COMP3211

02. Software Processes

Dr. Max Yu Pei

Intended Learning Outcomes

- Upon completion of this lecture, you will
  1. understand the concepts of software processes and software process models;
  2. have been introduced to three general software process models and when they might be used;
  3. know about the fundamental process activities of software requirements engineering, software development, testing, and evolution;
  4. understand why processes should be organized to cope with changes in the software requirements and design;
  5. understand the notion of software process improvement and the factors that affect software process quality.

Software Process

- A structured set of activities required to develop a software system.
- There are many different software processes, but they all involve:
  - Specification – defining what the system should do;
  - Development – producing the software that meets the specification;
  - Validation – checking that it does what it should do;
  - Evolution – changing the system in response to changing needs.

Software Process Descriptions

- When we describe and discuss processes, we usually talk about the activities in these processes such as specifying a data model, designing a user interface, etc., and the ordering of these activities.

- Process descriptions may also include:
  - Products, or deliverables, which are the outcomes of a process activity;
  - Roles, which reflect the responsibilities of the people involved in the process;
  - Pre- and post-conditions, which are statements that are true before and after a process activity has been enacted or a product produced.

Plan-Driven and Agile Processes

- Plan-driven processes are processes where all of the process activities are planned in advance and progress is measured against this plan.

- In agile processes, planning is incremental, and changing the process to reflect changing customer requirements is easier.
  - In practice, most practical processes include elements of both plan-driven and agile approaches.
  - There are no right or wrong software processes.

SOFTWARE PROCESS MODELS

The Waterfall Model

- There are separate identified phases in the waterfall model, and in principle, a phase must be completed before moving on to the next phase
  - Requirements analysis and definition
  - System and software design
  - Implementation and unit testing
  - Integration and system testing
  - Operation and maintenance

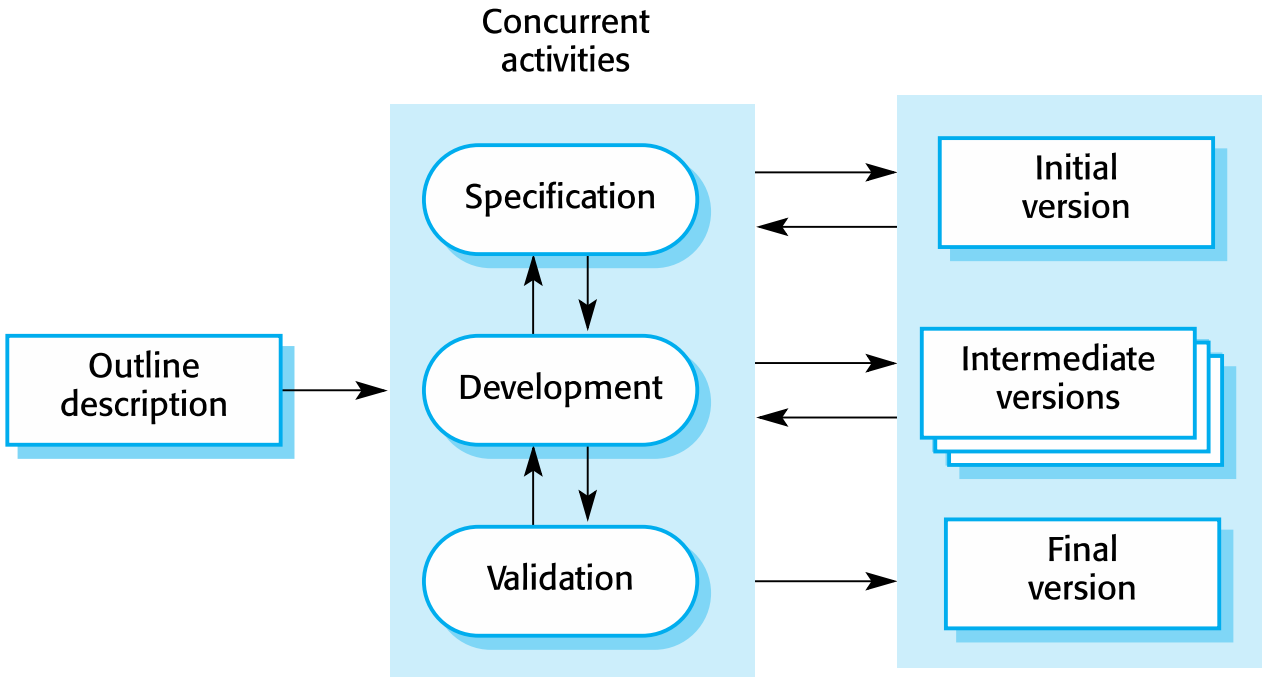
Waterfall Model Problems

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.
  - Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
  - Few business systems have stable requirements.

- Types of systems for which the waterfall model is appropriate
  - Embedded systems where the software has to interface with hardware systems
  - Critical systems where there is a need for extensive safety and security analysis of the software specification and design
  - Large software systems that are part of broader engineering systems developed by several partner companies

Incremental Development

- Incremental development is based on the idea of developing an initial implementation, getting feedback from users and others, and evolving the software through several versions until the required system has been developed
  - This approach can be either plan-driven, agile or, more usually, a mixture of these approaches.
  - You don't have to deliver each increment to the system customer.



Incremental Development Benefits and Problems

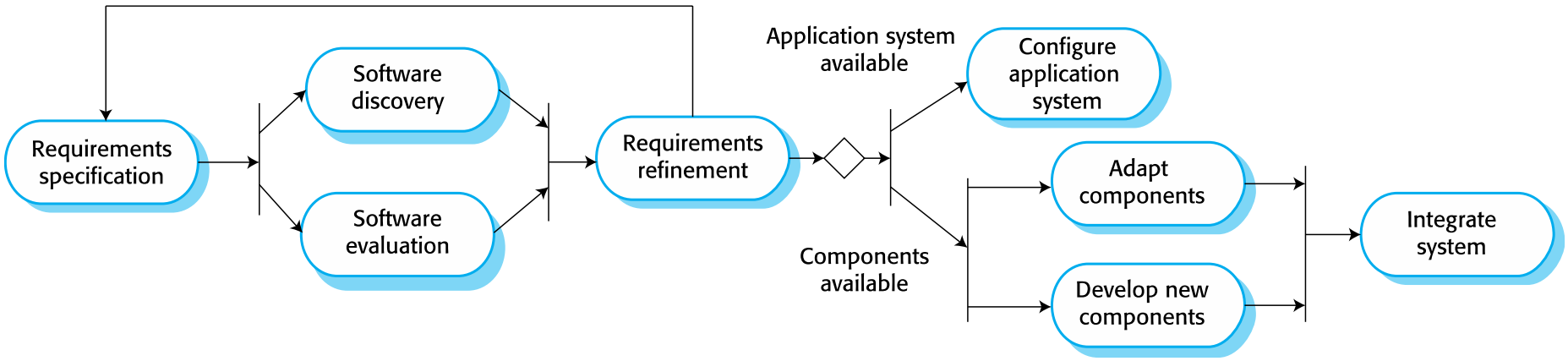
- Benefits
  - It is easier to get customer feedback on the development work that has been done.
  - The cost of accommodating changing customer requirements is reduced.
  - More rapid delivery and deployment of useful software to the customer is possible.

- Problems
  - There may be a mismatch between the organizations' bureaucratic procedures and a more informal iterative or agile process
  - System structure tends to degrade as new increments are added.

Integration and Configuration

- Based on software reuse where systems are integrated from existing components or application systems, or COTS (Commercial-off-the-shelf) systems.
  - Reused elements may be configured to adapt their behaviour and functionality to a user's requirements
  - Reuse is now the standard approach for building many types of business systems
- Types of reusable software
  - Stand-alone application systems that are configured for use in a particular environment.
  - Collections of objects that are developed as a component or a package to be integrated with frameworks like .NET or J2EE.
  - Web services that are developed according to service standards and that are available for remote invocation.

Reuse-Oriented Software Engineering



- Advantages and disadvantages
  - Reduced costs and risks as less software is developed from scratch
  - Faster delivery and deployment of systems
  - Requirements compromises are inevitable so the system may not meet the real needs of users
  - Loss of control over the system evolution

PROCESS ACTIVITIES

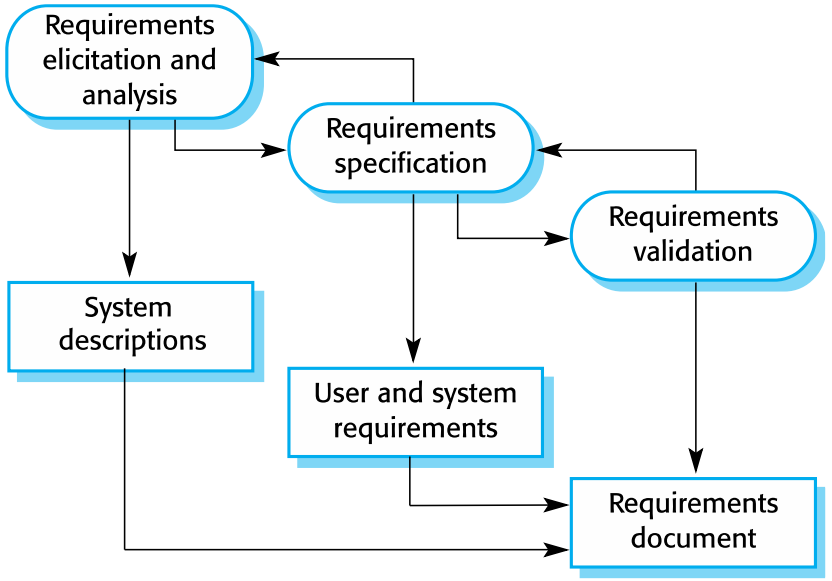
Process Activities

- Real software processes are inter-leaved sequences of technical, collaborative and managerial activities with the overall goal of specifying, designing, implementing and testing a software system.
- The four basic process activities of *specification*, *development*, *validation* and *evolution* are organized differently in different development processes.
- For example, in the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.

Software Specification

- The process of establishing what *services* are required and identifying the *constraints* on the system's operation and development.

- Requirements engineering process
  - Requirements elicitation and analysis
    - What do the system stakeholders require or expect from the system?
  - Requirements specification
    - Defining the requirements in detail
  - Requirements validation
    - Checking the validity of the requirements



The requirements engineering process

Software Development

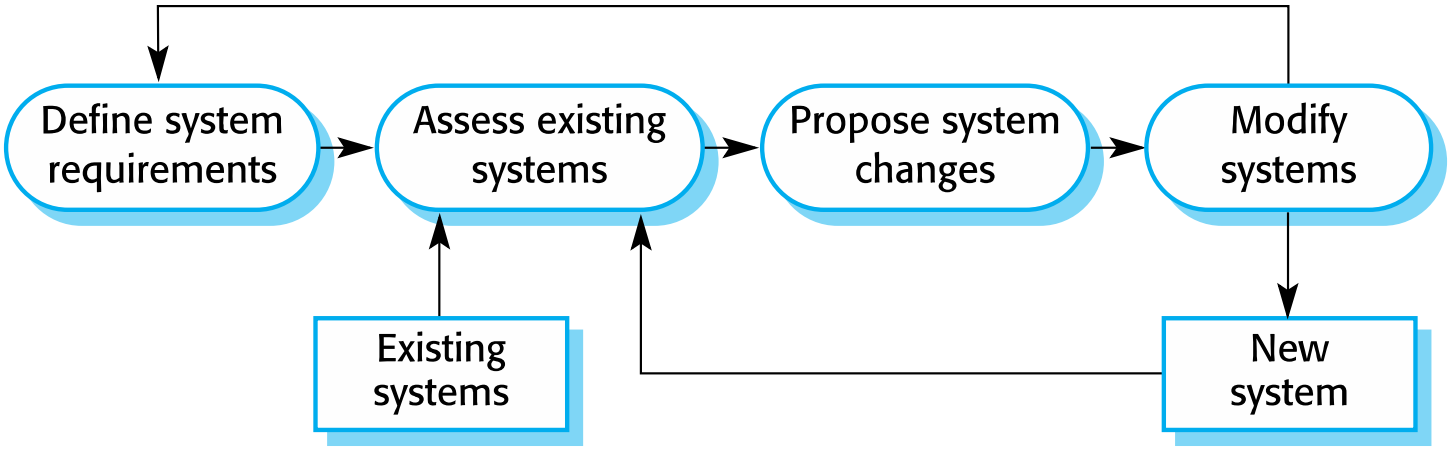
- The process of developing an executable system for delivery to the customer.
  - Design and implementation are interleaved activities for most types of software systems.
  - Software design
    - Design a software structure to realize the requirements
    - Architectural design, database design, interface design, component selection and design, …
  - Implementation
    - Translate this structure into an executable program
    - Programming is an individual activity with no standard process

Software Validation

- Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the system customer.
  - It involves checking processes (such as inspections and reviews) and system testing (static vs. dynamic)
  - Testing is the most commonly used V & V activity.
  - Defect testing and debugging are different processes. Testing establishes the existence of defects. Debugging is concerned with locating and correcting these defects.

Software Evolution

- Software is inherently flexible and can change.
  - As requirements change, the software that supports the business must also evolve and change.
  - Although there has been a demarcation between development and evolution (maintenance), the distinction is increasingly irrelevant as fewer and fewer systems are completely new.
  - It is more realistic to think of software engineering as an evolutionary process where software is continually changed in response to changing requirements and customer needs.



COPING WITH CHANGE

Coping with Change

- Change is inevitable in all large software projects.
  - A software product is a model of the real world, which is continually changing
  - Software professionals are human, and they make mistakes.
- The moving target problem: The requirements change while the software product is being developed
  - Any change made to a software product can potentially cause a regression fault
  - If there are too many changes, the entire product may have to be redesigned and reimplemented
  - There is no solution to the moving target problem
- Change leads to rework so the costs of change include both rework (e.g., re-analyzing requirements) as well as the costs of implementing new functionality

Reducing The Costs of Rework

- Two related approaches
  - Change anticipation, where the software process includes activities that can anticipate possible changes before significant rework is required.
    - For example, a prototype system may be developed to show customers some key features of the system.
  - Change tolerance, where the process is designed so that changes can be accommodated at a relatively low cost.
    - This normally involves some form of incremental development.
    - The longer a change is delayed, the more it costs to implement.

Two Ways of Coping with Changing Requirements

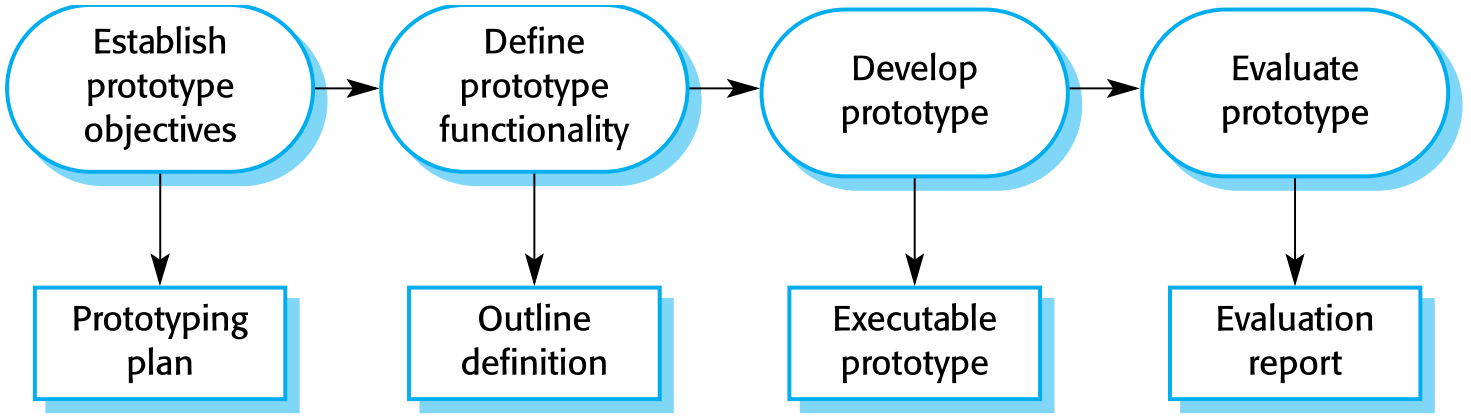
- System prototyping
  - A version or part of the system is developed quickly to check the customer's requirements and the feasibility of design decisions.
  - This is a method of change anticipation as it allows users to experiment with the system before delivery and refine their requirements.

- Incremental delivery
  - System increments are delivered to the customer for comment and experimentation.
  - This supports both change anticipation and change tolerance.

Software Prototyping

- A prototype is an initial version of a system used to demonstrate concepts, try out design options, and find out more about the problem and its possible solutions.
- A prototype can be used
  - in the requirements engineering process to help with requirements elicitation and validation;
  - in the design process to explore options and develop a UI design;
- Benefits
  - Improved system usability.
  - A closer match to users' real needs.
  - Improved design quality.
  - Improved maintainability.
  - Reduced development effort.

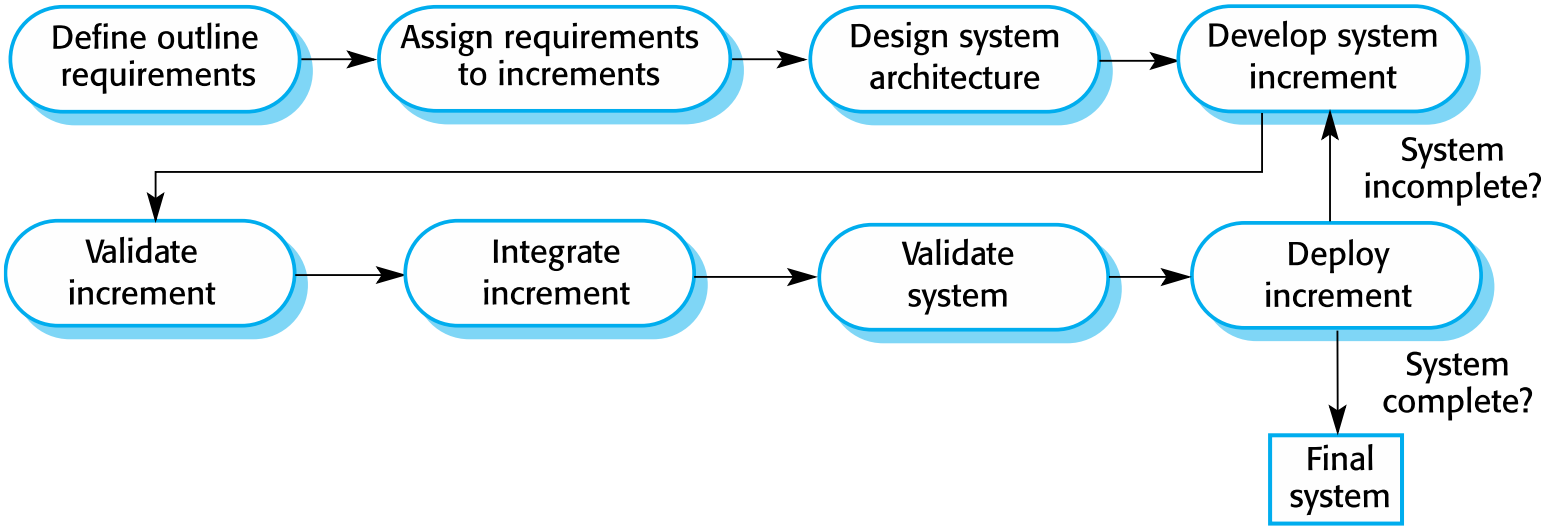
Prototype Development



- May be based on rapid prototyping languages or tools
- May involve leaving out functionality
  - Prototype should focus on areas of the product that are not well-understood;
  - Error checking and recovery may not be included in the prototype;
  - Focus on functional rather than non-functional requirements such as reliability and security

Incremental Delivery

- The development and delivery is broken down into increments with each increment delivering part of the required functionality.
  - User requirements are prioritised, and the highest priority requirements are included in early increments.
  - Once the development of an increment is started, the requirements are frozen though requirements for later increments can continue to evolve.



Incremental Delivery Advantages and Problems

- Advantages
  - Early increments act as a prototype to help elicit requirements for later increments.
  - Customer value can be delivered with each increment, so system functionality is available earlier.
  - It should be relatively easy to incorporate changes into the system.
  - The highest priority system services tend to receive the most testing.
- Problems
  - Incremental delivery is difficult to implement for replacement systems as increments have less functionality than the system being replaced.
  - Most systems require a set of basic facilities that are used by different parts of the system, but common facilities can be difficult to identify with incremental delivery.
  - Its essence is that the spec is developed in conjunction with the software, which conflicts with the procurement model of many organizations

PROCESS IMPROVEMENT

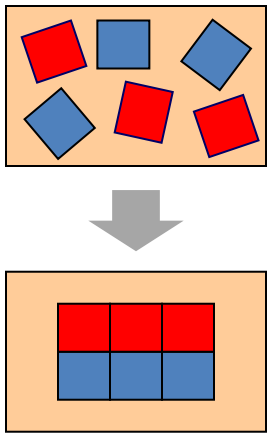
Software Process Performance and Capability

- Software Process Performance
  - Represents the actual results *achieved* by following a software process.
  - E.g., our actual defect rate is 2 defect/KLOC
  - Software process performance focuses on the achieved results

- Software Process Capability
  - Describes the expected results that *can be achieved* by following a software process.
  - E.g., the final defect rate will likely be 1.5 defect/KLOC
  - Software process capability focuses on the expected results

Software Process Maturity

- Process Maturity
  - It is the extent to which an organization's processes are explicitly defined, managed, measured, controlled, and effective.
  - Maturity implies a potential for growth in capability
  - Immature software organization
    - Processes being improvised by practitioners
    - Managers being forced to solve immediate crises (fire fighting)
    - No objective basis for judging quality
  - Mature software organization
    - Processes are documented, usable, and consistent with the way the work is actually done



Quality Assurance (QA)

- Quality Assurance
  - the identification, planning, adoption, and implementation of the disciplines and actions (the process) necessary to assure that the product satisfies its users.
  - Focus on improving the process
  - Use error and trend analysis to locate weaknesses in:
    - Development process
    - Test process
    - Maintenance process
    - Support process (Quality, Configuration management)

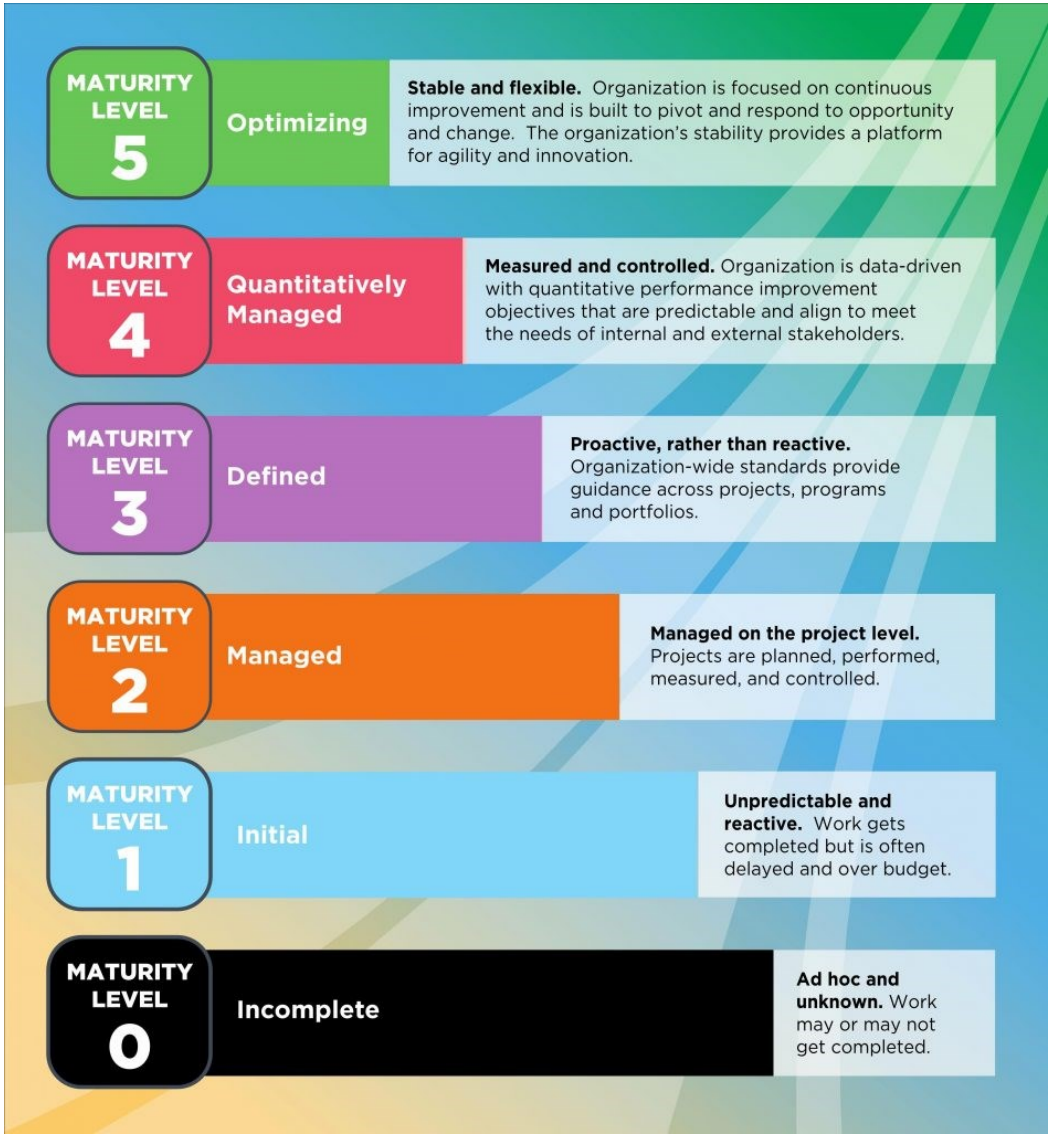
CMMI (Capability Maturity Model Integration)

- CMMI is a framework to improve product quality and development efficiency for systems (both hardware and software)
  - CMMI has been established as a model to improve business results
  - Many companies use CMMI

  - CMMI supports 2 representations: staged and continuous. To initiate the process improvement, an organization must first select a representation.
    - A process area is a cluster of related practices in an area that, when implemented collectively, satisfies a set of goals considered important for making improvement in that area.

Staged Representation of CMMI

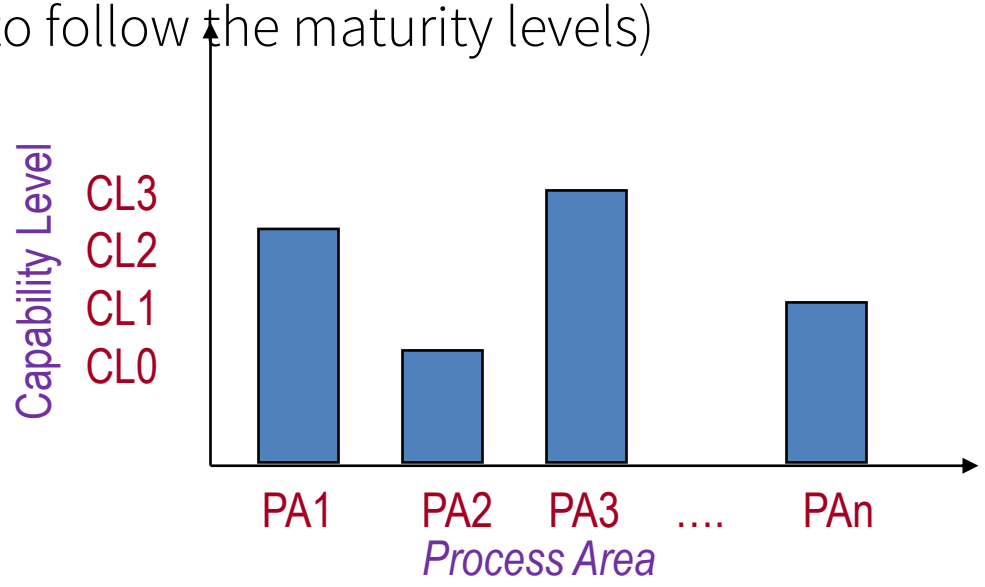
- Staged models provide a proven sequence of improvements, beginning with basic management practices and progressing through successive levels.



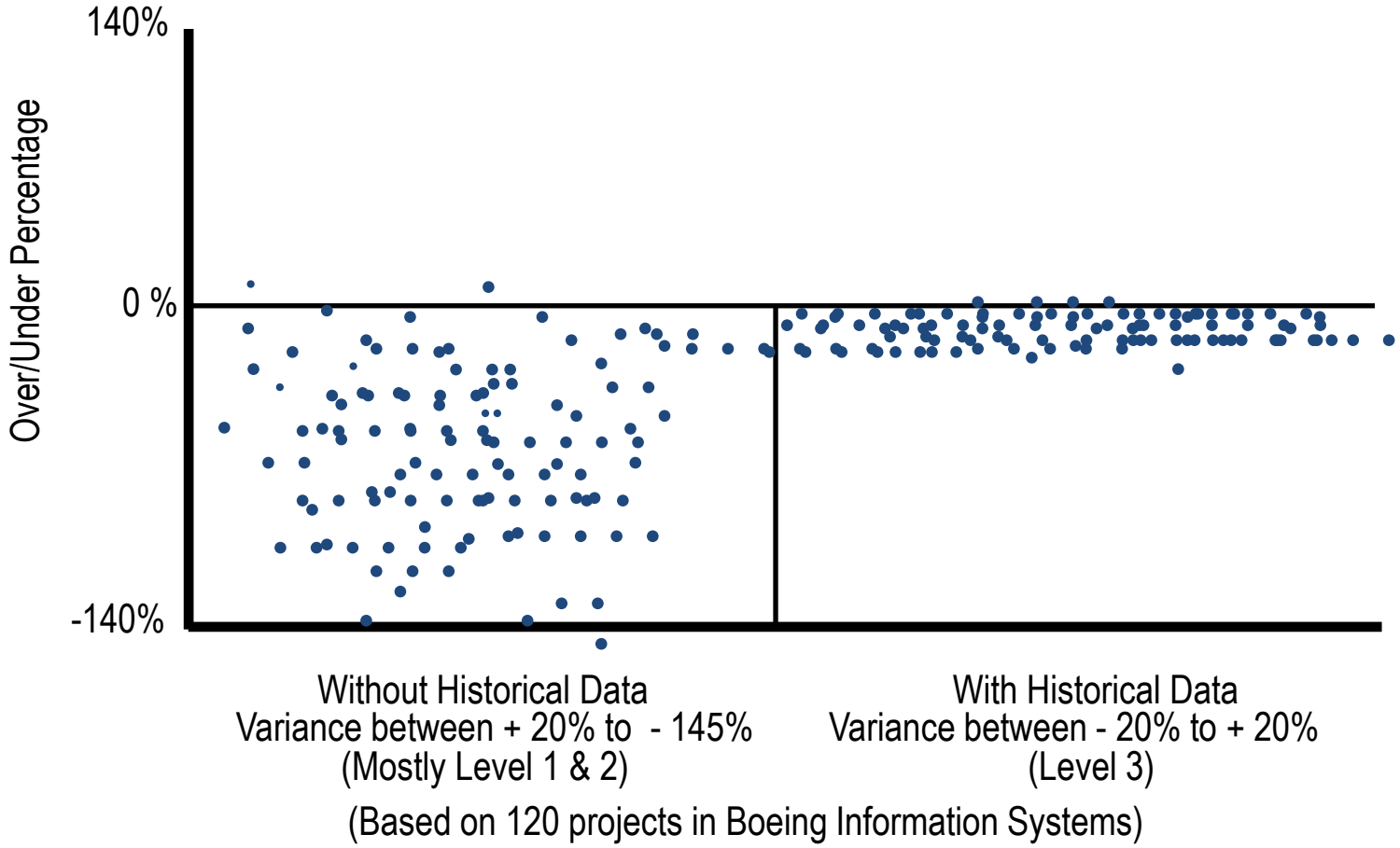
Continuous Representation of CMMI

- Continuous representation designates capability levels (numbered 0 through 3: incomplete, performed, managed, and defined) for process improvement within each process area.
  - An organization may select the order of improvement that best meets its business objectives and mitigates the organization's areas of risk. (No need to follow the maturity levels)

No need to follow a particular sequence in implementing the process areas.



Improved Schedule and Budget Predictability



(Based on 120 projects in Boeing Information Systems)

Ref: John D. Vu. "Software Process Improvement Journey: From Level 1 to Level 5." 7th SEPG Conference, San Jose, March 1997.

Key Points (1)

- Software processes are the activities involved in producing a software system. Software process models are abstract representations of these processes.
- General process models describe the organization of software processes.
  - Examples of these general models include the ‘waterfall’ model, incremental development, and reuse-oriented development.
- Requirements engineering is the process of developing a software specification.

Key Points (2)

- Design and implementation processes are concerned with transforming a requirements specification into an executable software system.
- Software validation is the process of checking that the system conforms to its specification and that it meets the real needs of the users of the system.
- Software evolution takes place when you change existing software systems to meet new requirements. The software must evolve to remain useful.
- Processes should include activities such as prototyping and incremental delivery to cope with change.

Key Points (3)

- Processes may be structured for iterative development and delivery so that changes may be made without disrupting the system as a whole.
- The principal approaches to process improvement are agile approaches, geared to reducing process overheads, and maturity-based approaches based on better process management and the use of good software engineering practice.
- The CMMI model identifies maturity levels that essentially correspond to the use of good software engineering practice.

THE END