Software Process

- The set of activities to be carried out in order to build a successful software product by meeting quality needs, as well as time and cost expectations
- It defines a framework to:
  - apply methods, techniques and tools
  - build artifacts (both intermediate and final)
  - manage software development projects
  - provide quality assurance
  - manage software changes

Process phases

- A software process adopts a lifecycle that consists of 3 stages (development, maintenance and retirement)
- The development stage consists of two types of phases:
  - **definition**-oriented phases
  - **production**-oriented phases
- The definition-oriented phases deal with "***WHAT***" software has to provide
  - software requirements are defined and specified (analyzed)
- The production-oriented phases deal with "***HOW***" to implement what defined in the previous phases
  - software design, coding, integration and release activities are carried out
- The *maintenance* stage supports the released product by executing both definition-oriented phases and production-oriented phases
- At each phase a testing activity has to be carried to verify what has been produced, according to appropriate *verification and validation* (*V&V*) techniques that are applied to both intermediate artifacts and the final artifact

Types of maintenance

- **Corrective** maintenance, to discover and remove the defects that have produced failures
- **Adaptive** maintenance, to adapt the software product to changes of the operational environment
- **Perfective** maintenance, to extend the software product and include additional functionality
- **Preventive** maintenance (or *software reengineering*), which consists of modify the product so as to simplify and reduce the effort of maintenance activities

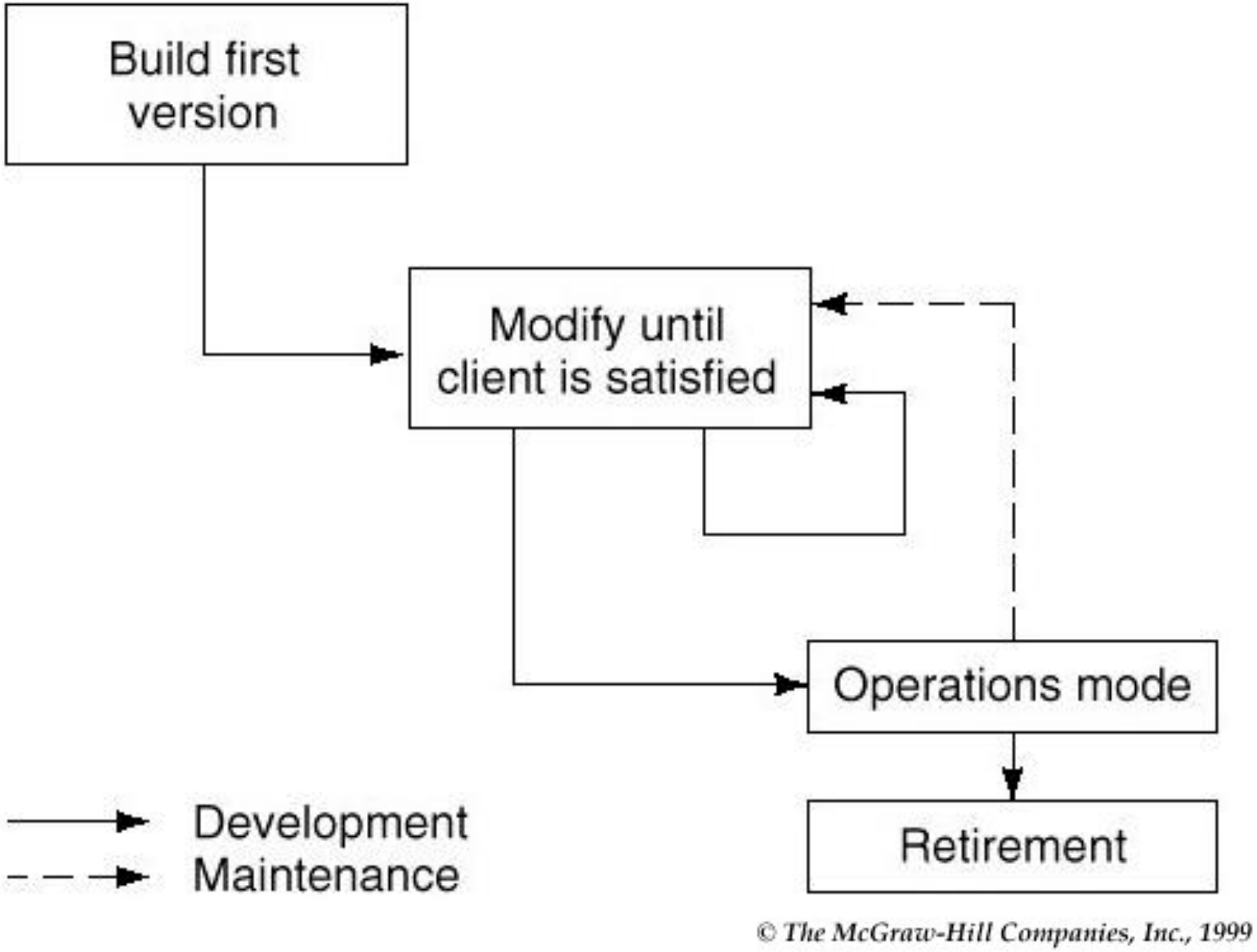
Lifecycle definition

- Def. IEEE Std 610-12 (*Software Eng. Terminology*)
  - the period of time that begins when a software product is conceived and ends when the software is no longer available for use
  - it includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance and, sometimes, retirement phase
  - *Note*: these phases may overlap or be performed iteratively

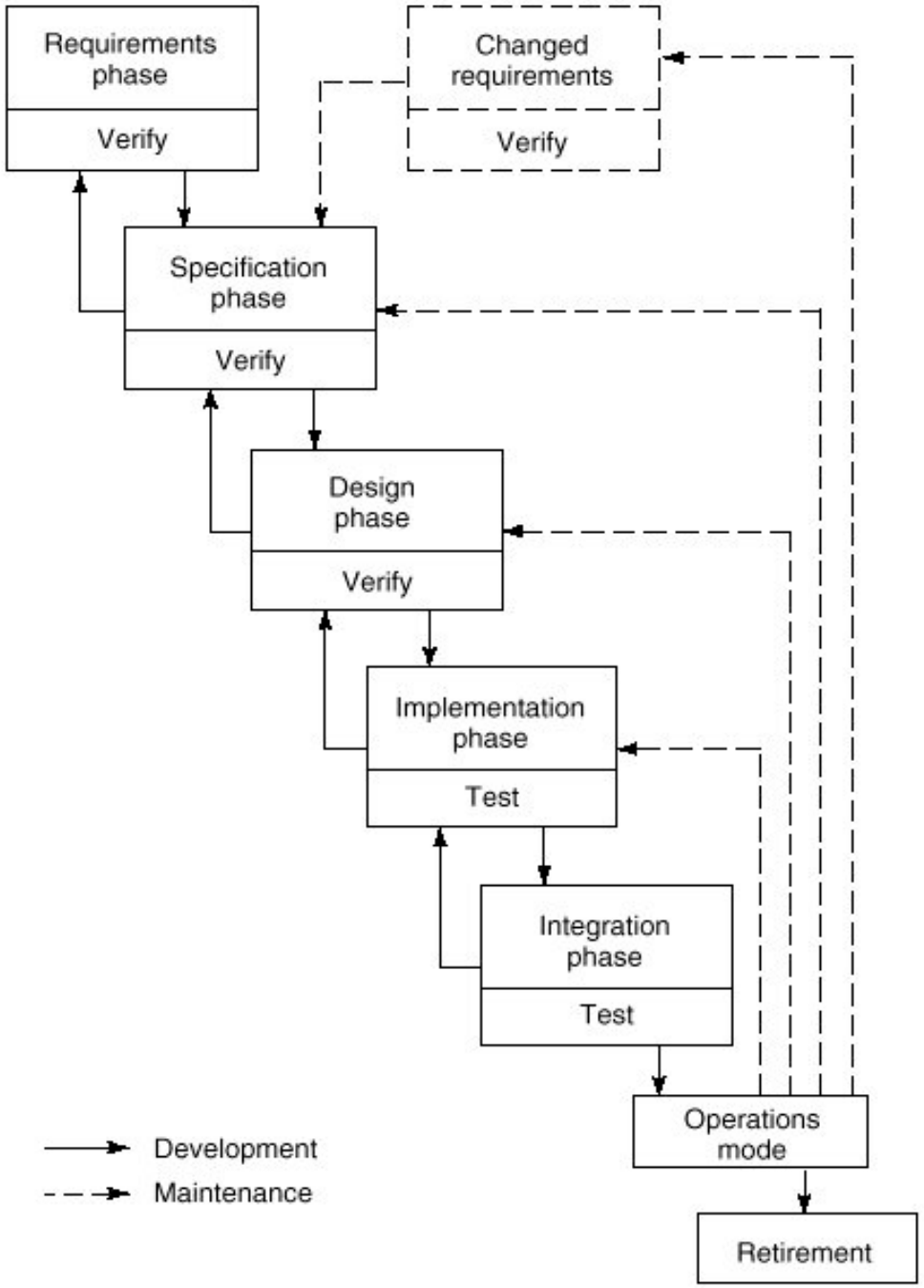
Lifecycle Models

- A **software lifecycle model** specifies the series of phases through which the software development advances and the order in which they are executed, from the requirements phase down to retirement
- The choice of a given model depends on the type of software, the maturity of the organization that develops the software, the methods and technologies used and possible constraints imposed by the customer
- The absence of lifecycle model conventionally corresponds to a software development approach referred to as "*build & fix*" (or "*fix-it-later*"), in which the software product is implemented and then reworked until the customer needs are satisfied

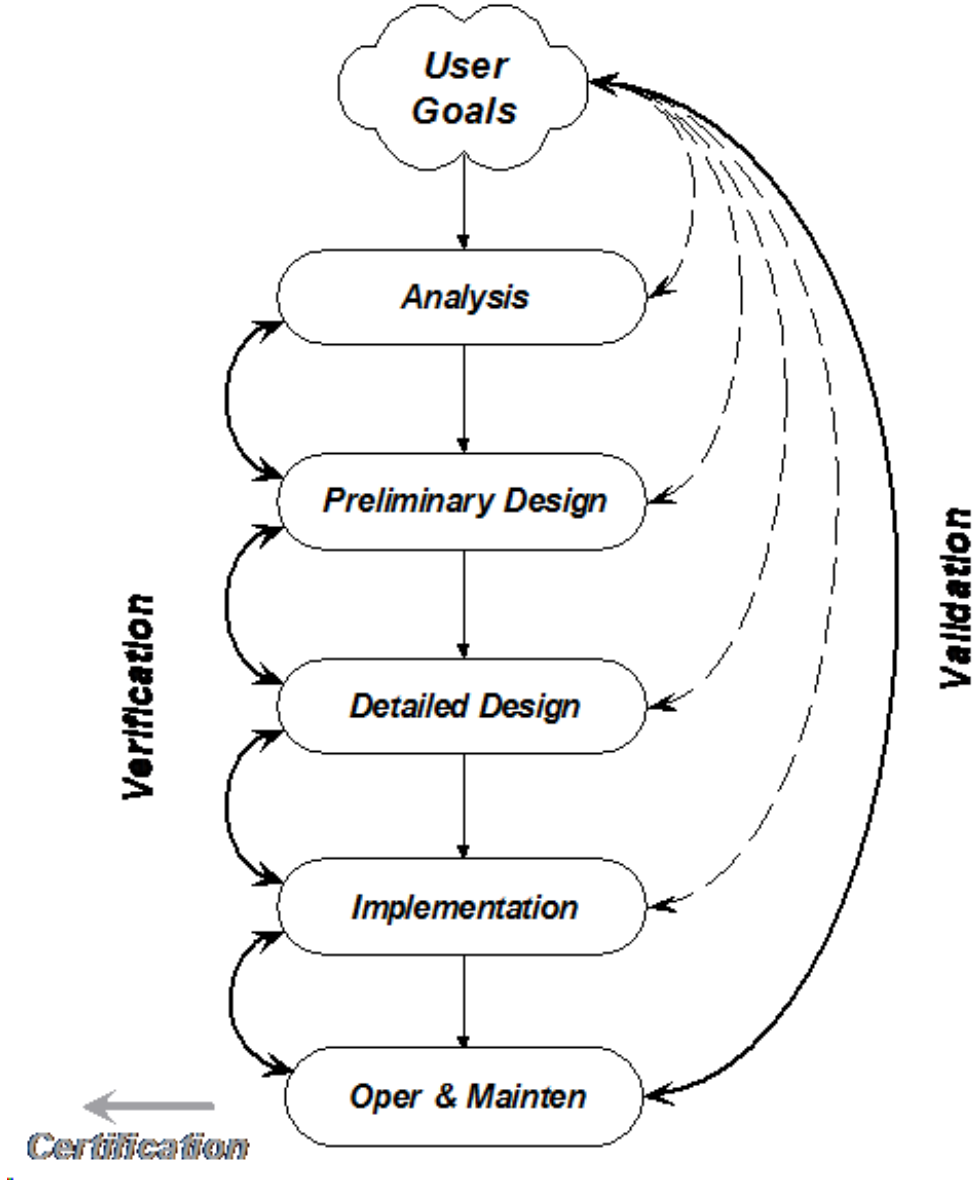
Build&Fix



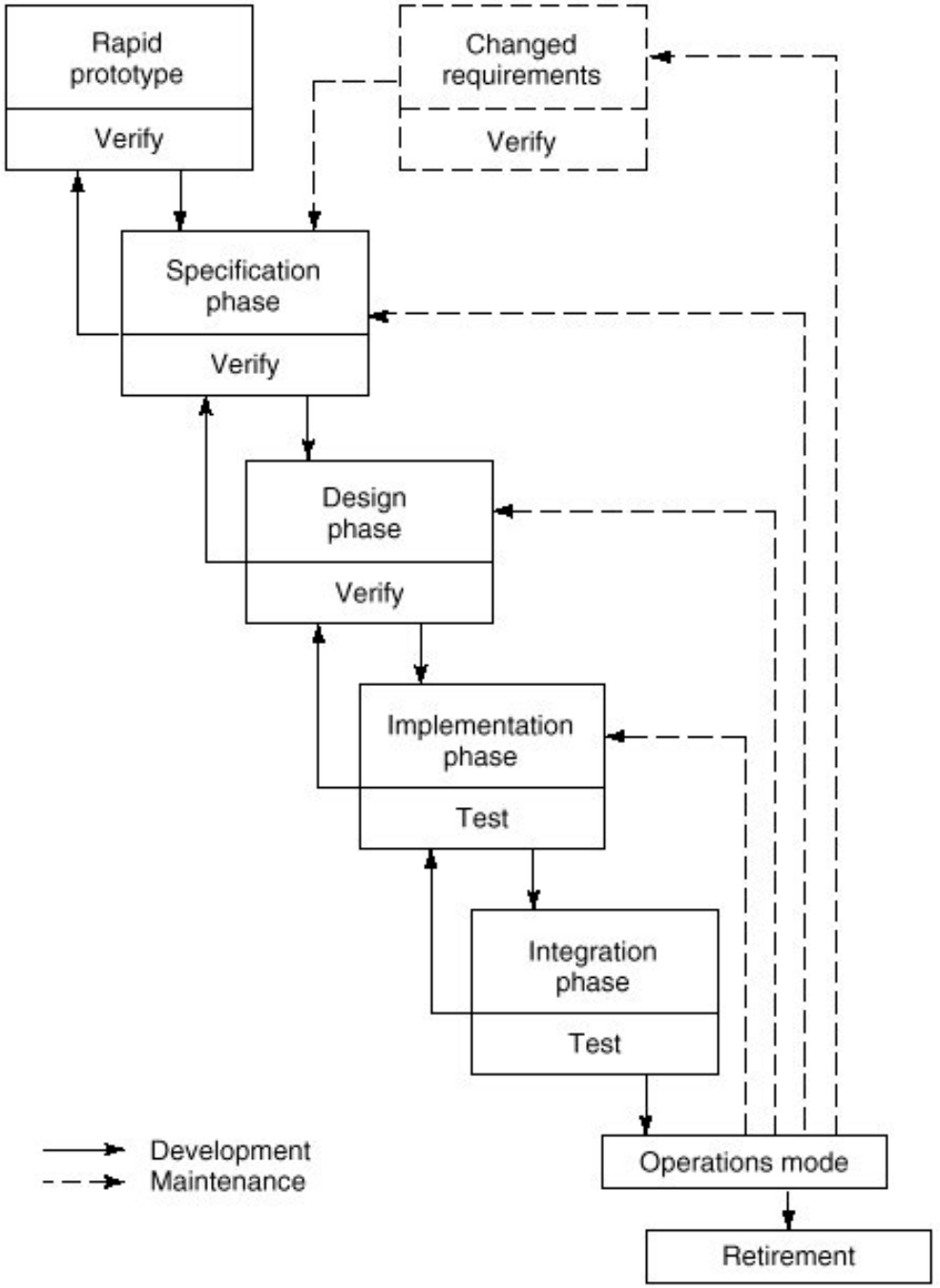
Waterfall Model



Verification & Validation (V&V) activities



Rapid Prototyping Model



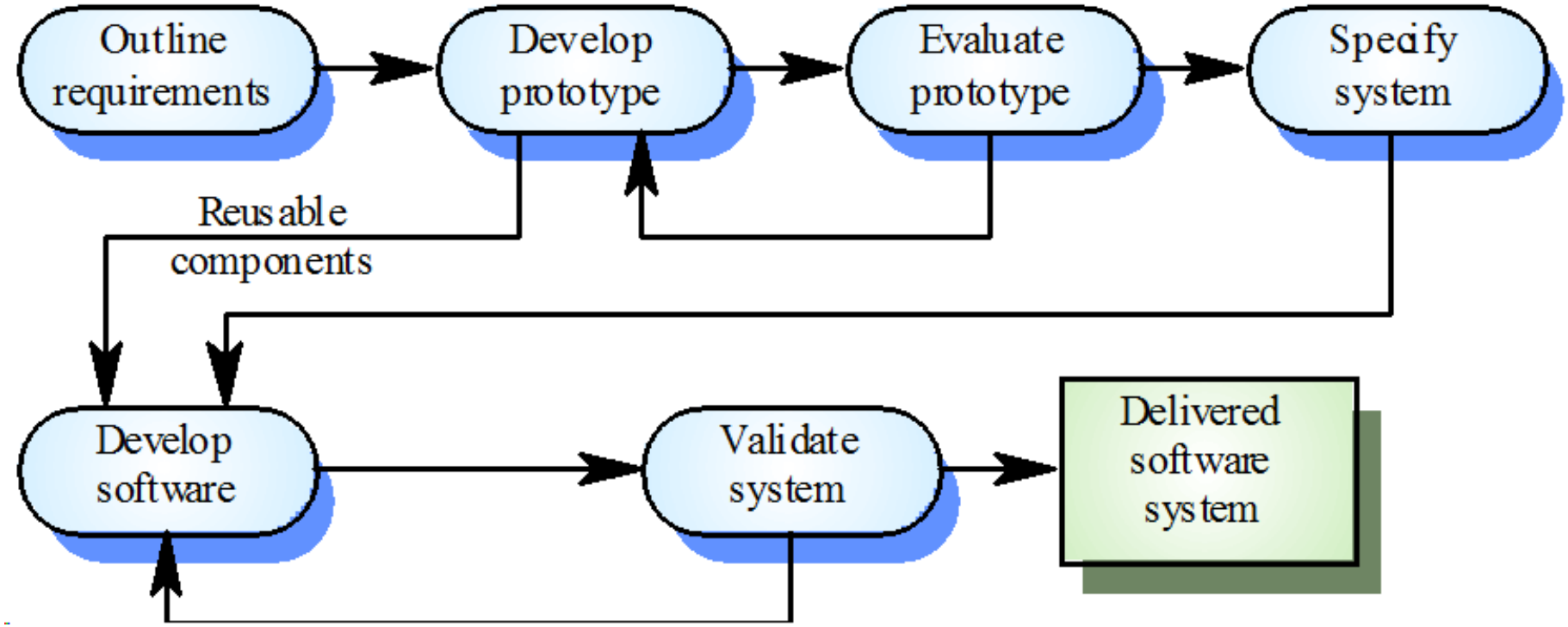
Uses of system prototypes

- The principal use is to help customers and developers understand the software requirements
  - Requirements elicitation: users can experiment with a prototype to see how the system supports their work
  - Requirements validation: the prototype can reveal errors and omissions in the requirements
- Prototyping can be considered as a risk reduction activity which reduces requirements risks

Throw-away prototyping

- A prototype which is usually a practical implementation of the product is produced to help discover requirements problems and then discarded. The product is then developed using some other development process
- Used to reduce requirements risk
- The prototype is developed from an initial requirement, delivered for experiment then discarded
- The throw-away prototype should NOT be considered as a final product
  - Some characteristics may have been left out
  - There is no specification for long-term maintenance
  - The product will be poorly structured and difficult to maintain

Throw-away prototyping process



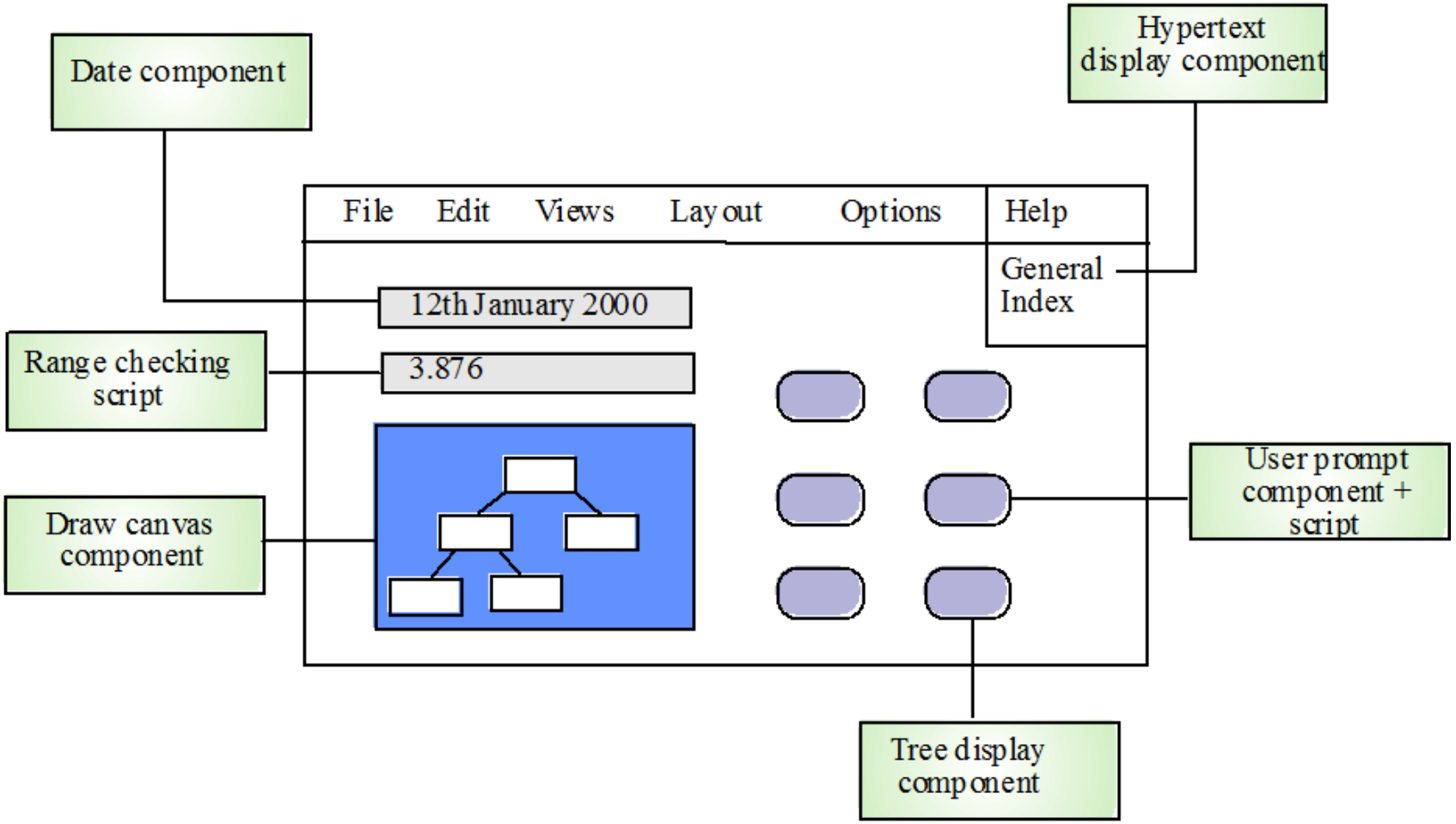
Prototyping key points

- A prototype can be used to give end-users a concrete impression of the product's capabilities
- Prototyping is becoming increasingly used for product development where rapid development is essential
- Throw-away prototyping is used to understand the product requirements
- Rapid development of prototypes is essential. This may require leaving out functionality or relaxing non-functional constraints
- Visual programming is an inherent part of most prototype development methods

Visual programming

- Scripting languages such as Visual Basic support visual programming where the prototype is developed by creating a user interface from standard items and associating components with these items
- A large library of components exists to support this type of development
- These may be tailored to suit the specific application requirements

Visual programming (2)



Process iteration

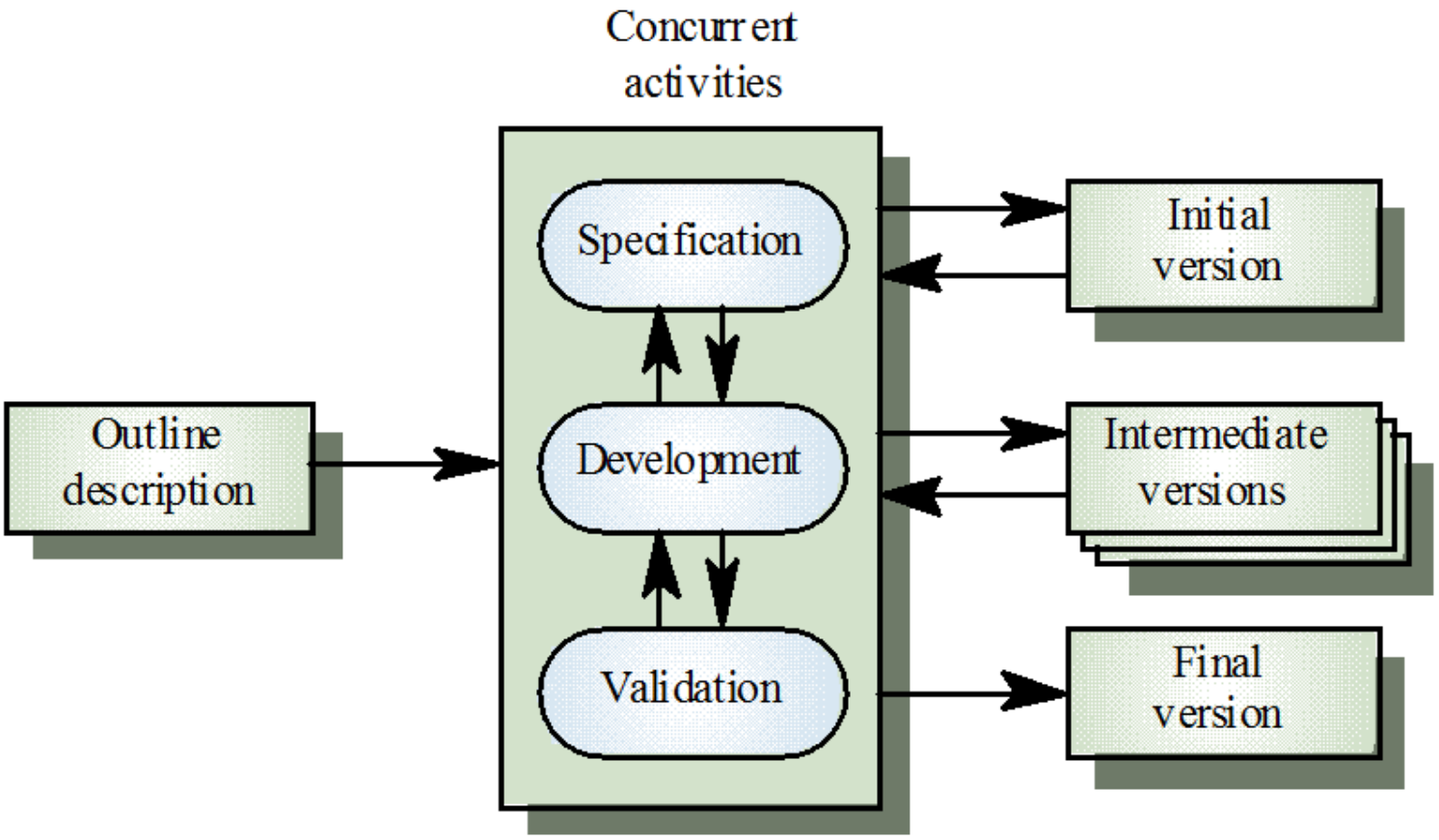
- Requirements ALWAYS evolve in the course of a project so process iteration where earlier stages are reworked is always part of the process for large products
- Iteration can be applied to any of the generic process models
- Two (related) approaches
  - Incremental development
  - Spiral development

Incremental development

- The product is developed and delivered in increments after establishing an overall architecture
- Requirements and specifications for each increment may be developed
- Users may experiment with delivered increments while others are being developed. Therefore, these serve as a form of prototype
- Intended to combine some of the advantages of prototyping but with a more manageable process and better structure

Incremental Model

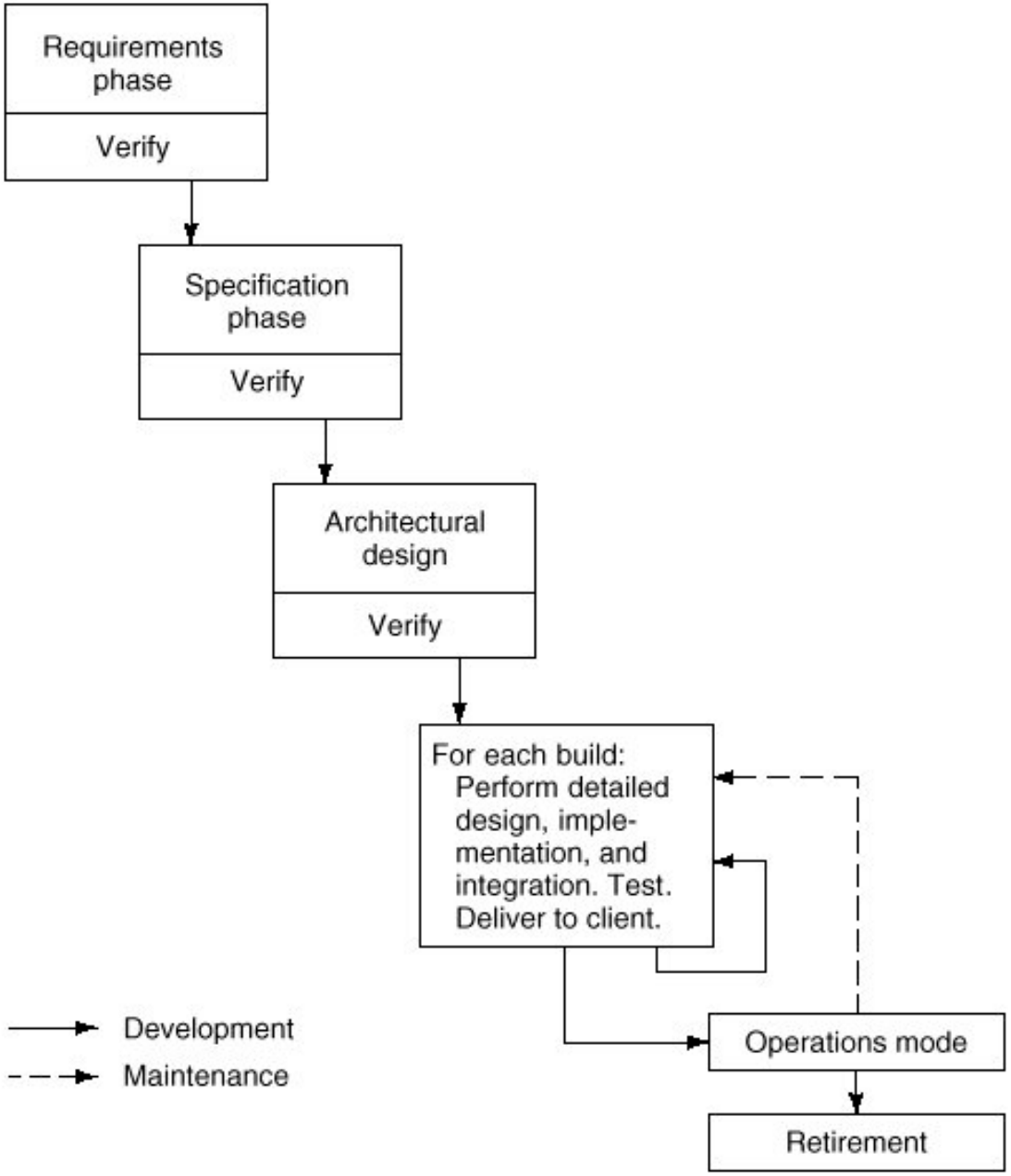
- The software product is developed and released as a set of consecutive *builds*



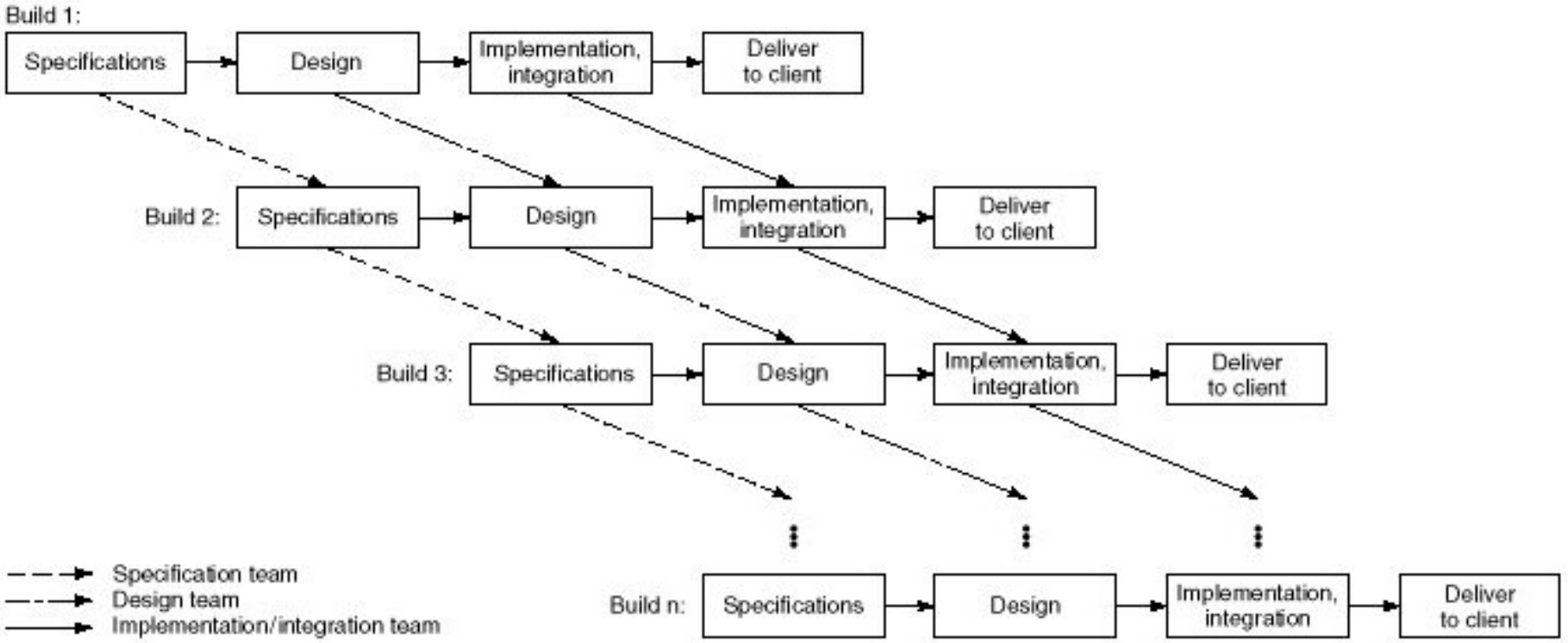
Incremental Model (cont.)

- Includes some of the advantages of the *rapid prototyping* model (the user may experiment with the delivered product builds, while the remaining ones are under development)
- Is effective when the customer wants to be kept up to date about the development advances, as well as when requirements change
- May be carried out according to two alternatives:
  - version with *overall architecture*
  - version without *overall architecture* (more risky)

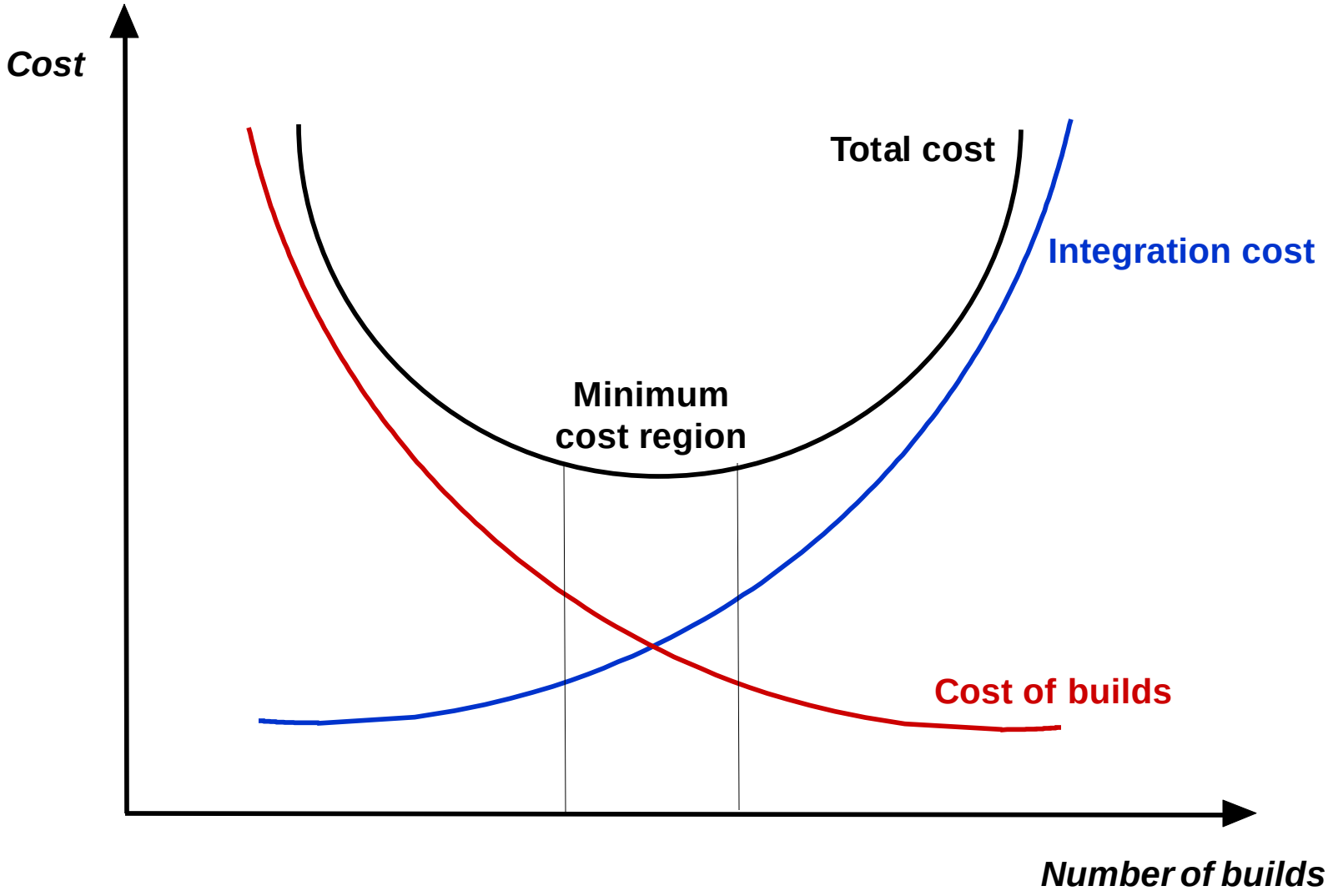
Version WITH *overall architecture*



Version WITHOUT *overall architecture*



Impact on software costs



Incremental vs. Waterfall

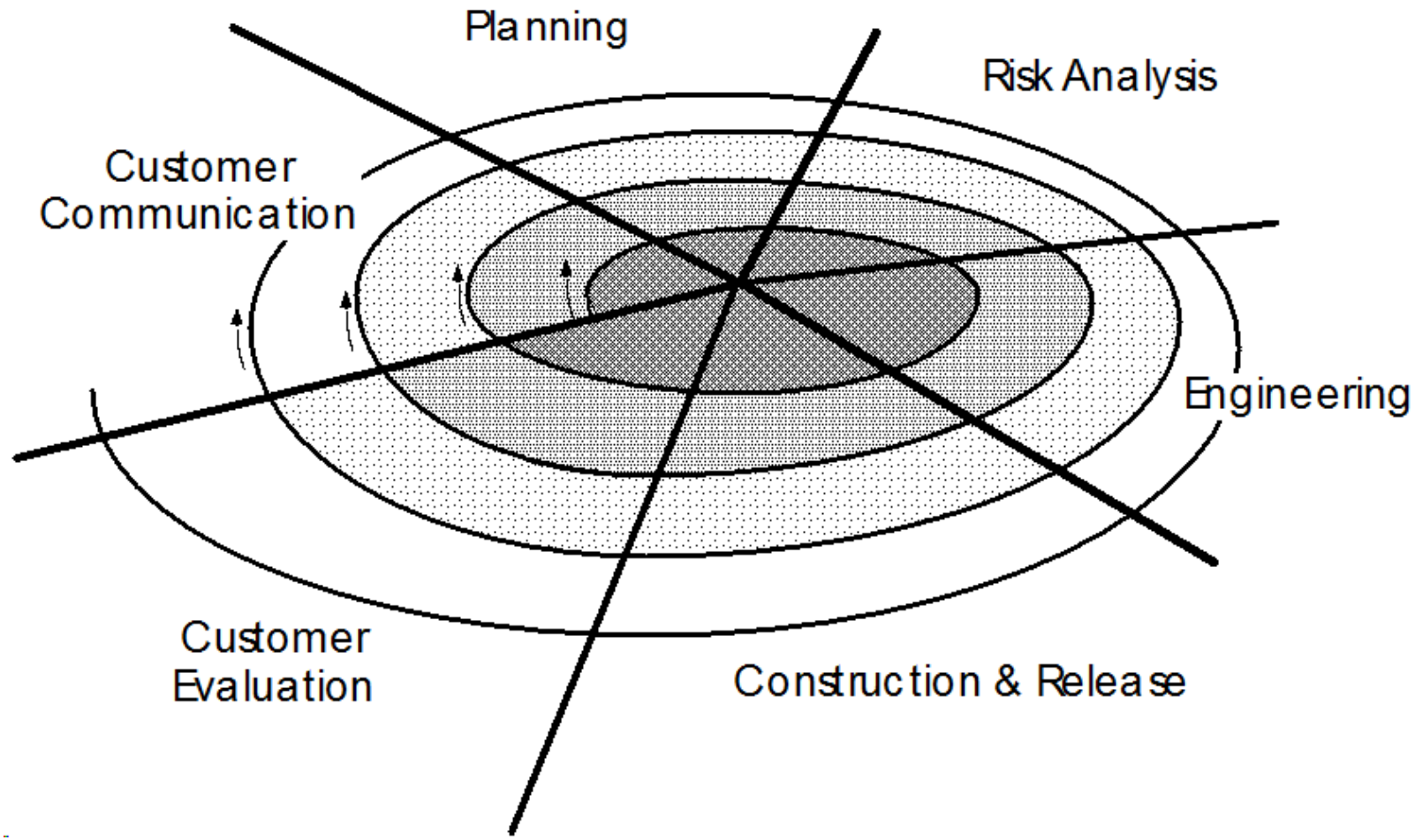
Waterfall Model

- Requirements “frozen” at the end of specification phase
- Customer feedback only at the end of development
- Phases to be executed sequentially (output of a phase taken as input by the next phase)
- Detailed design and coding on the entire product
- A single development team composed of a large number of people

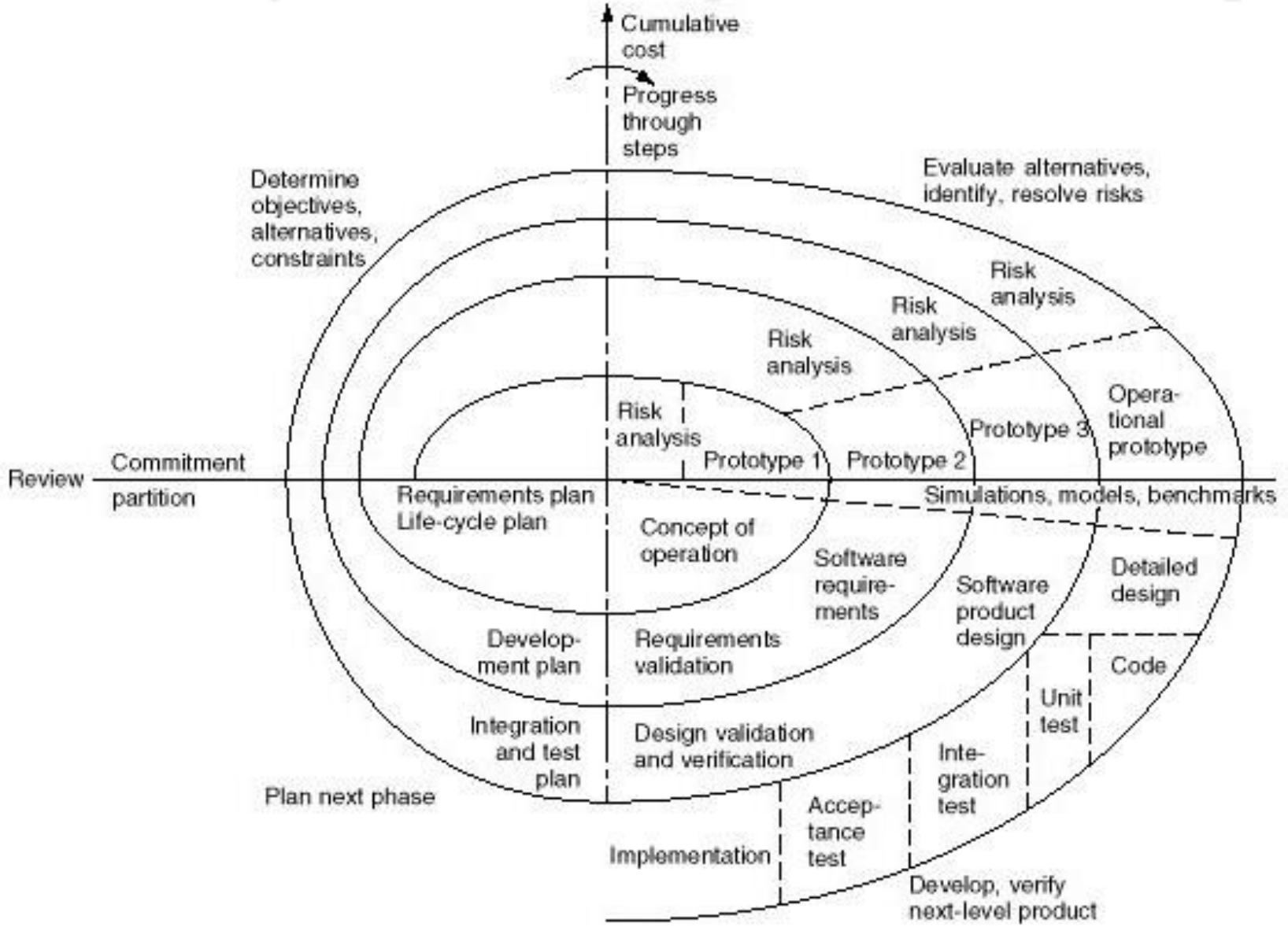
Incremental Model

- Requirements prioritized and easier to modify
- Continuous customer feedback
- Phases may be executed concurrently
- Detailed design and coding on single *builds*
- Various development teams, each composed of a small number of people

Spiral Model



Full-Spiral Model [Boehm, 1988]



Risk management

- Risk management is concerned with identifying risks and drawing up plans to minimise their effect on a project
- A *risk* is a *probability* that some adverse circumstance will occur
- Categories of risk
  - *Project* risks affect schedule or resources
  - *Product* risks affect the quality or performance of the software being developed
  - *Business* risks affect the organisation developing or procuring the software

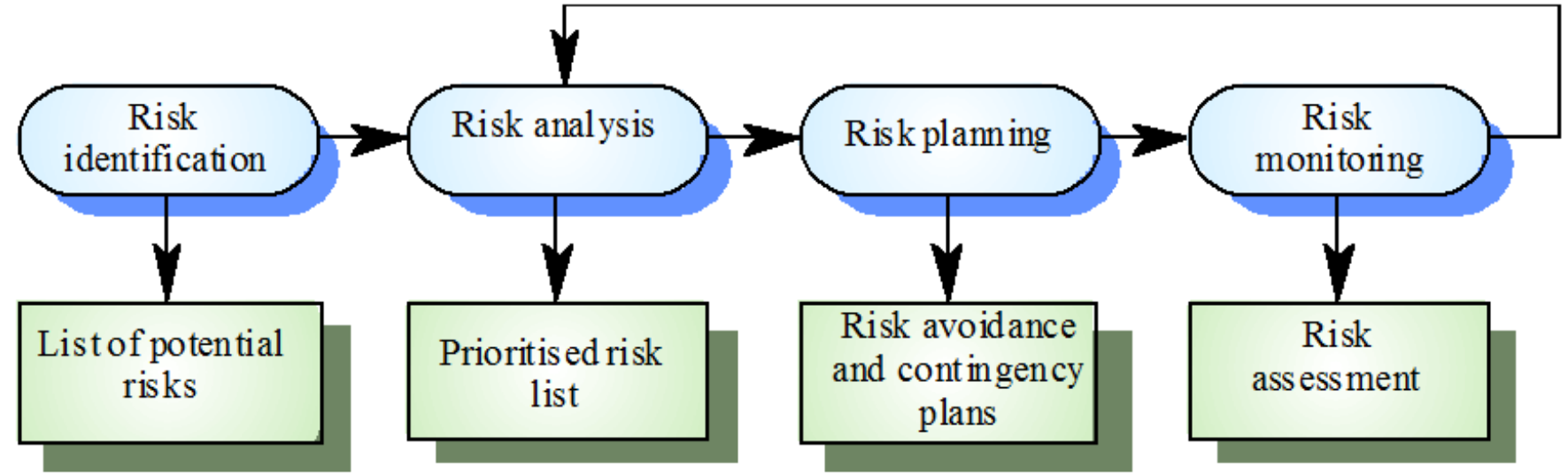
Risks by category

Risk	Risk type	Description
Staff turnover	Project	Experienced staff will leave the project before it is finished.
Management change	Project	There will be a change of organisational management with different priorities.
Hardware unavailability	Project	Hardware which is essential for the project will not be delivered on schedule.
Requirements change	Project and product	There will be a larger number of changes to the requirements than anticipated.
Specification delays	Project and product	Specifications of essential interfaces are not available on schedule
Size underestimate	Project and product	The size of the system has been underestimated.
CASE tool under-performance	Product	CASE tools which support the project do not perform as anticipated
Technology change	Business	The underlying technology on which the system is built is superseded by new technology.
Product competition	Business	A competitive product is marketed before the system is completed.

The risk management process

- Risk identification
  - Identify project, product and business risks
- Risk analysis
  - Assess the likelihood and consequences of these risks
- Risk planning
  - Draw up plans to avoid or minimise the effects of the risk
- Risk monitoring
  - Monitor the risks throughout the project

The risk management process (2)



Risk identification (1)
Risk types

- Technology risks
- People risks
- Organisational risks
- Tools risks
- Requirements risks
- Estimation risks

Risk identification (2)

Risk type	Possible risks
Technology	The database used in the system cannot process as many transactions per second as expected. Software components which should be reused contain defects which limit their functionality.
People	It is impossible to recruit staff with the skills required. Key staff are ill and unavailable at critical times. Required training for staff is not available.
Organisational	The organisation is restructured so that different management are responsible for the project. Organisational financial problems force reductions in the project budget.
Tools	The code generated by CASE tools is inefficient. CASE tools cannot be integrated.
Requirements	Changes to requirements which require major design rework are proposed. Customers fail to understand the impact of requirements changes.
Estimation	The time required to develop the software is underestimated. The rate of defect repair is underestimated. The size of the software is underestimated.

Risk analysis (1)

- Assess probability and seriousness of each risk
- Risk *probability* may be:
  - very low (<10%)
  - low (10-25%)
  - moderate (25-50%)
  - high (50-75%)
  - very high (>75%)
- Risk *effects* might be catastrophic, serious, tolerable or insignificant

Risk analysis (2)

Risk	Probability	Effects
Organisational financial problems force reductions in the project budget.	Low	Catastrophic
It is impossible to recruit staff with the skills required for the project.	High	Catastrophic
Key staff are ill at critical times in the project.	Moderate	Serious
Software components which should be reused contain defects which limit their functionality.	Moderate	Serious
Changes to requirements which require major design rework are proposed.	Moderate	Serious
The organisation is restructured so that different management are responsible for the project.	High	Serious
The database used in the system cannot process as many transactions per second as expected.	Moderate	Serious
The time required to develop the software is underestimated.	High	Serious
CASE tools cannot be integrated.	High	Tolerable
Customers fail to understand the impact of requirements changes.	Moderate	Tolerable
Required training for staff is not available.	Moderate	Tolerable
The rate of defect repair is underestimated.	Moderate	Tolerable
The size of the software is underestimated.	High	Tolerable
The code generated by CASE tools is inefficient.	Moderate	Insignificant

Risk analysis (3)

- Identify e.g., the *top-ten risks* by considering:
  - all *catastrophic* risks
  - all *serious* risks that have more than a *moderate* probability of occurrence
- Rank such risks by order of importance

Risk planning

- Consider each risk and develop a strategy to manage that risk
- Avoidance strategies
  - The probability that the risk will arise is reduced
- Minimisation strategies
  - The impact of the risk on the project or product will be reduced
- Contingency plans
  - If the risk arises, contingency plans are strategies to deal with that risk

Risk management strategies

Risk	Strategy
Organisational financial problems	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Recruitment problems	Alert customer of potential difficulties and the possibility of delays, investigate buying-in components.
Staff illness	Reorganise team so that there is more overlap of work and people therefore understand each other's jobs.
Defective components	Replace potentially defective components with bought-in components of known reliability.
Requirements changes	Derive traceability information to assess requirements change impact, maximise information hiding in the design.
Organisational restructuring	Prepare a briefing document for senior management showing how the project is making a very important contribution to the goals of the business.
Database performance	Investigate the possibility of buying a higher-performance database.
Underestimated development time	Investigate buying in components, investigate use of a program generator.

Risk monitoring (1)

- Assess each identified risks regularly to decide whether or not it is becoming less or more probable
- To perform assessment look at ***risk factors*** (see next slide)
- Also assess whether the effects of the risk have changed (in such case go back to risk analysis)
- Each key risk should be discussed at management progress meetings

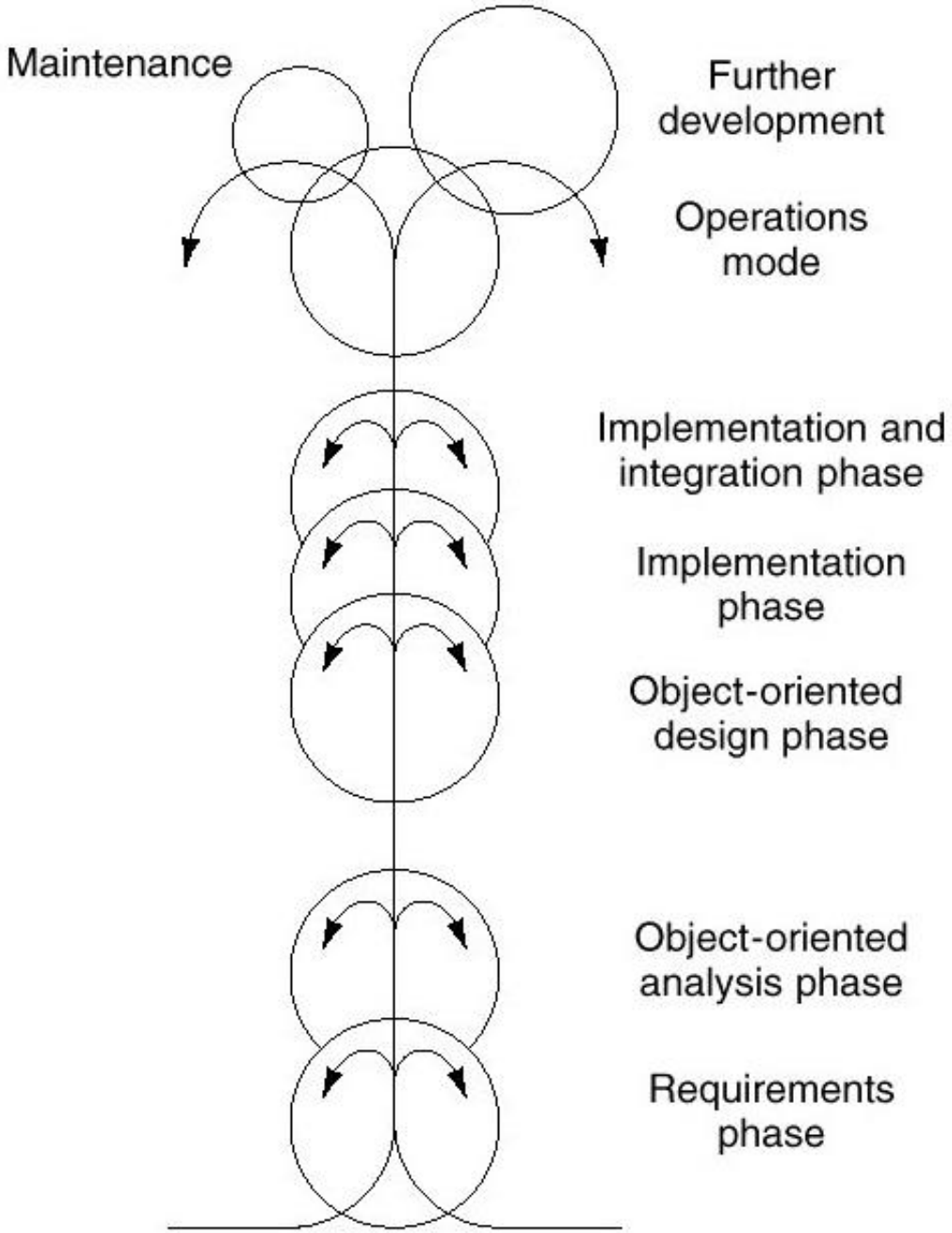
Risk monitoring (2)

Risk factors

Risk type	**Potential indicators**
Technology	Late delivery of hardware or support software, many reported technology problems
People	Poor staff morale, poor relationships amongst team member, job availability
Organisational	organisational gossip, lack of action by senior management
Tools	reluctance by team members to use tools, complaints about CASE tools, demands for higher-powered workstations
Requirements	many requirements change requests, customer complaints
Estimation	failure to meet agreed schedule, failure to clear reported defects

Additional models (1)

Object-oriented Model



Additional Models (2)

- ***Concurrent engineering*** **model**
  - intended to reduce development time and cost, by use of a systematic approach to the integrated and concurrent design of both the product and the associated process
  - the development phases coexist rather than being executed sequentially
- **Model based on *formal methods***
  - includes a set of activities that lead to the formal specification of the software product, in order to avoid ambiguity, incompleteness and inconsistency and to make easier software verification by use of algebraic techniques and tools
  - A significant example is the ***Cleanroom Software Engineering*** (1987), which emphasizes the possibility of detecting defects earlier with respect to conventional models

Agile Methods

- In the early 2000's, a reaction was witnessed against the importance of carefully planned software processes, stating that such processes are too restrictive on the developers
- The term *agile method* was introduced to extend the original concept of iterative and incremental development to concepts such as intensive communication within the project, fast feedback, few external rules for the way of working, etc.
- The common values and principles of agile methods are summarized in the *Agile Manifesto*

Agile Manifesto (2001)

- The Agile Manifesto defines *agile values*

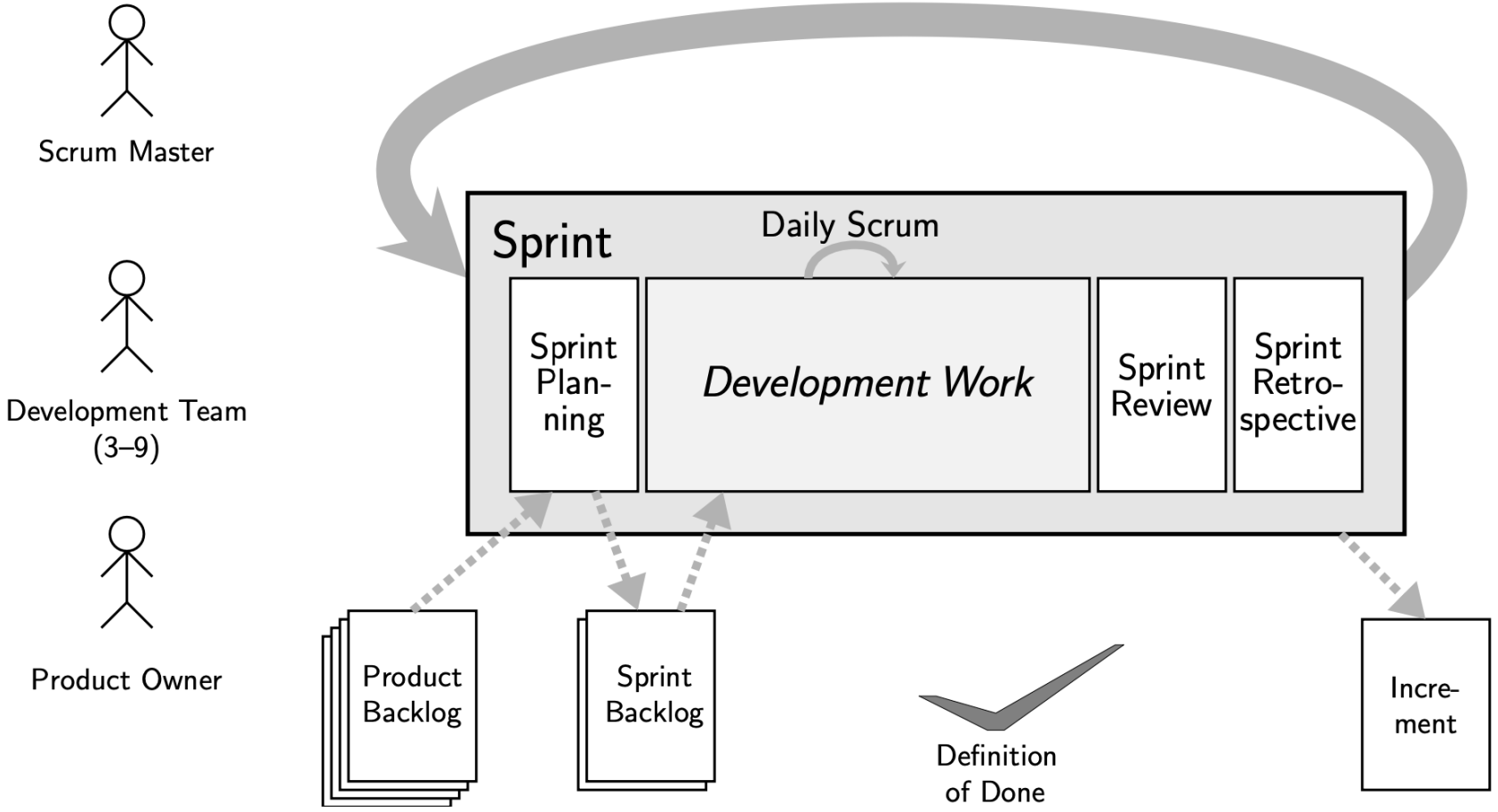
  "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

  – **Individuals and interactions** over *processes and tools*
  – **Working software** over *comprehensive documentation*
  – **Customer collaboration** over *contract negotiation*
  – **Responding to change** over *following a plan*

  That is, while there is value in the items on the right, we value the items on the left more."

- In addition to values, the Agile Manifesto contains the *twelve agile principles* that provide additional guidance on the use of agile methods
- Together, these values and principles define the basic concepts that are today known as *agile development*

Scrum


Scrum Master
Development Team
(3–9)
Product Owner
Sprint
Daily Scrum
Sprint Plan-ning
Development Work
Sprint Review
Sprint Retro-spective
Product Backlog
Sprint Backlog
Definition of Done
Incre-ment

Scrum Roles

- The *Scrum master*:
  - ensures that the methodology is understood and properly implemented by the development team and the product owner
  - supports the team by helping everyone else to interact with the team according to the Scrum rules
- The *product owner* manages and helps prioritizing the requirements to be implemented as documented in the product backlog
- The *development team* is responsible for developing the product, including all relevant tasks (e.g., designing, coding and testing)

Sprints

- A ***sprint*** is carried out to deliver a new increment of working software, and typically takes 2 to 4 weeks
- It is started with the *sprint planning* meeting where the Scrum team transfers the items to be developed from the product backlog and transfers them into the sprint backlog
- During the sprint, the development team works on the increment, with short *daily Scrum* meetings (aka stand-up meeting) of the development team to synchronize work and address any problems
- At the end of a sprint, the increment is presented to the product owner and other stakeholders in a *sprint review*
- Finally, the *sprint retrospective* meeting is performed to identify and plan any improvements for the next sprint

Definition of *Done*

- Scrum requires a “definition of done”, where the development team defines for itself what it means for a work item to be done (i.e., before it is allowed to be integrated into the main branch)
- Typical minimum requirements included in this “definition of done” include an adequate number of test cases, as well as checking the integration of the new code to ensure that it does not break the main development branch
- The definition of done also requires that the code has been documented adequately, where the team defines for itself what ”adequate” means

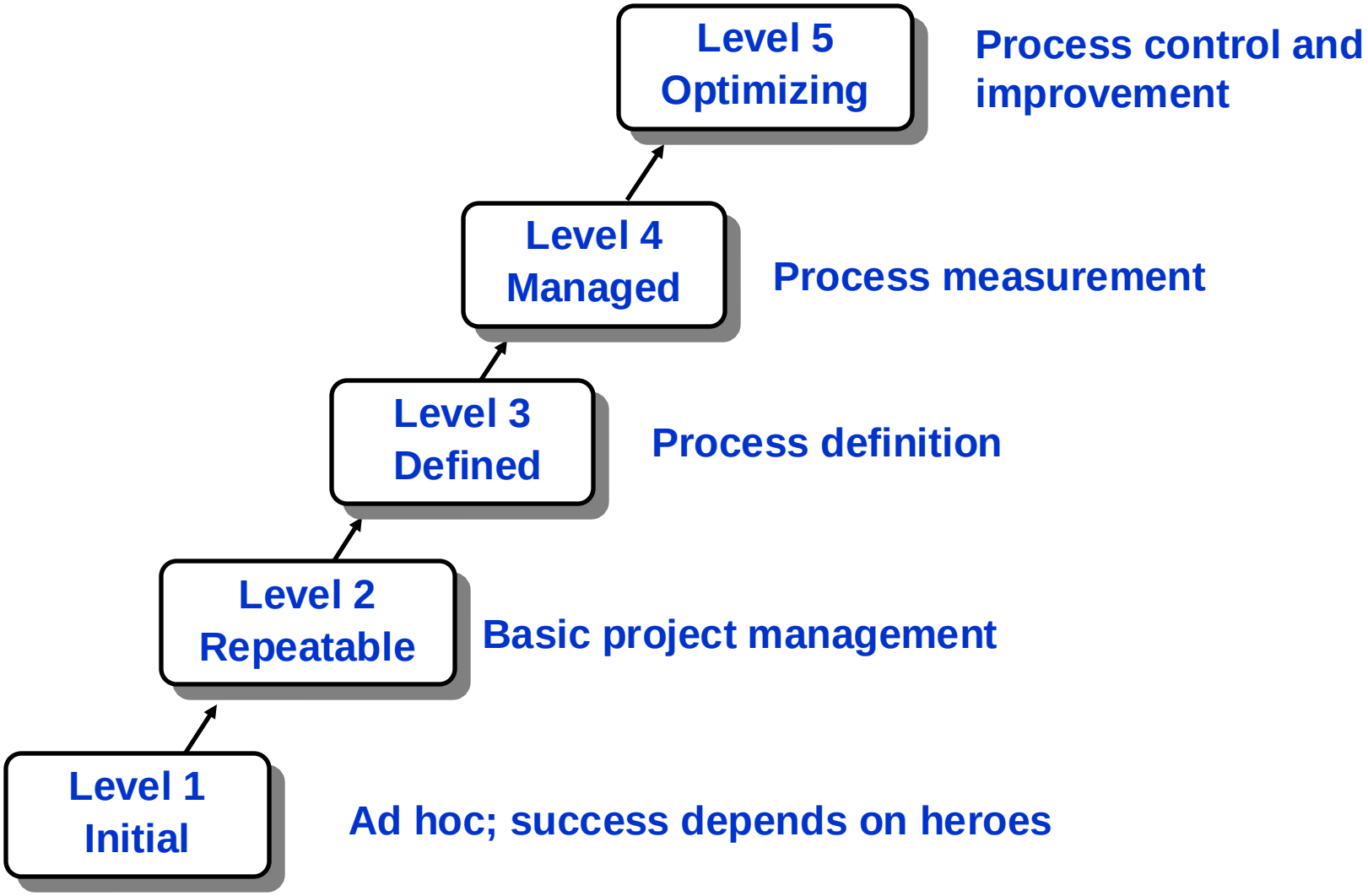
User Stories

- A common practice, used in agile development, often in combination with Scrum (but not defined in the Scrum Guide)
- A user story:
  - is a format for describing user requirements as a "story". Is should be short, typically just one sentence, and described from the user point of view
  - uses a common template, such as
    - *As a <role>, I want <goal> so that <benefit>*
      - Example: "*As a process engineer, I want to see the dependencies between different process steps so that I can easily verify and validate them*"
- Large user stories which are later broken down into smaller ones are often called *epics*

Capability Maturity Model (CMM)

- In 1993 the SEI (Software Engineering Institute) has developed a model to determine the maturity level of software development organizations (i.e., a global measure of effectiveness in the application of software engineering best practices)
- Capability assessment is questionnaire-based
- The model consists of five levels and each level includes all the characteristics of the lower level

The 5 CMM levels

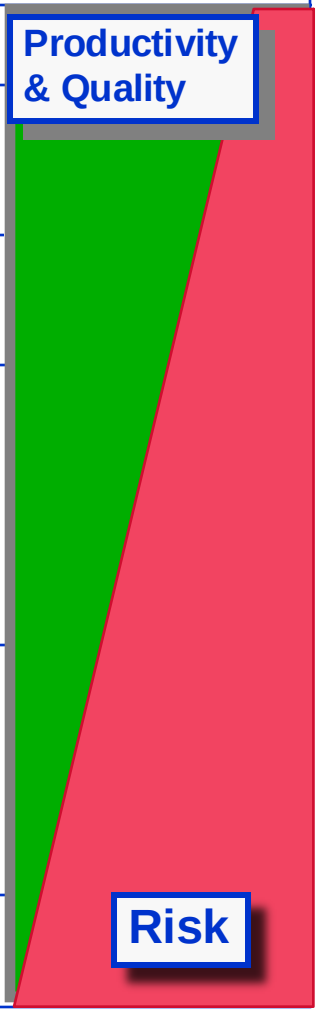


Key Process Areas

- A set of so-called KPAs (Key Process Areas) is associated to each CMM level
- A total 18 KPAs have been defined
- Each KPA describes a cluster of related practices that have to be implemented in order to satisfy a set of goals considered important for making significant improvement in that area
- A KPA description includes:
  - goals
  - responsibilities and duties
  - implementation capabilities and resources
  - activities to be carried out
  - implementation monitoring methods
  - implementation verification methods

CMM KPAs

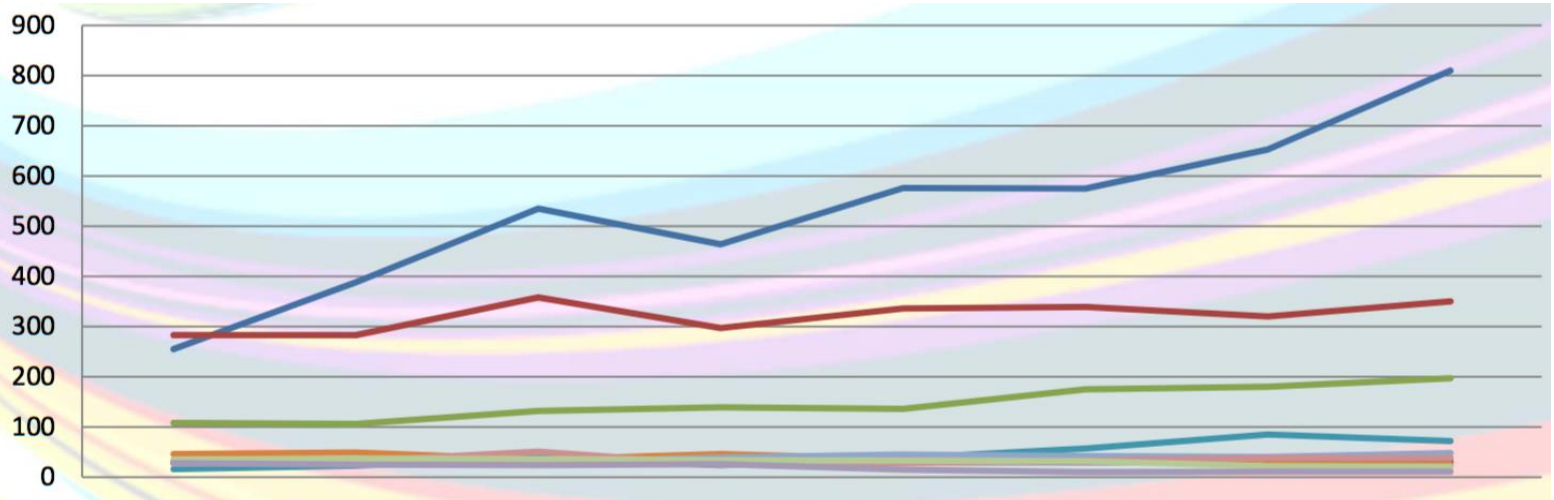
Level	Characteristic	Key Process Areas
Optimizing (5)	Continuous process capability improvement	Process change management Technology change management Defect prevention
Managed (4)	Product quality planning; tracking of measured software process	Software quality management Quantitative process management
Defined (3)	Software process defined and institutionalized to provide product quality control	Peer reviews Intergroup coordination Software product engineering Integrated software management Training program Organization process definition Organization process focus
Repeatable (2)	Management oversight and tracking project; stable planning and product baselines	Software configuration management Software quality assurance Software subcontract management Software project tracking & oversight Software project planning Requirements management
Initial (1)	Ad hoc (success depends on heroes)	"People"



Some statistics (February 2000)

- High maturity (either Level 4 or Level 5) organizations include:
  - 71 organizations in the USA
    - 44 at Level 4 (e.g., Oracle, NCR, Siemens Info Systems, IBM Global Services)
    - 27 at Level 5 (e.g., Motorola, Lockeed-Martin, Boeing, Honeywell)
  - 25 organizations outside USA
    - 1 organization at Level 4 in Australia
    - 14 organizations at Level 4 in India
    - 10 organizations at Level 5 in India

Number of appraisals by country (06/15)



	2008	2009	2010	2011	2012	2013	2014	2015
China	255	388	535	464	576	575	653	810
United States	283	283	358	297	336	339	320	350
India	108	106	132	139	136	175	180	197
Spain	30	30	30	30	30	30	30	30
Mexico	16	23	39	24	38	57	85	72
Japan	46	49	35	46	36	32	26	25
Korea, Republic Of	27	36	29	39	45	42	40	48
Brazil	28	32	51	25	28	33	38	38
France	35	37	36	34	33	32	22	21
United Kingdom	28	25	24	26	15	10	11	11

Trends (as of June 2015)