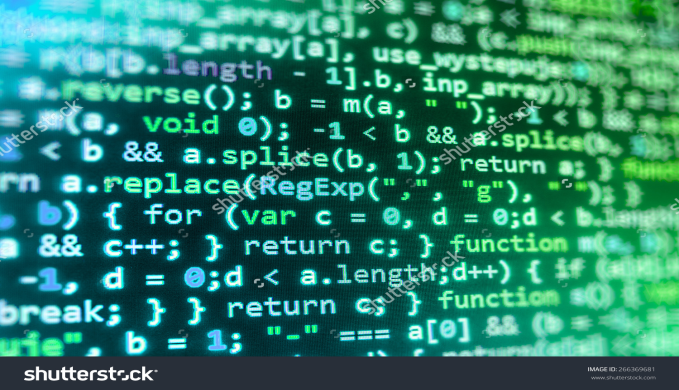
**Chapter - Six**

**Implementation, Testing, Maintenance and Deployment**

1. **Implementation (Coding)**

The actual task of developing the software starts here with data recording going on in the background. Once the software is developed, the stage of implementation comes in where the product goes through a pilot study to see if it’s functioning properly.

* Implementation = Software development or assembling according to previously created design.
  + Don’t confuse the term with “system deployment”



* Modularized design (objects and packages; components) – interaction among system components must be implemented using certain standard (SOAP/XML, CORBA, COM)\*
* Coding Standards give the program a common look and feel

**Coding standards typically address**

* Use of comments
* Variable names
* Function names
* Maximum length of a routine (lines of code)
* Maximum number of routines within a class
* Degree of complexity allowed (nested loops, compound boolean testing, etc)
* Naming convention of source code files
* Source code directory structure for developer machines, build machines and source code control tools
* Source code file contents (i.e. one C++ class per file)
* Ways to indicate incomplete code in source.

**Pair Programming**

A dialog between two people trying to simultaneously program (and analyze and design and test) and understand how to program better”.

* Encourages communication
* Reduces coding errors
* Improves code quality

(While one person types, the other thinks at a more strategic level – where is this line of development going? Is there an opportunity to refactor?)

Good software development organizations normally require their programmers to adhere to some well-defined and standard style of coding called coding standards. Most software development organizations formulate their own coding standards that suit them most, and require their engineers to follow these standards rigorously. The purpose of requiring all engineers of an organization to adhere to a standard style of coding is the following:

* A coding standard gives a uniform appearance to the codes written by different engineers.
* It enhances code understanding.
* It encourages good programming practices.

A coding standard lists several rules to be followed during coding, such as the way variables are to be named, the way the code is to be laid out, error return conventions, etc.

**Coding standards and guidelines**

Good software development organizations usually develop their own coding standards and guidelines depending on what best suits their organization and the type of products they develop. The following are some representative coding standards.  
**Rules for limiting the use of global:** These rules list what types of data can be declared global and what cannot.

**Contents of the headers preceding codes for different modules:** The information contained in the headers of different modules should be standard for an organization. The exact format in which the header information is organized in the header can also be specified. The following are some standard header data:

* Name of the module.
* Date on which the module was created.
* Author’s name.
* Modification history.
* Synopsis of the module.
* Different functions supported, along with their input/output parameters.
* Global variables accessed/modified by the module.

**Naming conventions for global variables, local variables, and constant** **identifiers:** A possible naming convention can be that global variable names always start with a capital letter, local variable names are made of small letters, and constant names are always capital letters.  
**Error return conventions and exception handling mechanisms:** The way error conditions are reported by different functions in a program are handled should be standard within an organization. For example, different functions while encountering an error condition should either return a 0 or 1 consistently. The following are some representative coding guidelines recommended by many software development organizations.

**Do not use a coding style that is too clever or too difficult to understand:** Code should be easy to understand. Many inexperienced engineers actually take pride in writing cryptic and incomprehensible code.

Clever coding can obscure meaning of the code and hamper understanding. It also makes maintenance difficult  
**Avoid obscure side effects:** The side effects of a function call include modification of parameters passed by reference, modification of global variables, and I/O operations. An obscure side effect is one that is not obvious from a casual examination of the code. Obscure side effects make it difficult to understand a piece of code. For example, if a global variable is changed obscurely in a called module or some file I/O is performed which is difficult to infer from the function’s name and header information, it becomes difficult for anybody trying to understand the code.

**Do not use an identifier for multiple purposes:** Programmers often use the same identifier to denote several temporary entities. For example, some programmers use a temporary loop variable for computing and a storing the final result. The rationale that is usually given by these programmers for such multiple uses of variables is memory efficiency, e.g. three variables use up three memory locations, whereas the same variable used in three different ways uses just one memory location. However, there are several things wrong with this approach and hence should be avoided. Some of the problems caused by use of variables for multiple purposes as follows:

**The code should be well-documented:** As a rule of thumb, there must be at least one comment line on the average for every three-source line.

**The length of any function should not exceed 10 source lines:** A function that is very lengthy is usually very difficult to understand as it probably carries out many different functions. For the same reason, lengthy functions are likely to have disproportionately larger number of bugs

**Do not use goto statements:** Use of goto statements makes a program unstructured and makes it very difficult to understand

**Code Review**

Code review for a model is carried out after the module is successfully compiled and the all the syntax errors have been eliminated. Code reviews are extremely cost-effective strategies for reduction in coding errors and to produce high quality code. Normally, two types of reviews are carried out on the code of a module. These two types’ code review techniques are code inspection and code walk through.

**Code Walk Through**

Code walk through is an informal code analysis technique. In this technique, after a module has been coded, successfully compiled and all syntax errors eliminated. A few members of the development team are given the code few days before the walk through meeting to read and understand code. Each member selects some test cases and simulates execution of the code by hand (i.e. trace execution through each statement and function execution). The main objectives of the walk through are to discover the algorithmic and logical errors in the code. The members note down their findings to discuss these in a walk through meeting where the coder of the module is present.

**Code Inspection**

In contrast to code walk through, the aim of code inspection is to discover some common types of errors caused due to oversight and improper programming. In other words, during code inspection the code is examined for the presence of certain kinds of errors, in contrast to the hand simulation of code execution done in code walk throughs. For instance, consider the classical error of writing a procedure that modifies a formal parameter while the calling routine calls that procedure with a constant actual parameter. It is more likely that such an error will be discovered by looking for these kinds of mistakes in the code, rather than by simply hand simulating execution of the procedure. In addition to the commonly made errors, adherence to coding standards is also checked during code inspection. Good software development companies collect statistics regarding different types of errors commonly committed by their engineers and identify the type of errors most frequently committed. Such a list of commonly committed errors can be used during code inspection to look out for possible errors. Following is a list of some classical programming errors which can be checked during code inspection:

* Use of uninitialized variables.
* Jumps into loops.
* Nonterminating loops.
* Incompatible assignments.
* Array indices out of bounds.
* Improper storage allocation and deallocation.
* Mismatches between actual and formal parameter in procedure calls.
* Use of incorrect logical operators or incorrect precedence among operators.
* Improper modification of loop variables.
* Comparison of equally of floating point variables, etc.

**Clean room testing**

Clean room testing was pioneered by IBM. This type of testing relies heavily on walk throughs, inspection, and formal verification. The programmers are not allowed to test any of their code by executing the code other than doing some syntax testing using a compiler. The software development philosophy is based on avoiding software defects by using a rigorous inspection process. The objective of this software is zero-defect software.

**Software Documentation**

When various kinds of software products are developed then not only the executable files and the source code are developed but also various kinds of documents such as users’ manual, software requirements specification (SRS) documents, design documents, test documents, installation manual, etc are also developed as part of any software engineering process. All these documents are a vital part of good software development practice.

Good documents are very useful and server the following purposes:

* Good documents enhance understandability and maintainability of a software product. They reduce the effort and time required for maintenance.
* Use documents help the users in effectively using the system.
* Good documents help in effectively handling the manpower turnover problem. Even when an engineer leaves the organization, and a new engineer comes in, he can build up the required knowledge easily.
* Production of good documents helps the manager in effectively tracking the progress of the project. The project manager knows that measurable progress is achieved if a piece of work is done and the required documents have been produced and reviewed.

Different types of software documents can broadly be classified into the following:

• Internal documentation

• External documentation

**Internal documentation** is the code comprehension features provided as part of the source code itself. Internal documentation is provided through appropriate module headers and comments embedded in the source code. Internal documentation is also provided through the useful variable names, module and function headers, code indentation, code structuring, use of enumerated types and constant identifiers, use of user-defined data types, etc. Careful experiments suggest that out of all types of internal documentation meaningful variable names is most useful in understanding the code. This is of course in contrast to the common expectation that code commenting would be the most useful.

Some research finding is obviously true when comments are written without thought. For example, the following style of code commenting does not in any way help in understanding the code.  
**a = 10; /\* a made 10 \*/**

But even when code is carefully commented, meaningful variable names still are more helpful in understanding a piece of code. Good software development organizations usually ensure good internal documentation by appropriately formulating their coding standards and coding guidelines.

**External documentation** is provided through various types of supporting documents such as users’ manual, software requirements specification document, design document, test documents, etc. A systematic software development style ensures that all these documents are produced in an orderly fashion

### System Testing

* **Testing** = a process of identifying system defects
* Develop test cases and test data
  + A test case specifies starting & ending state, and events to which the software must respond
  + Test data – data in the defined starting state, which must reach a defined ending state after being exposed to certain event (input).
  + Modern approach: build to predefined test (Agile Methodology)

When testing systems built using object technology it is important to understand that your ***source code*** is composed of several constructs, including methods (operations), classes, and inheritance relationships. The following are object oriented testing techniques that used to test **system.**

**Testing Your System methods and classes**

1. **Method testing.** Method testing is the act of ensuring that your methods, called operations or member functions in C++ and Java, perform as defined. The closest comparison to method testing in the structured world is the unit testing of functions and procedures. Issues to address during method testing include the following:
   * Ensuring that your getter and setter methods manipulate the value of a single property work as intended;
   * Ensuring that each method returns the proper values, including error messages and exceptions;
   * Basic checking of the parameters being passed to each method; and
   * Ensuring that each method does what the documentation says it does.
2. **Class testing/Unit testing.** It is unit testing because you are testing the class and its instances as single units in isolation. An important class test is to validate that the attributes of an object are initialized properly.
3. **Class-integration testing/integration testing.** Also known as component testing, this technique addresses the issue of whether the classes in your system, or a component of your system, work together properly. The only way classes or, to be more accurate, the instances of classes, can work together is by sending each other messages. Therefore, some sort of relationship must exist between those objects before they can send the message, implying that the relationships between classes can be used to drive the development of integration test cases. In other words, your strategy should be to look at the association, aggregation, and inheritance relationships that appear on your class diagram and in formulating class-integration test cases.

**Testing Your System in its Entirety**

System testing is a testing process in which you aim to ensure that your overall system works as defined by your requirements. System testing comprises the following techniques:

1. **Function testing.**In function testing, development staff verifies that their application meets the defined needs of their users. The idea is that developers, typically test engineers, work through the main functionality that the system should exhibit to assure themselves that their application is ready for user-acceptance testing (UAT).
2. **Installation testing.** The goal is to determine whether your application can be installed successfully. The installation utility/process for your application is part of your overall application package and, therefore, must be tested. Several important issues should be considered:
   * Can you successfully install the application into an environment that it has not been installed into before?
   * Can you successfully install the application into an environment where it, or a previous version, already exists?
   * Is configuration information defined correctly?
   * Is previous configuration information taken into account?
   * Is online documentation installed correctly?
   * Are other applications affected by the installation of this one?
   * Are there adequate computer resources for the application? Does the installation utility detect this and act appropriately?
3. **Operations testing.** The goal of operations testing is to verify that the requirements of operations personnel are met. The main goal of operations testing is to ensure that your operations staff will be able to run your application successfully ***once it is installed.***
4. **Stress testing.** Sometimes called **volume testing,** this is the process of ensuring that your application works with high numbers of users, high numbers of transactions (testing of high numbers of transactions is also called volume testing), high numbers of data transmissions, high numbers of printed reports, and so on. The goal is to find the stress points of your system under which it no longer operates, so you can gain insights into how it will perform in unusual and/or stressful situations.
5. **Unit testing** - individual methods, classes, or components before they are integrated with other software
   * Defect example: A method for calculating taxes outputs incorrect results.
6. **Integration testing** - evaluates the behavior of a group of methods or classes (interface compatibility, unexpected parameter values or state interaction, run-time exceptions)
   * Applied to system or sub-system
   * Defect example: Results of retrieving a record significantly differ when real database as opposed to when dummy database is retrieved.

**Testing by Users**

User testing, which follows system testing, is composed of testing processes in which members of your user community perform the tests. The goal of user testing is to have the users verify that an application meets their needs.

User testing comprises the following techniques:

1. **Alpha testing.** Alpha testing is a process in which you send out software that is not quite ready for prime time to a ***small group of your customers*** to enable them work with it and report back to you the problems they encounter. Although the software is typically buggy and may not meet all their needs, they get a heads-up on what you are doing much earlier than if they waited for you to release the software formally.
2. **Beta testing.** Beta testing is basically the same process as alpha testing, except the software has many of the bugs identified during alpha testing (beta testing follows alpha testing) fixed and the software is distributed to a ***larger group.*** The main goal of both alpha and beta testing is to test run the product to identify and then fix any bugs before you release your application.
3. **Pilot testing.** Pilot testing is the ***"in-house" version of alpha/beta testing***, the only difference being that the customers are typically ***internal to your organization***. Companies that sell software typically use alpha/beta test, whereas IT organizations that produce software for internal use will pilot test.
4. **User-acceptance testing (UAT).** After your system testing proves successful, your users must perform user-acceptance testing, a process in which they determine whether your application truly meets their needs. This means you have to let your users work with the software you produced. Because the only person who truly knows your own needs is you, the people involved in the user-acceptance test should be the actual users of the system—the people who will work daily with the application. Although you may have to give them some training to gain the testing skills they need, actual *users* are the only people who are qualified to do *user*-acceptance testing. The good news is, if you have function tested your application thoroughly, then the UAT process will take only a few days to a week at the most.
5. **Usability testing** – system works but fails some of the usability requirements (easy to learn, use & navigate, task efficiency); user interface focus. Can be part of Acceptance testing.

**Software Testers**

* Programmers
  + Unit testing
  + Testing buddies can test other’s programmer’s code
* Quality assurance personnel
  + Integration, Usability, Acceptance
* Users
  + Usability, Acceptance testing
  + Volunteers for testing beta version of software

1. **Software Maintenance**

* Is the set of activities, both technical and managerial, that ensures that software continues to meet organizational and business objectives in a cost effective way.
* Difference between Software Product and Software Maintenance is:
  + Software Product is the result of the Software development.
  + Software Maintenance results in a service being delivered to the customer.

**Software Maintenance Objectives**

* Corrective,

**These are also Maintenance Types**

* Adaptive,
* Perfective,
* Inspection*.*

**Corrective:** Taking existing code and correcting a fault that causes the code to behave in some way that deviates from its documented requirements

**Adaptive:** Taking existing code and adapting it to provide new features and functionality. These are typically part of a new release of the code and part of a larger development effort.

**Perfective:** These are typically made to improve the maintainability of the code such as restructuring it to make it more easily understood or to remove ambiguities.

**Inspection:** These are usually made as a result of code inspections and focus more of adhering to coding standards or to reduce the likelihood of a failure.

**Problems of Maintenance**

* Maintenance Attributes,
* Maintenance Organization,
* Respect of Metrics,
* Requirements volatility.

**Maintenance Attribute**

* five factors drive the difficulty of delivering software:
  + Product ,
  + Documentation,
  + End users,
  + Process,
  + Environment.
* Important attributes to be considered during Maintenance:
  + product age,
  + design,
  + Language,
  + current failure rate,
  + Staff experience.

**Maintenance Organization**

* five factors drive the difficulty of delivering software:
  + Input Vs. output,
  + cycle-time,
  + cost/change,
  + schedule,
  + flexibility,
  + Quality.

**Respect Maintenance**

* Software maintenance should be measured and managed using metrics to reach a quality software.
* However, we don't know how to measure maintainability because it’s a service.
* Approaches were made to get values that can be useful during maintenance (surveys).

**Requirements Volatility**

* Requirements are the foundation of the software release process.
  + Changing requirements during the software maintenance process impacts the cost, schedule, and quality of the resulting product.
  + Build model to make planning of customer communications (predictions).
* A focus is made on non-volatile requirements.

**Design for Maintenance**

* A strategy to set a certain rules during the Software development.
* It eases the maintability of the system.

**Design Attributes**

* Three main Factors that we have to ensure during the design of the Software:
  + Understandability,
  + Modifiability,
  + Stability.

**Design Strategies**

* Reasons to do a good Design in order to ease maintability:
  + Low Cost for future maintenance,
  + Avoid long time periods of maintenance,
  + Ability to discover rapidly the failures.

**Design Data Collection**

* Collecting some characteristics during the commencement of the project about :
  + The Behavior of the requirements,
  + Case tool and the Programming Language used,
  + The environment.
* Emergency Rework on Modules
  + Design independent modules in order to substitute them in failures and correct the deficiencies rapidly.
  + Design a structure with independent components in order to be easy in maintenance

**Key Points about Maintenance**

* It’s very hard to maintain a system than to design it.
* It’s even harder to design a maintainable system, because it’s difficult to predict future changes in the business environment.
* Software maintenance is a critical activity in the life cycle of a system.

1. **Deployment**

One of the biggest application life cycle management challenges is deployment phase.

Al the activities that make a software system available for use.

Activity responsible for movement of approved releases to test and production environment.

**Release:** a collection of hardware, software, documentation, processes or other components required to implement one or more approved changes to IT services. The contents of each release are managed, tested and deployed as a single entity.

**Release and Deployment management**

Process responsible for planning, scheduling and controlling the movement of releases to test and production environments.

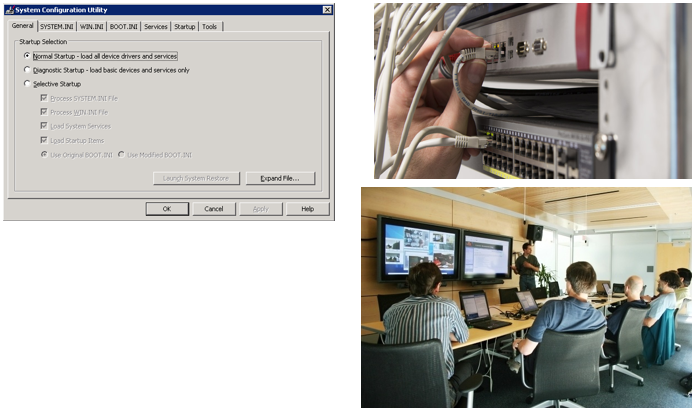
**Objectives of Deployment**

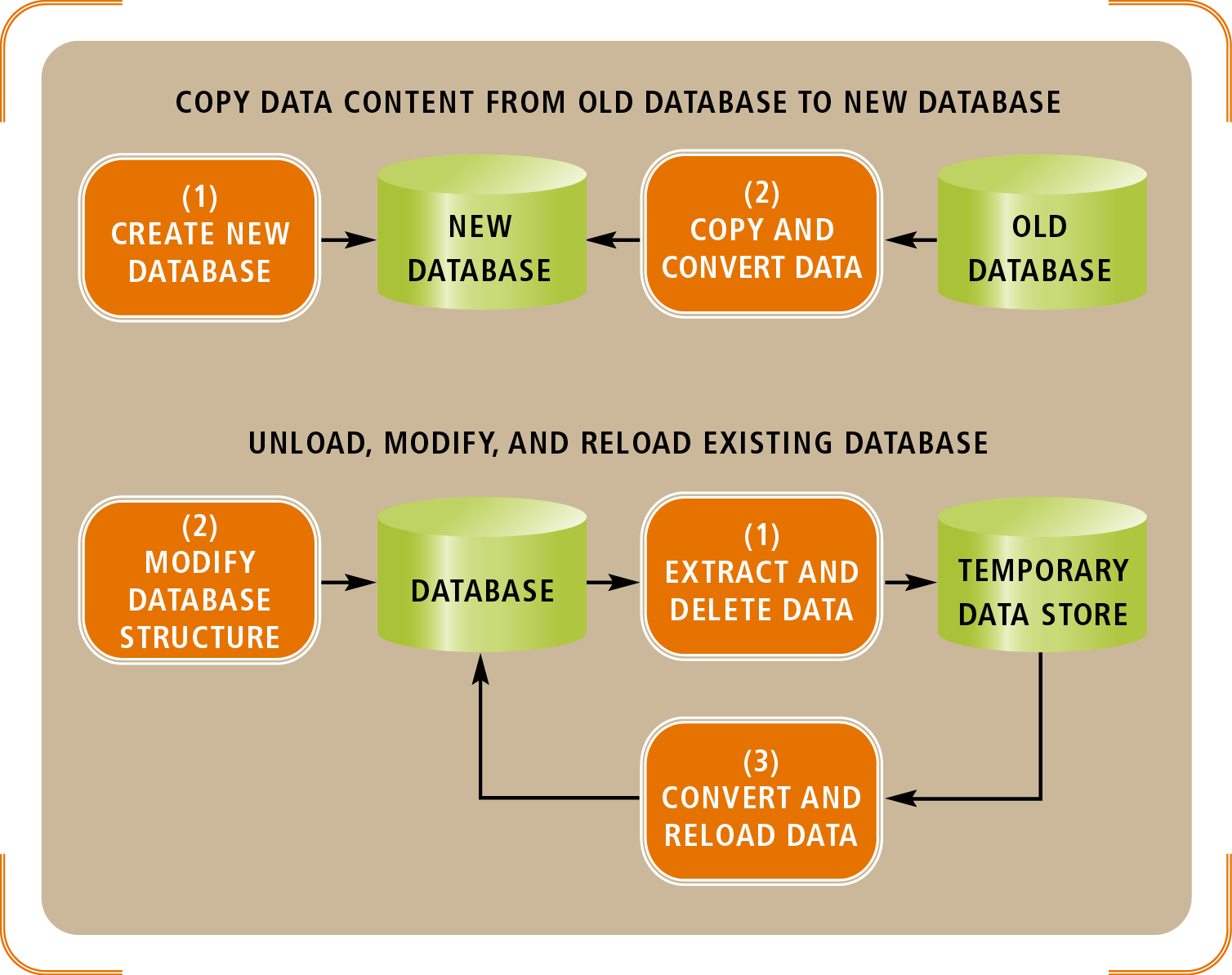
* Efficiently build, install, test and deploy releases to a target environment successfully and on schedule.
* Clear and comprehensive release and deployment plans.
* Controlled deployment of a new or changed service into production
* Minimal unpredicted impact on the production services, operations and support organization.

**Issues Related to Deployment**

* Selection of the knowledge base problem
* Ease of understanding the KM System
* Knowledge transfer
* Integration alternatives
* The issue of maintenance
* Organizational factors

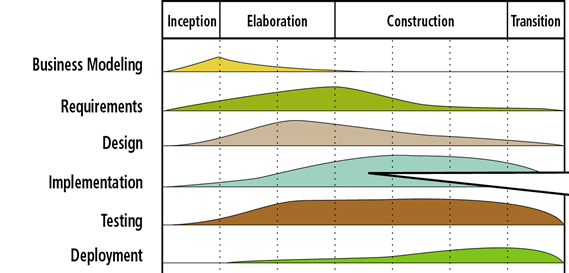
**User Training and Deployment**

* Preparing for KM system training via advance demos and easy to follow training
* Combating resistance to change
* Watch for knowledge hoarders
* Watch for troublemakers and narrow-minded “superstars”
* Look for resistance via projection, avoidance, and aggression
* Activities for making a new system operational
* **System Deployment Include:**
  + Installation and configuration
  + User training
  + Porting and Converting data
  + Deployment strategy

**Data porting & converting**

**Deployment strategy**

* Can be defined more clearly with the structured methodology than with rapid methodologies of system development
  + With rapid methodologies deployment is gradual, module by module



* + Agile approach: IT Staging - shuffling between build testing, development and deployment. Finally deployed parts are said to be in production.

In general, software deployment is the final releasing of software product as a service for users.