Software Engineering

The Software Development Process

Software Process Definition

- The sequence of steps required to manage, develop or maintain software (as standard software engineering practice)
- A set of (partially ordered) activities needed to transform a user's requirements into a software system (product)

Software Process

- The process involves
  - Translating user needs into software requirements
  - Transforming software requirements into design
  - Implementing the design in code
  - Testing the code
  - Sometimes: installing and checking out the software for operational use
  - Evolution

The Software Process

- Software process activities may overlap or be performed iteratively
- Activities vary depending on organization and type of system developed
- In order to be managed, must be explicitly modeled

Software Process Models

- A (software) process model is
  - An abstraction of similar software processes
  - A blueprint (a pattern) used to organize and perform a software process

The need For a Process

- Provide guidance to the order of a team’s activities
- Direct tasks of individual developers and coordinate the team as a whole
- Specify what artifacts should be developed
- Offer criteria for monitoring and measuring a project’s products and activities
- Goal: reduce risk

Process Characteristics

- Understandability
  - Clearly defined
- Visibility
  - Is the process progress externally visible?
- Supportability
  - Can CASE tools support the process?
- Acceptability
  - Is the process acceptable to the involved staff?

Process Characteristics

- Reliability
  - Are process errors discovered before they result in product errors?
- Robustness
  - Can process continue when unexpected problems occur?

Process Characteristics

- Maintainability
  - Can the process meet changing needs?
- Rapidity
  - How fast is the system produced?

Note

Processes are like habits: hard to establish, harder to break

Software Development Activities

- Analysis
  - What are the problems of the customer?
  - Where and how can software help to solve the problems?
  - What are the main functionalities of the software?

Software Development Activities

- Specification of requirements
  - What are the requirements on the software
  - Get an agreement between customer and contractor
  - High quality requirements are the foundation for software development
  - Result: requirements specification

Software Development Activities

- Top-level/detailed-level design
  - Defining the building blocks of the software (components, modules)
  - Designing the software architecture
  - Defining the interfaces between the building blocks
  - Defining the functionality of each building block
  - Result: Software Architecture

Software Development Activities

- "**I know you believe you understood** what you think I said, but I am not sure you realize that what you heard is not what I **meant**!"

  -- George Romney, 1967, U.S. Presidential Candidate
- Requirements
  - Analysis: What does the software do
  - Design: How does it do it
- Requirements are most associated with analysis
  - What does the customer want?
  - Even more important: what does the customer need?
    - Requirements elicitation techniques

Software Development Activities

- Coding and test
  - Each building block is implemented
  - Each building block is tested based on its specification (unit test)
  - Detected errors are corrected
  - Result: coded and tested building blocks

Software Development Activities

- Integration, test, acceptance
  - Assembly of the system based on the software architecture
  - Test of all defined interfaces
    - Integration test
  - Performing the acceptance test showing that the system works well in the environment of the customer
  - Result: delivered system

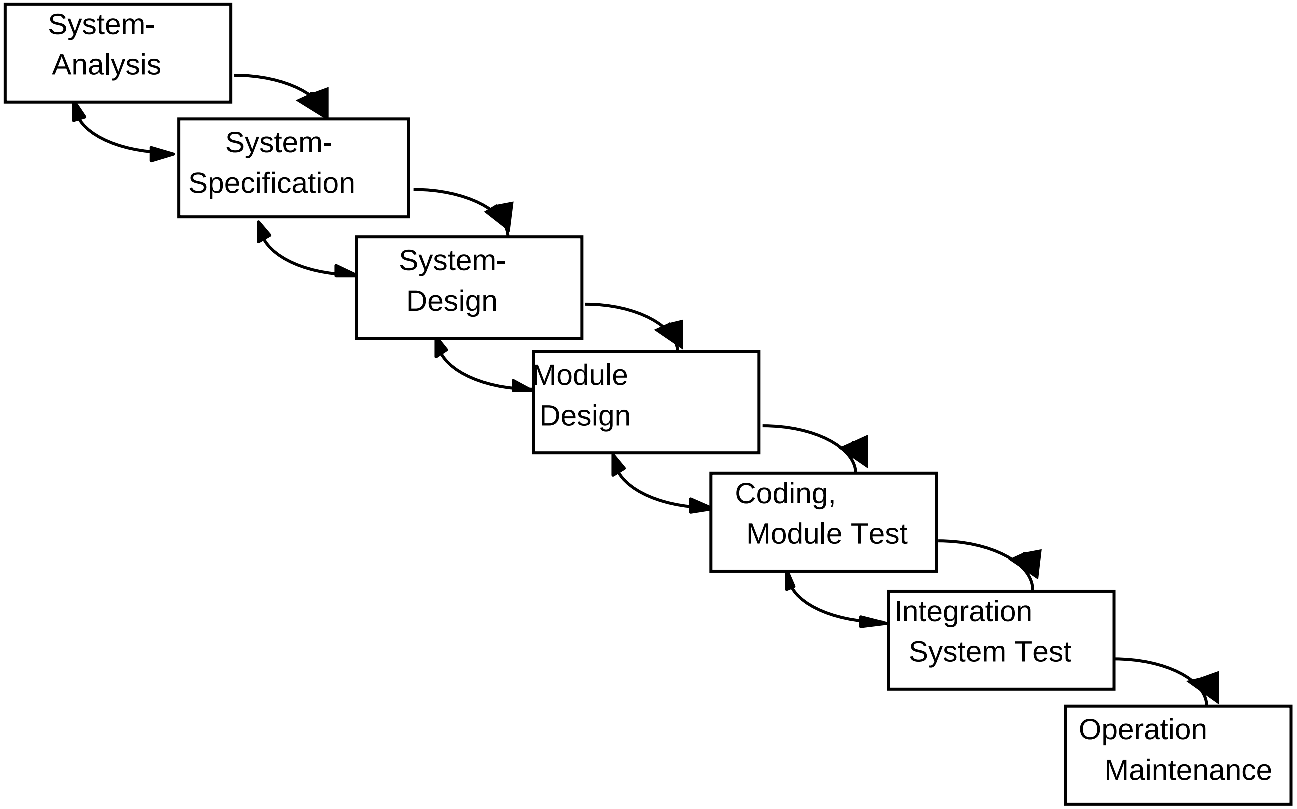
Software Development Activities

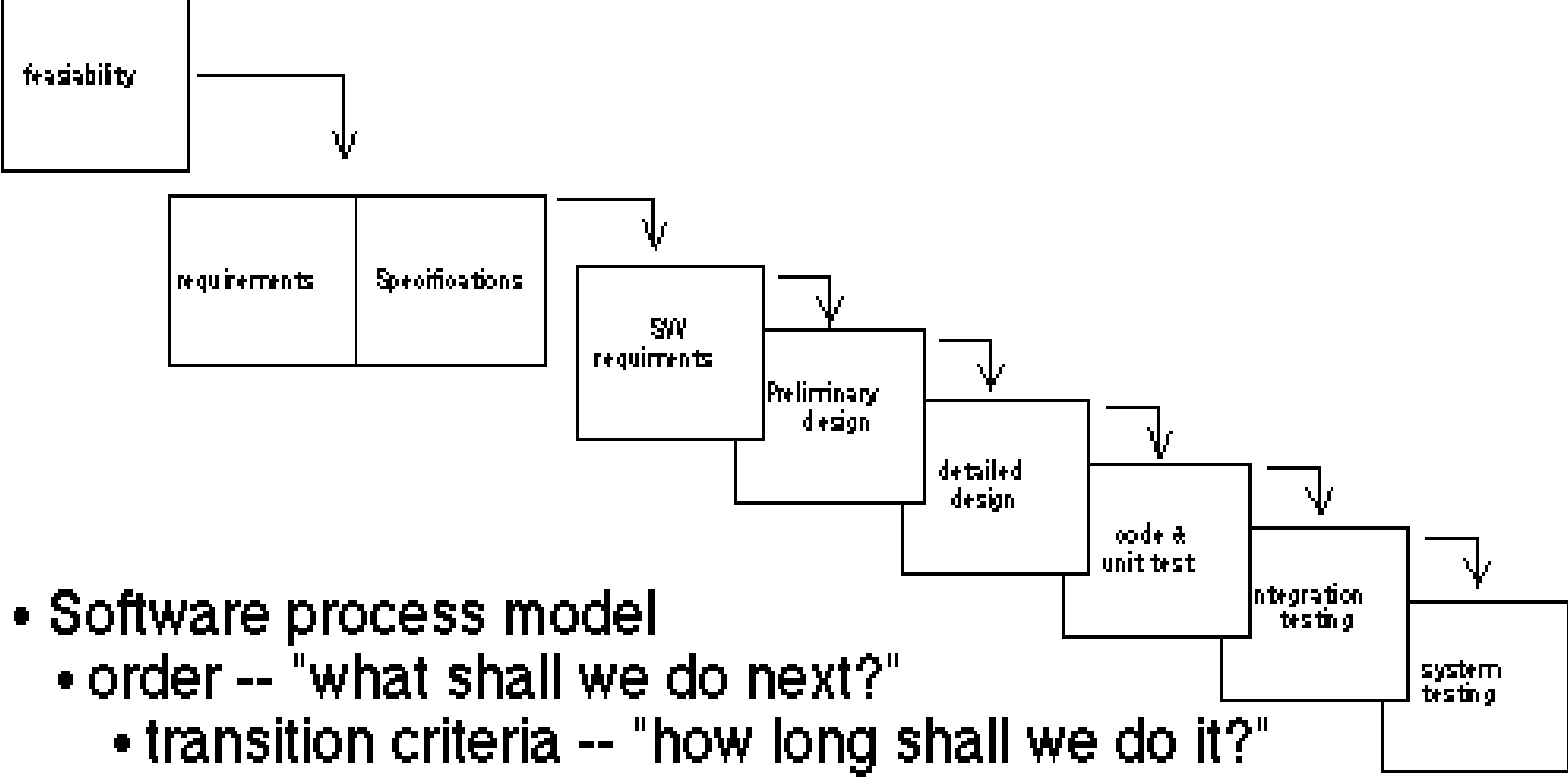
- Operation and Maintenance
  - Installation of the system
  - Putting the system in operation
  - Correction of all detected errors during operation
- Replacement
  - Due to software aging software has to be replaced eventually
  - It is important to start the development of a new system replacing the old one in good time

Software Development Approaches

- The waterfall model
- Spiral model
- Prototyping model
- Formal transformations (a mathematical system model is formally transformed to an implementation)
- Reuse-based development (assembly of existing components)

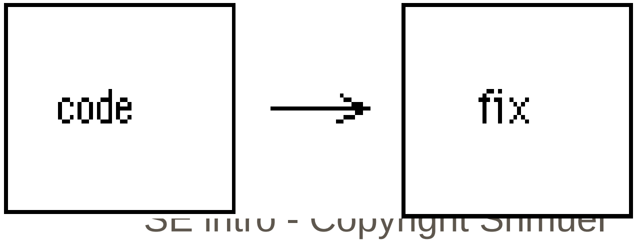
The Waterfall Model



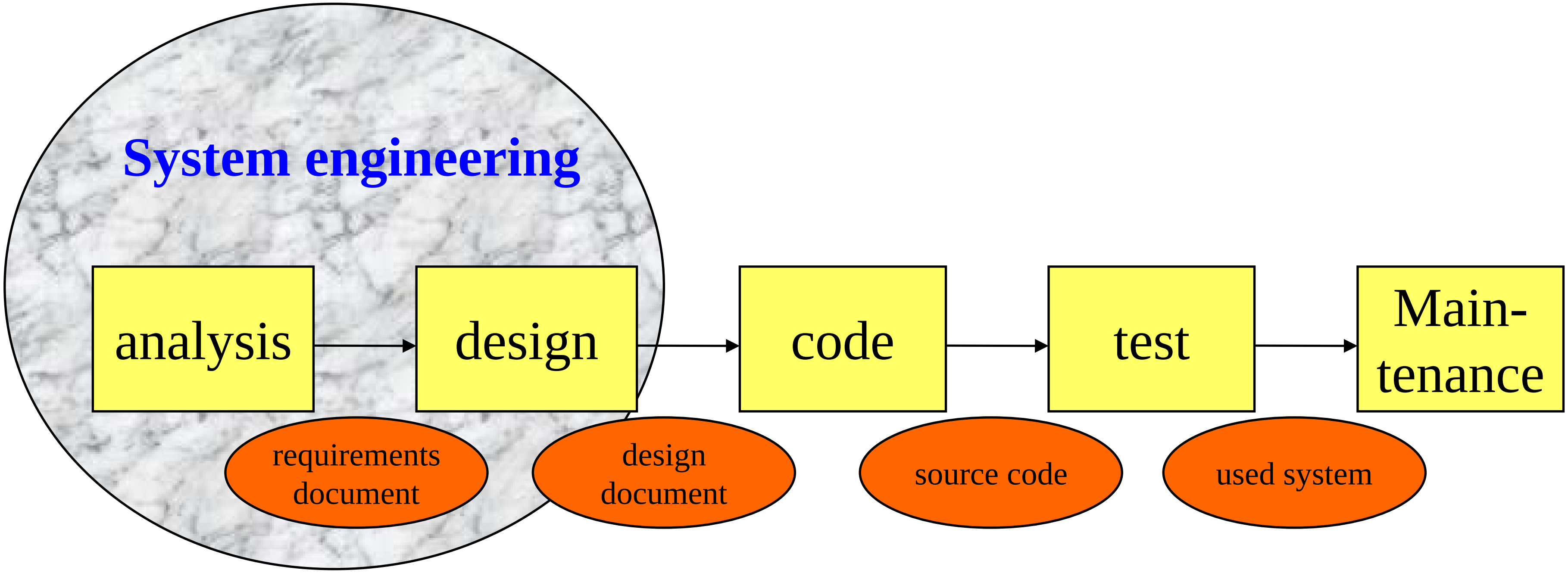


- Software process model
  - order -- "what shall we do next?"
    - transition criteria -- "how long shall we do it?"

Earliest Model



The Waterfall Model



The Waterfall Model

- System engineering
  - software as part of a larger system
- Analysis
  - software requirements
    - functional, performance, interface
- Design
  - architecture; procedural details; interfaces (possibility: rapid prototyping)

The Waterfall Model

- Coding
- Testing
  - verification and validation
- Maintenance
  - corrective (fixing errors)
  - adaptive (different environment)
  - enhancement (new functionality)

The Waterfall Model

- Basically sequential
  - reality enforces backtracking
- Specifications are incomplete, ambiguous, anomalous
- Customer receives a system only late into the project
  - recall: customer is not sure what (s)he needs ...
  - stating all requirements at the beginning of the project is difficult

The Waterfall Model

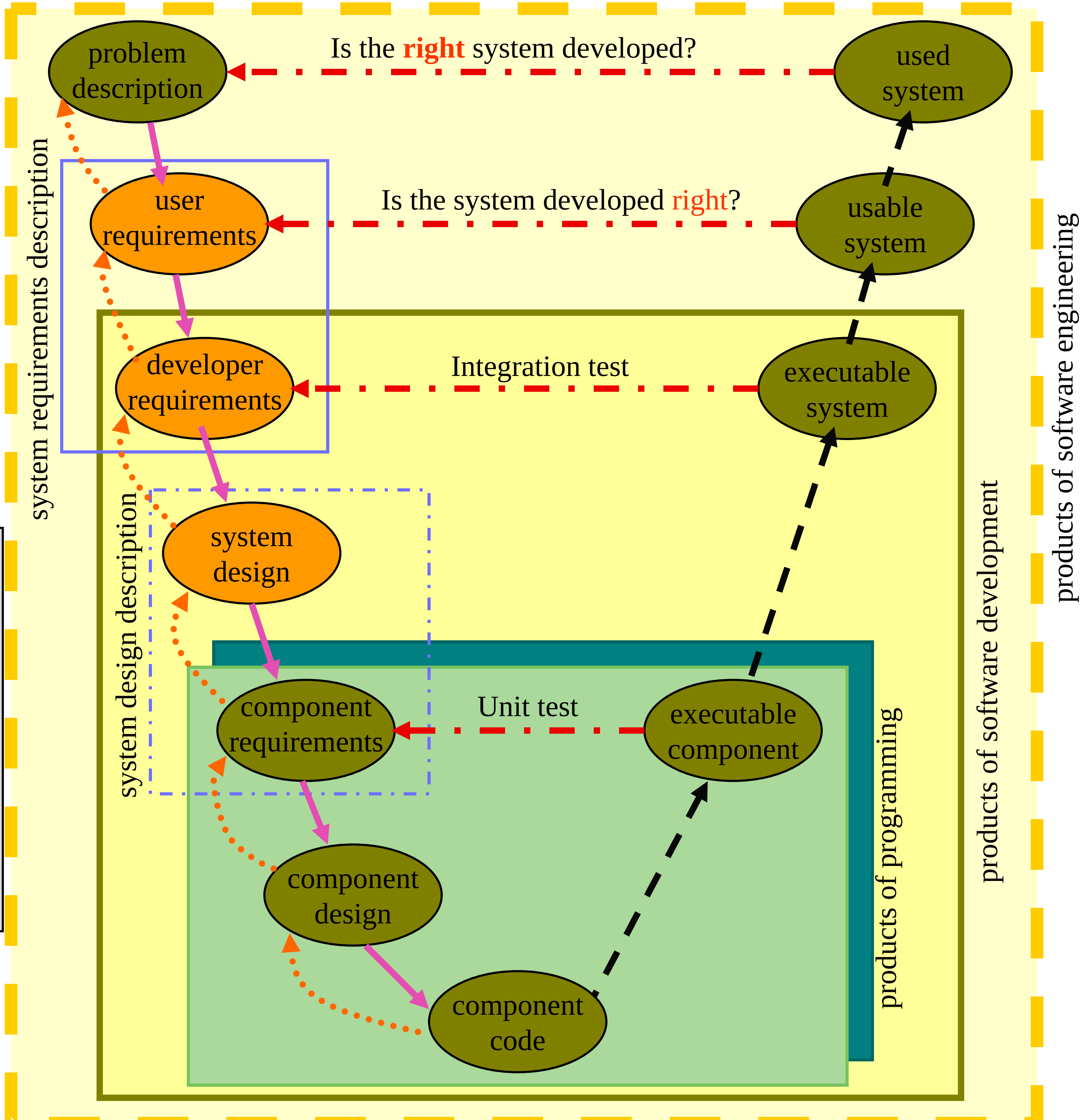
- Since software does not wear out, maintenance is not component replacement
- Each step results in documentation
- Suitable for well-understood developments
- Difficulty in accommodating changes after the process has started

return to software development approaches

The
V - model

legend

- ◀⋯⋯ is verified against
- ——▶ is developed in
- ◀– – – is integrated in
- –·–▶ is validated against



Is the right system developed?

problem description

used system

Is the system developed right?

user requirements

usable system

Integration test

developer requirements

executable system

system design

Unit test

component requirements

executable component

component design

component code

system requirements description

system design description

products of programming

products of software development

products of software engineering

The Prototyping Model

- A working model of (possibly part of) software system, emphasizing certain aspects
- Helps customers and developers to understand system requirements
- Maybe used for user training
- Exposes misunderstandings (users-developers)

The Prototyping Model

- Tool for requirements analysis and validation
  - building a mock-up of the system and obtain user feedback
  - check algorithms
- Important when aspects of system cannot be described due to lack of insight
- Detects missing services
- Identifies difficult-to-use parts

The Prototyping Model

- Basis for deriving (executable) specifications
- Can(not) serve as a contract
- Non-functional requirements cannot be adequately tested
- An iterative process
- Used for small interactive systems, part of large systems

The Prototyping Model

- A vehicle to experiment with different solutions proposed
- Evolutionary development (versions)
  - sacrifices: documentation, robustness

Prototyping Techniques

- Executable specification languages
  - dual semantics, formal/mathematical
  - good for functional requirements, not for GUI
  - executable, powerful data management facilities, inefficient
  - examples: Lisp, Prolog, SETL, ML, Miranda, Scheme

Prototyping Techniques

- 4GL
  - application generators
  - screen generators (GUI builders)

Evolutionary Prototype

- Delivers a working system to end users
- Starts with best understood requirements
- Aimed at implementing a system in stages (versions)

Evolutionary Prototype

- Exploratory programming, used for systems where specification cannot be fully developed in advance
  - e.g. AI, UI, research projects
- A vehicle for validating theories and/or for obtaining better conjectures

Throw-Away Prototype

- Understand, validate, or derive system requirements
- Starts with poorly understood requirements
- Aimed at gathering insights
- Output is executable specification
- Reduces requirements risk

Throw-Away Prototype

- It is not the final system
  - some aspects may have been left out
  - no maintenance specification
  - quick and dirty development

Prototyping Process

- Activities of specification, development, and validation are concurrent
- Prototype objectives
  - UI, validate requirements, etc.
- Requirements gathering
  - initial statement of requirements
  - revised in the iterative process

Prototyping Process

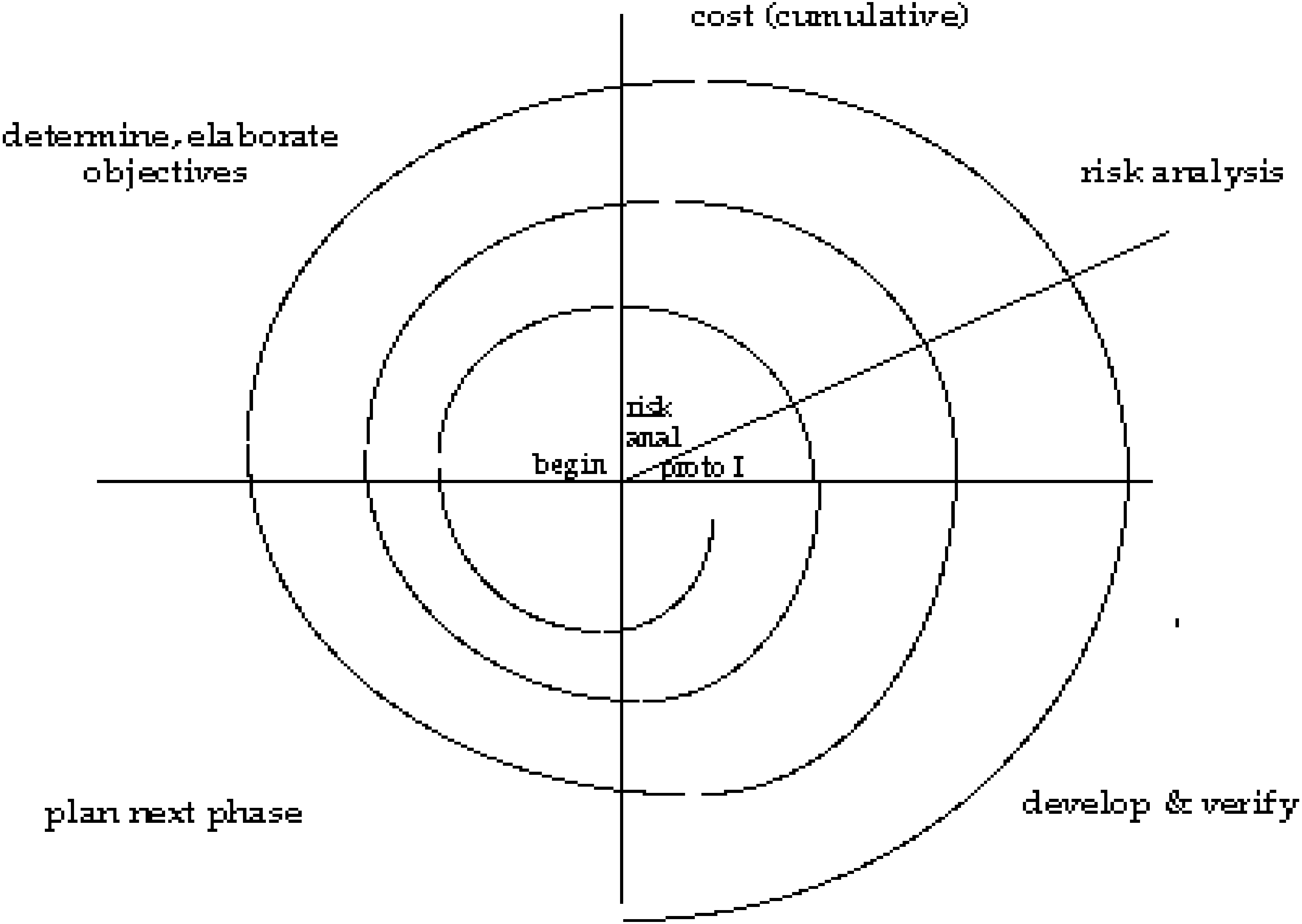
- Quick design
- Developing (quick and dirty) a prototype
  - reduced level functionality, subset of functionality
- Prototype evaluation
  - by customer and developer

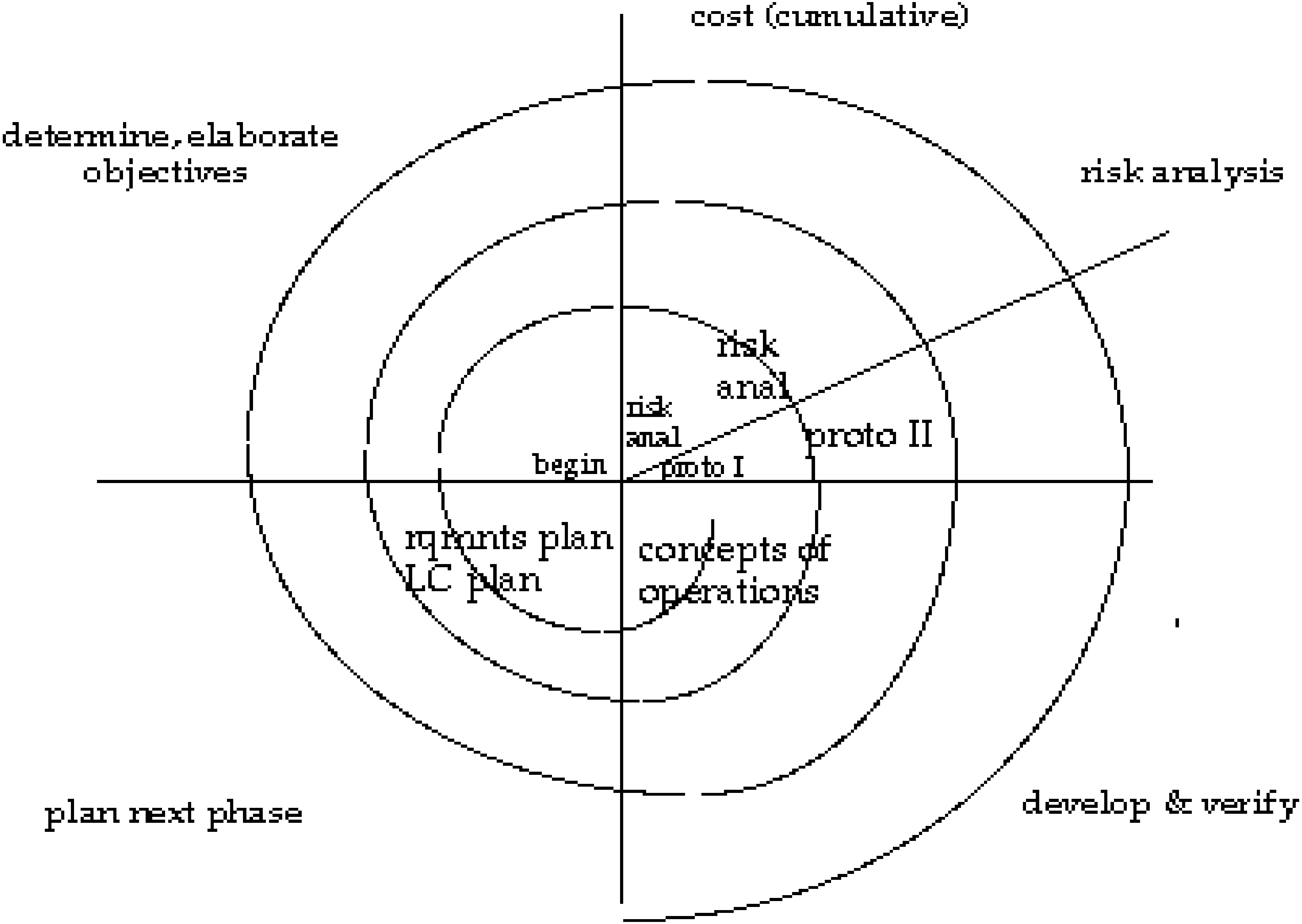
Problems with Prototyping

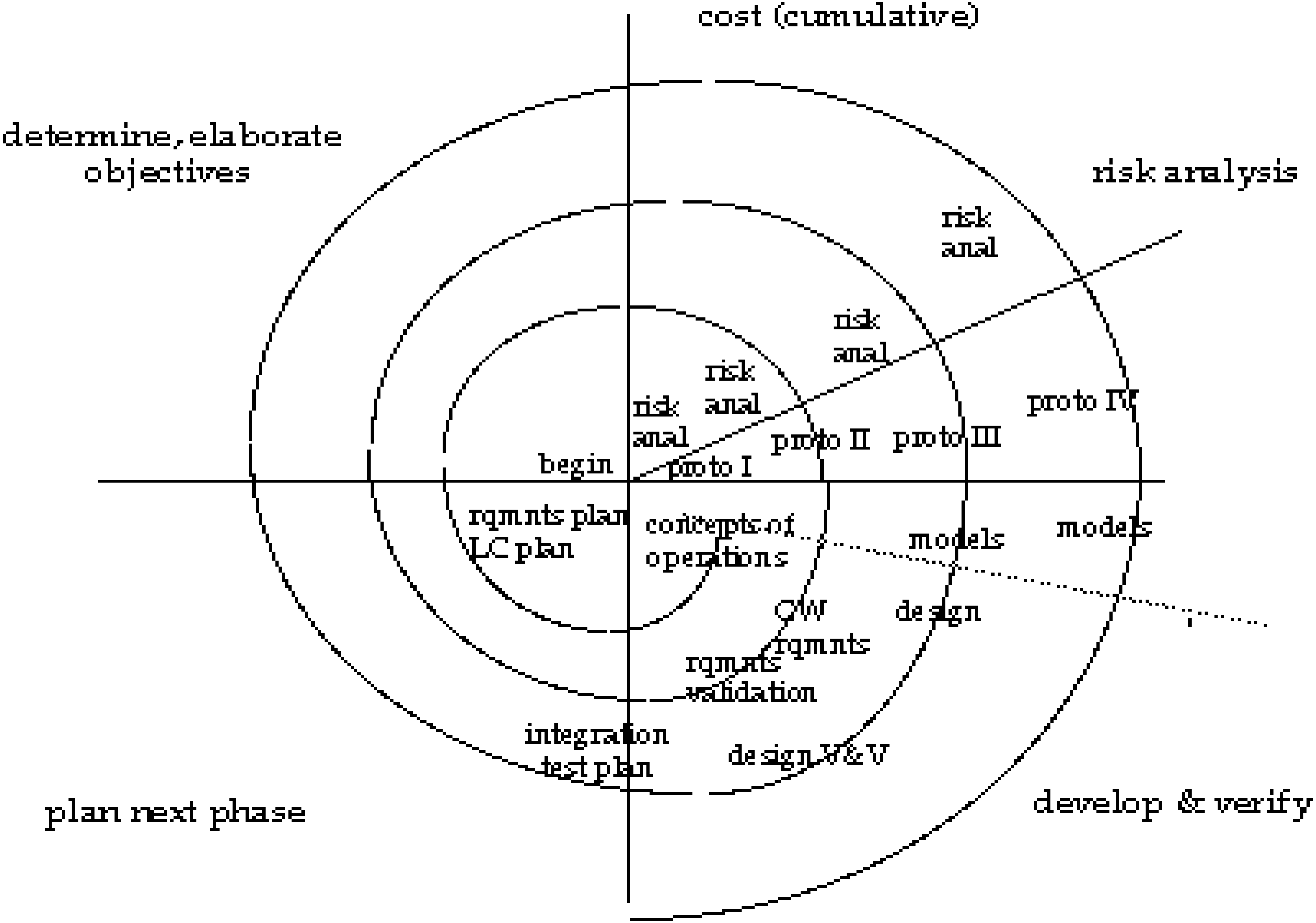
- Lack of process visibility
- Poorly structured systems
  - quick and dirty
- Additional skills needed
  - e.g. 4GL

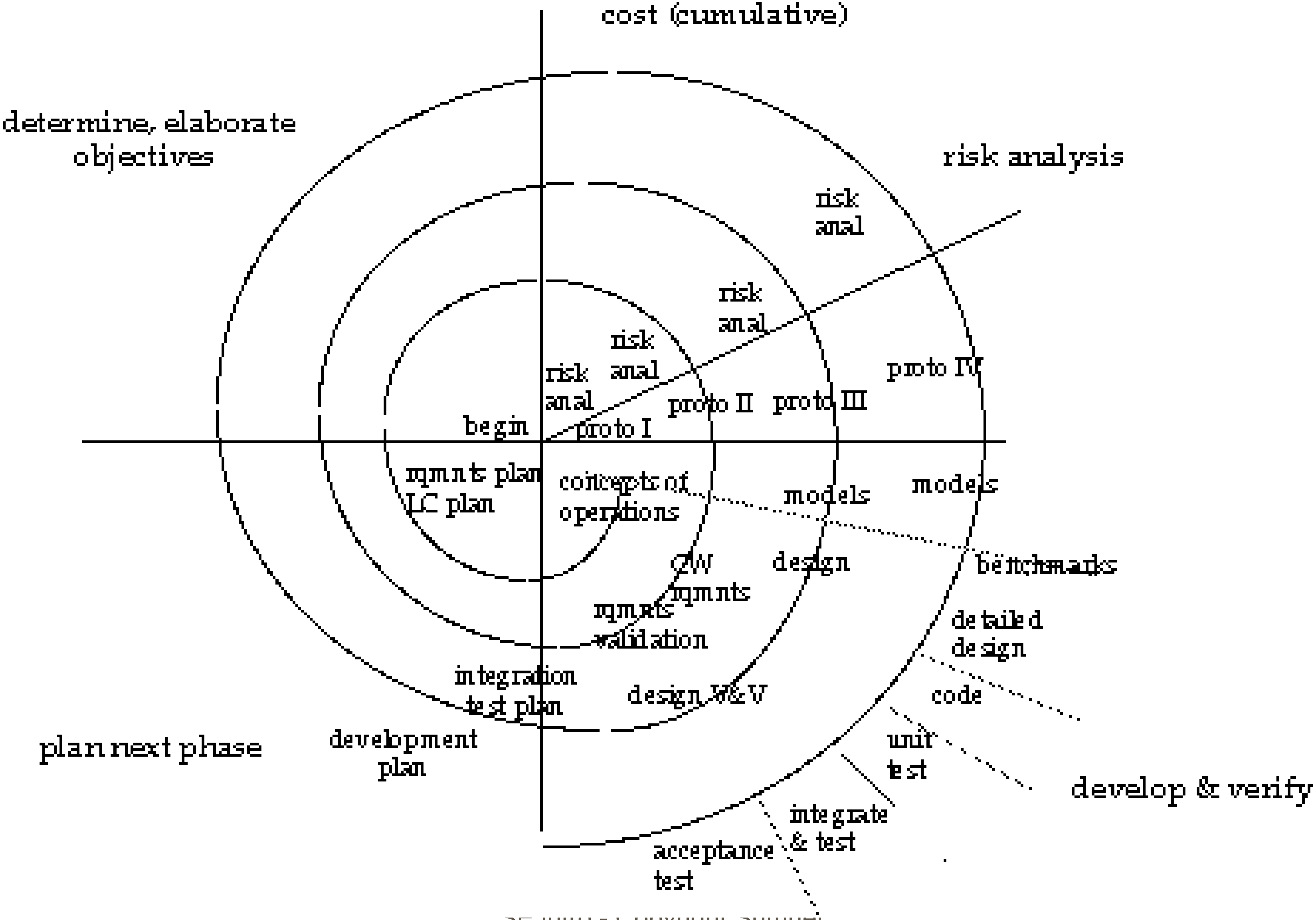
return to software development approaches

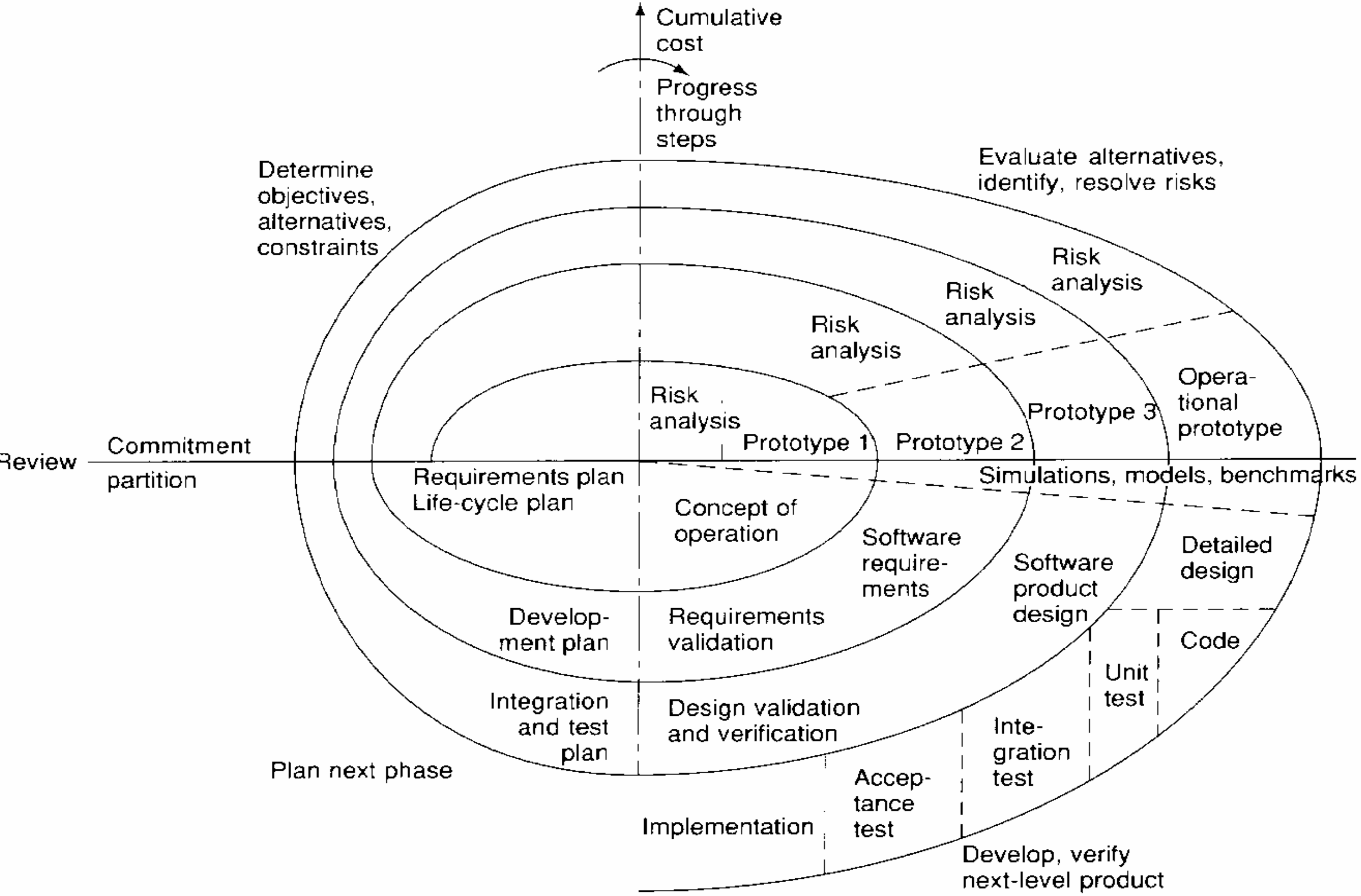
The Spiral Model


cost (cumulative)
determine, elaborate objectives
risk analysis
risk anal
begin
proto I
plan next phase
develop & verify


cost (cumulative)
determine, elaborate objectives
risk analysis
risk anal
risk anal
begin
proto I
proto II
rqmnts plan
LC plan
concepts of operations
plan next phase
develop & verify


cost (cumulative)
determine, elaborate objectives
risk analysis
risk anal
risk anal
risk anal
risk anal
proto IV
proto III
proto II
proto I
begin
rqmnts plan
LC plan
concepts of operations
models
models
design
SW rqmnts
rqmnts validation
design V&V
integration test plan
plan next phase
develop & verify


cost (cumulative)
determine, elaborate objectives
risk analysis
risk anal
risk anal
risk anal
risk anal
proto IV
proto II
proto III
begin
proto I
rqmnts plan
LC plan
concepts of operations
models
models
benchmarks
design
CW rqmnts
rqmnts validation
detailed design
integration test plan
design V&V
code
plan next phase
development plan
unit test
develop & verify
integrate & test
acceptance test


Cumulative cost
Progress through steps
Determine objectives, alternatives, constraints
Evaluate alternatives, identify, resolve risks
Risk analysis
Risk analysis
Risk analysis
Risk analysis
Operational prototype
Prototype 3
Prototype 2
Prototype 1
Review
Commitment
partition
Requirements plan
Life-cycle plan
Concept of operation
Simulations, models, benchmarks
Software requirements
Software product design
Detailed design
Development plan
Requirements validation
Code
Unit test
Integration and test plan
Design validation and verification
Integration test
Plan next phase
Acceptance test
Implementation
Develop, verify next-level product

The Spiral Model Iterations First Loop (spiral center)

- Project plan
  - initial requirements gathering
  - objective setting
- Risk assessment/analysis
  - go/no go decision
  - stop if no go

The Spiral Model Iterations First Loop (spiral center)

- Development
  - appropriate model, e.g. initial prototype
- Evaluation
  - validation

The Spiral Model Iterations N-th (N > 1)

- Requirements refinement
  - prototype feedback
  - more project planning
- Risk analysis
  - go/no go
- Prototype (N-th version)

The Spiral Model Iterations N-th (N > 1)

- Evaluation
  - at some point: customer evaluation
- System building

The Spiral Model

- Focuses on
  - reuse options
  - early error elimination
  - quality
- Integrates development and maintenance
- Provides a framework for HW/SW development

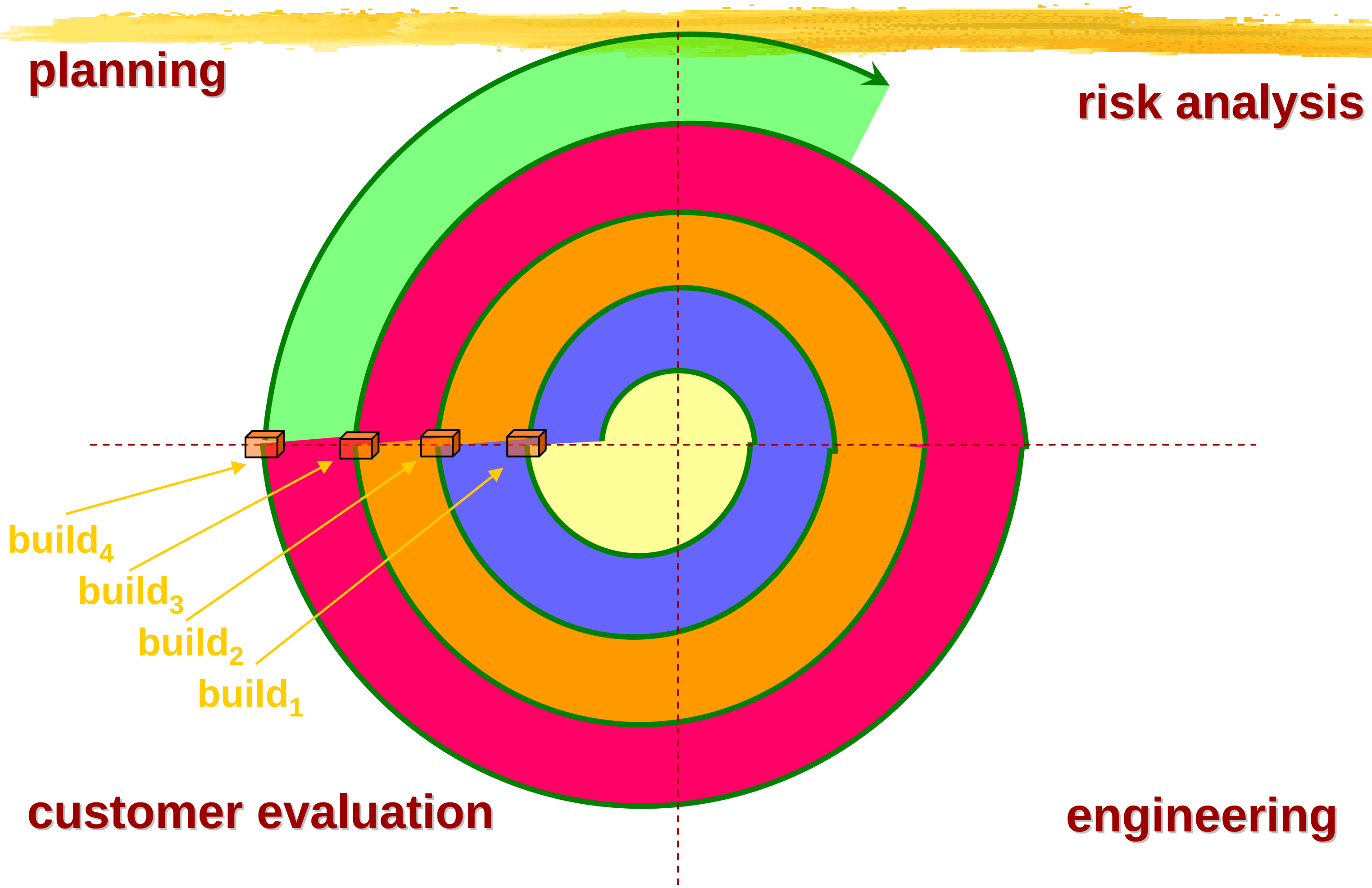
The Spiral Model

- Requires risk assessment expertise
- Needs refinement for general use
- After each increment:
  - functionality review (QA team)
  - updated schedule
  - updated resource requirements
  - new risk analysis
- Compare to evolutionary prototype

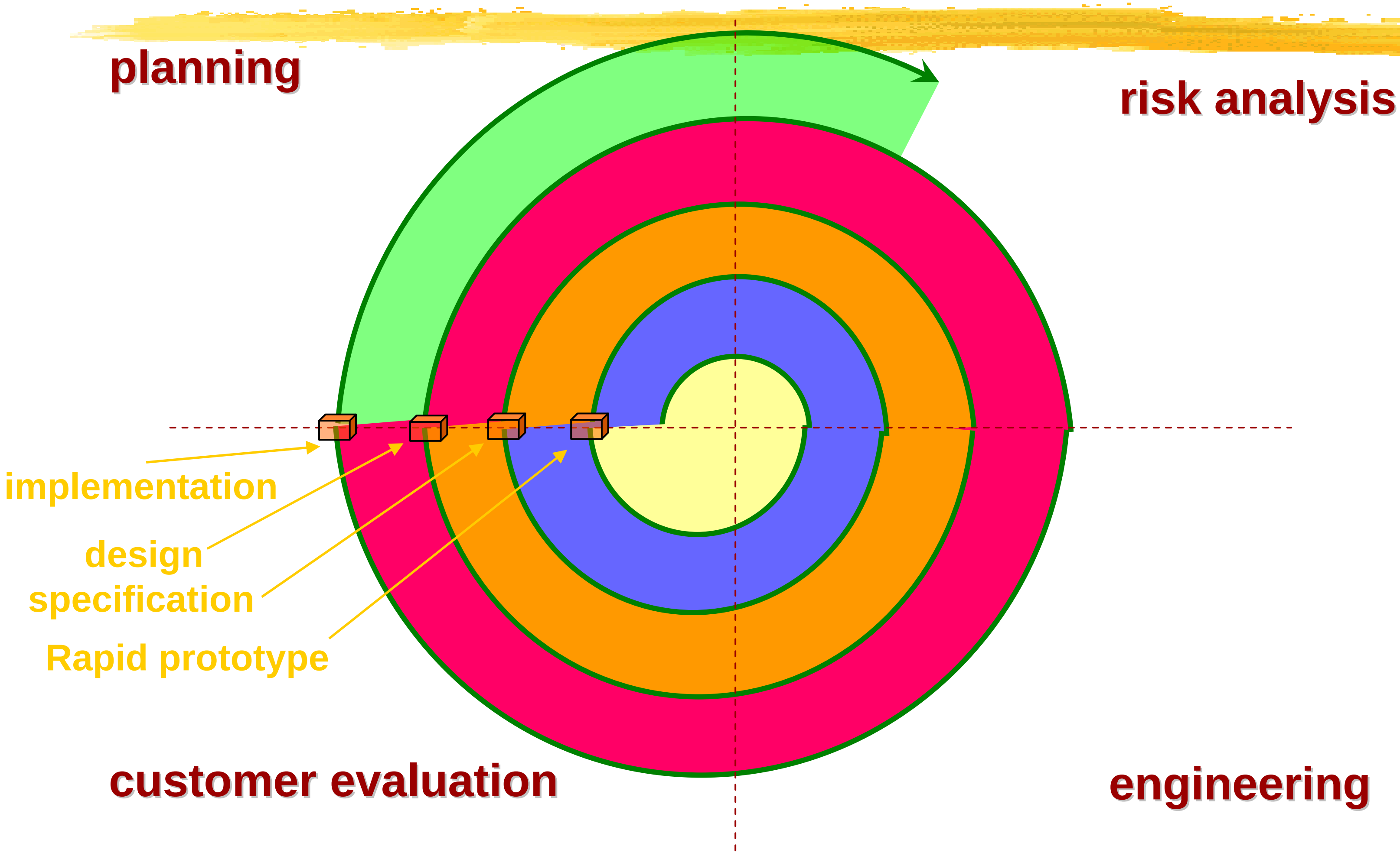
The Spiral Model

- Can use prototype during any phase
- Is a realistic approach to tackle problems with large scale systems development

Spiral Model for Incremental Development



Spiral Model for Prototyping



Which Model to Use?

- Rules of thumb:
  - waterfall model for well understood systems (low technical risks)
  - prototype model for high UI risk, incomplete specification
  - formal transformations for safety critical systems

Common Phases for All Models

- Definition/concept
  - system analysis
  - software planning
  - requirements analysis

  Note
  - delay HW/SW decisions as much as possible

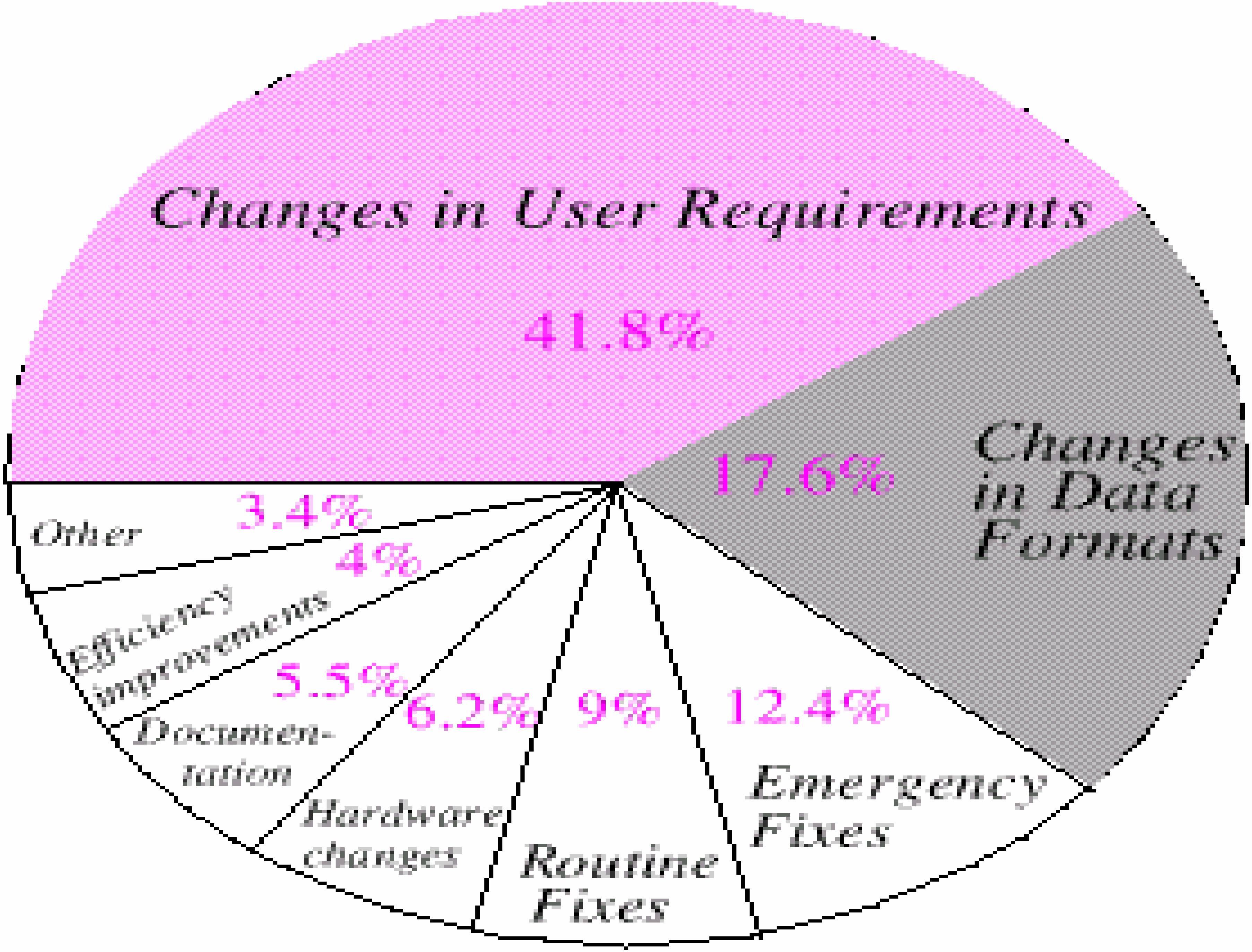
return to software development approaches

Common Phases for All Models

- Development
  - design
  - code
  - test

Common Phases for All Models

- Maintenance
  - correction (bugs): 21%
  - adaptive (new environments-HW/OS/DB): 25%
  - enhancement/perfection (user demands or more facilities/functionality): 50%
  - prevention (updating documents, adding comments, improving modularity): 4%


Changes in User Requirements
41.8%
Changes in Data Formats
17.6%
Emergency Fixes
12.4%
Routine Fixes
9%
Hardware changes
6.2%
Documentation
5.5%
Efficiency improvements
4%
Other
3.4%

Causes of Maintenance Problem

- Poor (unstructured) code
- Inflexible designs, coding and documentation
  - e.g. modules are tightly coupled
  - code is write-only (cannot be read)
- Programmers with no knowledge of the system/application domain
  - maintenance is usually done by inexperienced staff unfamiliar with the application
- No documentation, or worse, out-of-date
  - most software is between 10 and 15 years old

Causes of Maintenance Problem

- Changes often cause new faults in the system
- Changes tend to degrade the structure of a program

Reduce Maintenance Problem

- Higher quality code
- Better test procedures
- Adherence to standards/conventions
- Prototypes
- User participation
- Reuse of code

Types of Maintenance

- Corrective Maintenance
  - bug fixing
- Adaptive Maintenance
  - evolution of the system to meet new needs, caused by
    - internal needs
    - external competition
    - external requirements e.g. changes in law
  - introducing new requirements to the system
  - responses to changes in the environment
    - a new compiler, operating system, and/or hardware

Types of Maintenance

- Perfective Maintenance
  - improving the quality of a program that already works
  - polish or refine the quality of the software or the documentation
  - changes to improve product effectiveness
    - additional functionality
    - make product run faster
    - improve maintainability

Types of Maintenance

- Preventative Maintenance
  - converting a legacy system structure
  - converting a legacy system to a new language
    - old system starts as a specification for the new system
  - common method: wrappers where an entire system is placed in an OO wrapper and treated as one large object