Software processes

Vũ Thị Hồng Nhạn

(vthnhan@vnu.edu.vn)

Dept. of Software Engineering, FIT, UET

Vietnam National Univ., Hanoi

Contents

- Software process vs. software development processes
- Fundamental activities
- Coping with change
- Process improvement

Software process

- is **a structured set of activities** that cover **all aspects of software development** from early requirement for that system to development deployment, evolution after the system has gone into use and finally decommissioning of that system

- There are several **formal software processes** but there are 4 fundamental activities in all these processes (discuss later), sometimes in different forms
  - **Specification:** defining what the software should do
  - **Design:** defining the organization and structure of the system
  - **Implementing & testing**: developing the programs and testing they are free of bugs and meet the requirements
  - **Evolution:** changing the system in response to changing the customer needs

Plan-driven approach vs. Agile approach

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

- **Agile processes** don't have a detailed project plan, but rather software's developed in a series of increments with the functionality of each increment dependent on overall progress in the development

Software development processes

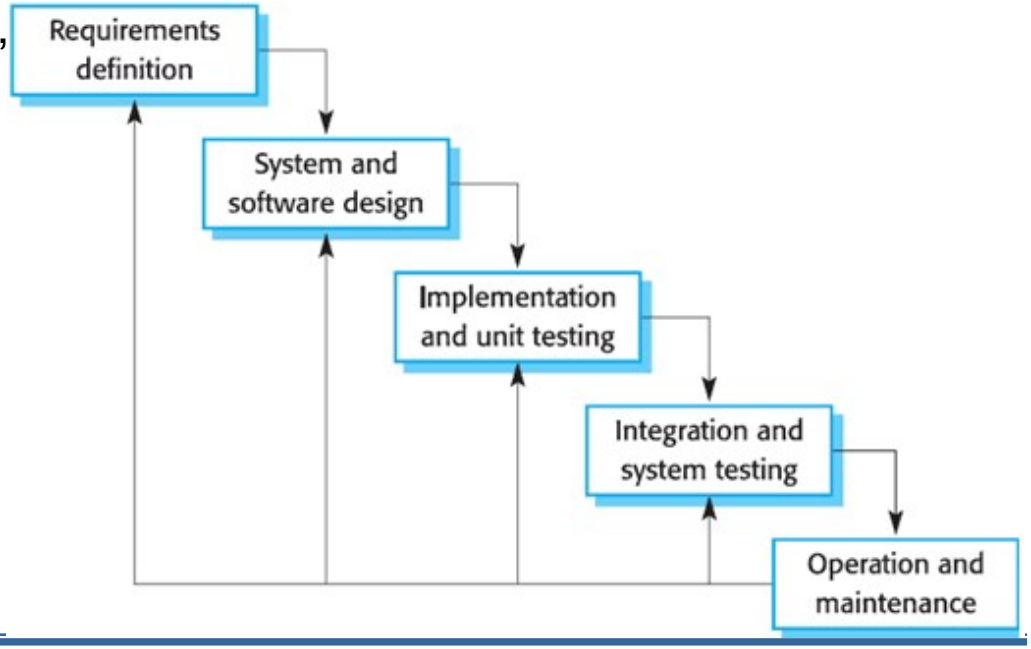
Software development processes

- The waterfall model (applies)
  - **Plan-driven (approach)**
- Incremental development
  - May be **plan-driven** or **agile**
- Integration and configuration
  - The system is assembled from existing configurable components
  - May (also) be **plan-driven** or **agile**

Waterfall model

- The original, best-known and still wildly used **plan-based process**, was introduced around 1970
- This came from **a standard engineering process** which has been used across the engineering industry for many years

Waterfall model has several phases, including ….



Problems with waterfall model

- It's a **document-driven process**, so each of these phases in the process gives us **an output**,
  - And if any change happens at a later phase, it requires a lot of work to modify the **documents** earlier in the process
  - This often means that we would freeze the software after we finish the development process
    ➔ This way would **avoid** making changes even if these changes would come with some benefits to the company or organization who buy the software)

Problems with waterfall model...

- Waterfall model is only suitable when the requirements are well-known, i.e.
  - We're not gonna have a lot of changes in that system →
  - However, most business systems change very quickly! →So the assumption of **no change** is not typical of most business systems
- Waterfall process is widely used in engineering projects where a system is developed at several sites by different teams
  - It's very difficult to organize an informal coordination,

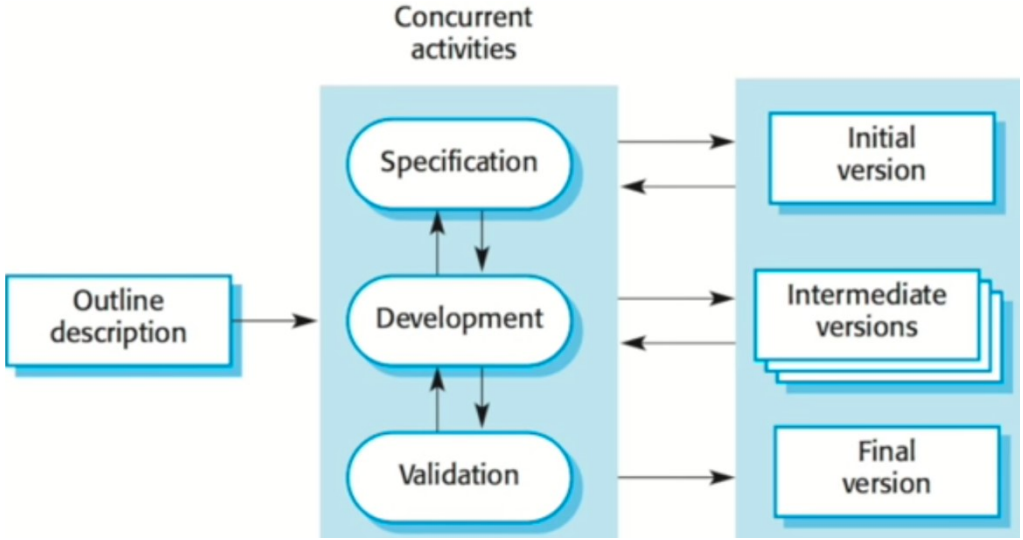
  → So **the document-driven process** is used to coordinate the work, **but still** there are often elements of agile development involved

Agile processes

- **Agile methods** are incremental or iterative approach to development based on series of iterations which typically lasts 2 or 4 weeks, each of which develops some discrete units of system functionality
- **The aim** of agile methods is to minimize documentation and communication overhead **to focus** simply on developing the operational program
- It's an approach that is generally much more responsive to **change**
- **The goal** is to deliver **useful functionality** to users more quickly than using the waterfall process

Agile processes

- in Agile development, we still have **the activities of** specification, design & implementation, and validation
- **But instead** of happening in a sequence as we've seen in the waterfall model these are interleaved activities
  - We do *a little bit of specification, a little bit of design and a little bit of implementation*, and then go around and repeat that process again and again
- **Agile methods have several benefits.**
  - The cost of implementing *changes to customer requirements* **is reduced**
  - The amount of *analysis and documentation* that has to be redone is **less** than it is required with the waterfall model



  - Easy to get customer feedback because there's always a workable version of the system → Customers can see what's going on, can check if it's what they want

However, agile approach has some problems

- The process is not visible because documents are avoided
  - So, it's hard to know how the system is progressing
  - Also, it's difficult to coordinate activities across different teams
- Unless time & money is spent on **refractoring** to improve the software, **regular change** usually corrupts its structure

- **Incremental development**
  - May be **plan-driven** or **agile**

Process based on Integration & reuse

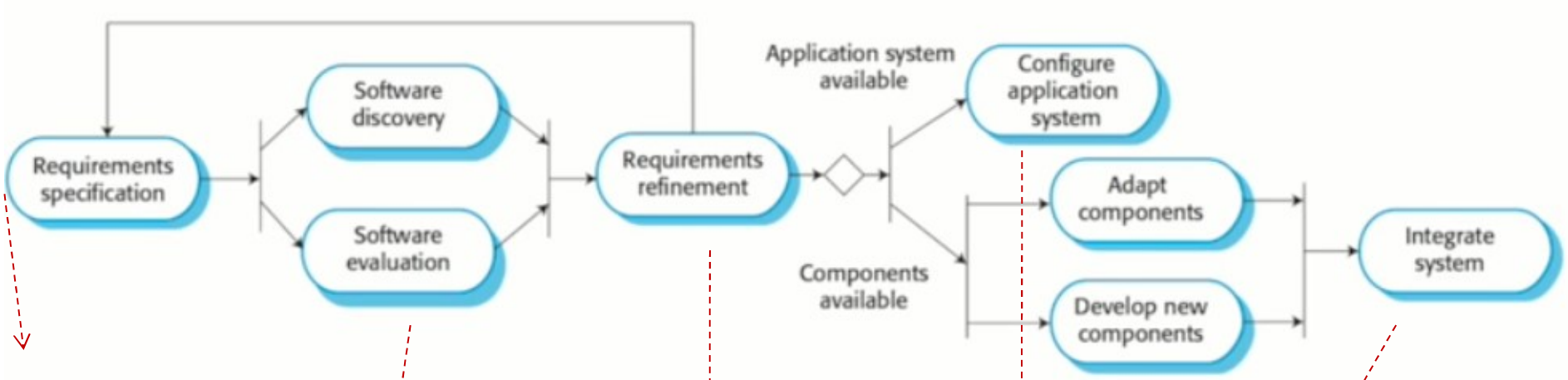
- Over the last 15 years or so, we've seen a huge change in the way software has been developed **where** there is **much more reuse of existing software** rather than developing software from scratch

- So an approach to development is...
  - Find the existing software systems or components to integrate these into a system
  - **Reused elements** may have to be **configured** to adapt their behaviors and functionality to **user requirements**

What types of software can be reused?

- Stand-alone application systems can be configured for use in a particular environment
- We can integrate Reusable components with other reusable and specially written components
- 3rd type is Web services that are developed according to service standards and that are available for remote invocation

This is the model for Reuse-oriented SE



try to understand what the system has to do

look for reusable software & Evaluate if it meets the needs of the customer

- Go back the customer,
- Modify the requirements because the reused software doesn't do what's required

2 cases: 1) in case the application system is already available, we only need to adapt it by doing some configurations to make it work in the present context

integrate them with the adapted existing ones to make a new system

Mixed approach

- involves **a mix** of plan-based and agile development
  - it's plan-based because requirements are planned **in advance**
  - but an iterative and agile approach can be taken to design and implement
- It's really common to develop and deliver the software incrementally
  - There's no a hard line between plan-based and agile processes
- But as I ever mentioned, when it comes to critical systems like safety critical systems we need to have **a very detailed analysis of requirements**
  - So we need an up front requirement specification (in the form of documents) to ensure that the work has been done properly and & we can demonstrate that the system is safe
  - These kinds of systems are usually developed using plan-based process

Agile software products & apps

- **In contrast, Off-the-shelf products** are universally developed using an agile process
  - This approach is suitable for this type of product because the specification is quite flexible
  - And It's not that the specification come from an external company, but come from the company that is developing the product themselves

Fundamental activities in SE

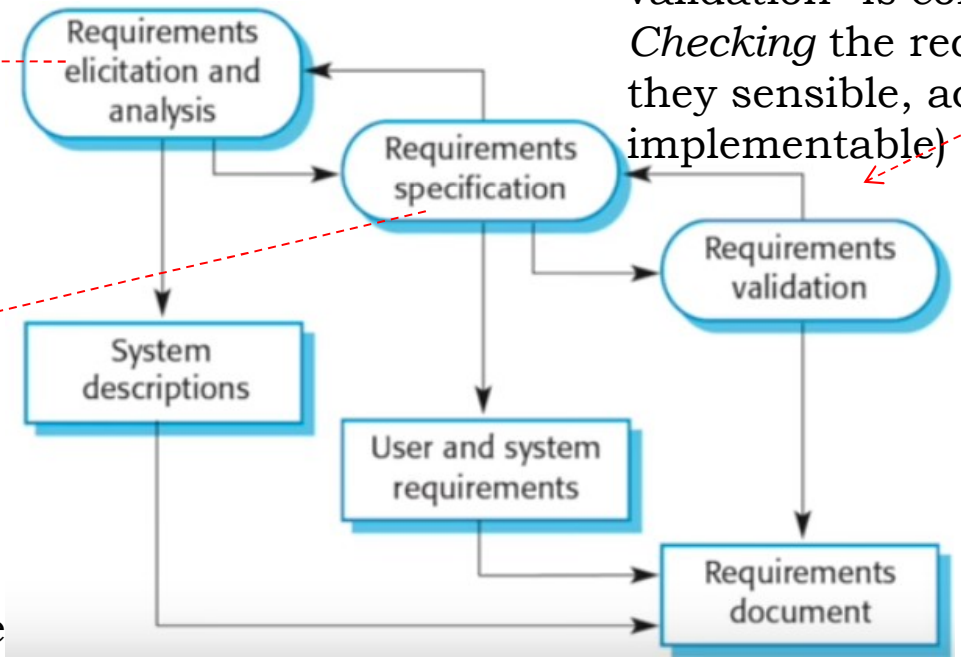
- Software specification is about
  - setting out what the system should do, what should be implemented and the expected behavior of the software
- Design & implementation is about
  - organizing the software, data structure, and implementing the system in some programming language
- Validation
  - *involves* testing the system for the bugs and checking if it meets the users requirements
- Software evolution
  - happens after the software has been deployed, when it's changed in response to changing user requirements

1. Specification

- The 1st activity Specification sometimes we call Requirement Engineering
  - Basically, the requirements engineering process has a number of sub-processes or activities within it as represented in this figure….

At the 1st phase: Engineers work with users & customers to discover what they want from the system

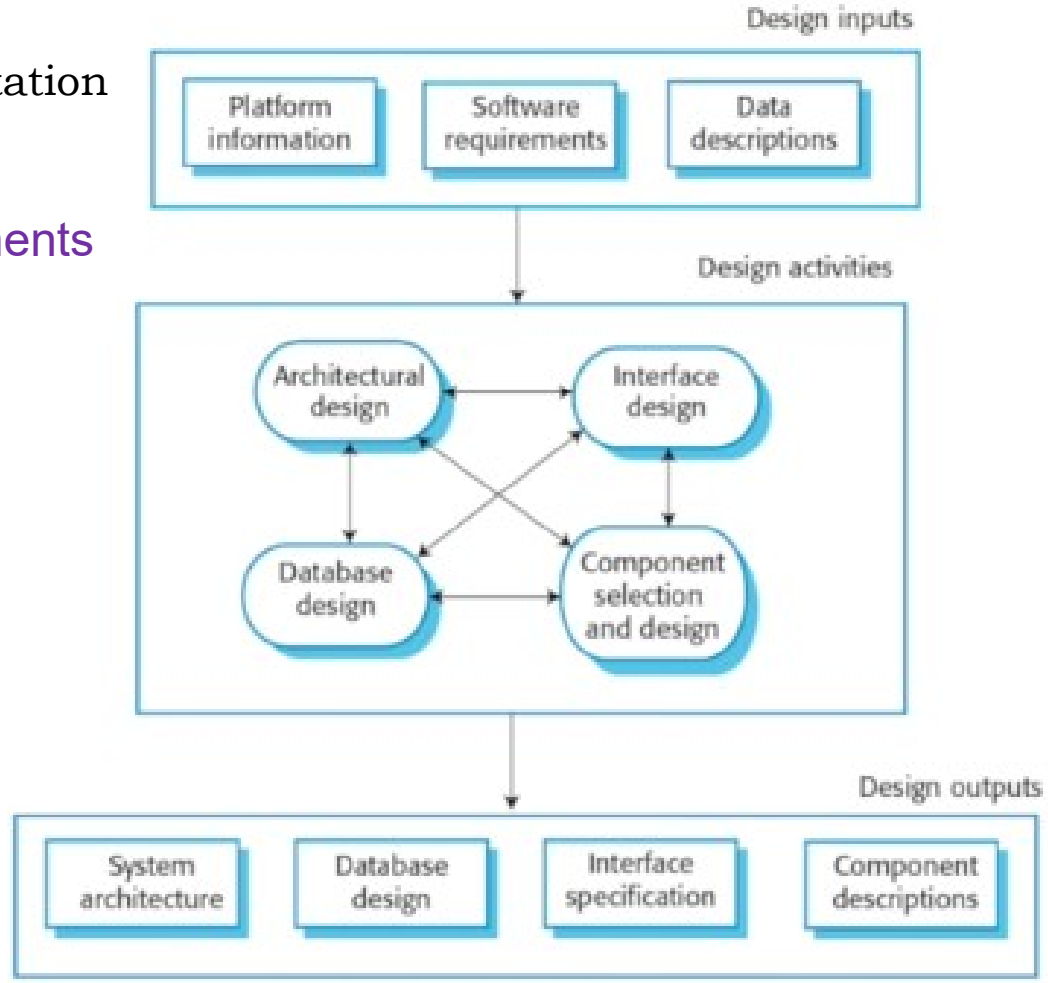
2nd phase "requirements specification" involves *taking the analysis* and *defining the document* which can be used by the development team as a basis for the design of the system



3rd phase "requirements validation" is concerned with *Checking* the requirements (are they sensible, achievable, implementable)

2. Design process

Software design and implementation starts with actually working out an architecture and major components of the system



2. Design activities

- **Architectural design**, where you identify the overall structure of the system, the principle components (subsystems or modules), their relationships and how they are distributed
- **Database design**, where you design the system data structures and how they are represented in a database
- **Interface design**, where you define the interfaces between system components
- **Component selection & design**, where you search for reusable component

3. System implementation

Implementation involves adding details to the design and programming the system

- **Software** is implemented either by developing a program or programs, or by configuring a reusable application system
- **Programming** is an individual activity with no standard process
  - Different people program in different ways
- **Debugging** is the activity of using testing to reveal program faults and then correcting these faults
  - It's an inherent part of many agile methods that **testing & programming** are very closely linked
  - so that sometimes we do **test-first development** (i.e., we prepare scenarios and tools for testing even before we develop the software, which will be introduced in the later lecture)

3.1 V&V and testing

- **V & V** is intended to show that **a system** meets its stated requirements (verification: kiểm định – theo tài liệu ghi) and also meets the real needs of the system customer (validation: thẩm định – hỏi trực tiếp xem đúng ko)

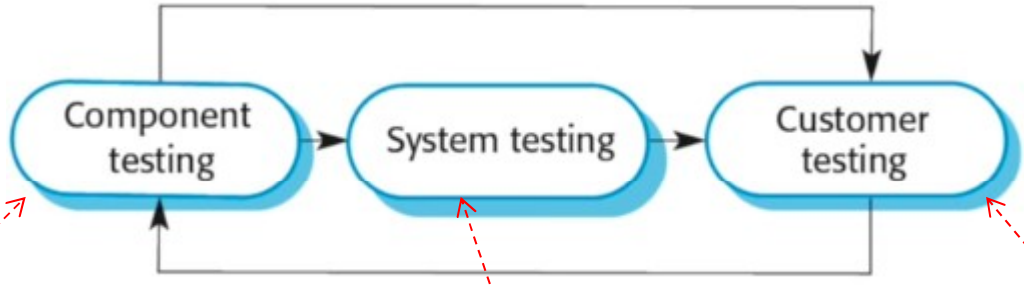
- **System testing** involves executing the system with test cases that are derived from the specification of the data to be processed by the system

  - It's not real data, but made up by the developers or the testing team which seems to be consistent with the user specification

- As well as system testing, **system validation** may involve **other reviews** and sometimes supported by **automated tools**

3.2. Testing process

- **Testing process** varies quite widely from one organization to another
  - Within the testing process, there're always really 3 activities



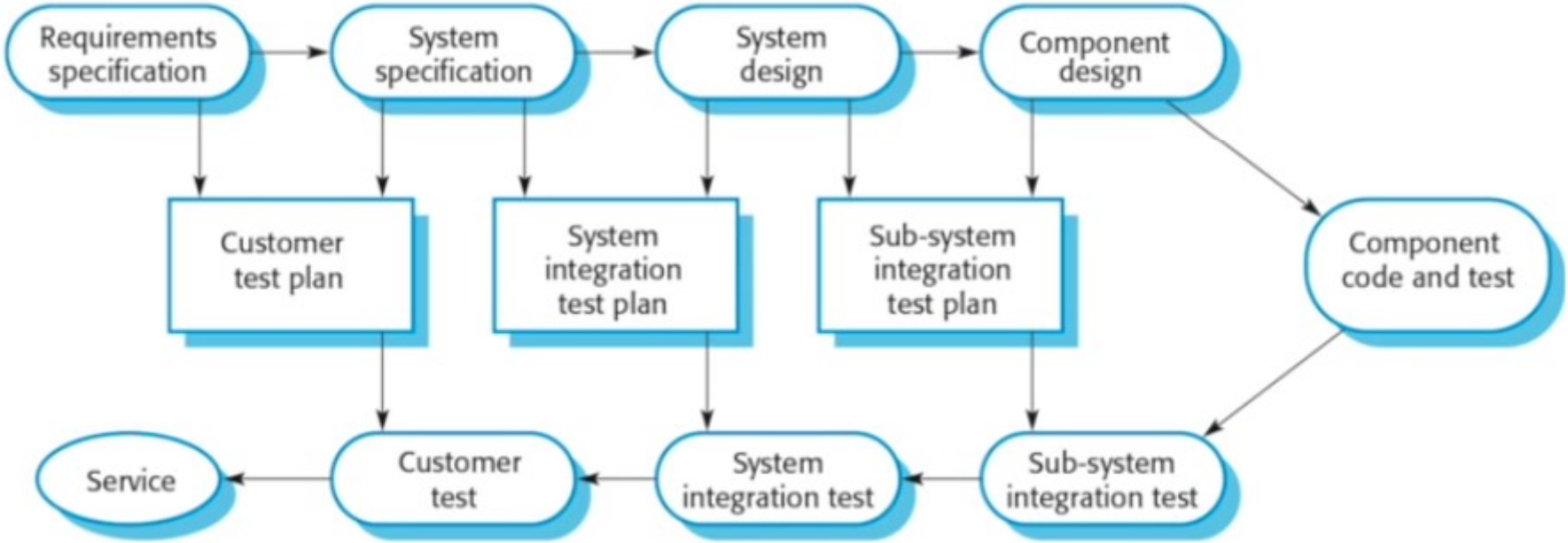
Which is done by developers, sometimes techniques such as *test-first development* are applied

This takes place at **a number of levels**, that's used to test *API*, check if the system *meets customer requirements*, *check the non-functional behavior of the system* (performance, reliability…)

Customers decide if *the software is what they want*

3.2 Testing process...

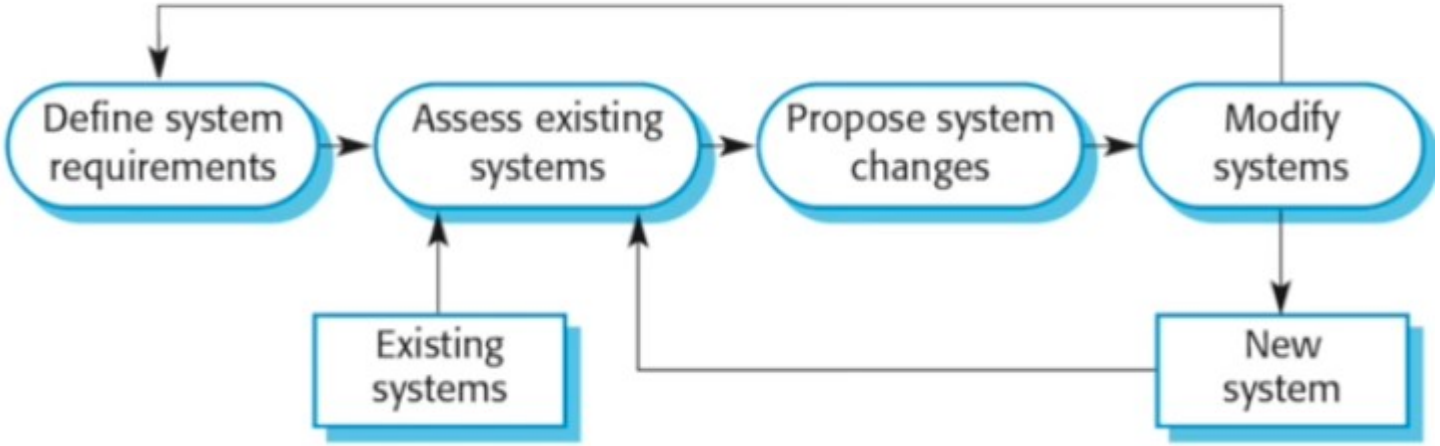
- **In a plan-driven process** we sometimes see this model of testing which is called **V model** (you can see it in a vertical alignment)



- **Each stage** in the development process has its corresponding testing activity

4. Software evolution process

- **Software** is inherently flexible and can be much easier to change than the **hardware**
- **Requirements** change through changing business circumstances, and the software need to support the business, so it must evolve and change



- Sometimes **changes** introduce new bugs into the system and causes existing bugs that have been hidden to **come to light** → the system must be modified and put into use
- and then the cycle starts again!

Coping with change

Coping with change

- **Change** is inevitable in all large software project
  - **Business changes** lead to new and changed system requirements
  - **New technologies** open up new possibilities for improving implementations
  - **Changing platform** requires application change
- **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

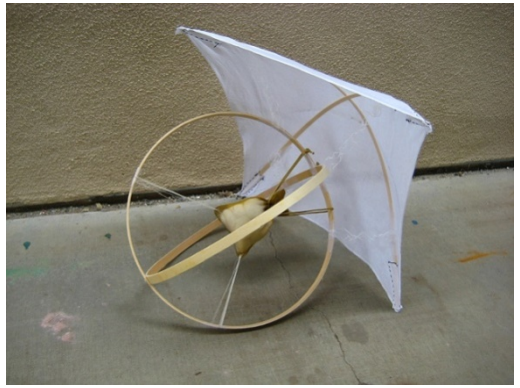
- **Change anticipation**, where the software process includes activities that can anticipate possible changes before significant rework is required
  - E.g., a prototype system may be developed to show some key features of the system to customers
- **Change tolerance**, where the process is designed so that changes can be accommodated at relatively low cost
  - This normally involves some form of **incremental development**
  - **Proposed changes** may be implemented in **increments** that have **not** been developed
  - If this is impossible, then only a single increment may have to be altered to incorporate the change

Coping with changing requirements

- **System prototyping**, where a version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of design decision
  - This approach supports **change anticipation**
- **Incremental delivery**, where **system increments** are delivered to the customer for comments and experimentation
  - this support both change avoidance and change tolerance

Software prototyping

- **A prototype** is **an initial version** of a system used to demonstrate concepts and try out design options



Software prototyping

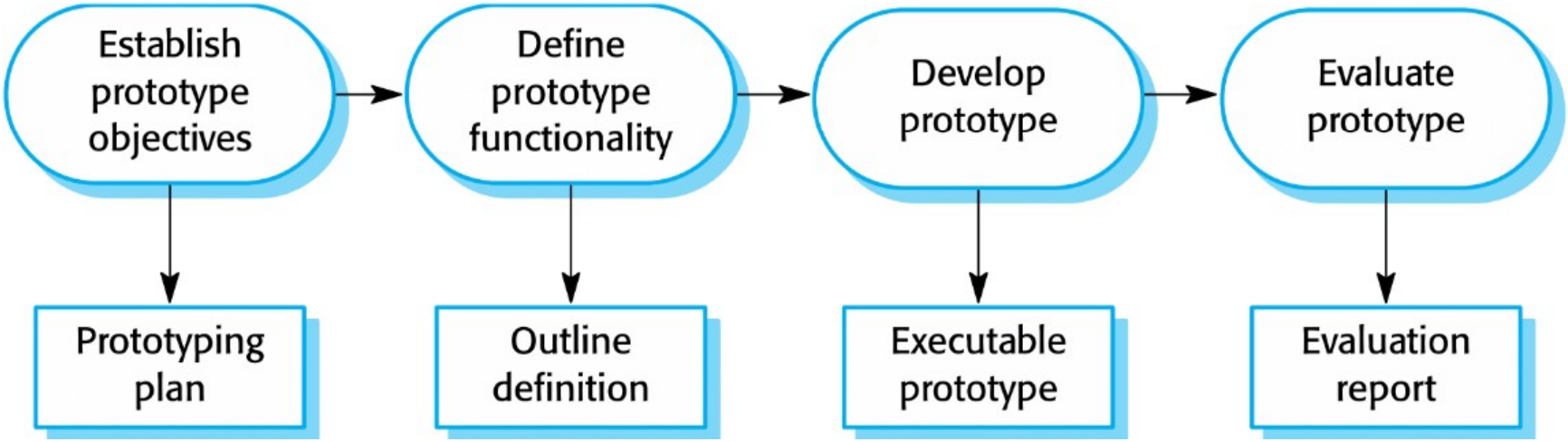
- A prototype can be **used in** ...
  - The **requirements engineering process** to help with requirements elicitation and validation
  - In design processes to explore options and develop a UI design
  - In the testing process to run back-to-back tests

smart energy monitoring system

Benefits of prototyping

- Improved system usability
- A closer match to users' real needs
- Improved design quality
- Improved maintainability
- Reduced development effort

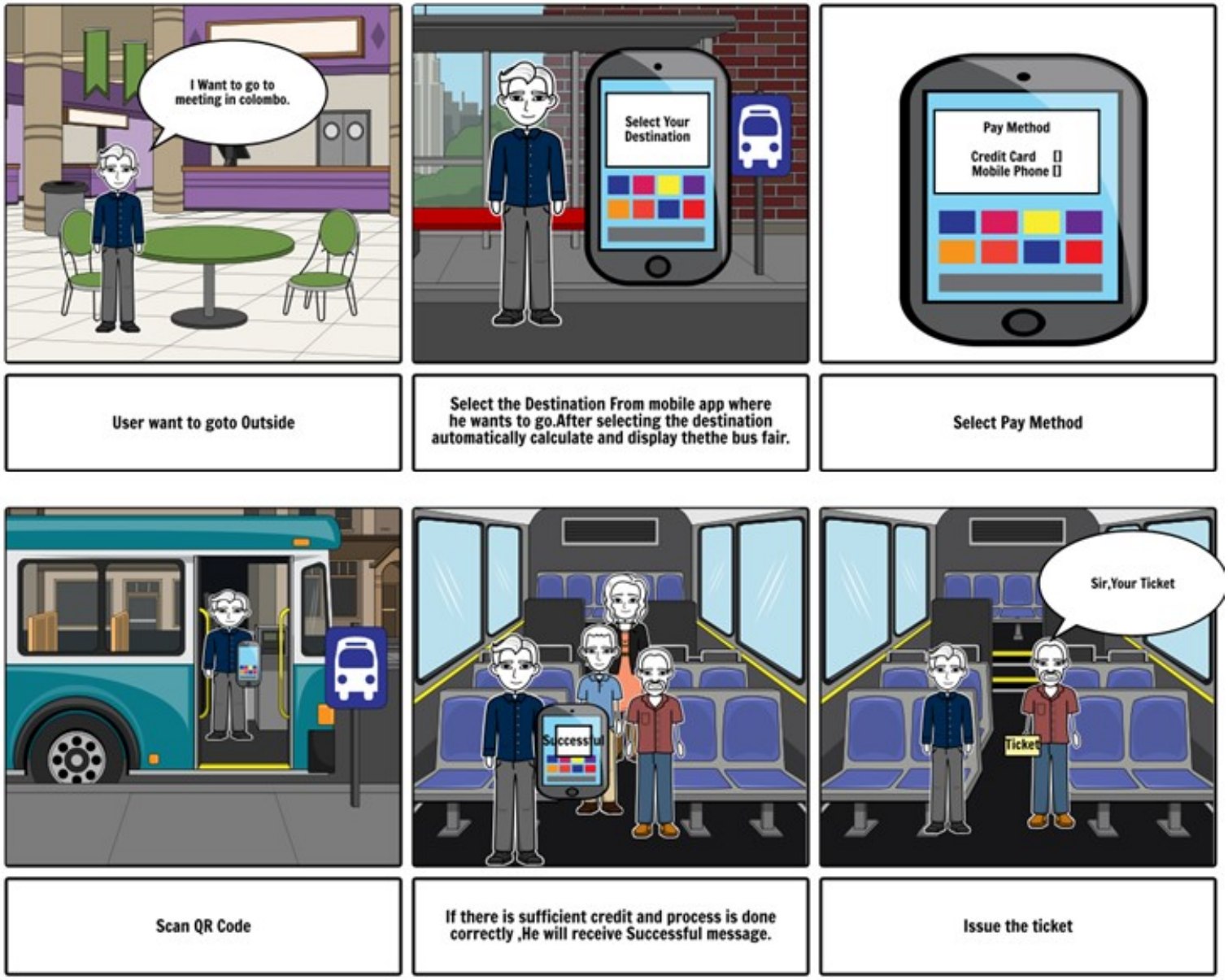
The process of prototype development



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

Prototype development: example



Throw-away prototype

- **Prototypes** should be discarded after development as they are not a good basis for a production system
  - it may be impossible to tune the system to meet non-functional requirements
  - **Prototypes** are normally undocumented
  - **The prototype structure** is usually degraded through rapid change
  - **The prototype** probably will not meet normal organizational quality standards

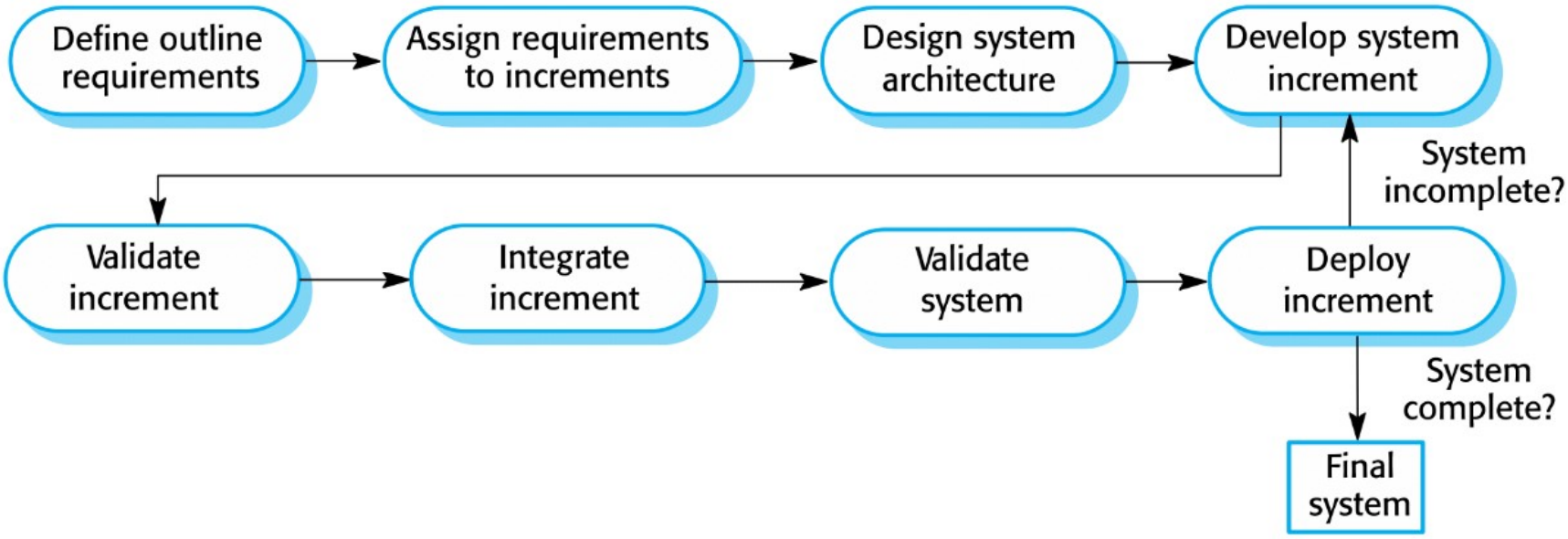
Incremental delivery

- Rather than deliver the system as **a single delivery**, the development and delivery is broken down into increments with each increment delivering part of the required functionality
- **User requirements** are prioritized and the highest priority requirements are included in early increments
- **Once** the development of an increment is started, the requirements are frozen for later increments can continue to evolve

Incremental development & delivery

- Incremental development
  - Develop the system in **increments** and evaluate each increment **before proceeding to** the development of the next increment
  - **Normal approach used is agile methods**
  - Evaluation done by user/customer proxy
- Incremental delivery
  - Deploy an increment for use by end-users & more realistic evaluation about practical use of software

Incremental delivery of a system



Incremental delivery advantages

- **Customer value** can be delivered with each increment so system functionality is available earlier
- Early increments act as a prototype to help elicit requirements **for later increments**
- **Lower** risk of overall project failure
- The highest priority system services tend to receive **the most** testing

Incremental delivery problems

- Most systems require **a set of basic facilities** that are used by different parts of the system
  - As requirements are not defined in detail **until** an increment is to be implemented, it can be hard to identify **common facilities** that are needed by **all** increments
- The essence of iterative processes is that the specification is developed in conjunctions with the software
  - However, this conflicts with the procurement model of many organizations, where **the complete system specification** is part of the system development contract

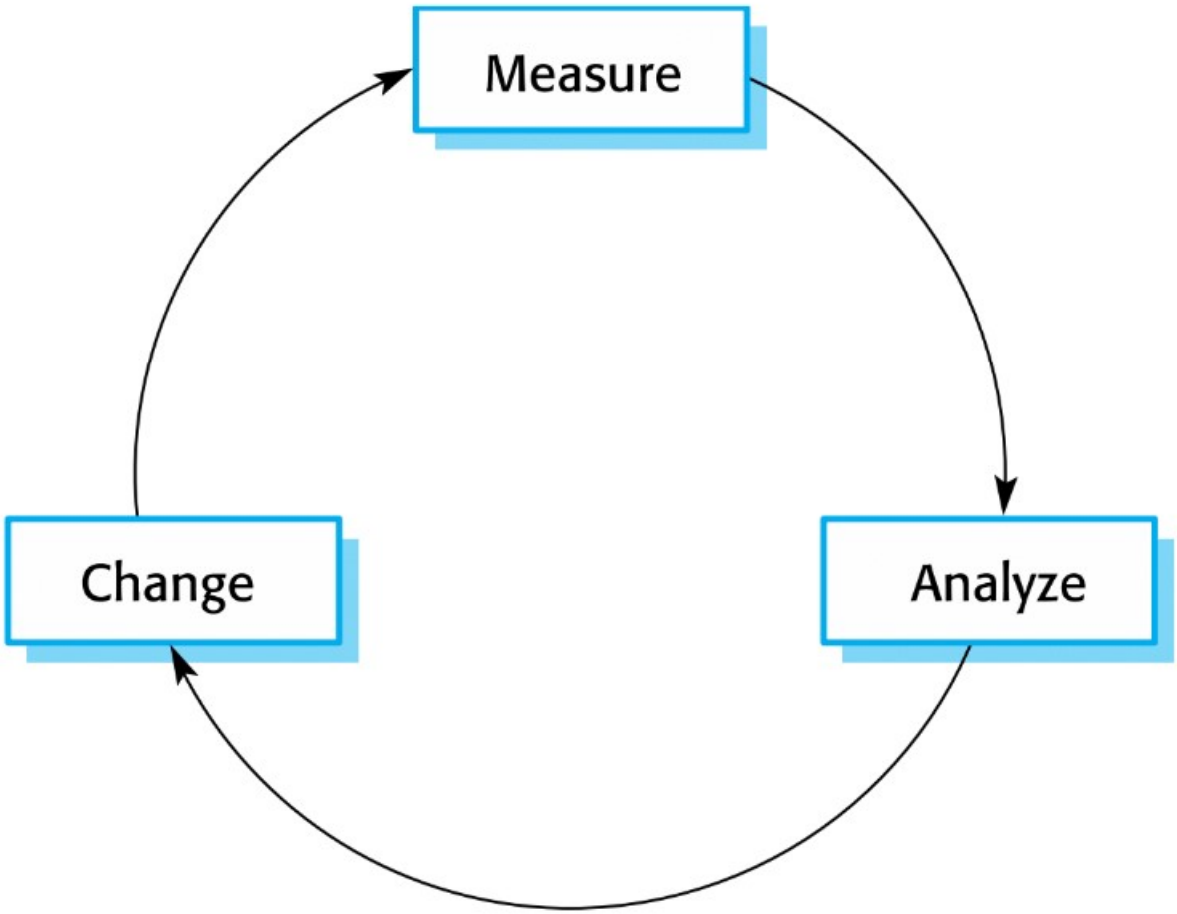
Process improvement

- **Many software companies** have tuned to software process improvement as **a way** of enhancing the quality of their software, reducing costs or accelerating their development processes
- **Process improvement** means
  - **Understanding** existing processes
  - and **changing** these processes to **increase** product quality and/or **reduce** costs and development time

Approach to improvement

- **The process maturity approach**, which focuses on...
  - improving process and project management
  - and introducing good software engineering practice
  - **The level** of process maturity reflects the extent to which good technical and management practice has been adopted in organization of software development processes

The process of improvement cycle



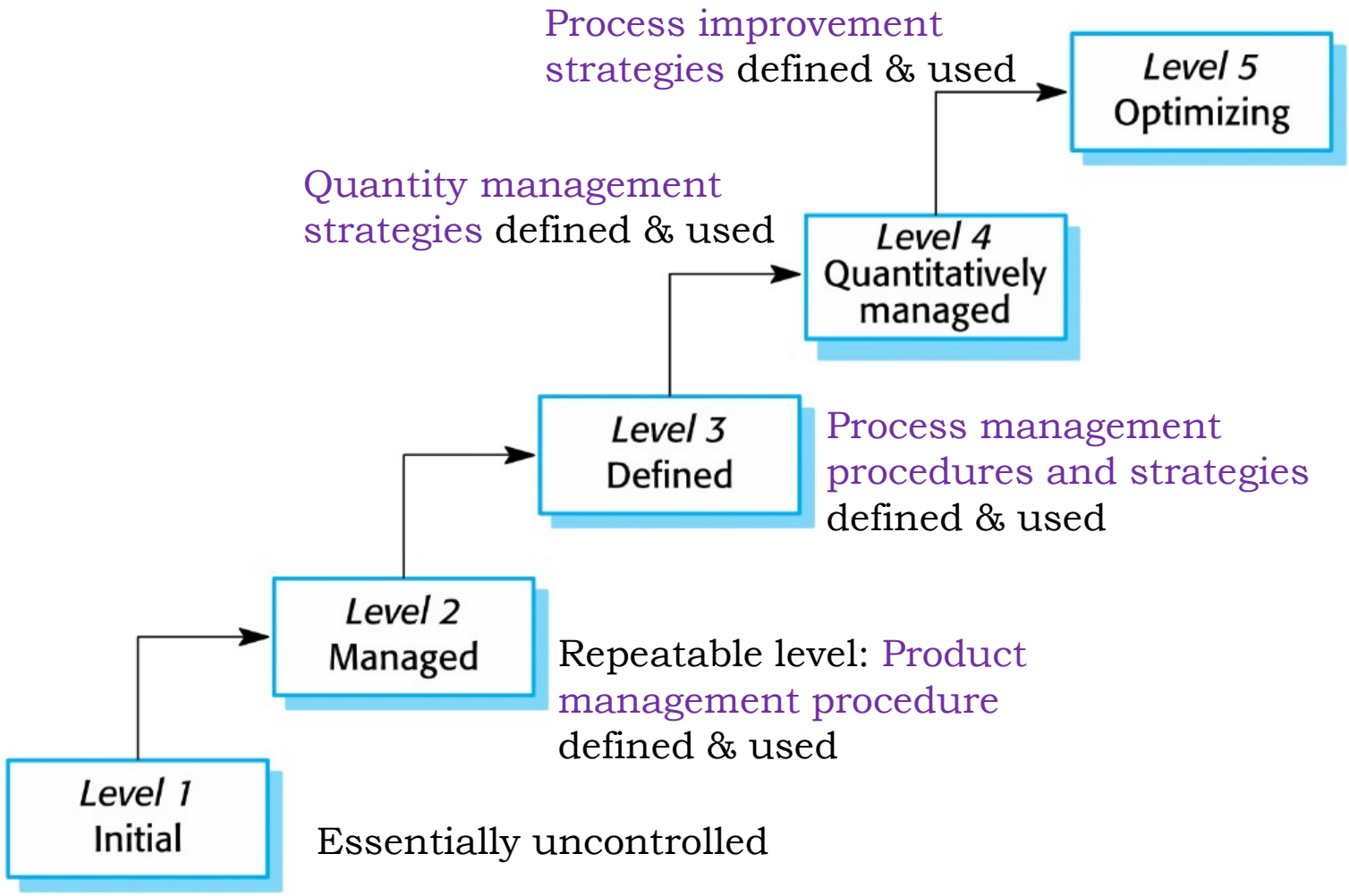
Process improvement activities

- Process measurement
  - You measure one or more attributes of the software process or product
  - These measurements form a baseline that helps you decide if process improvements have been effective
- Process analysis
  - The current process is assessed
  - and process weakness and bottlenecks are identified
  - Process models (sometimes called process maps) that describe the process may be developed
- Process change
  - Changes are processed to address some of the identified process weakness.
  - These are introduced and the cycle resumes to collect data about the effectiveness of the changes

Process measurement

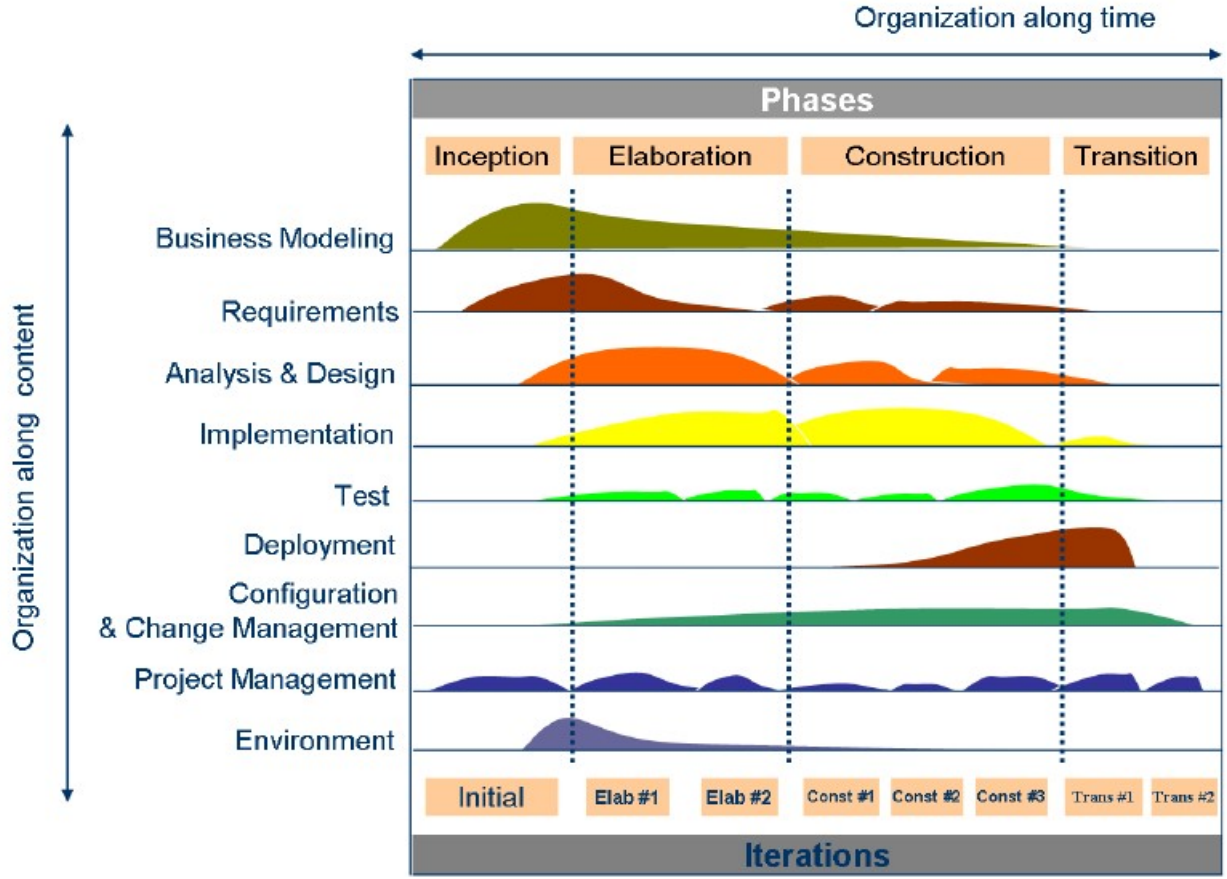
- Whenever possible, quantitative process data should be collected
  - However, where organizations do not have *clearly defined process standards* this is very difficult as you don't know what to measure
- Process measurements should be used to assess process improvements
  - But this does not mean that **measurements** should drive the improvements
  - The improvement driver should be the organizational objectives

SEI capability maturity Model



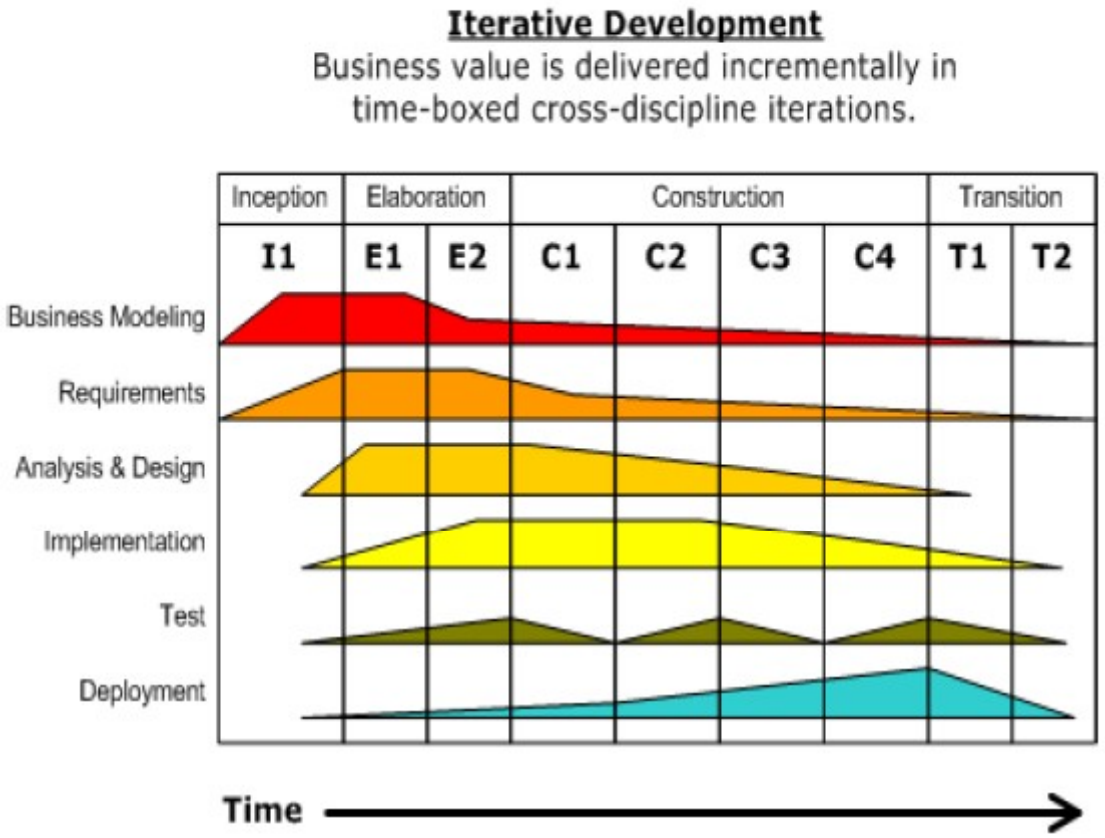
The Rational Unified Process (RUP)

- **A modern process model** derived from the work on the UML and associated process
- Normally described from **3 perspectives**
  - a dynamic perspective that shows phases over time
  - a static perspective that shows process activities
  - A practice perspective that suggests good practice



RUP phases

- Inception
  - establish the business cases for the system
- Elaboration
  - Develop an understanding of the problem domain and the system architecture
- Construction
  - System design, programming and testing
- Transition
  - Deploy the system in its operating environment



RUP good practice

- Develop software iteratively
- Manage requirements
- Use component-based architectures
- Visually model software
- Verify software quality
- Control changes to software

Static workflows

Workflow	Description
Business modeling	The business processes are modeled using business use cases
Requirements	Actors who interact with the system are identified and use cases are developed to model the system requirements
Analysis and design	A design model is created and documented using architectural models, component models, object models and sequence models
Implementation	The components in the system are implemented and structured into sub-systems. Automatic code generation from design models help accelerate this process
Test	Testing is an iterative process that is carried out in conjunction with implementation
Deployment	A product release is created, distributed to users and installed in their workplace
Configuration and change mgnt	This supporting workflow manages changes to the system
Project management	This supporting workflow manages the system development
Environment	This workflow is concerned with making appropriate software tools available to the software development team

Computer-aided software engineering

- Computer-aided software engineering (CASE) is **software** to support software development and evolution processes
- Activity automation
  - Graphical editors for system model development
  - Data dictionary to manage design entities
  - Graphical UI builder for user interface construction
  - Debuggers to support program fault finding
  - Automated translators to generate new versions of a program

CASE technology

- CASE technology had led to significant improvements in the software process
  - However, there are not the order of magnitude improvements that were once predicted
- SE requires creative thought- this is not readily automated
- SE is a team activity
  - **For large projects**, much time is spent in team interactions
  - CASE technology does not really support these

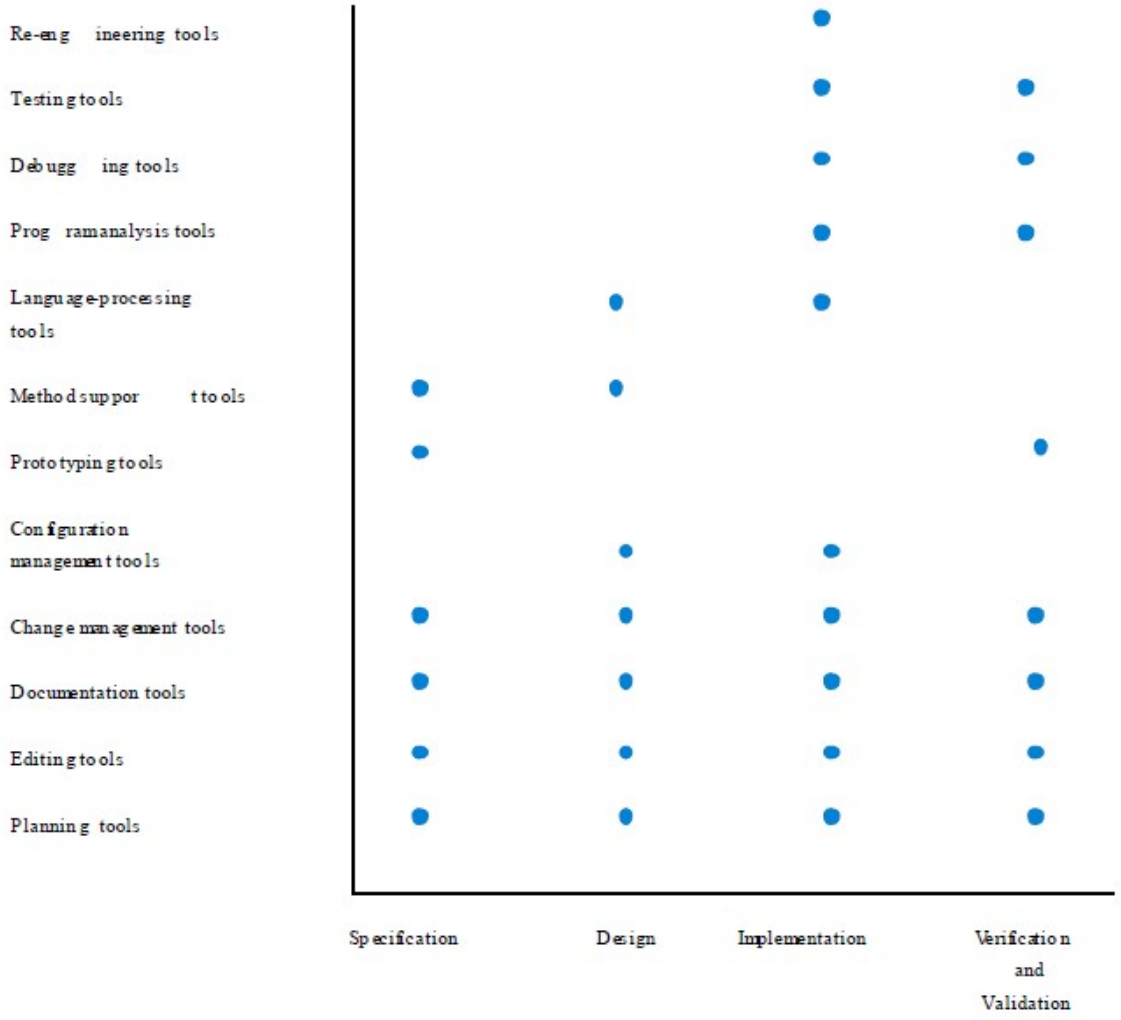
CASE classification

- **Classification** helps us understand the different types of CASE tools and their support for process activities
- Functional perspective
  - **Tools** are classified according to their specific function
- Process perspective
  - Tools are classified according to process activities that are supported
- Integrated perspective
  - Tools are classified according to their organization into integrated units

Functional tool classification

Tool type	
Planning tools	PERT tools, estimation tools, spreadsheets
Editing tools	Text editors, diagram editors, word processors
Change management tools	Requirements traceability tools, change control systems
Configuration management tools	Version management system, system building tools
Prototyping tools	Very high-level languages, user interface generators
Method-support tools	Design editors, data dictionaries, code generators
Language processing tools	Compilers, interpreters
Program analysis tool	Cross reference generators, static analyzers, dynamic analyzers
Testing tools	Test data generators, file comparators
Debugging tools	Interactive debugging systems
Documentation tools	Page layout programs, image editors
Re-engineering tools	Cross-reference systems, program re-structuring systems

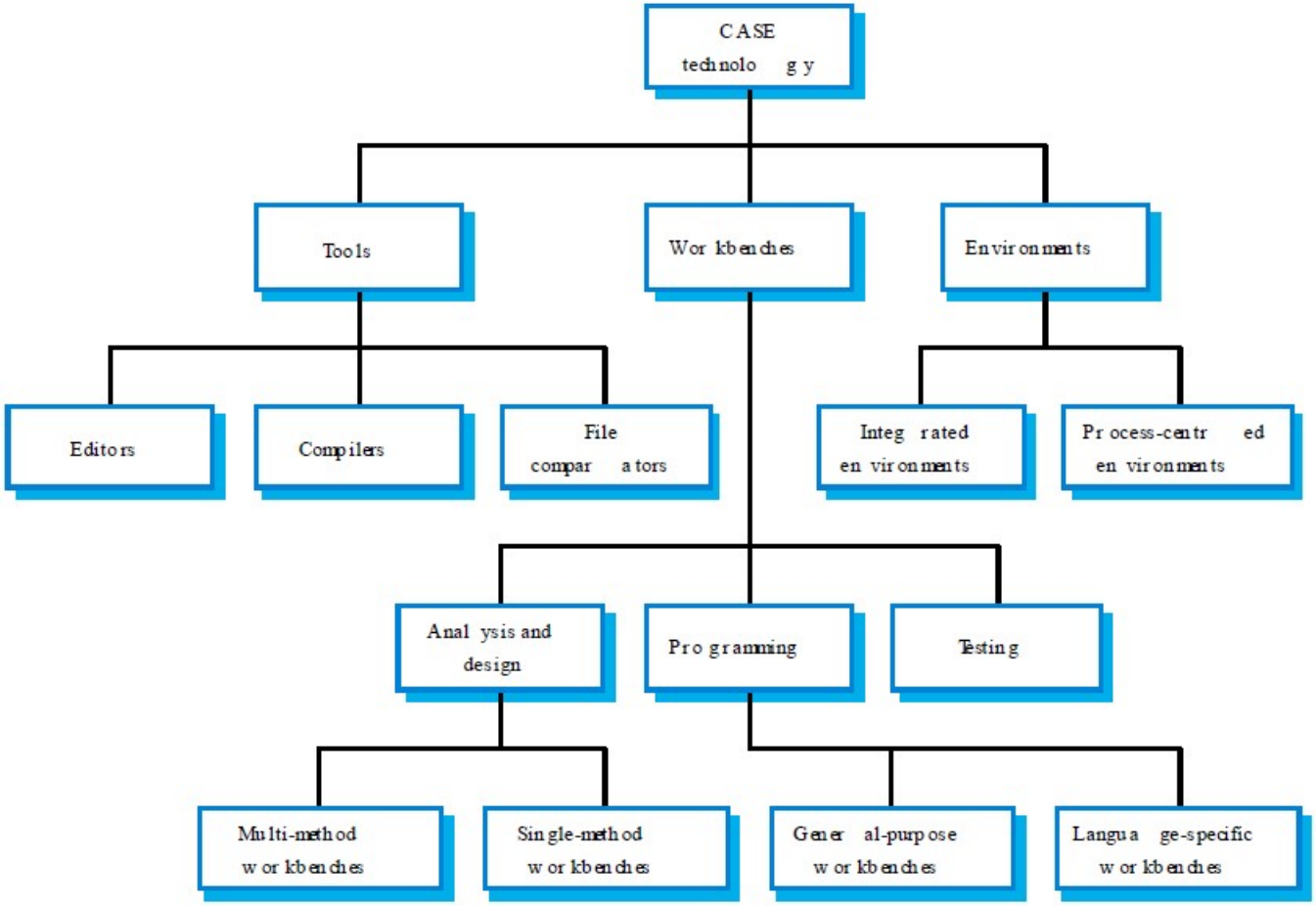
Activity-based tool classification



CASE integration

- Tools
  - Support **individual process tasks** such as design consistency checking, text editing, etc.
- Workbenches
  - Support **a process phase** such as specification or design, normally include a number of integrated tools
- Environments
  - Support **all of a substantial part** of an entire software process
  - Normally include several integrated workbenches

Tools, workbenches, environments



Key points

- **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 (cont)

- **Design and implementation processes** are concerned with transforming a requirements specification into an executable software program
- **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 changes

Key points

- **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 improvements are agile approaches, geared to reducing process overheads, and maturity-based approaches based on better process management and then the use of good SE practice
- The SEI process maturity framework identifies maturity levels that essentially correspond to the use of good SE practice