

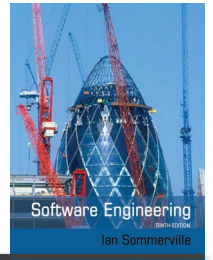
Chapter 2 – Software Processes

Ian Sommerville,
Software Engineering, 10th Edition
Pearson Education, Addison-
Wesley

Note: These are a slightly modified version of Chapter 2 slides available from the author's site

<http://iansommerville.com/software-engineering-book/>

Topics covered



- Software process models
- Process activities
- Coping with change
- Process improvement

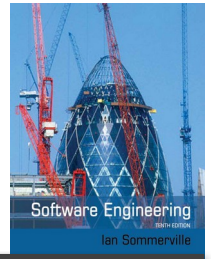
The software process



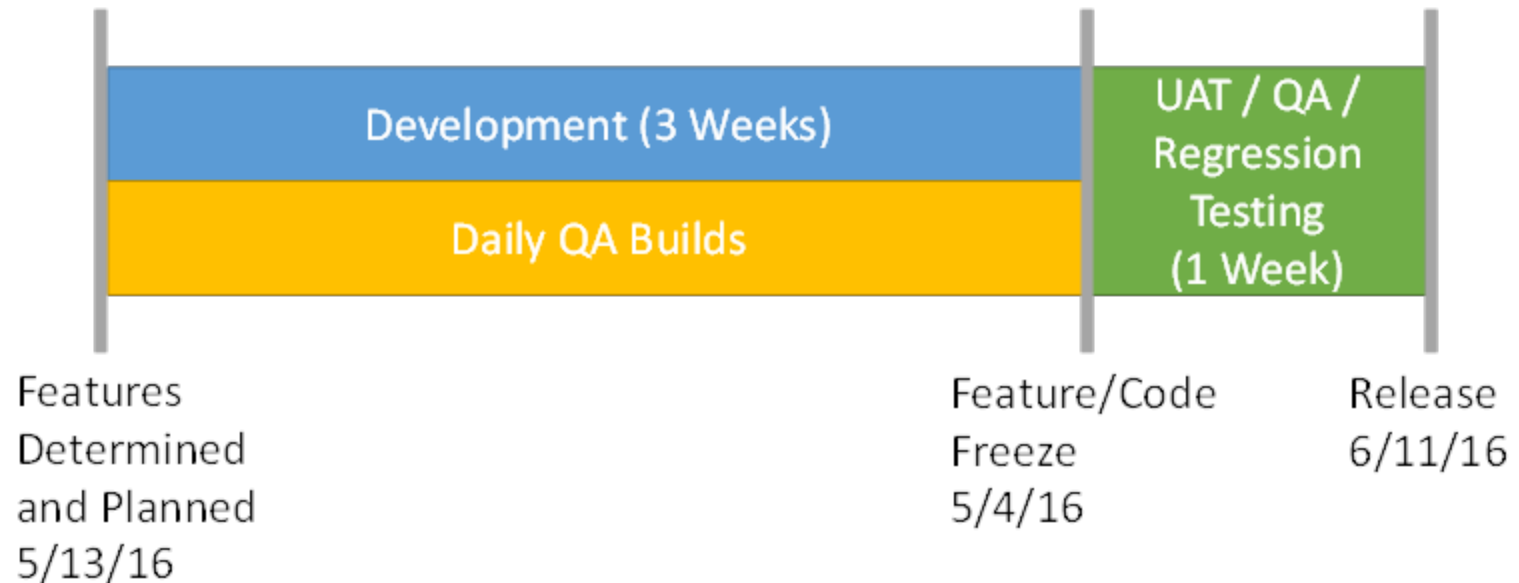
- *Software process*: a structured set of activities required to develop a software system
- Many different software processes but all involve:
 - *Specification* – defining what the system should do;
 - *Design and implementation* – defining the organization of the system and implementing the system;
 - *Validation* – checking that it does what the customer wants;
 - *Evolution* – changing the system in response to changing customer needs.
- A *software process model* is an abstract representation of a process. It presents a description of a process from some particular perspective.

Phase	Required	Nice to have
INITIATION		
	Document/Page Containing the following to initiate the Project:	
	Description (3 to 5 bullet Points)	Customer/market target
	Purpose (Goals and objectives)	Assumptions & exclusions
	Success Factors	Business Sponsor
	Key Resources Identified (i.e. Project manager, Dev Lead, QA Lead)	In scope and out of the scope
	Project Kick-off	
PLANNING		
	Requirements / Specs (1-100%)	
	Project Schedule	Key dates & process
	Work Estimates	Risk Analysis Log
	Major Milestones Identified	Project Management Plan
	Dependencies Identified	
EXECUTION		
	Infrastructure (1) - Development	
	Core Development	Tools
		Issue Management Log
	Demonstration(s)	Iterations & tempo
	Status Updates	Executive Sign-Off
TESTING		
	QA Cycle	
	(Risk Analysis Statement)	Customer (UAT)
	Acceptance Criteria and Process Defined	
LAUNCH		
	Code Freeze	
	Infrastructure (2) - PROD	Training
	Customer Training/Communication	Time-frame: begin & end
		Validation & Hot-Fixes
	GO-LIVE	
CLOSURE		
	Release Assessment	Post-mortem
		Next steps
	NSE (Source code escrow)	
	PROJECT COMPLETE	

Sprint Example



Monthly Sprint Example 16.6 (June)



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 it is easier to change the process to reflect changing customer requirements.
- 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



Software process models



□ The waterfall model (1)

- Plan-driven model. Separate and distinct phases of specification and development.

□ Incremental development (2)

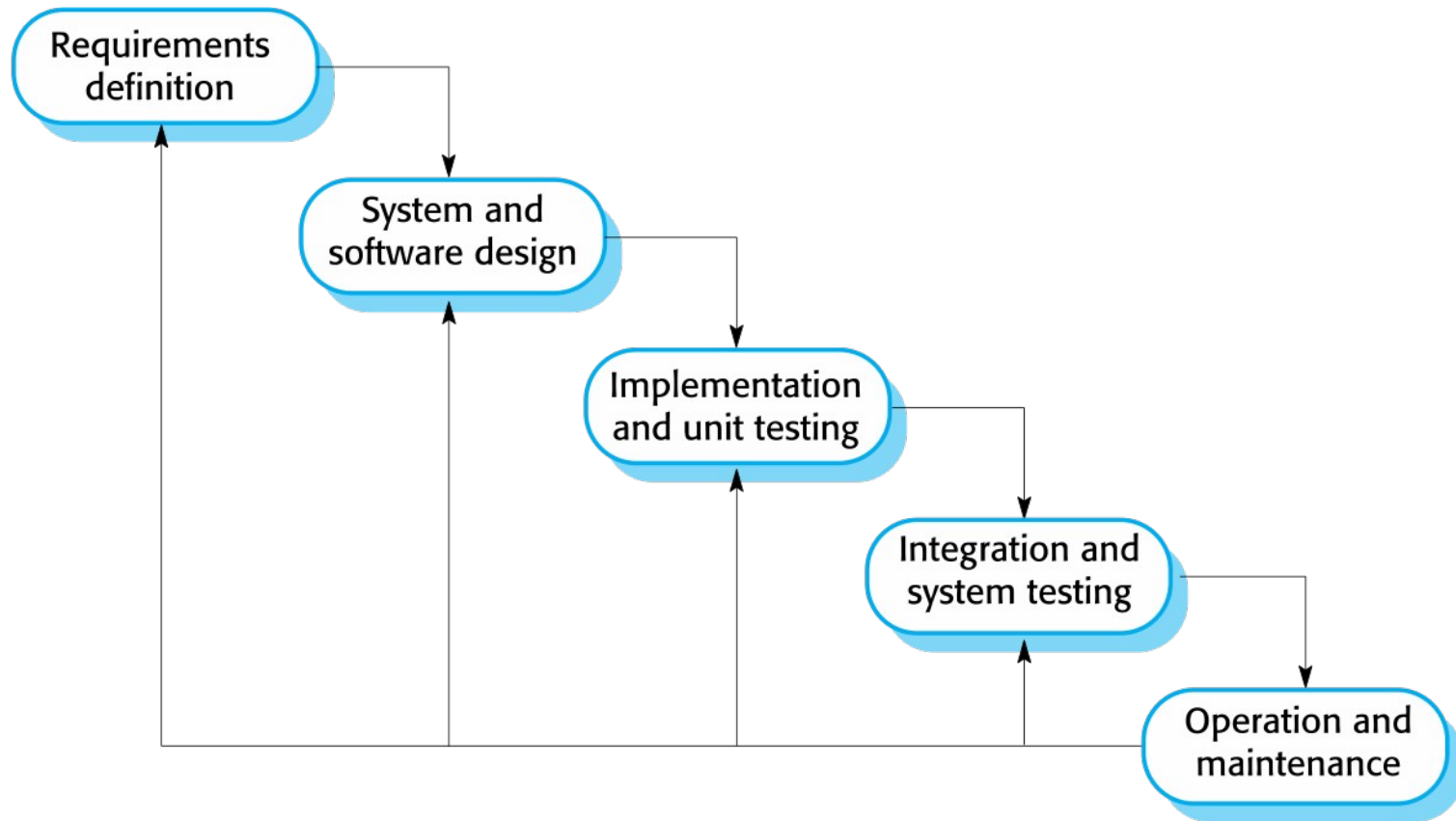
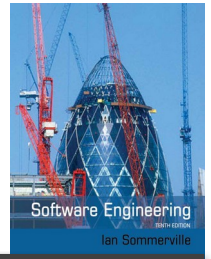
- Specification, development and validation are interleaved. May be plan-driven or agile.

□ Integration and configuration (3)

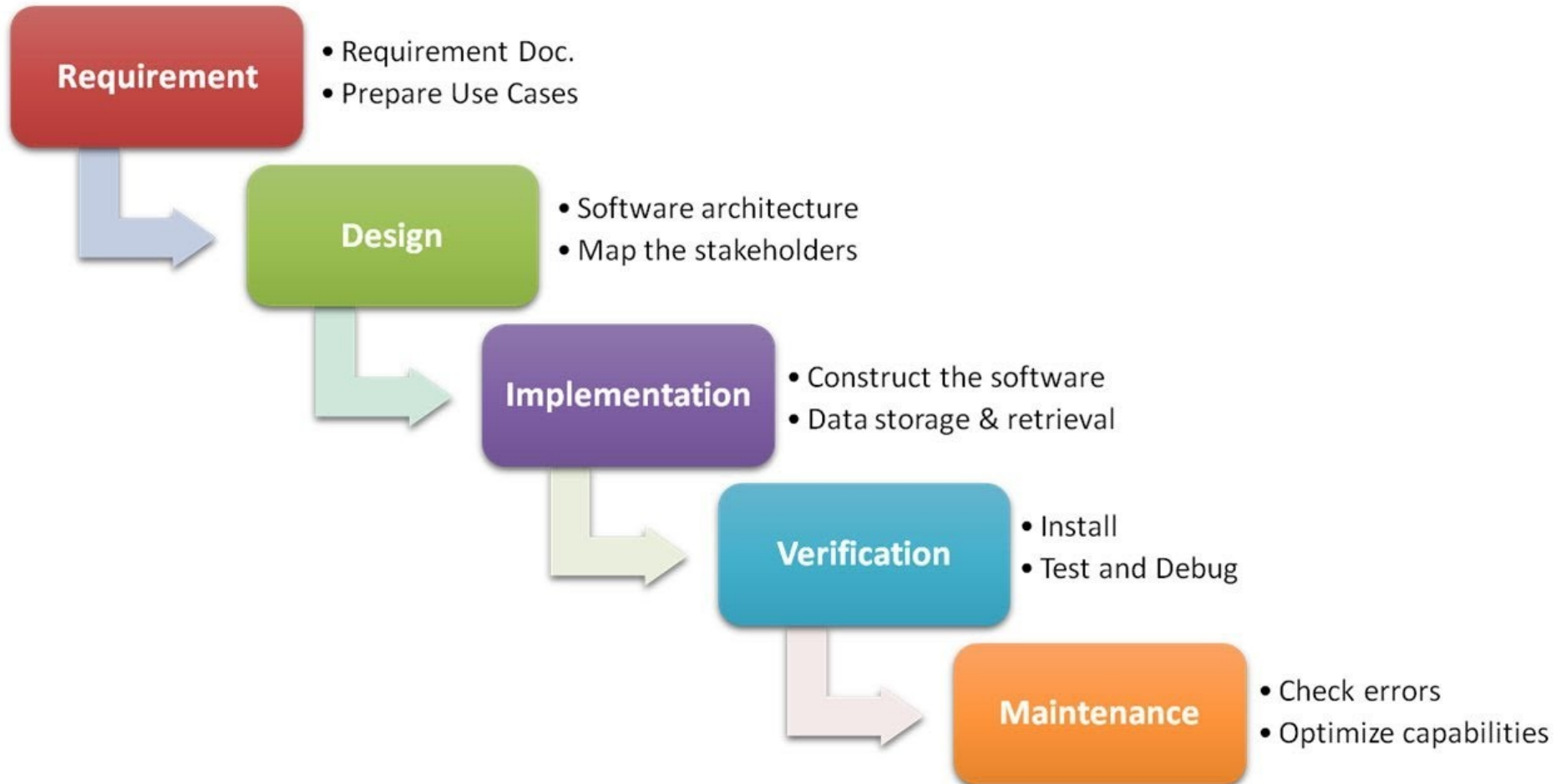
- The system is assembled from existing configurable components. May be plan-driven or agile.

□ In practice, most large systems are developed using a process that incorporates elements from all of these models.

The waterfall model



The waterfall model Phases

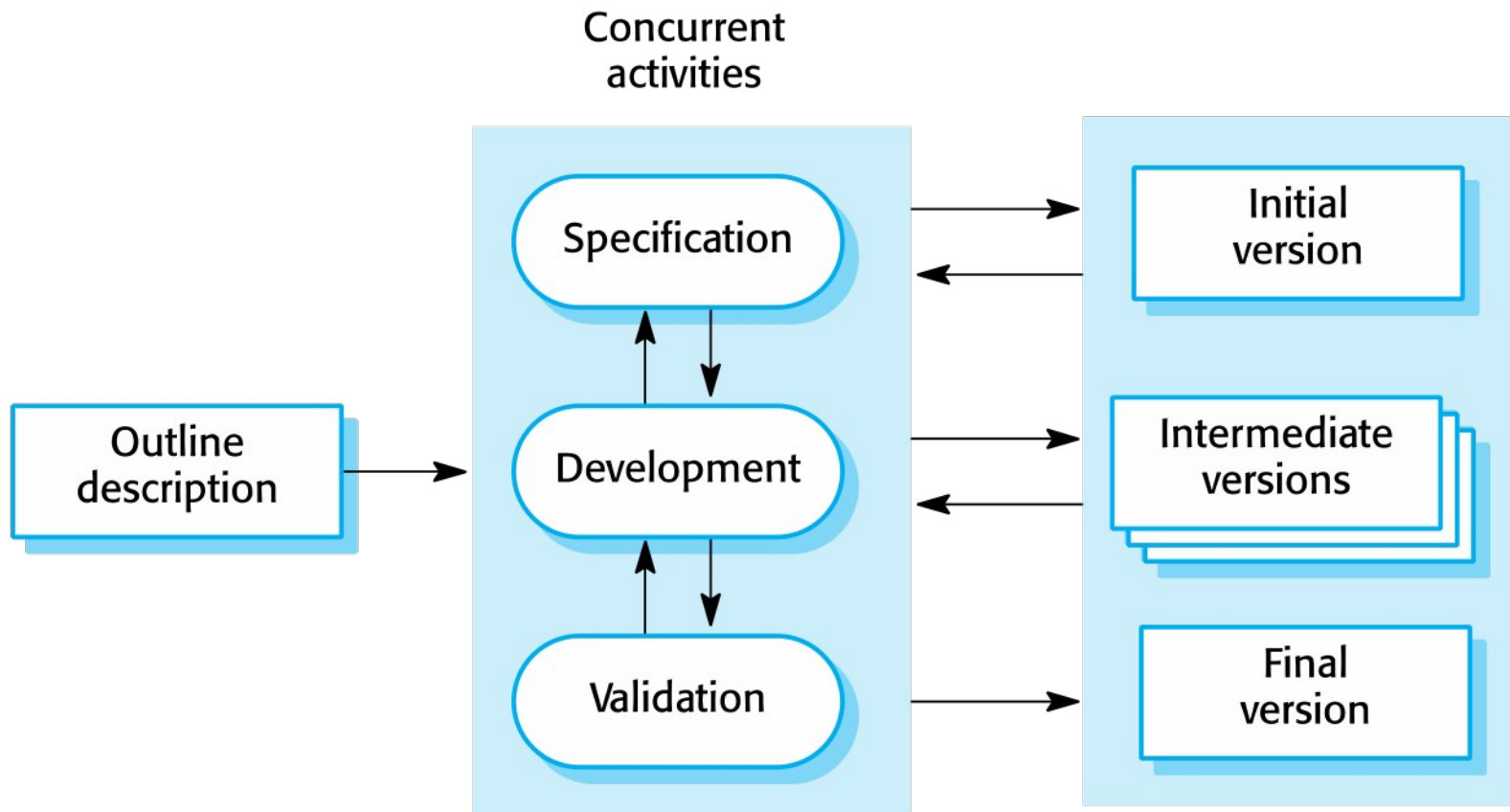
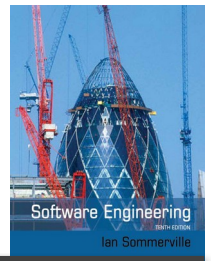


Waterfall model problems



- 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.
- The waterfall model is mostly used for large systems engineering projects where a system is developed at several sites.
 - In those circumstances, the plan-driven nature of the waterfall model helps coordinate the work.

Incremental development



Incremental development benefits



- The cost of accommodating changing customer requirements is reduced
 - The amount of analysis and documentation that has to be redone is much less than is required with the waterfall model
- It is easier to get customer feedback on the development work that has been done
 - Customers can comment on demonstrations of the software and see how much has been implemented
- More rapid delivery and deployment of useful software to the customer is possible
 - Customers are able to use and gain value from the software earlier than is possible with a waterfall process

Incremental development problems



- The process is not visible
 - Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost-effective to produce documents that reflect every version of the system
- System structure tends to degrade as new increments are added
 - Unless time and money is spent on refactoring to improve the software, regular change tends to corrupt its structure. Incorporating further software changes becomes increasingly difficult and costly

Integration and configuration



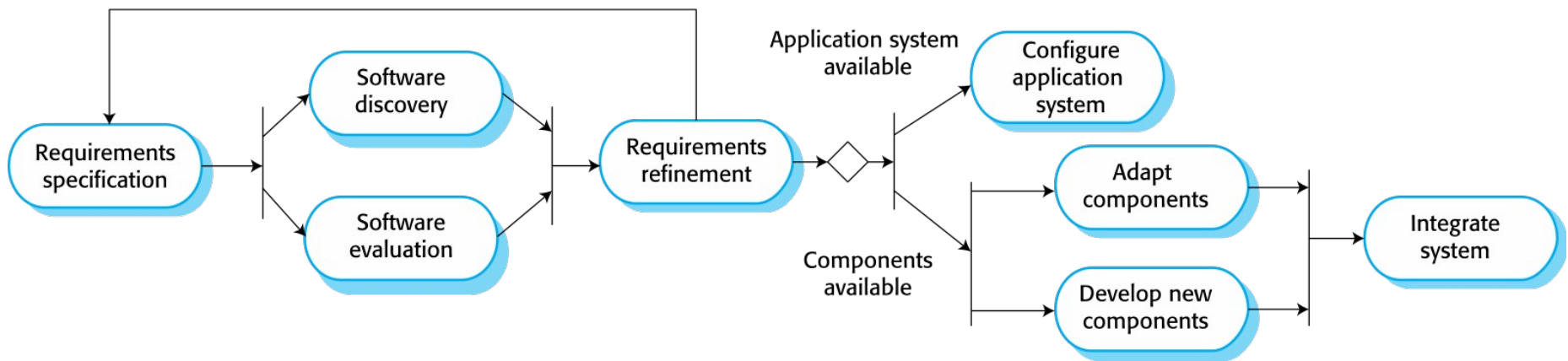
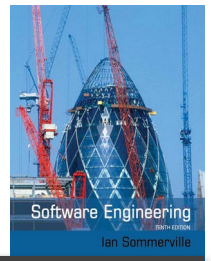
- Based on software reuse where systems are integrated from existing components or application systems (**COTS** - commercial-off-the-shelf).
- Reused elements may be configured to adapt their behaviour and functionality to a user's requirements
- Reuse is now the standard approach for building many types of business system
 - Reuse covered in more depth in Chapter 15

Types of reusable software

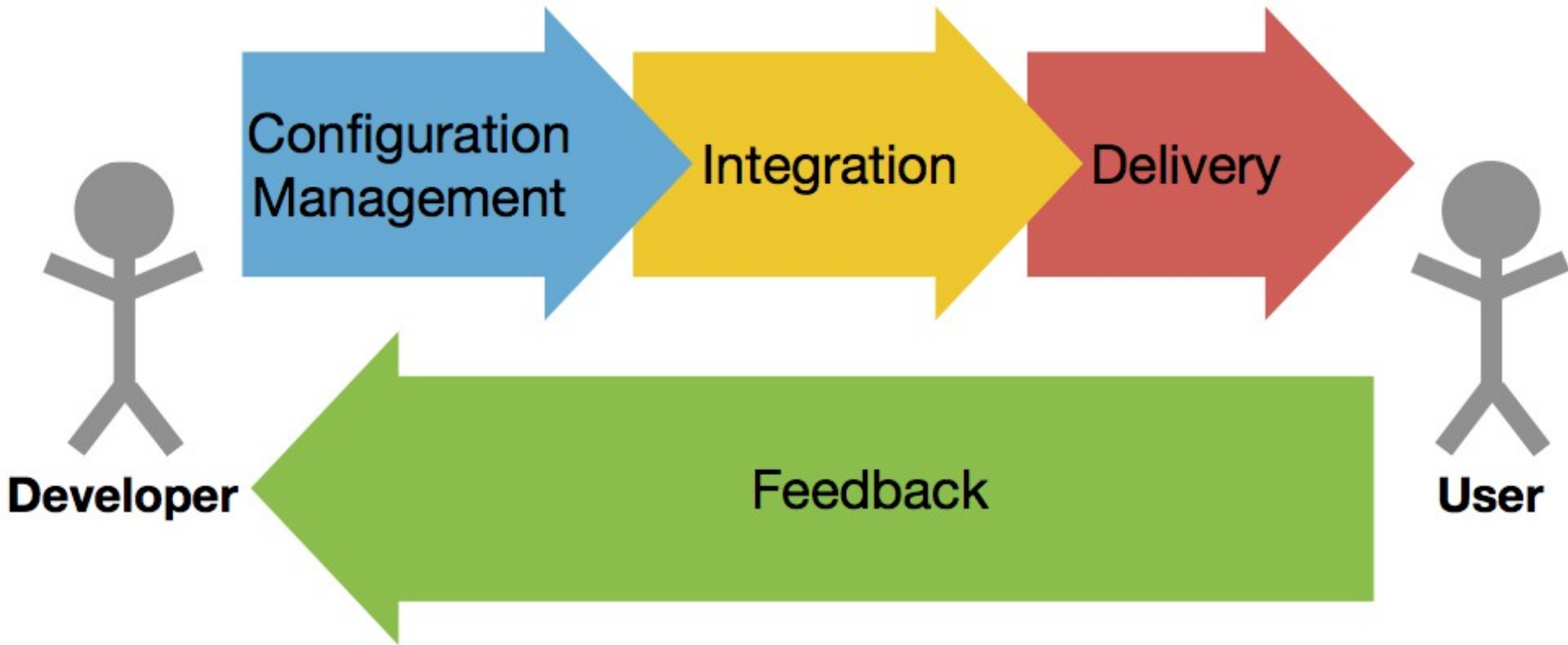
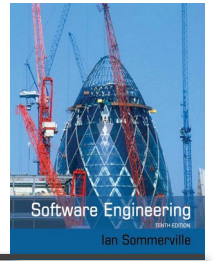


- Stand-alone application systems (sometimes called COTS) that are configured for use in a particular environment.
- Collections of objects that are developed as a package to be integrated with a component framework such as .NET or J2EE.
- Web services that are developed according to service standards and which are available for remote invocation.

Reuse-oriented software engineering



Integration and configuration Simplified



Key process stages



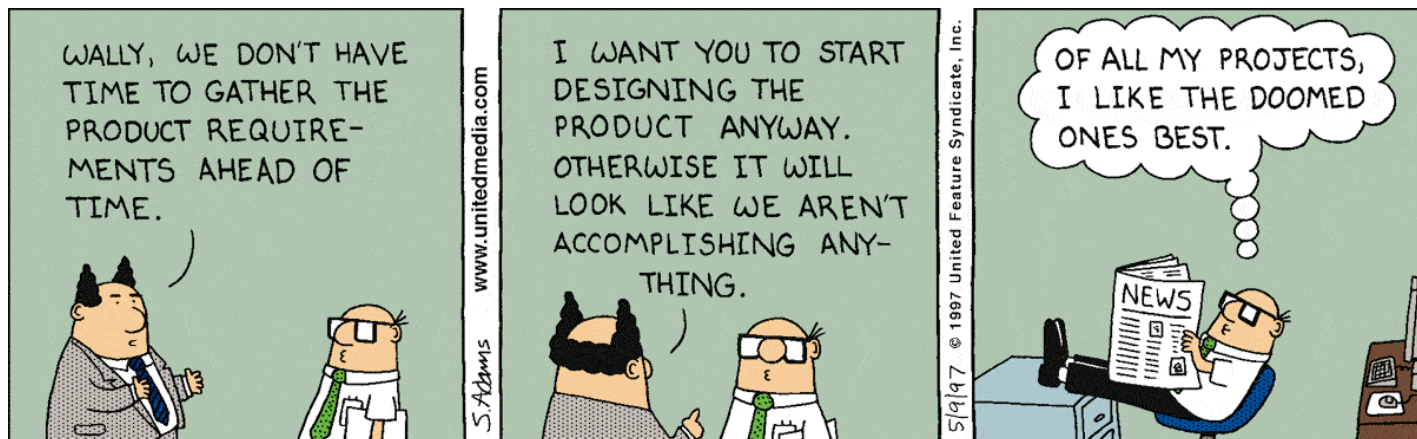
- Requirements specification
- Software discovery and evaluation
- Requirements refinement
- Application system configuration
- Component adaptation and integration

Advantages and disadvantages

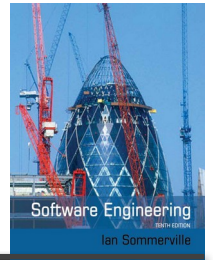


- ❑ Reduced costs and risks as less software is developed from scratch
- ❑ Faster delivery and deployment of system
- ❑ But requirements compromises are inevitable so system may not meet real needs of users
- ❑ Loss of control over evolution of reused system elements

Process activities



Process activities



Specification

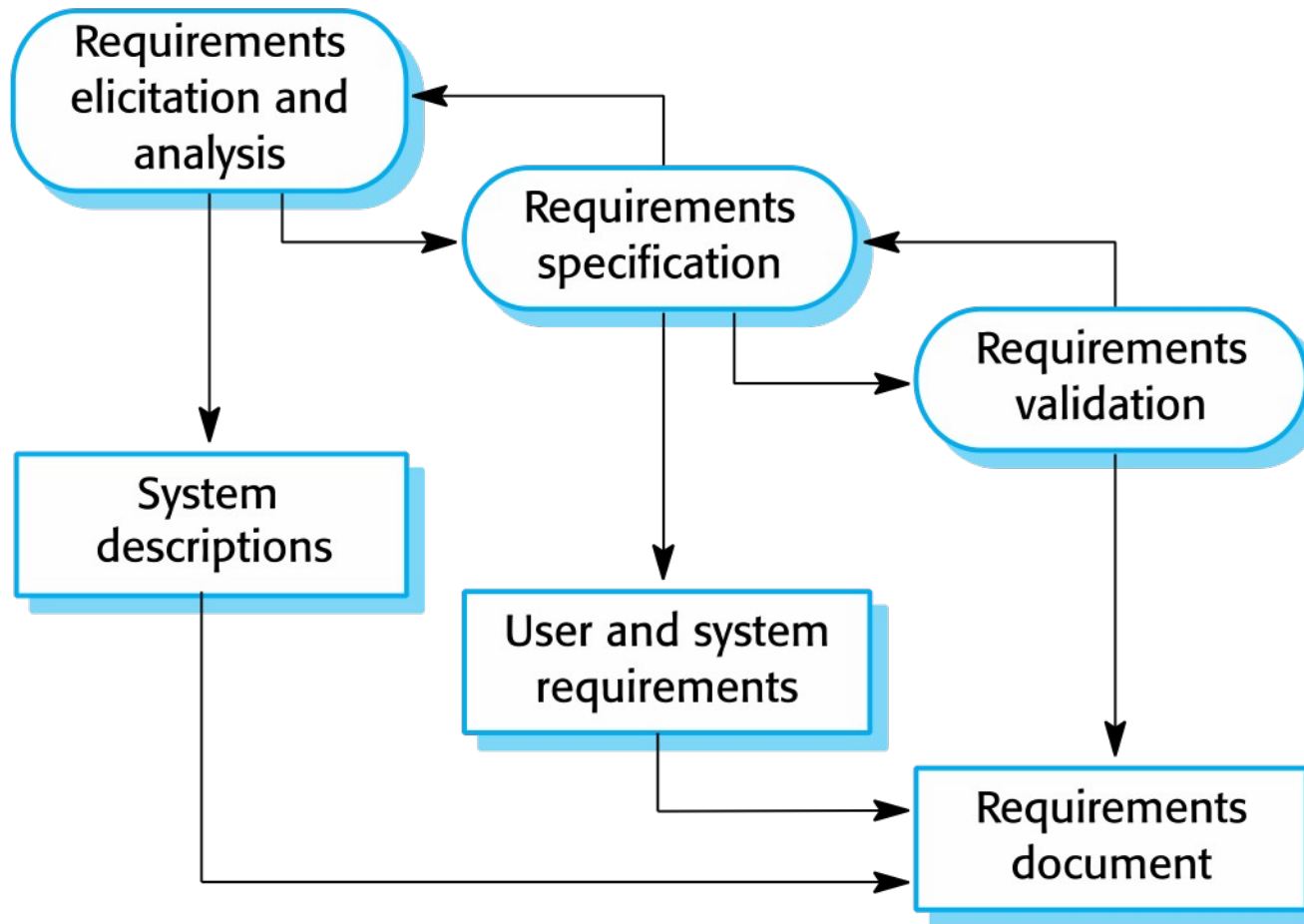
Development

Validation

Evaluation

Four Activities of Software Process Framework

The requirements engineering process



Software specification



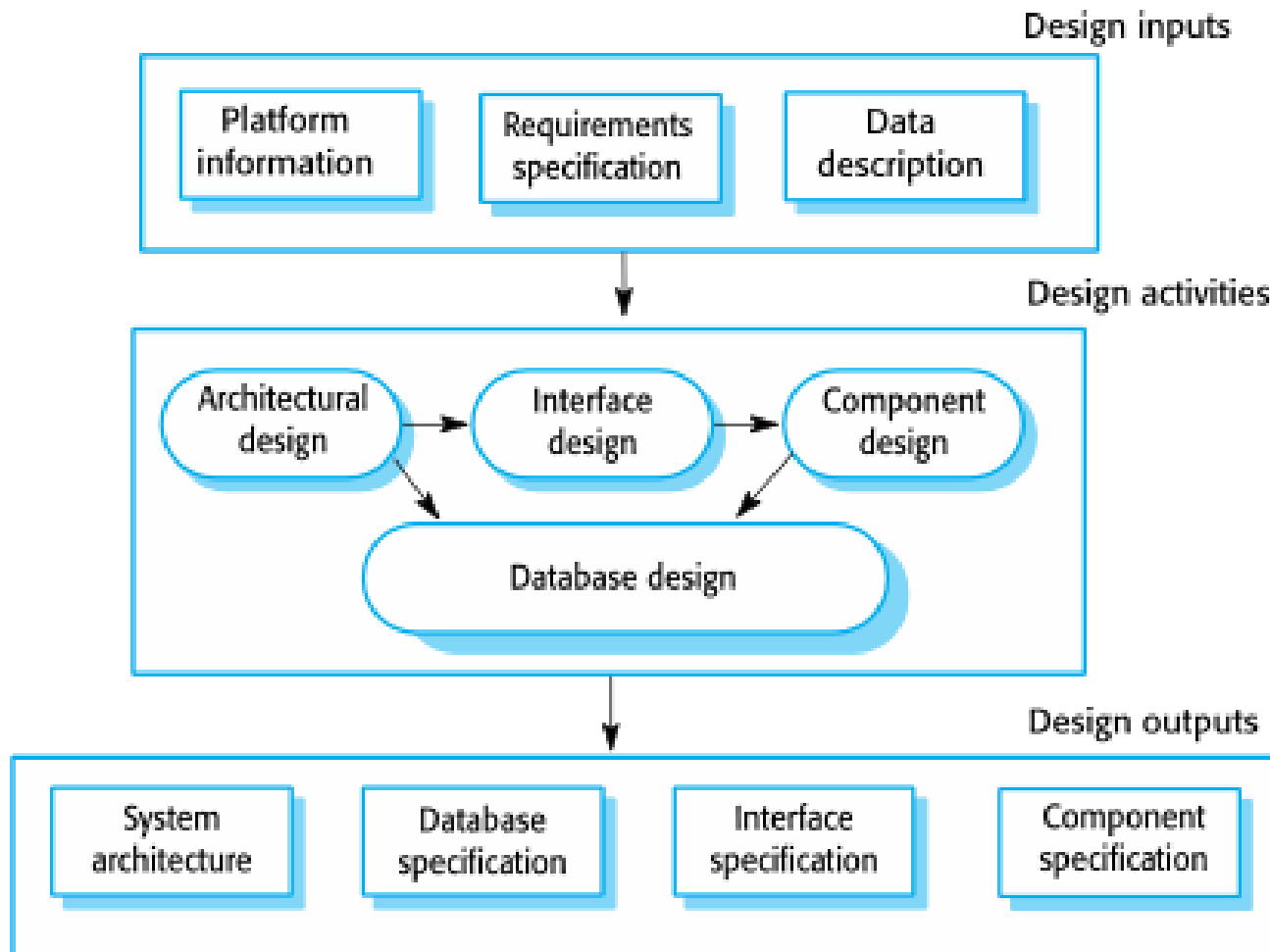
- The process of establishing **what services** are required and the **constraints** on the system's operation and development
- Requirements engineering process
 - Requirements elicitation and analysis
 - What do the system stakeholders require or expect from the system?
 - Requirements specification
 - Defining the requirements in detail
 - Requirements validation
 - Checking the validity of the requirements

Software **design** and **implementation**



- The process of converting the system specification into an executable system
- **Software design**
 - Design a software structure that realizes the specification
- **Implementation**
 - Translate this structure into an executable program
- The activities of design and implementation are closely related and may be inter-leaved

A general model of the design process



Design activities



- ▮ *Architectural design*, where you identify the overall structure of the system, the principal components (subsystems or modules), their relationships and how they are distributed
- ▮ *Database design*, where you design the system data structures and how these are to be represented in a database
- ▮ *Interface design*, where you define the interfaces between system components
- ▮ *Component selection and design*, where you search for reusable components. If unavailable, you design how it will operate.

System implementation



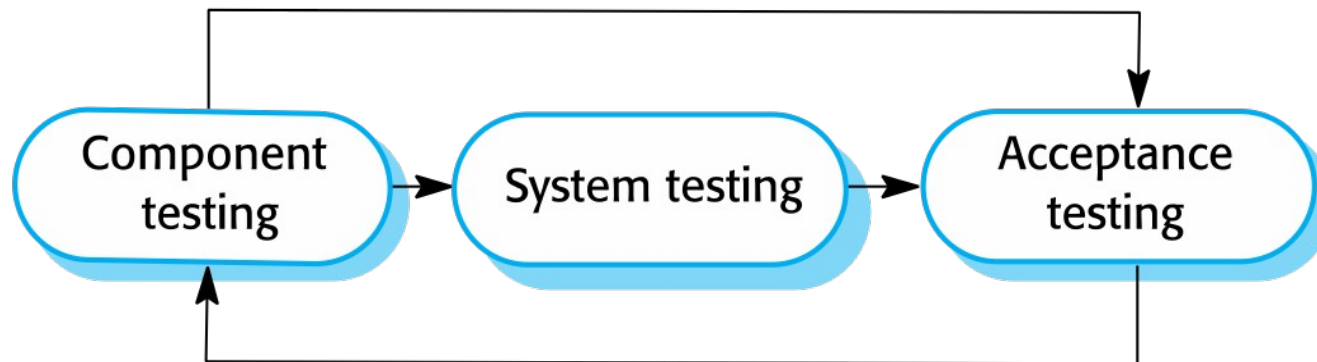
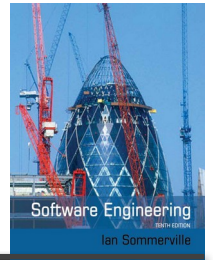
- The software is implemented either by **developing** a program or programs or by **configuring** an application system
- **Design and implementation** are interleaved activities for most types of software system
- **Programming** is an individual activity with no standard process
- **Debugging** is the activity of finding program faults and correcting these faults

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
- Involves checking and review processes and system testing
- System testing involves executing the system with test cases that are derived from the specification of the real data to be processed by the system
- Testing is the most commonly used V & V activity

Stages of testing



Testing stages



□ Component testing

- Individual components are tested independently
- Components may be functions or objects or coherent groupings of these entities

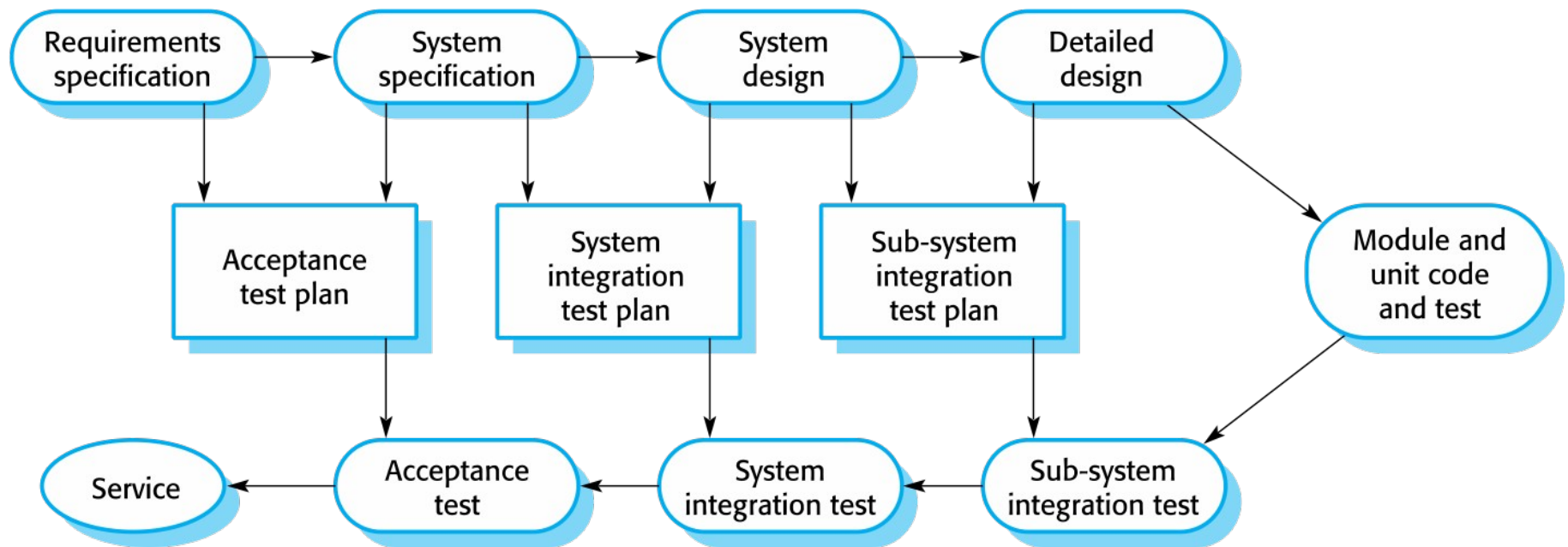
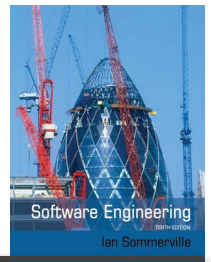
□ System testing

- Testing of the system as a whole. Testing of emergent properties is particularly important.

□ Customer testing

- Testing with customer data to check that the system meets the customer's needs

Testing phases in a plan-driven software process (V-model)

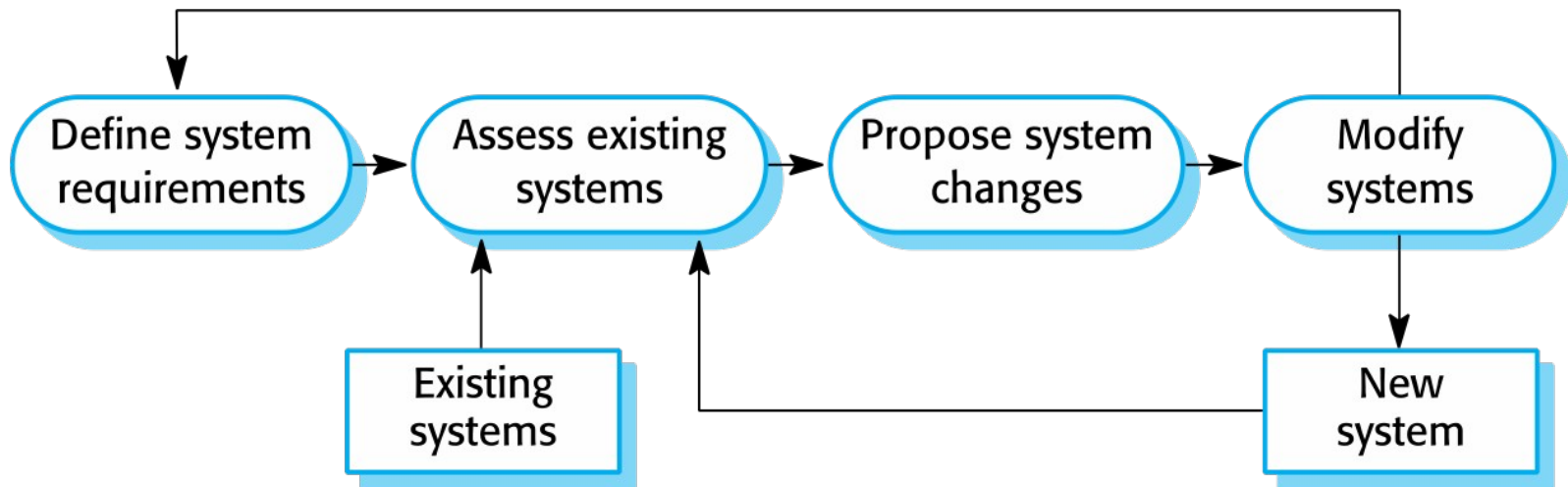


Software **evolution**



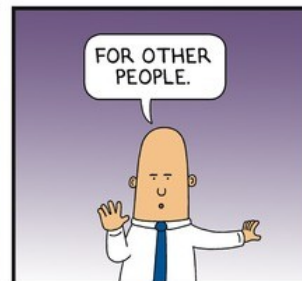
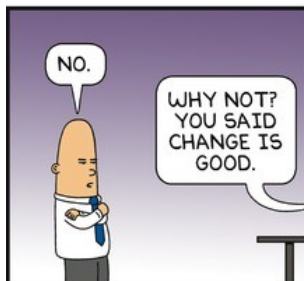
- Software is inherently flexible and can change
- As requirements change through changing business circumstances, the software that supports the business must also evolve and change
- Although there has been a demarcation between development and evolution (maintenance) this is increasingly irrelevant as fewer and fewer systems are completely new

System evolution



Coping with change

DILBERT



BY SCOTT ADAMS

Coping with change



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



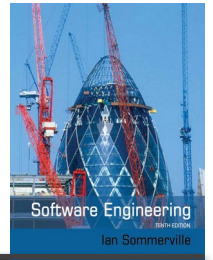
- **Change anticipation**, where the software process includes activities that can anticipate possible changes before significant rework is required
 - For example, a prototype system may be developed to show some key features of the system to customers
- **Change tolerance**, where the process is designed 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 yet been developed. If this is impossible, then only a single increment (a small part of the system) may have be altered to incorporate the change.

Coping with changing requirements



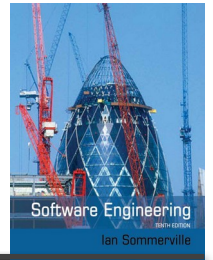
- ▮ **System prototyping**, where a version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of design decisions. This approach supports change anticipation.
- ▮ **Incremental delivery**, where system increments are delivered to the customer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

Software prototyping



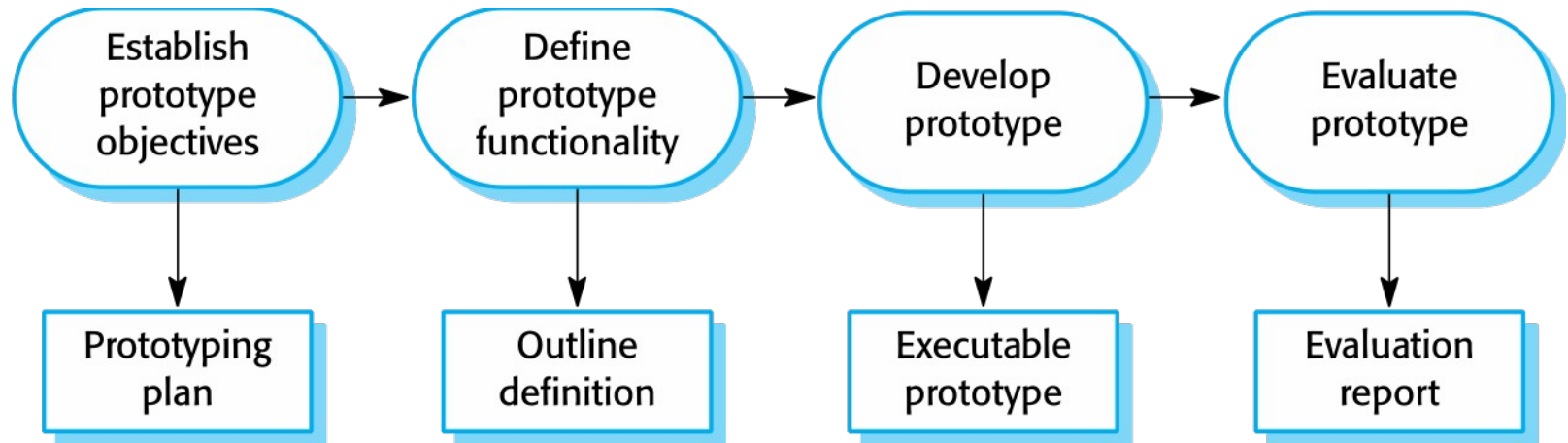
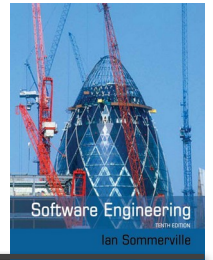
- A **prototype** is an initial version of a system used to demonstrate concepts and try out design options
- A prototype can be used in:
 - The requirements engineering process to help with requirements elicitation and validation
 - In design processes to explore options and develop a UI design
 - In the testing process to run back-to-back tests

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

Throw-away prototypes



- 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



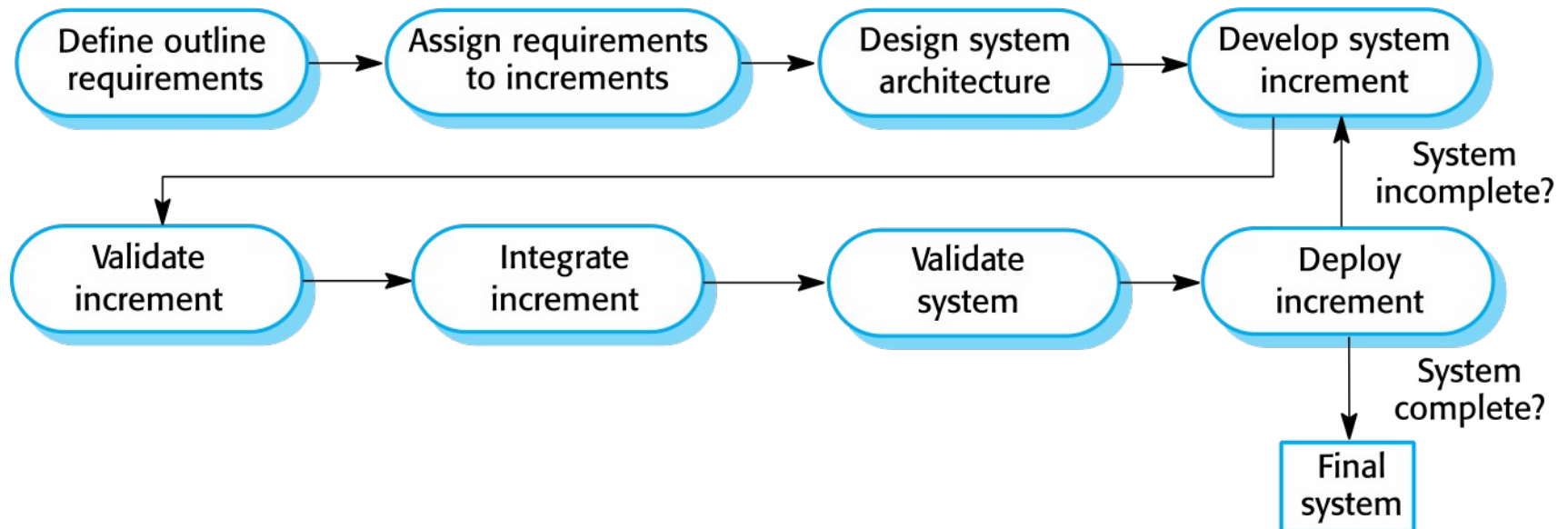
- Rather than deliver the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality
- User requirements are prioritized and the highest priority requirements are included in early increments
- Once the development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

Incremental development and delivery



- **Incremental development** (see earlier Software Process Model #3 – slides 10 and 14-16)
 - Develop the system in versions
 - Can work on parallel versions
 - Improved user/customer involvement (as compared with waterfall)
- **Incremental delivery** (considered Software Process Model #4)
 - Deploy an increment for use by end-users
 - More realistic evaluation about practical use of software
 - Difficult to implement for replacement systems as increments have less functionality than the system being replaced
 - Used in agile development

Incremental delivery



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

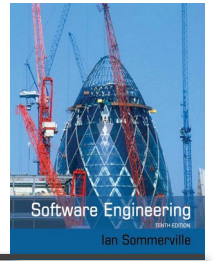


- Most systems require a set of **basic facilities** that are used by different parts of the system
 - As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments
- The essence of iterative processes is that the specification is developed in conjunction 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



Process improvement



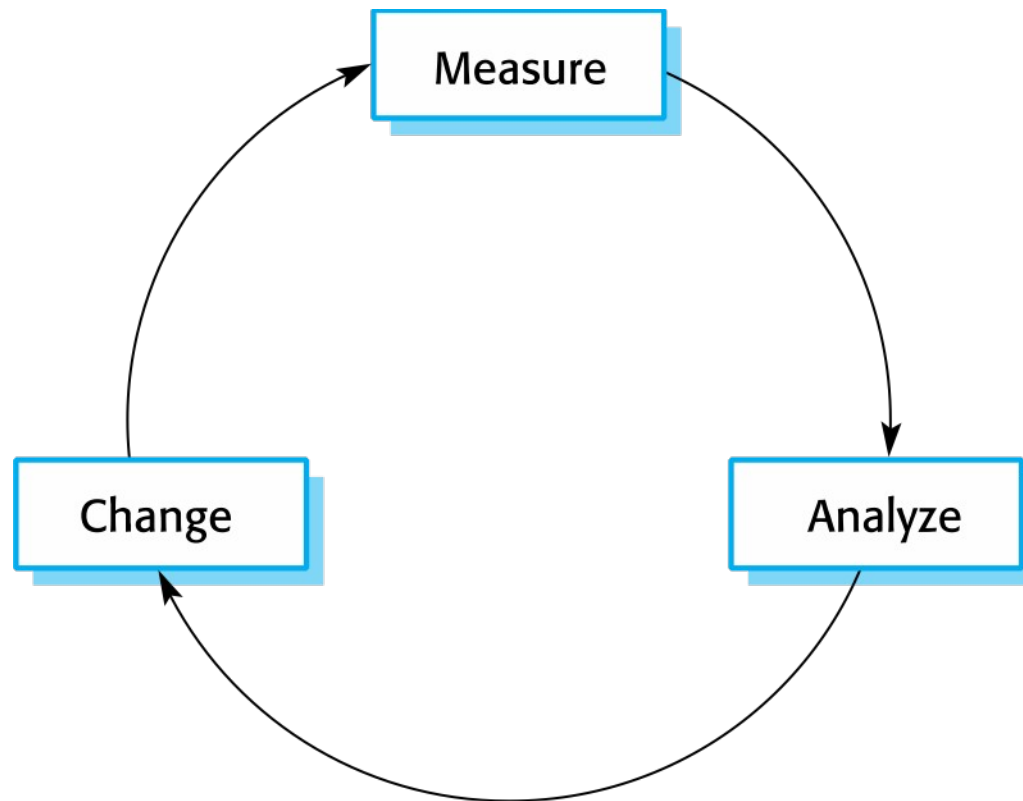
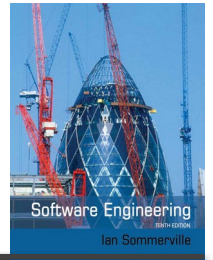
- Many software companies have turned to software **process improvement** as a way of enhancing the quality of their software, reducing costs or accelerating their development processes
- **Process improvement** means understanding existing processes and changing these processes to increase product quality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

Approaches 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 organizational software development processes
- The **agile approach**, which focuses on iterative development and the reduction of overheads in the software process
 - The primary characteristics of agile methods are **rapid delivery of functionality** and **responsiveness to changing customer requirements**

The process improvement cycle



Process improvement activities



□ *Process measurement*

- You measure one or more attributes of the software process or product. These measurements forms a baseline that helps you decide if process improvements have been effective.

□ *Process analysis*

- The current process is assessed, and process weaknesses and bottlenecks are identified. Process models (sometimes called process maps) that describe the process may be developed.

□ *Process change*

- Process changes are proposed to address some of the identified process weaknesses. These are introduced and the cycle resumes to collect data about the effectiveness of the changes.

Process measurement



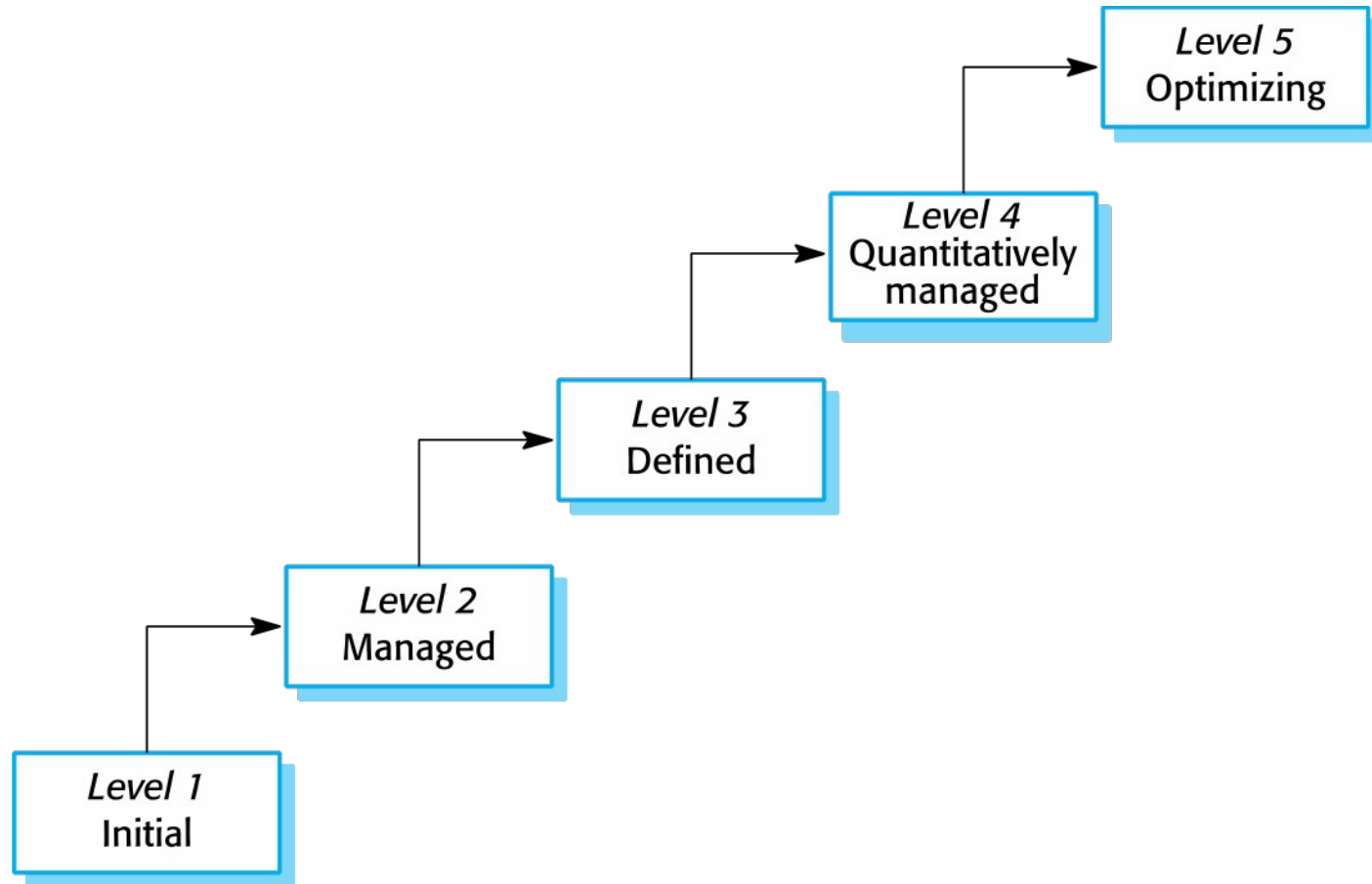
- Wherever possible, **quantitative process data** should be collected
 - However, where organizations do not have clearly defined process standards this is very difficult as you don't know what to measure. A process may have to be defined before any measurement is possible.
- Process **measurements 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.

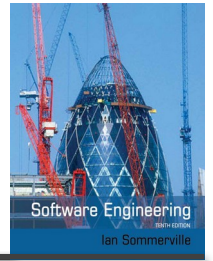
Process metrics



- **Time taken** for process activities to be completed
 - E.g., calendar time or effort to complete an activity or process
- **Resources required** for processes or activities
 - E.g., total effort in person-days
- **Number of occurrences** of a particular event
 - E.g., number of defects discovered.

Capability maturity levels





The SEI capability maturity model

- Initial
 - Essentially uncontrolled
- Repeatable
 - Product management procedures defined and used
- Defined
 - Process management procedures and strategies defined and used
- Managed
 - Quality management strategies defined and used
- Optimizing
 - Process improvement strategies defined and used

Key points



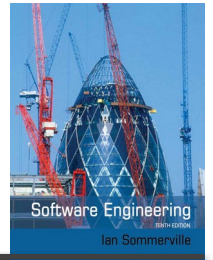
- **Software processes** are the activities involved in producing a software system. Software process models are abstract representations of these processes.
- General process models describe the organization of software processes
 - Examples of these general models include the '**waterfall**' model,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

Key points



- **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



- 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 **SEI process maturity framework** identifies maturity levels that essentially correspond to the use of good software engineering practice