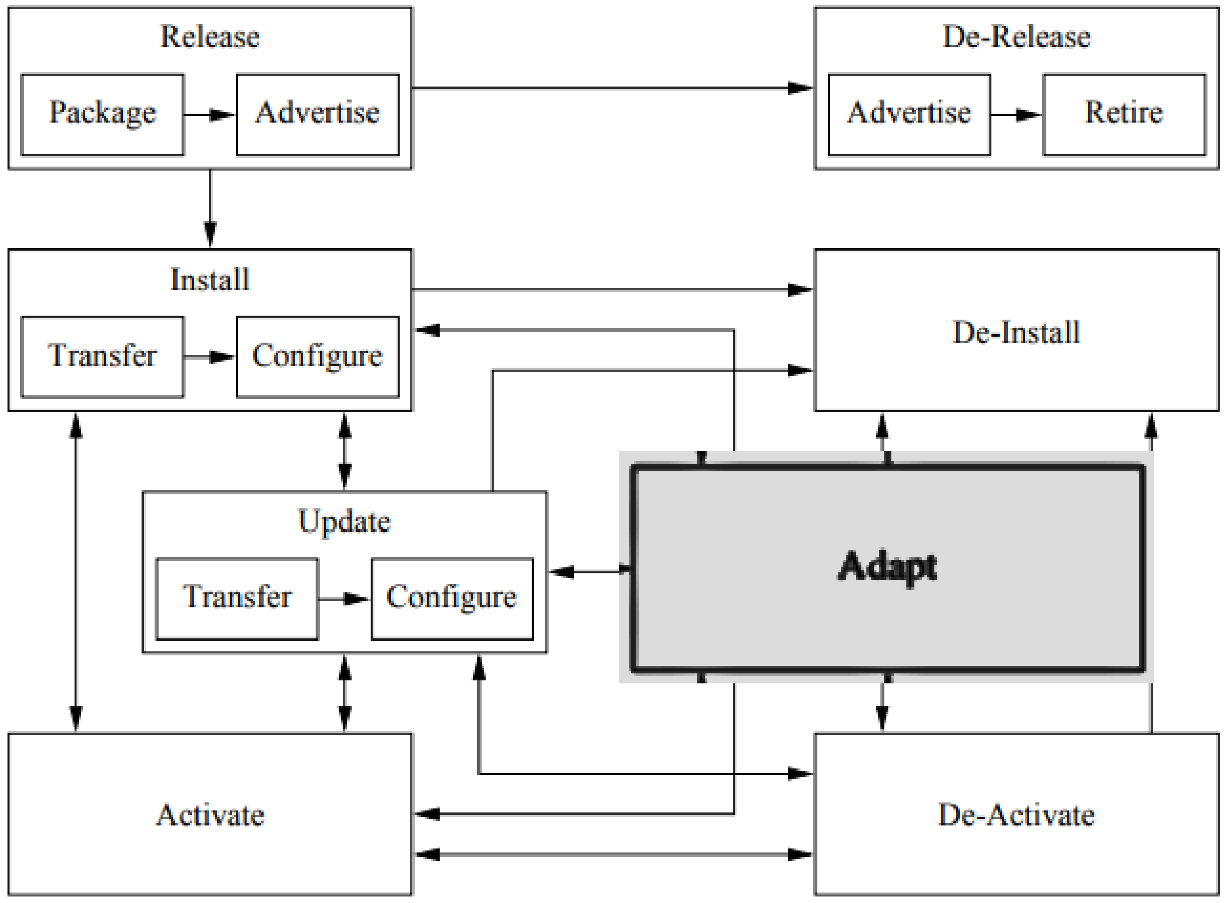


Figure 1: Activities of the Software Deployment Process.

https://www.ics.uci.edu/~andre/papers/T3.pdf

Adapt

- Adaptation is the ability of the software to adapt to an external change
- Potential aspects that can change:
  - External hardware and associated drivers (Video card or Sound driver)
  - External components (typically operating system update or web services that are used)
- Adaptation is generally triggered as part of the activation step
- Adaptation may trigger an update for new components (or) may need a fresh install

Software Deployment – De-Release

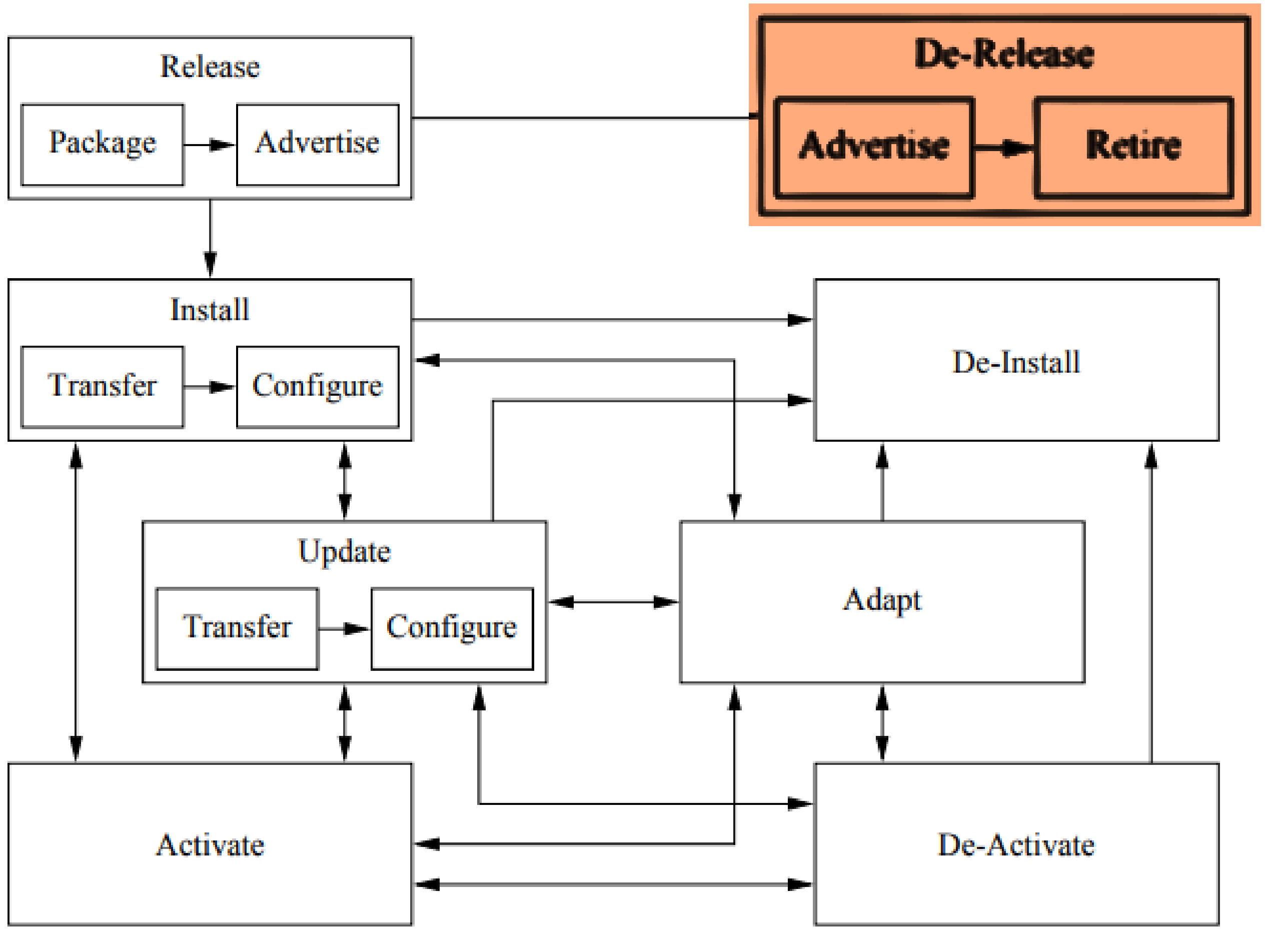


Figure 1: Activities of the Software Deployment Process.

https://www.ics.uci.edu/~andre/papers/T3.pdf

De-Release

- The process involves slowly removing support services and formally retiring a product
- Typically triggered when a new release is created, a previous release may be retired
- Most of the time this process involves advertising:
  - Removal of support services for a particular version of a product or a product line
  - Removal of development resources
  - Forcing licenses to expire
- The actual software itself is not formally removed

Timed Licensing

- Some software systems are released under a timed license
  - You lease the software for a period of time (e.g. 12 months)
  - After this period, the software will cease to function
- This model allows for a more effective de-release
- Commonly applied in large enterprise systems
  - Software running main-frames for instance.

Transactional Integrity

- The following activities must maintain transactional integrity and may need authorization:
  - Install, Update, Adapt and Activate
  - De-activate and De-install
- Ideally should be able to compensate for:
  - Power-failure
  - Component failures
- Must allow the user to return to previous state (in worst case scenario)
- Good improvements in Windows 7, Win 8, etc …

Data Migration

- Data migration is often needed when,
  - Installing over a previous version
  - Updating components
  - Adapting to changing external configuration
- Some types of migration can be undertaken by the configuration step (either during Install or Update)
  - A database schema change can be performed by running a set of'SQL' scripts and transferring data
- Some migration may require activation

Short Problem 5

- You have an application that has over I 2,000 tables in the database. A major revision has created another I,500 new tables and it also involves moving existing data between tables.

- What approach will you take for the data migration when you update the software? What are the risks in this approach?

Lecture Overview

- In this lecture we will cover,
  - Un-installing software
  - Updating and Adapting
  - Activating and De-activating
  - **Deployment Practices and Tools**
  - **Multi-tier and Web Applications**

Deployment Practices - I

- The following are certain deployment practices that should ideally be supported in good software:
  - Check configuration before install
  - Automatic resolution of dependency issues
  - Automatic (with consent) update process
  - Rollback of an update is possible
  - Rollback of an install is possible
  - Install and Update requires minimal/no downtime
  - Deployment configurations for test, dev, staging, live environments

Deployment Practices - 2

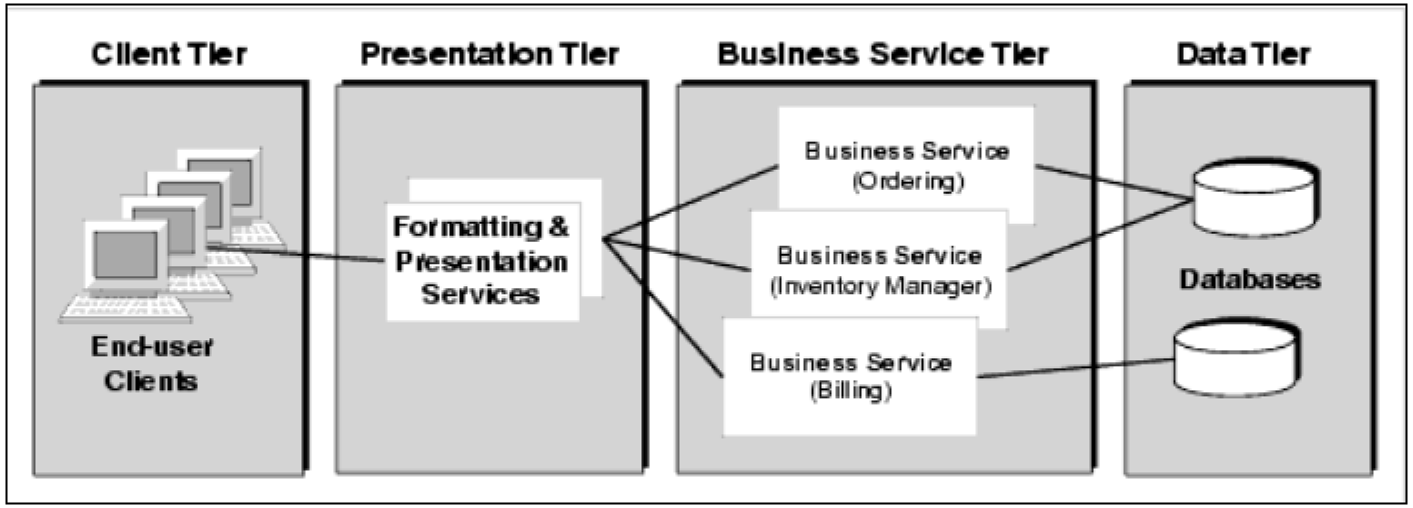
- Data updates are possible and do not require downtime (e.g. for anti-virus definitions)
- Logging of all transactions (install, update, adapt, activation, de-activation, de-install)
- Backups are automatically performed into an archive if product is removed (or) updated (or) upgraded software is able to monitor for a new version and inform user
- Multiple delivery models are available (Internet, DVD, phone)

Deployment Tools

- Some tools available to help automate deployment:
  - Windows Installer XML (WiX)
  - Chocolatey
  - RPM
  - NSIS (Nullsok Scriptable Installation System)
  - New ones …
- These tools support the deployment model to various levels -- some aspects are not covered by the tools (e.g. retire, activation and deactivation)

Deployment - Tiered Applications

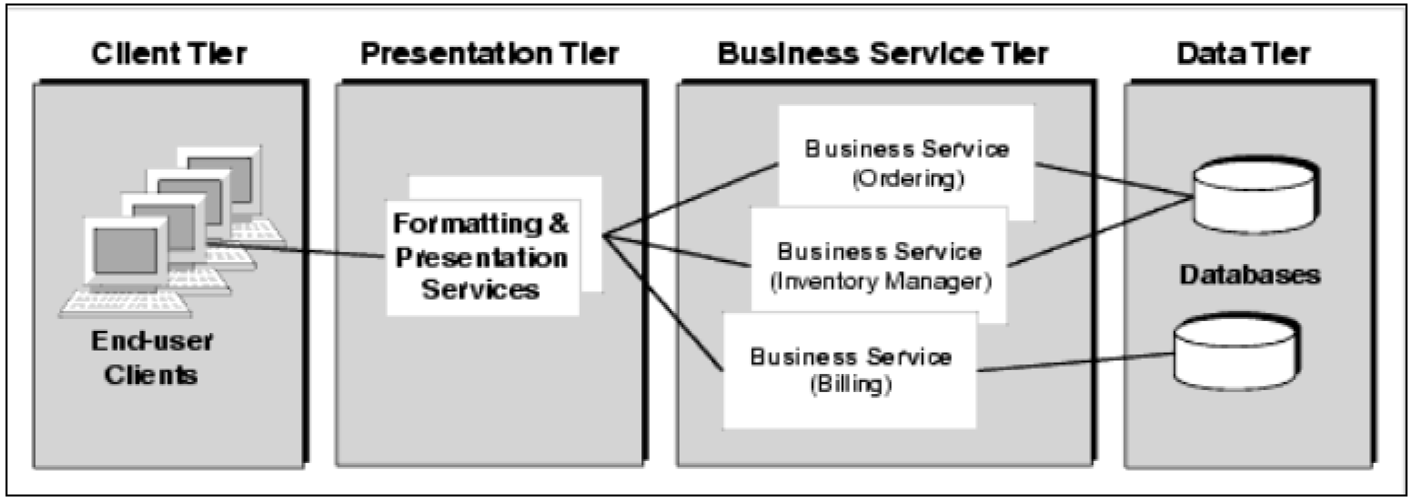
- Deployment may not update all tiers at the same time
- DB server may be updated independent of application


Client Tier
Presentation Tier
Business Service Tier
Data Tier
End-user Clients
Formatting & Presentation Services
Business Service (Ordering)
Business Service (Inventory Manager)
Business Service (Billing)
Databases

https://docs.oracle.com/cd/E19263-01/817-5764/architecture.html

Deployment - Tiered Applications

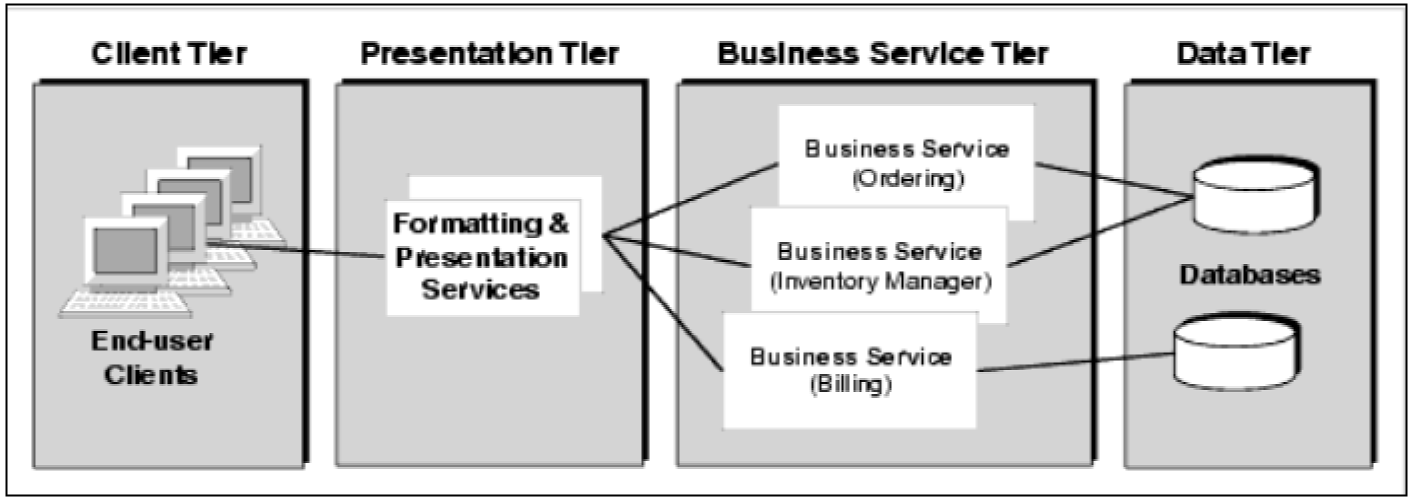
- Windows installer / RPM tools do not work well in these environments
- Custom deployment scripts need to be developed


Client Tier
Presentation Tier
Business Service Tier
Data Tier
End-user Clients
Formatting & Presentation Services
Business Service (Ordering)
Business Service (Inventory Manager)
Business Service (Billing)
Databases

https://docs.oracle.com/cd/E19263-01/817-5764/architecture.html

Middleware

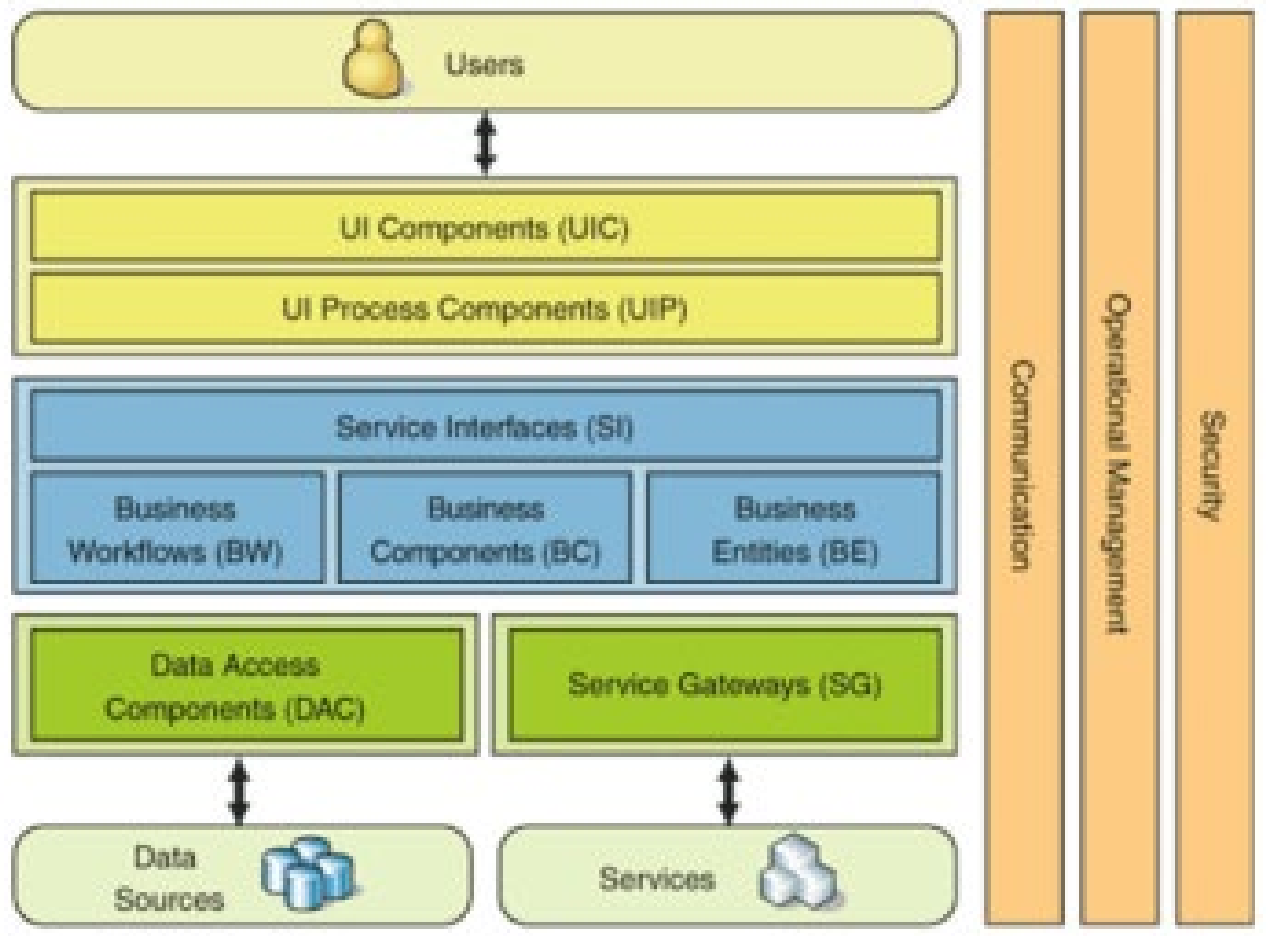
- Many Web applications make use of middle-ware to communicate between layers
- Middleware has to be installed and configured to work together


Client Tier
Presentation Tier
Business Service Tier
Data Tier
End-user Clients
Formatting & Presentation Services
Business Service (Ordering)
Business Service (Inventory Manager)
Business Service (Billing)
Databases

https://docs.oracle.com/cd/E19263-01/817-5764/architecture.html

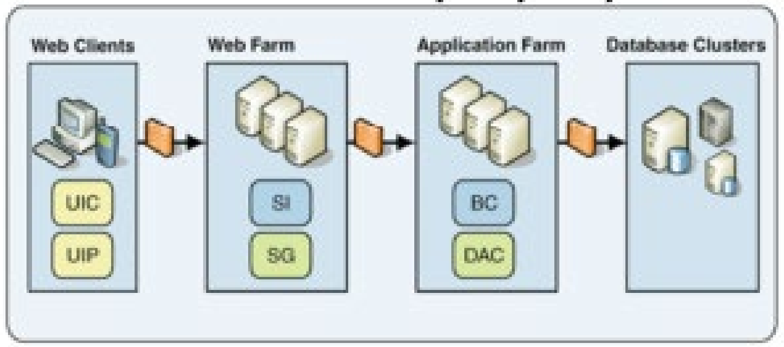
Deployment for Larger Systems

- The number of components that need to be carefully considered increases
- Current tools not mature for installations spanning many tiers -- but custom scripts can be created



Rich Web Applications

- The web clients can now be ~~Flash~~, Silverlight, WPF as well as HTML5/CSS/javascript -- this adds additional complexity
- The interfaces between tiers is often fire-walled
- Minimising downtime is a challenge -- esp. when many physical machines need to be updated
- Can be difficult if update needs re-booting and systems do not re-start properly



Minimising Downtime - Web Applications

- Large web applications achieve this by running parallel infrastructures
- New version is deployed to a “new live environment"
  - New user logins are directed to this infrastructure
  - Data migration is triggered at login time using specific data migration APIs
- User sign-off from old infrastructure (or) eventually their sessions will expire
  - Over a couple of weeks, the old infrastructure will have no further users and can be retired

Minimising Downtime - Web Applications

- Parallel infrastructures are expensive to run -- but possible if you have the user base to cover costs
- Product design implications:
  - It should be possible to perform data migration quickly and automatically -- must be engineered into the product
  - Functions need to be transactional and stateless for best results -- works where sessions eventually expire like for Web applications
- Partial updates (except for security patches) are generally avoided to infrastructure

Web Applications

- Activation in these application involves an additional aspect
  - The standard start-up process still applies
  - The login/sign-on process is treated as an activation of the application
  - Authentication check is performed
  - Data integrity checks are also triggered to help with migration of any data after update

Software Deployment Model

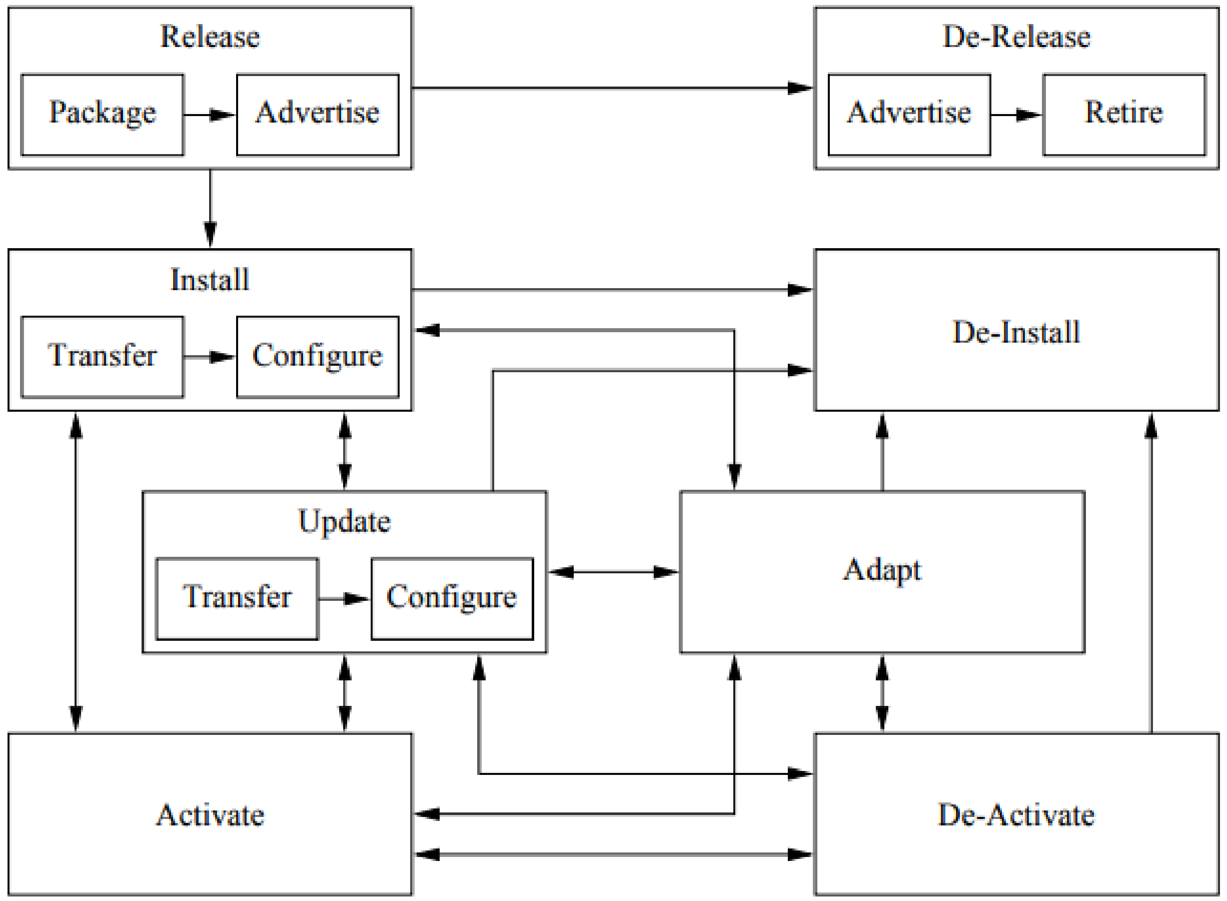


Figure 1: Activities of the Software Deployment Process.

https://www.ics.uci.edu/~andre/papers/T3.pdf

- NEXT …

Deployment - Story so far..

- The software deployment model
- Covers activities and relationships between these activities
- Work items inside each activity and issues that need attention
- The model broadly covers “what"
- We did not cover - how to execute the activities nor large scale environments
- Next, we will look at methods to deploy software

Related Learning objectives

1. Describe the activities in software deployment and apply these in a problem context.
2. Apply knowledge of software environments to plan development and deployment.
3. Describe and classify issues that drive software maintenance.

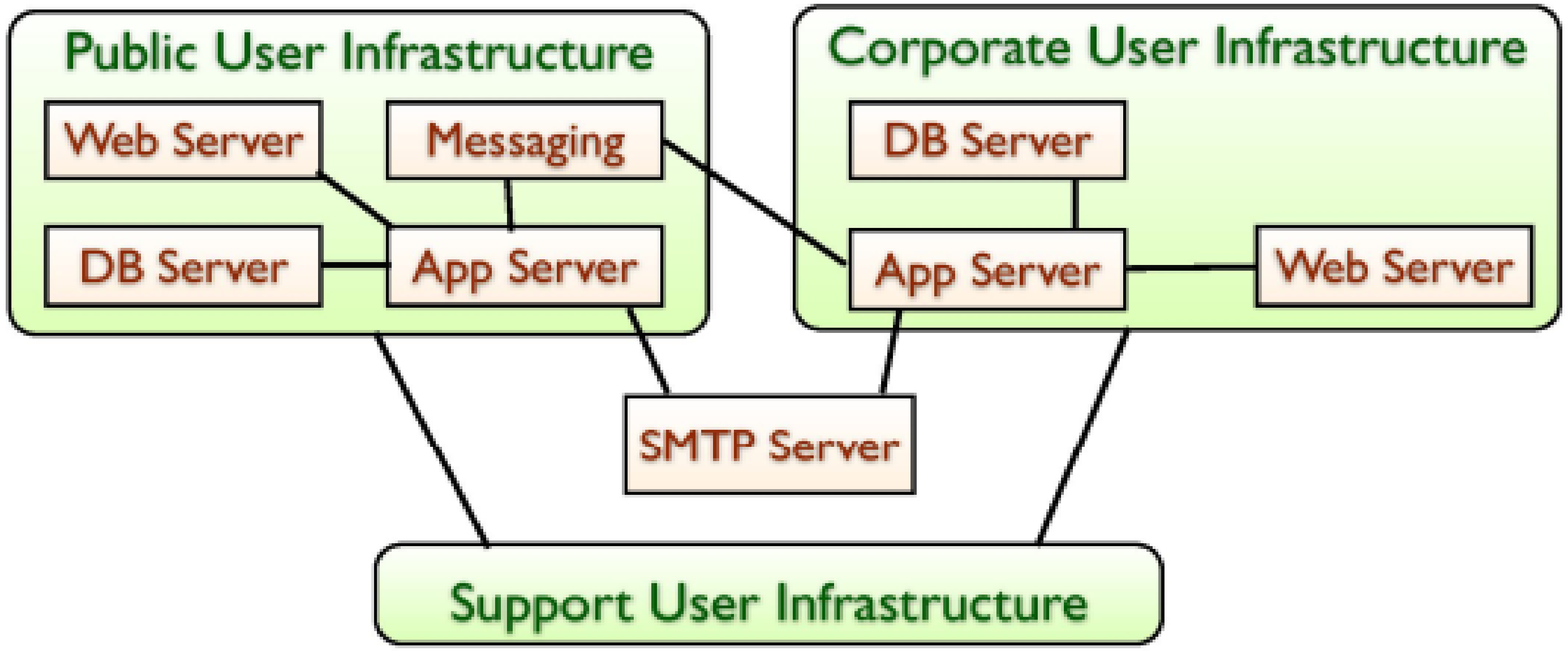
Lecture overview

This lecture will cover,

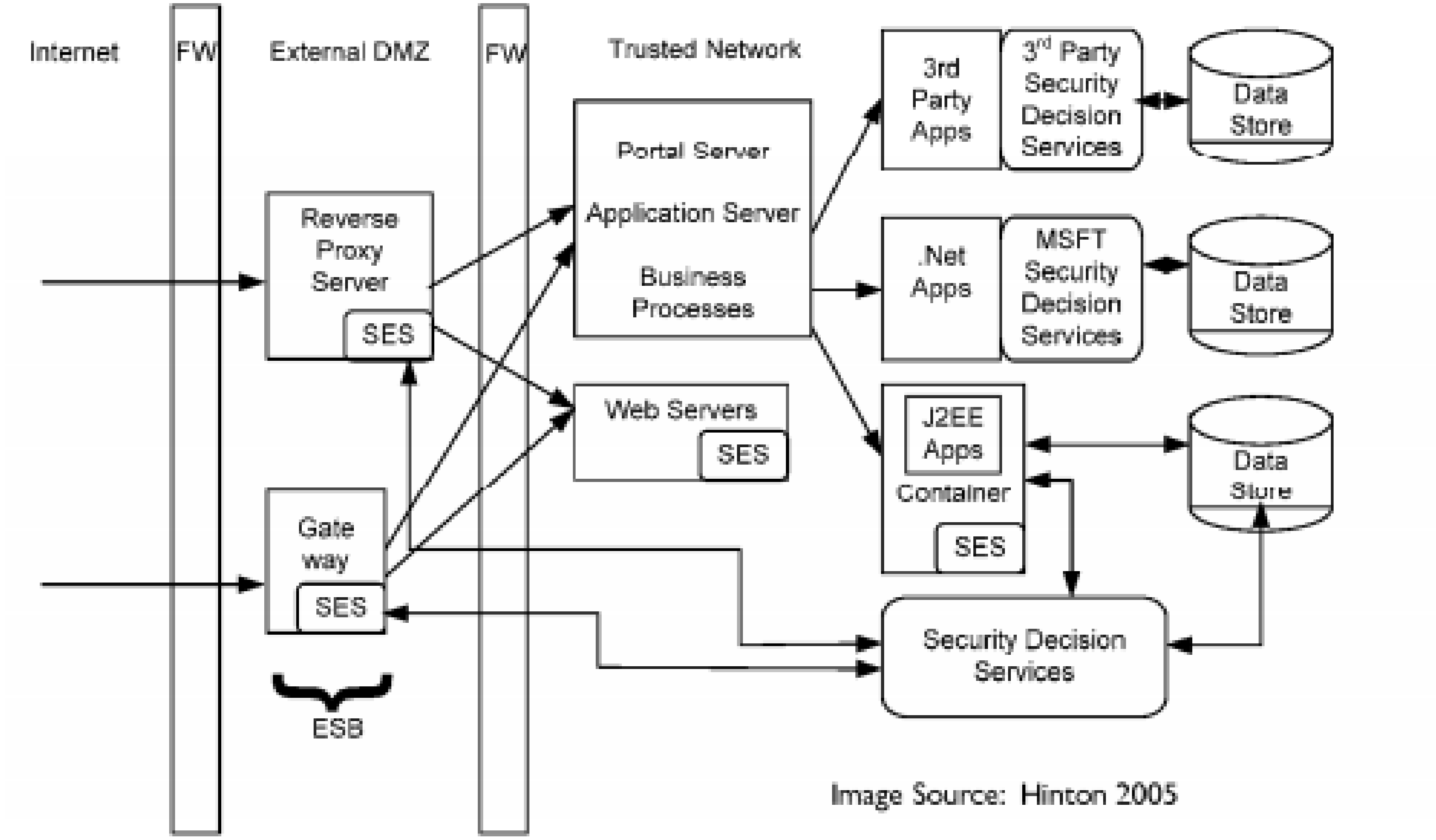
- **A motivating scenario (Jakarta EE Application)**
- Build Numbering and Management
- Deployment Approaches
  - Manual, Scripted, Language and Model-Based

High Level Context Diagram (Example)

- How do we plan for a deployment into this environment?


Public User Infrastructure
Web Server
Messaging
DB Server
App Server
Corporate User Infrastructure
DB Server
App Server
Web Server
SMTP Server
Support User Infrastructure

Real World System (Example)



ESB - Enterprise Service Bus, SES - Security Enforcement Service

It has a mix of .NET and Jakarta EE applications, Different DBMS as well as Security Services

Jakarta EE Deployment Scenario

- Lisa has to deploy a multi-tier JEE application:
  - MySQL Database server 5.0.x (clustered)
  - Active MQ (Message Queue)
  - JBoss Application Server
  - Spring Framework
  - 15 different java libraries for a range of tasks
  - Apache Web server
  - Monitoring and Administration applications
  - F5 load balance
  - LDAP server for authentication

Jakarta EE Deployment Scenario

- Staging/Live Environments:
  - 2 server grade machines for DB, 2 machines for web servers, and 1 machine for Application server
  - 1 desktop grade machine for monitoring and administration
  - All machines allow remote access from certain IP addresses (inside the DMZ)
  - All servers use Windows Server O/S

Jakarta EE Deployment Scenario

- Test Environment
  - Database, Web server and Application server all run on a single physical server grade machine
  - 5 testers each with desktop machines use the same test server
- Development Environment
  - Each developer has a high-end desktop (Windows 10) -- 10 developers
  - Git, Defect logging and Integration service runs on a single Linux server
- Integration service shares database

Jakarta EE Deployment Scenario

- Development schedule is 6 week time-boxed iterations
  - Team delivers a new build ready into Staging every 6 weeks
  - Team delivers and deploys a new build ready for Testing every fortnight
- Business decision causes one of the staging builds to be pushed into a live environment as an upgrade
- Deployment must support *full rollback* on all environments and must have an automated smoke test (i.e. to check everything works as expected)

Situation...

- Every year we need to undertake:
  - 26 deployments into test environment (with a rollback option)
  - 8 deployments into staging environment
  - Probably 4 deployments into live environment (certainly no more than 8)
  - Update Windows O/S 12 - 15 times (on average Microsoft releases an update once a month)
  - Update/Patch all other software in use (typically no more than once a month)
  - One major upgrade of some software systems

Lecture overview

This lecture will cover,

- A motivating scenario (Jakarta EE Application)
- **Build Numbering and Management**
- Deployment Approaches
    - Manual, Scripted, Language and Model-Based

    - We are taking a short de-tour, but will return back to the Jakarta EE scenario after this topic.

Build Management (Jakarta EE Scenario)

- Given the pace of development and the number of releases into various environment -- how does one keep track of things?
- Current good practice in these scenarios is to use Incremental single build numbering for Test/Staging and Live environments
  - Builds are numbers 1, 2, 3 ... 1341, 1342 ....
  - Environment is pre/post fixed to build
  - Example:T3987 (Test build 3987), S3991 for staging, L3991 for Live

Builds, Versions and Branches ...

- Incremental build numbering is easy to use -- especially when logging defects
  - Works quite well for Test/Staging/Live environments
  - Developers can still work with internal branches etc. -- but any build created for testers/QA/customers is tagged with a simple incremental build number
- Traditional version numbering (e.g. 5.0) is assigned to a build by Product Managers or Sales/Marketing

Builds and Revisions

- Incremental build numbers work well in most cases -- but have limitations for ‘product lines’
- Example Situation:
  - Let’s say, Build 2345 of Windows 8 goes Live
  - Revisions to Win 8 will be released for testing as 2345.1, 2345.2 ... (till product line is retired)
- When a new major product is started (say Windows 10) -- they seed the base code into the repository, prepare a Build 0 and start again

Builds – Revisions and Defects

- Consider this situation:
  - Windows 8 Build 2345.581 fixes serious flaws
  - Windows 10 build numbers are based on the year and month (1903=March 2019).
  - How do we ensure that Windows 10 code also fixes these serious issues?
- Solutions:
  - Merge code from Windows 8 to Windows 10: May work, but generally not practical
  - Realistically, Windows 10 testers have to replicate and add this to defect log with note on Windows 8 fix.

Builds - Revisions - Defects - Merge

- Merging code fixes from a branch into new product line may work -- only if this does not cause more problems
- In practice,
  - Testers will replicate and log defect for new system, may add a note about occurrence in Windows 8 (adds cost as we have to log defect twice)
  - Developers are left with choice to look at the fix and use that code (or) patch it again
- This is a trade-off - often faster to put in a *kludge* (workaround).

Back to Jakarta EE Scenario ...

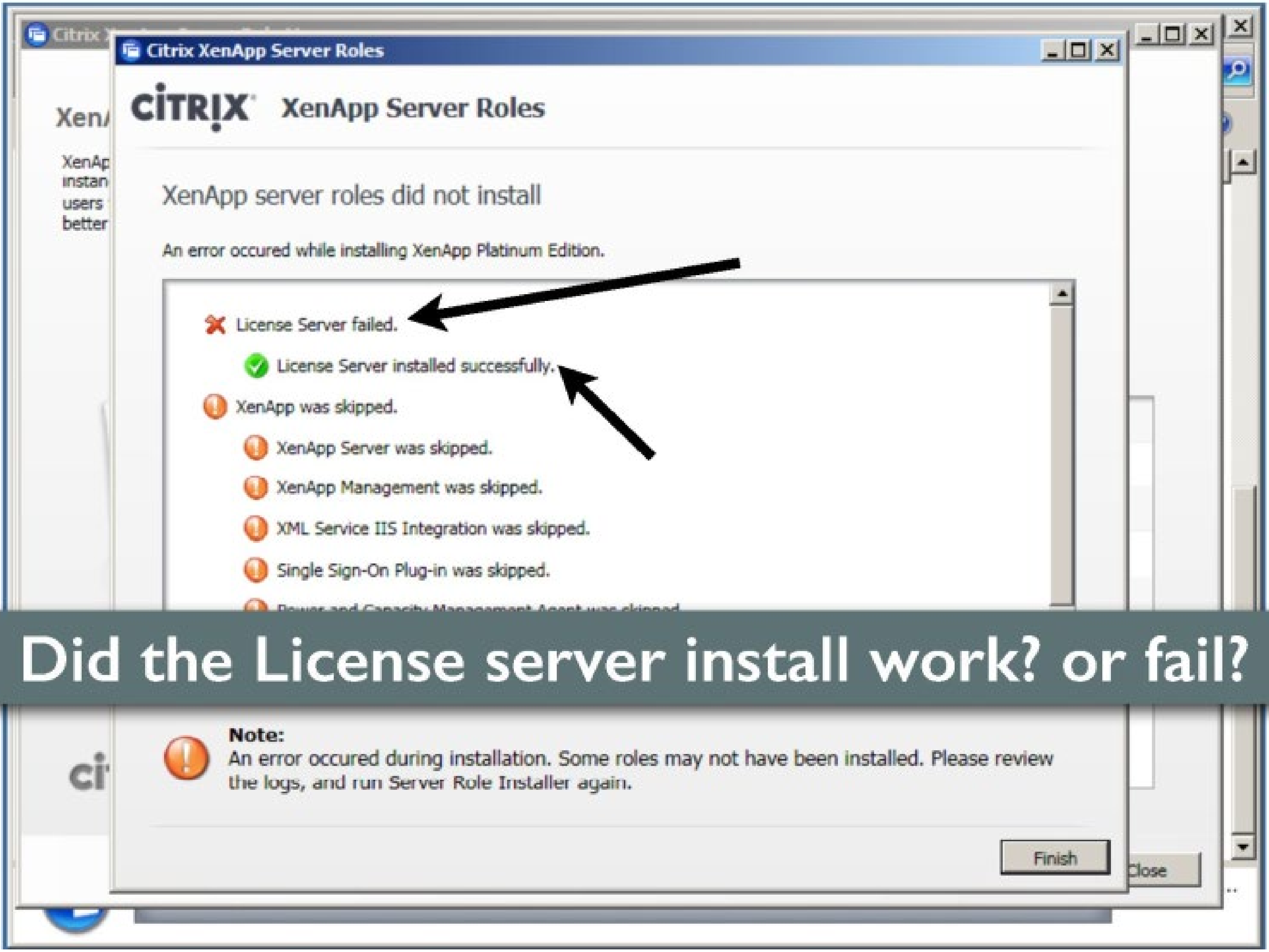
- In typical well managed enterprise scenarios
  - Incremental build numbering (some use ISO date format -- e.g. 20150826)
  - Minimum use of branches with revisions
- Build numbers are not just for the 'software being developed* -- they encapsulate a bit more
  - The exact version of all components, including the operating system, database server in use
  - This level of control is only found in mature organisations (as it adds cost)

Jakarta EE Scenario

- What are we dealing with?
  - Multiple environments (Dev/Test/Staging/Live)
  - Need to have the ability to rollback changes
  - Will need to migrate data
  - Relatively high-frequency deployment to test environment
  - Very low frequency live deployment
  - Staging environment may not mirror live (esp. if build is not pushed across)
  - Stable development environment

Jakarta EE Scenario

- How do we deploy the software and all associated components?
  - Assume Operating System is installed -- but nothing much else is configured/setup
- Options are:
  1. Write steps, manually execute deployment and manually verify configuration after activation
  2. Partial automation -- using scripts (Ant and Python)
  3. Use a deployment language
  4. Model environment and use model based deployment



Citrix XenApp Server Roles

XenApp Server Roles

XenApp server roles did not install

An error occured while installing XenApp Platinum Edition.

License Server failed.

License Server installed successfully.

XenApp was skipped.

XenApp Server was skipped.

XenApp Management was skipped.

XML Service IIS Integration was skipped.

Single Sign-On Plug-in was skipped.

Note:
An error occured during installation. Some roles may not have been installed. Please review the logs, and run Server Role Installer again.

Did the License server install work? or fail?

Next week…

- Deployment scripts and models in more detail