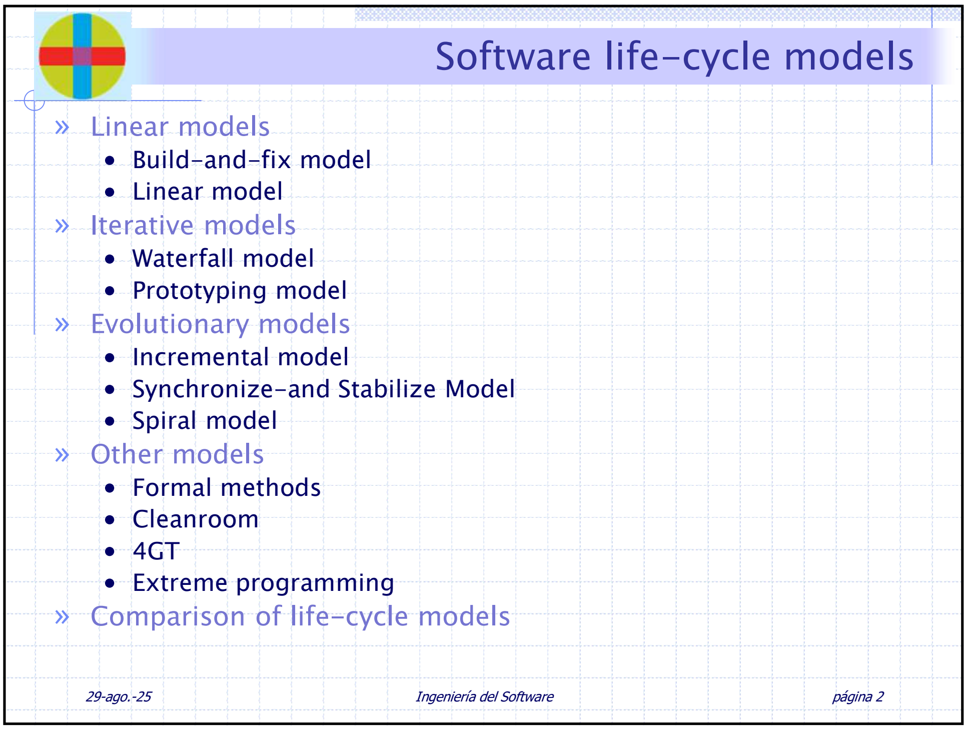
Metodologías clásicas de construcción de software

Ingeniería del Software

Curso 2025/2026
Universidad San Pablo-CEU
Escuela Politécnica Superior
Campus de Montepríncipe

1



Software life-cycle models

» Linear models
- Build-and-fix model
- Linear model

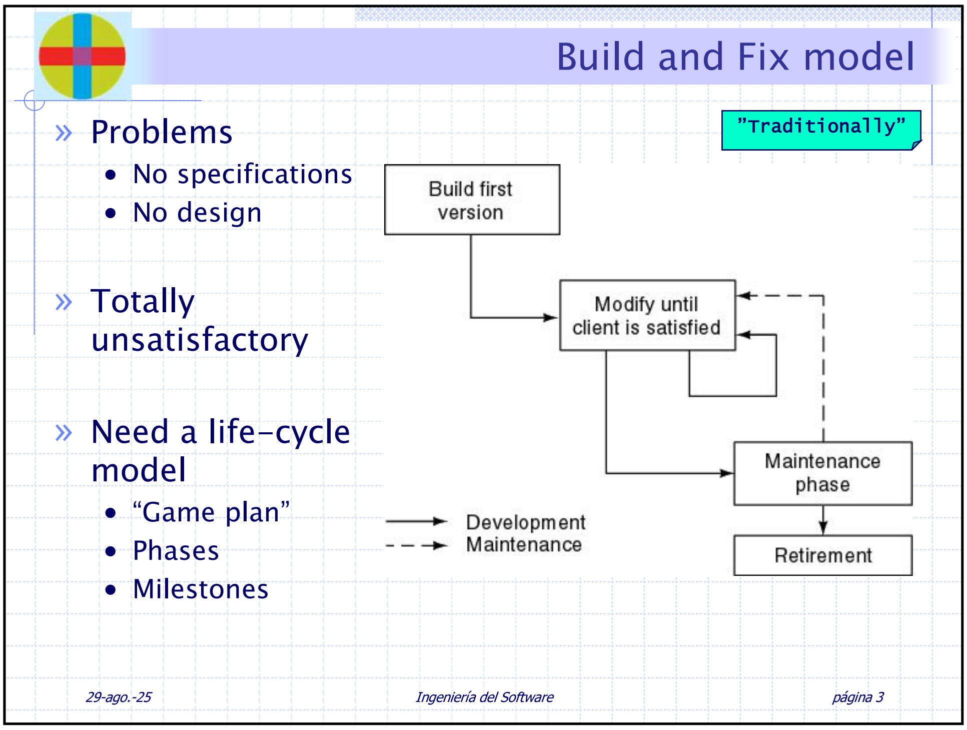
» Iterative models
- Waterfall model
- Prototyping model

» Evolutionary models
- Incremental model
- Synchronize-and Stabilize Model
- Spiral model

» Other models
- Formal methods
- Cleanroom
- 4GT
- Extreme programming

» Comparison of life-cycle models

2

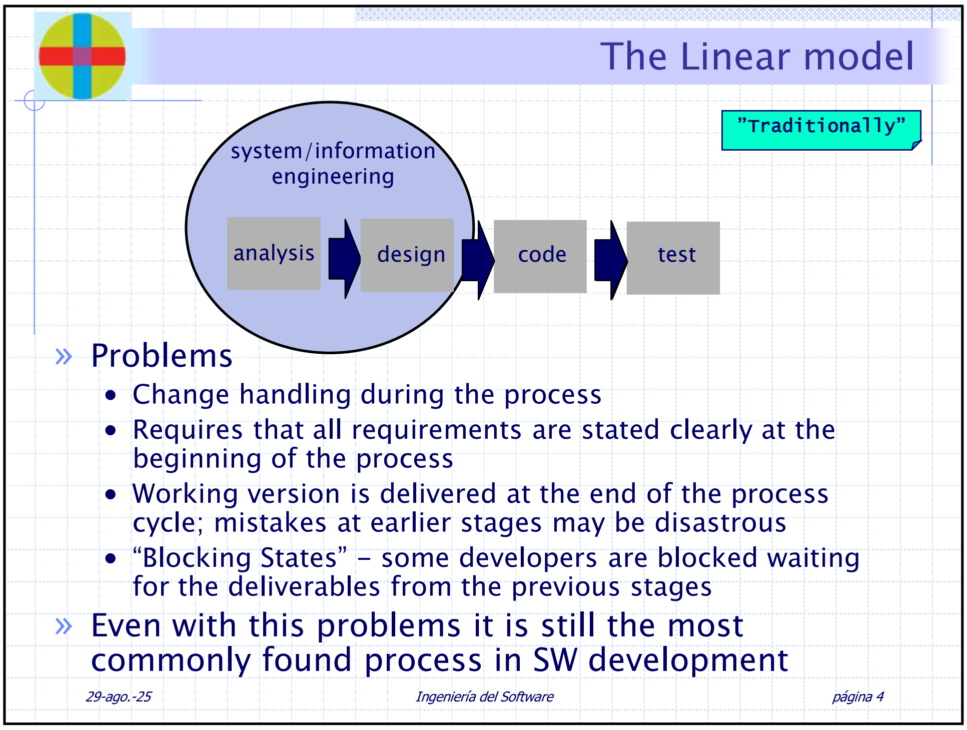


Build and Fix model

"Traditionally"

- Problems
  - No specifications
  - No design
- Totally unsatisfactory
- Need a life-cycle model
  - "Game plan"
  - Phases
  - Milestones

3

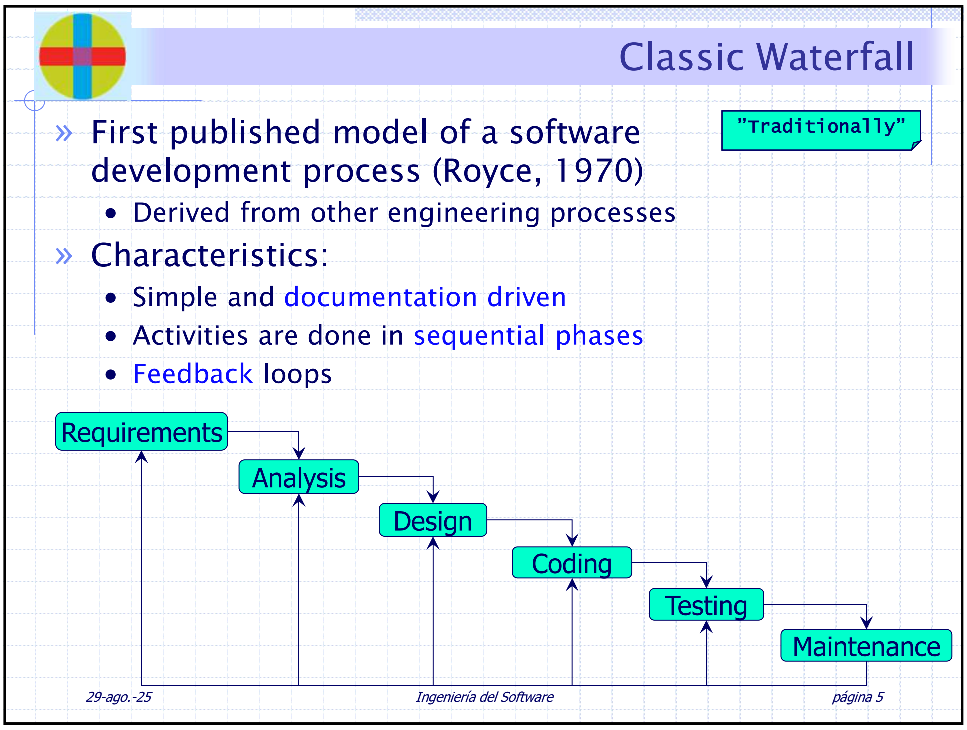


The Linear model

"Traditionally"

- Problems
  - Change handling during the process
  - Requires that all requirements are stated clearly at the beginning of the process
  - Working version is delivered at the end of the process cycle; mistakes at earlier stages may be disastrous
  - "Blocking States" – some developers are blocked waiting for the deliverables from the previous stages
- Even with this problems it is still the most commonly found process in SW development

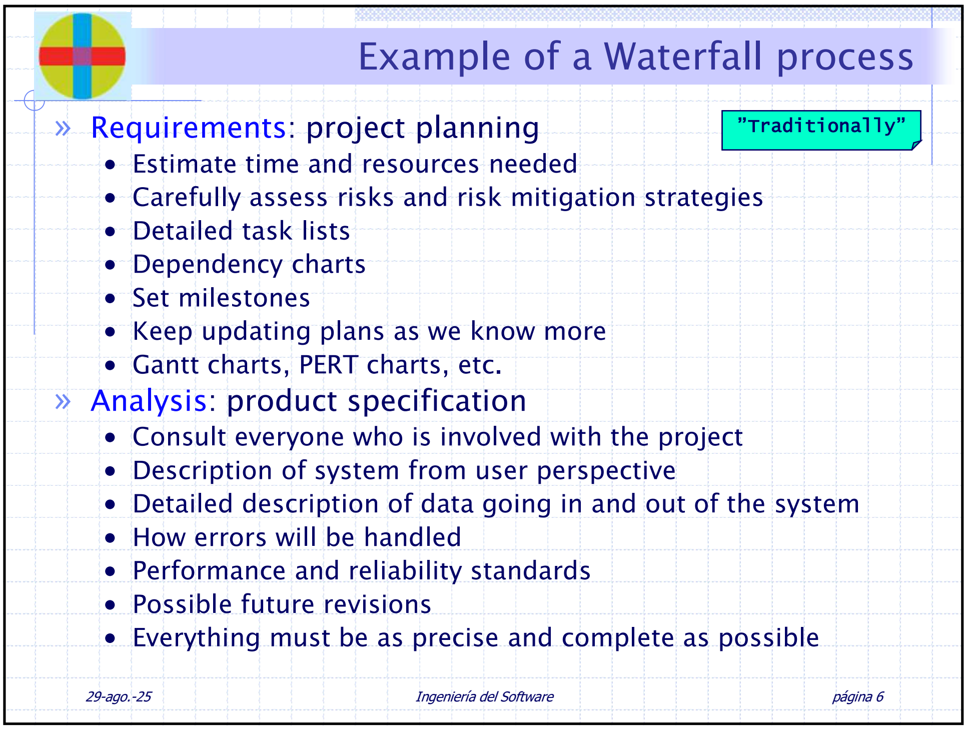
4



Classic Waterfall

"Traditionally"

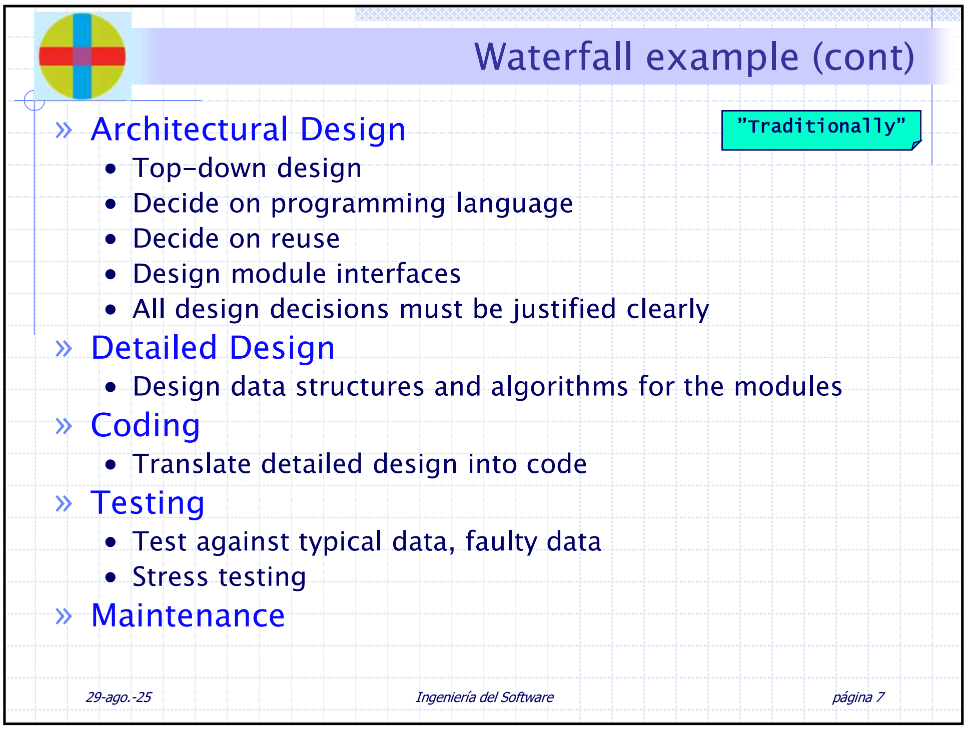
- » First published model of a software development process (Royce, 1970)
  - Derived from other engineering processes
- » Characteristics:
  - Simple and documentation driven
  - Activities are done in sequential phases
  - Feedback loops



Example of a Waterfall process

"Traditionally"

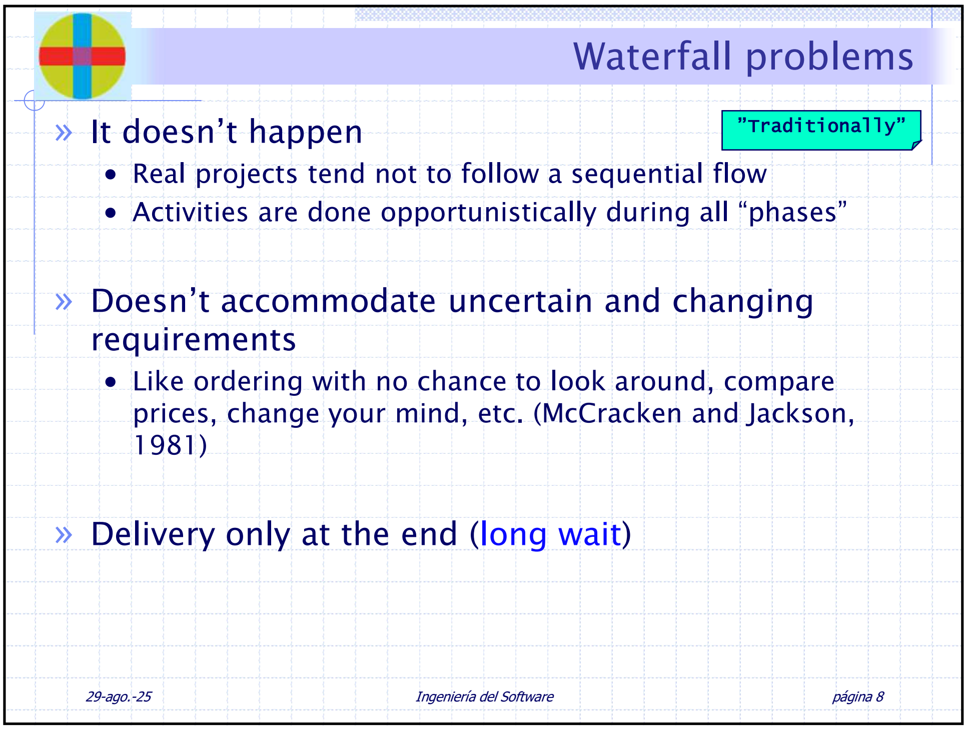
- » Requirements: project planning
  - Estimate time and resources needed
  - Carefully assess risks and risk mitigation strategies
  - Detailed task lists
  - Dependency charts
  - Set milestones
  - Keep updating plans as we know more
  - Gantt charts, PERT charts, etc.
- » Analysis: product specification
  - Consult everyone who is involved with the project
  - Description of system from user perspective
  - Detailed description of data going in and out of the system
  - How errors will be handled
  - Performance and reliability standards
  - Possible future revisions
  - Everything must be as precise and complete as possible



Waterfall example (cont)

"Traditionally"

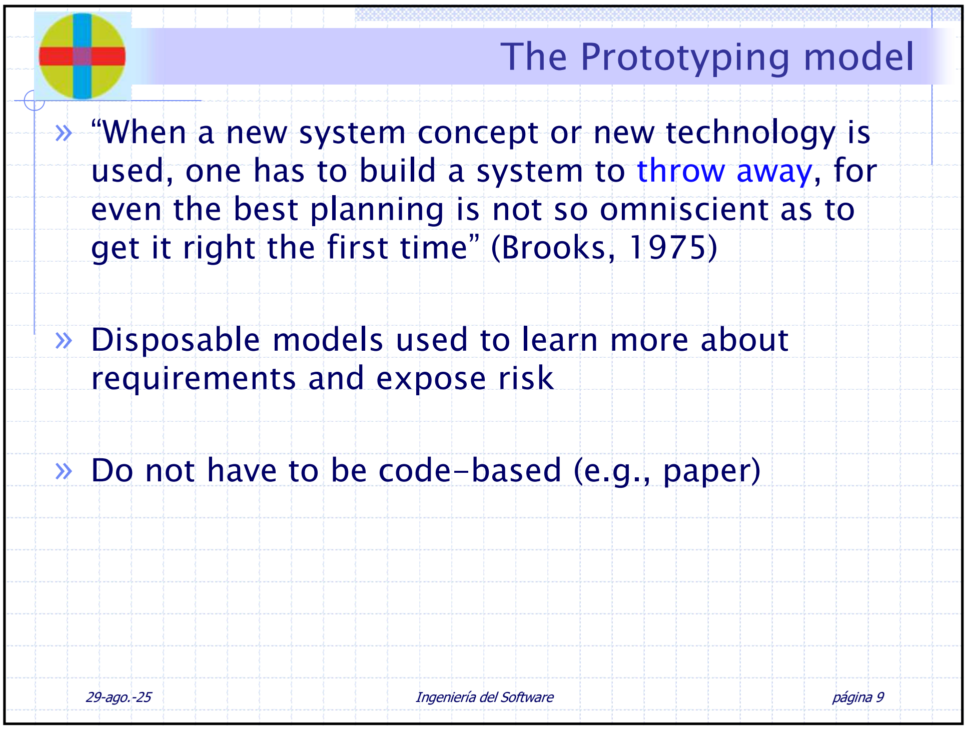
- Architectural Design
  - Top-down design
  - Decide on programming language
  - Decide on reuse
  - Design module interfaces
  - All design decisions must be justified clearly
- Detailed Design
  - Design data structures and algorithms for the modules
- Coding
  - Translate detailed design into code
- Testing
  - Test against typical data, faulty data
  - Stress testing
- Maintenance



Waterfall problems

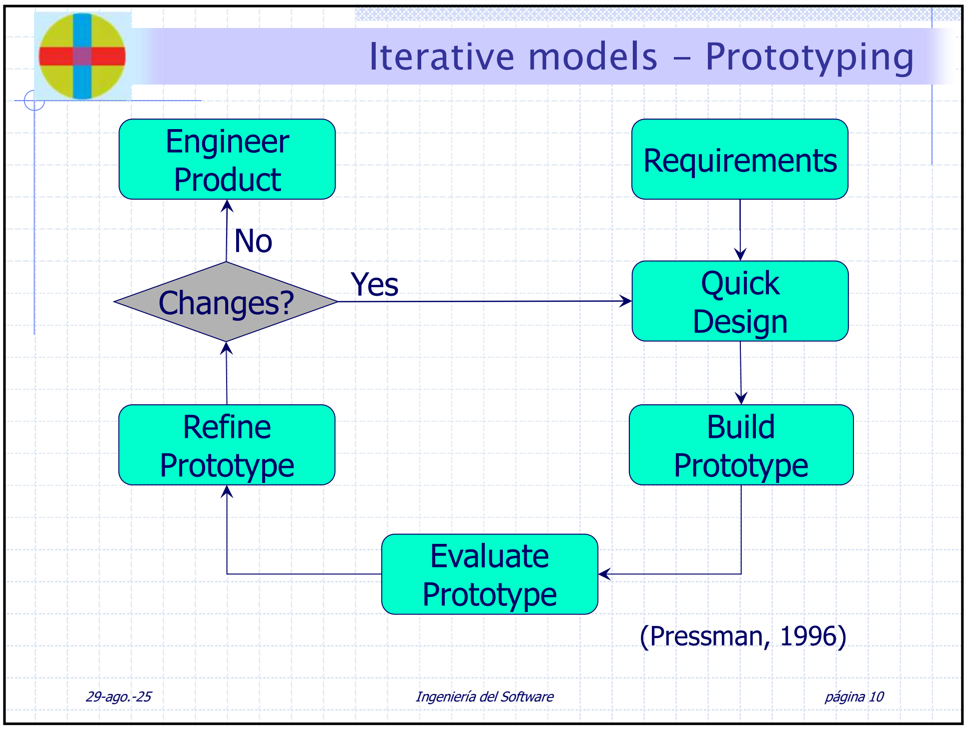
"Traditionally"

- It doesn't happen
  - Real projects tend not to follow a sequential flow
  - Activities are done opportunistically during all "phases"
- Doesn't accommodate uncertain and changing requirements
  - Like ordering with no chance to look around, compare prices, change your mind, etc. (McCracken and Jackson, 1981)
- Delivery only at the end (long wait)



The Prototyping model

- “When a new system concept or new technology is used, one has to build a system to throw away, for even the best planning is not so omniscient as to get it right the first time” (Brooks, 1975)
- Disposable models used to learn more about requirements and expose risk
- Do not have to be code-based (e.g., paper)



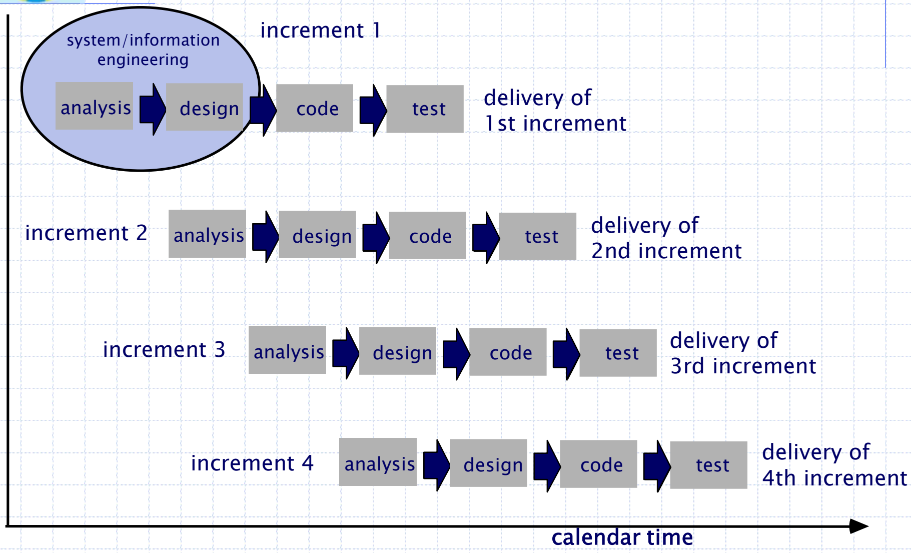
Iterative models – Prototyping

(Pressman, 1996)

Incremental model

- Original form: “Evolutionary delivery” (Gilb, 1988)
- Systems is delivered in pieces, highest priority first
- Early increments inform requirements for later increments
- Increment size varies (originally a maximum of a few weeks)

Evolutionary models – Incremental model



system/information engineering

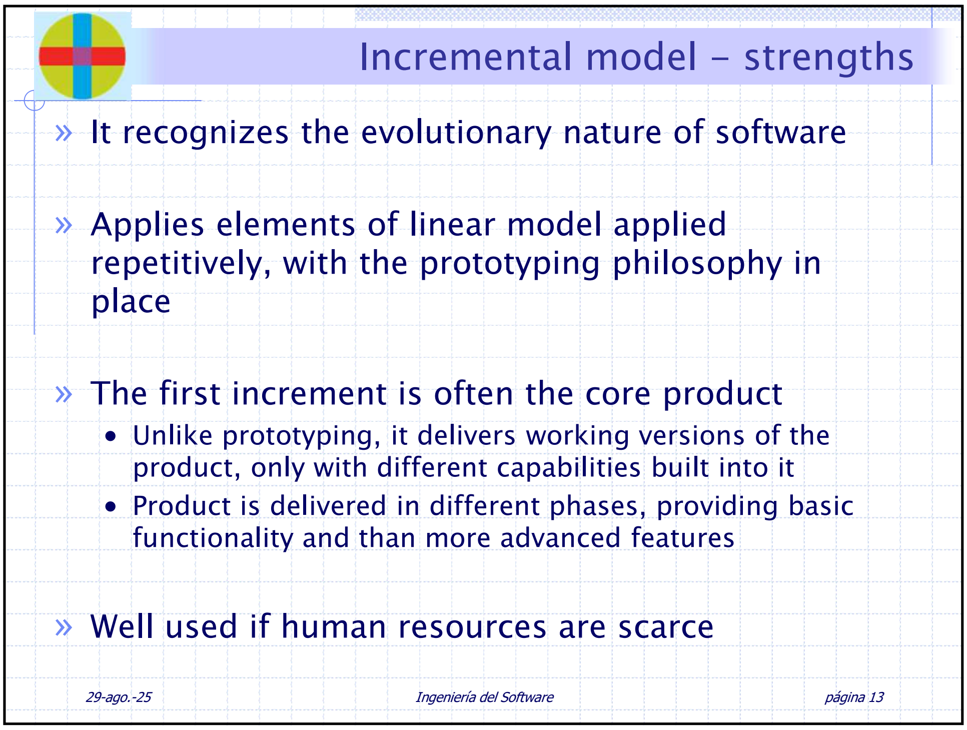
increment 1: analysis → design → code → test → delivery of 1st increment

increment 2: analysis → design → code → test → delivery of 2nd increment

increment 3: analysis → design → code → test → delivery of 3rd increment

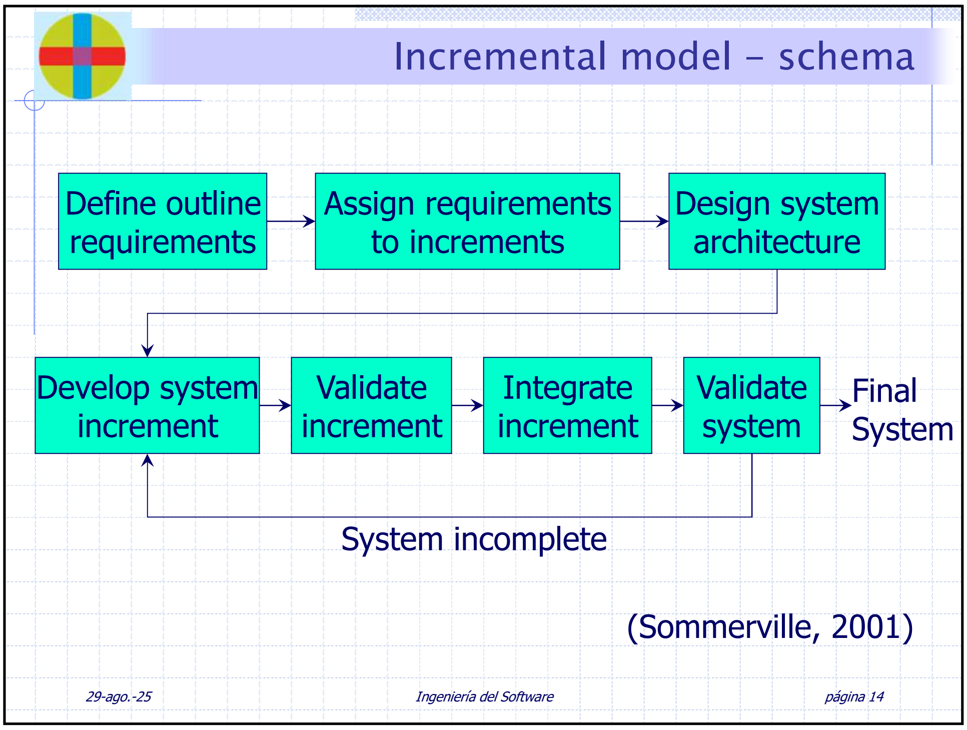
increment 4: analysis → design → code → test → delivery of 4th increment

calendar time



Incremental model – strengths

- It recognizes the evolutionary nature of software
- Applies elements of linear model applied repetitively, with the prototyping philosophy in place
- The first increment is often the core product
  - Unlike prototyping, it delivers working versions of the product, only with different capabilities built into it
  - Product is delivered in different phases, providing basic functionality and than more advanced features
- Well used if human resources are scarce

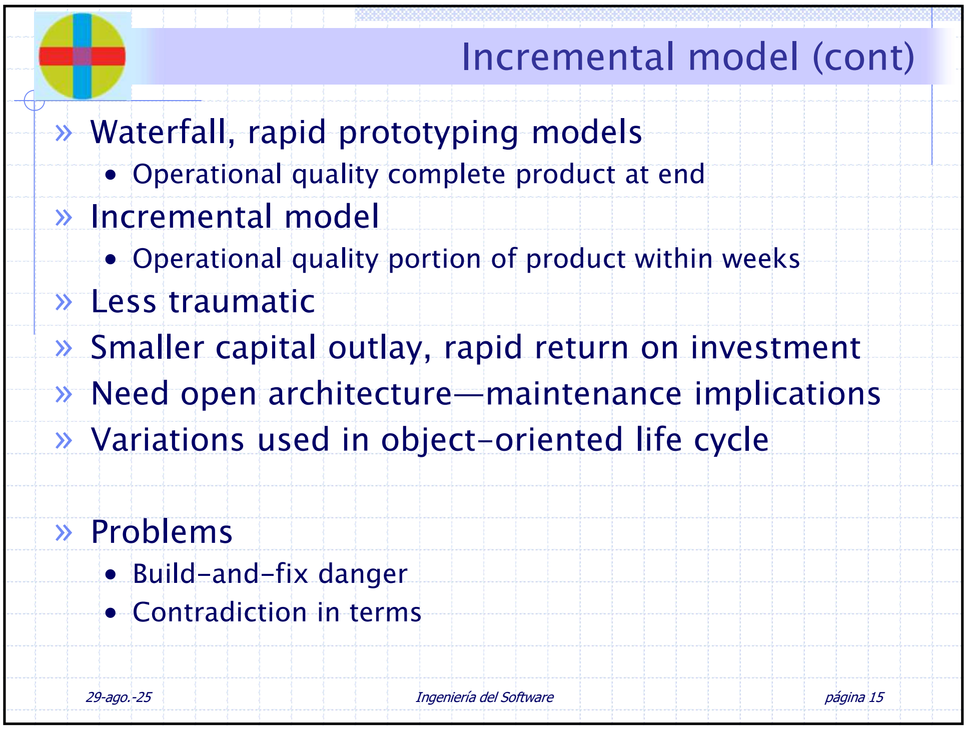


Incremental model – schema

Define outline requirements → Assign requirements to increments → Design system architecture → Develop system increment → Validate increment → Integrate increment → Validate system → Final System

System incomplete

(Sommerville, 2001)

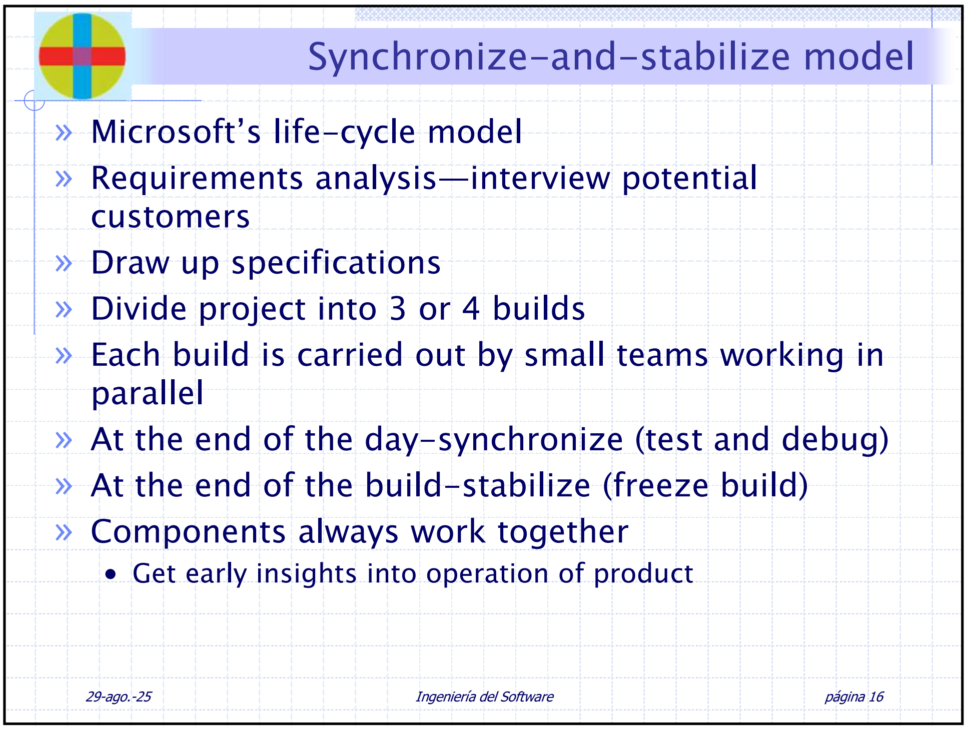


Incremental model (cont)

- Waterfall, rapid prototyping models
  - Operational quality complete product at end
- Incremental model
  - Operational quality portion of product within weeks
- Less traumatic
- Smaller capital outlay, rapid return on investment
- Need open architecture—maintenance implications
- Variations used in object-oriented life cycle

- Problems
  - Build-and-fix danger
  - Contradiction in terms

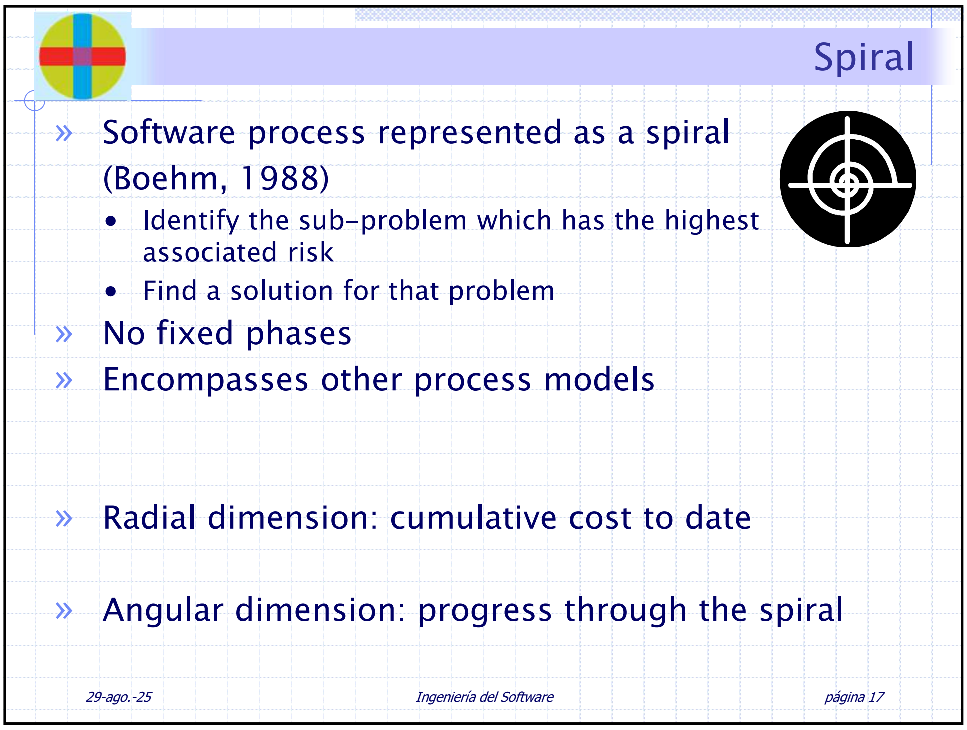
15



Synchronize-and-stabilize model

- Microsoft's life-cycle model
- Requirements analysis—interview potential customers
- Draw up specifications
- Divide project into 3 or 4 builds
- Each build is carried out by small teams working in parallel
- At the end of the day-synchronize (test and debug)
- At the end of the build-stabilize (freeze build)
- Components always work together
  - Get early insights into operation of product

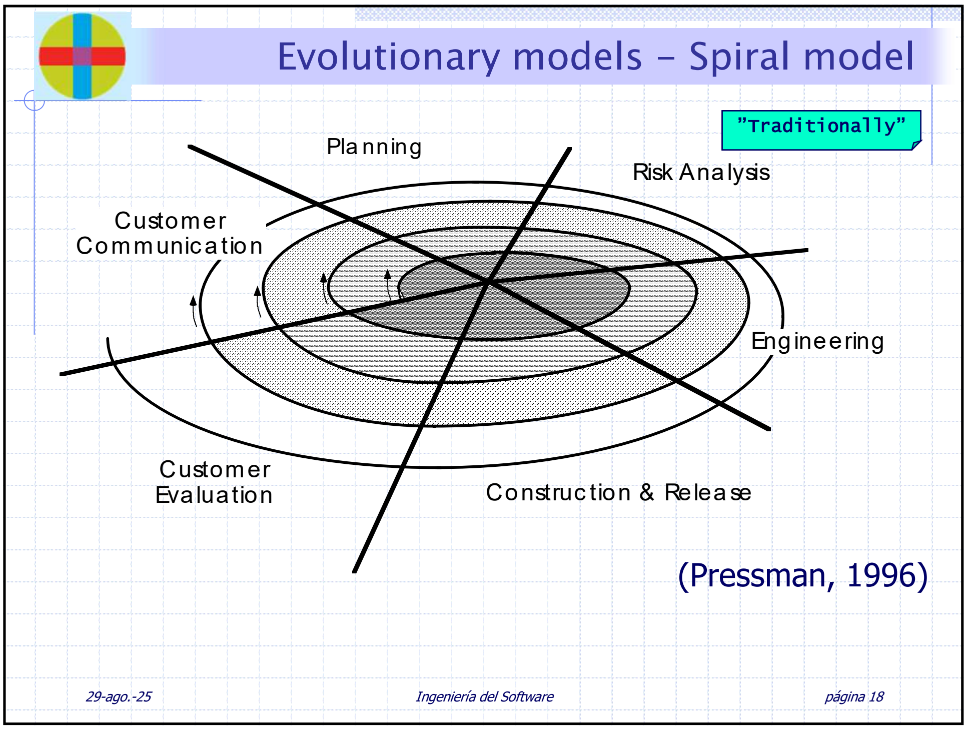
16



Spiral

- » Software process represented as a spiral (Boehm, 1988)
  - Identify the sub-problem which has the highest associated risk
  - Find a solution for that problem
- » No fixed phases
- » Encompasses other process models
- » Radial dimension: cumulative cost to date
- » Angular dimension: progress through the spiral

17

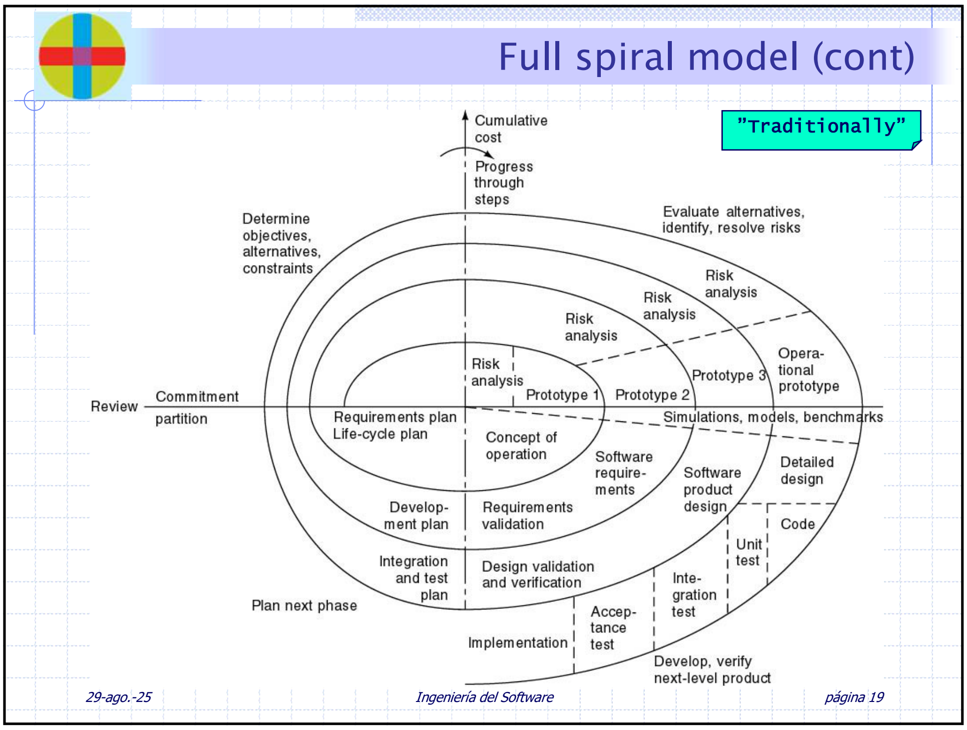


Evolutionary models – Spiral model

"Traditionally"

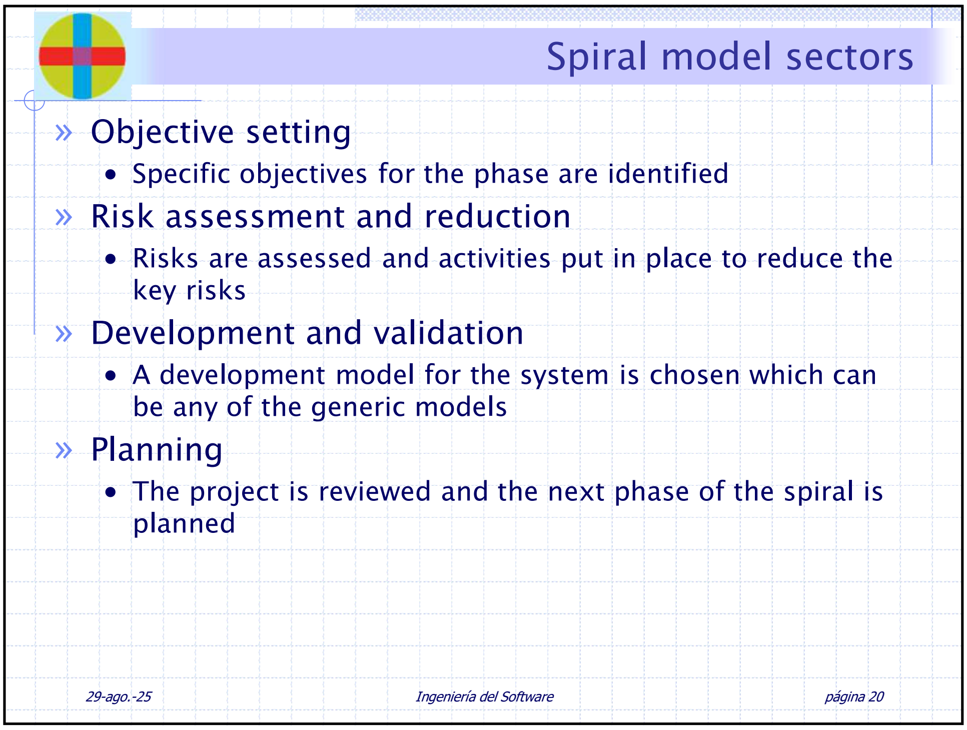
(Pressman, 1996)

18



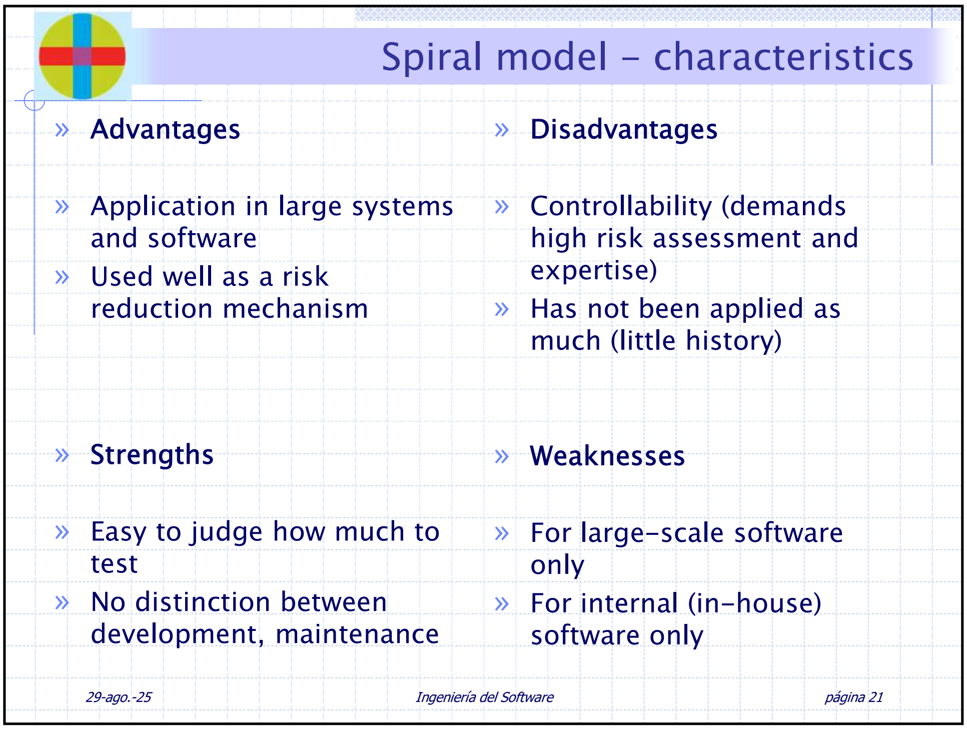
Full spiral model (cont)

"Traditionally"



Spiral model sectors

- Objective setting
  - Specific objectives for the phase are identified
- Risk assessment and reduction
  - Risks are assessed and activities put in place to reduce the key risks
- Development and validation
  - A development model for the system is chosen which can be any of the generic models
- Planning
  - The project is reviewed and the next phase of the spiral is planned



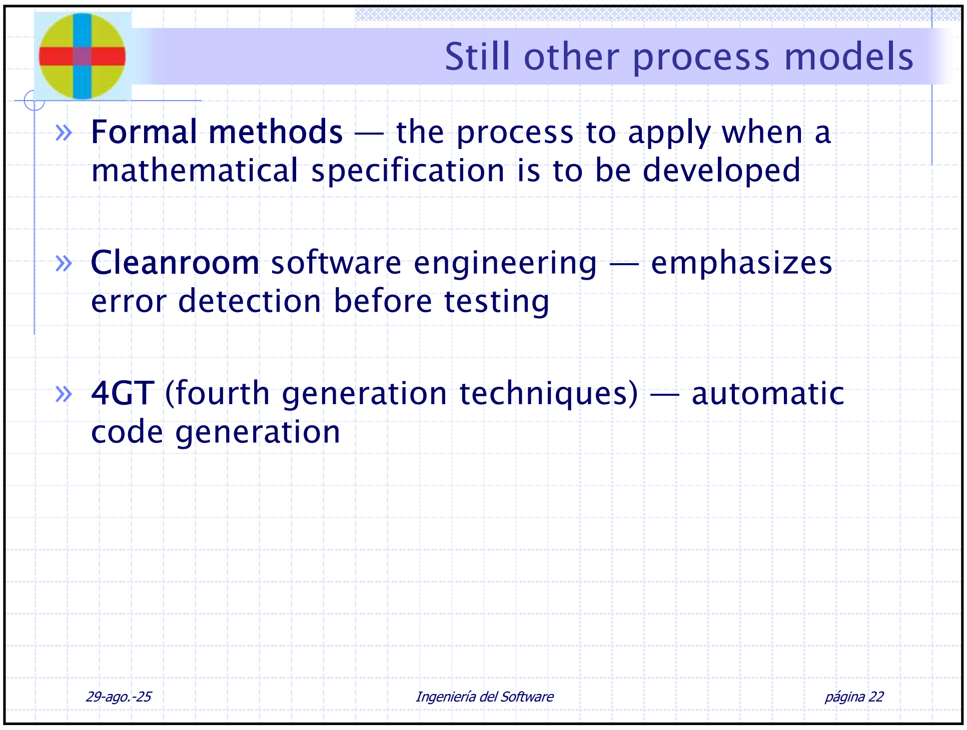
Spiral model – characteristics

- **Advantages**
- Application in large systems and software
- Used well as a risk reduction mechanism

- **Disadvantages**
- Controllability (demands high risk assessment and expertise)
- Has not been applied as much (little history)

- **Strengths**
- Easy to judge how much to test
- No distinction between development, maintenance

- **Weaknesses**
- For large-scale software only
- For internal (in-house) software only

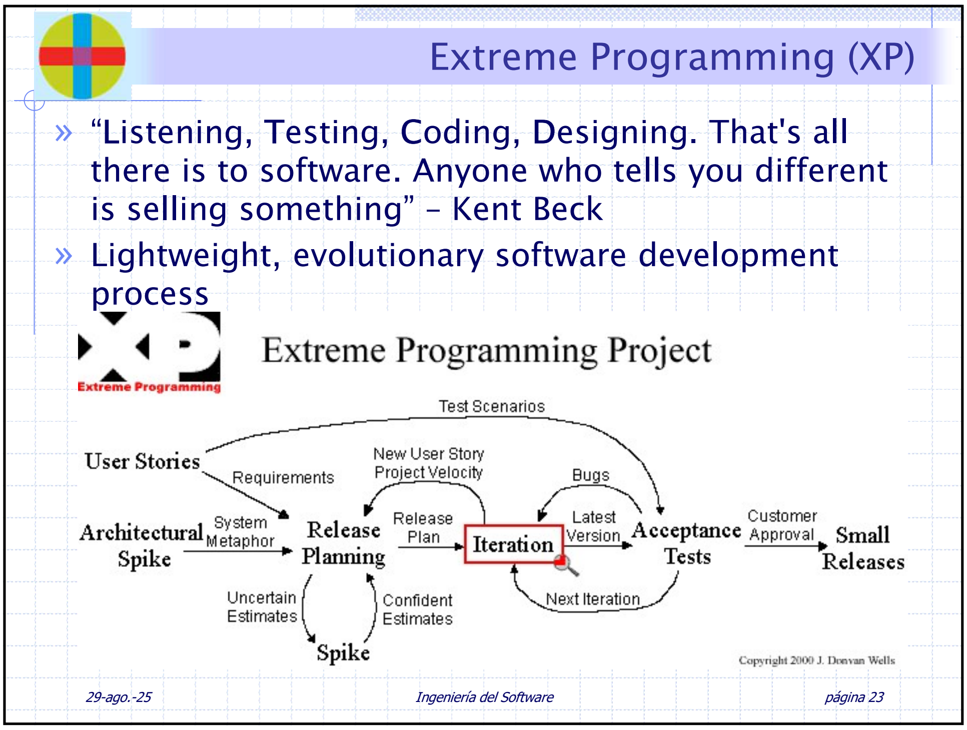


Still other process models

- **Formal methods** — the process to apply when a mathematical specification is to be developed

- **Cleanroom** software engineering — emphasizes error detection before testing

- **4GT** (fourth generation techniques) — automatic code generation



Extreme Programming (XP)

- “Listening, Testing, Coding, Designing. That's all there is to software. Anyone who tells you different is selling something” – Kent Beck
- Lightweight, evolutionary software development process

Extreme Programming Project

Extreme Programming

User Stories

Test Scenarios

Requirements

New User Story Project Velocity

Bugs

Architectural Spike

System Metaphor

Release Planning

Release Plan

Iteration

Latest Version

Acceptance Tests

Customer Approval

Small Releases

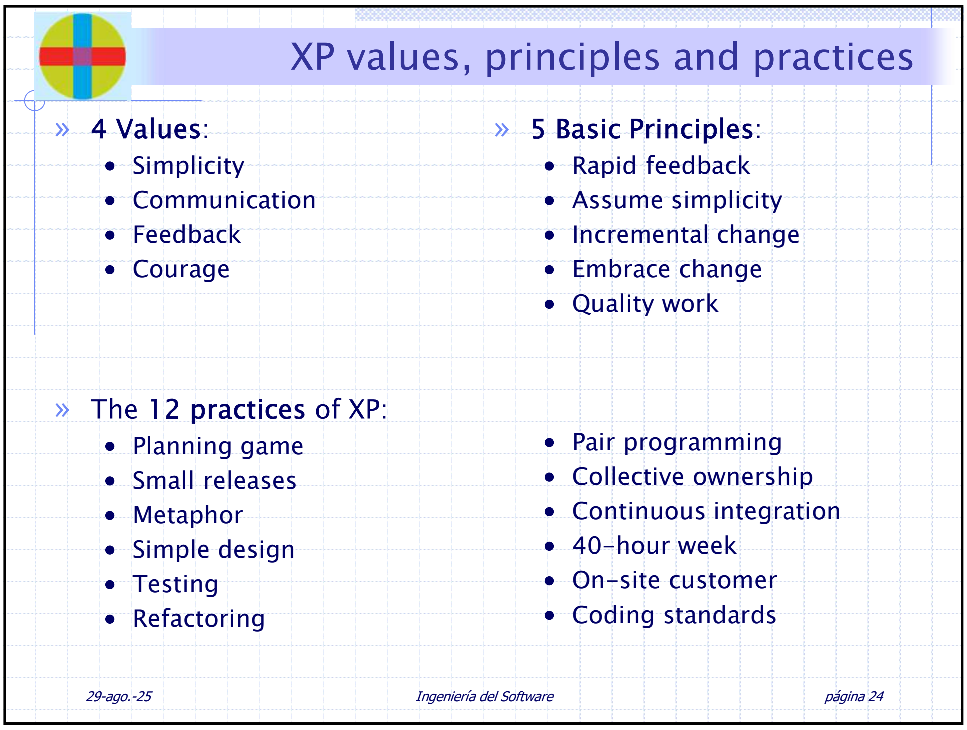
Uncertain Estimates

Confident Estimates

Spike

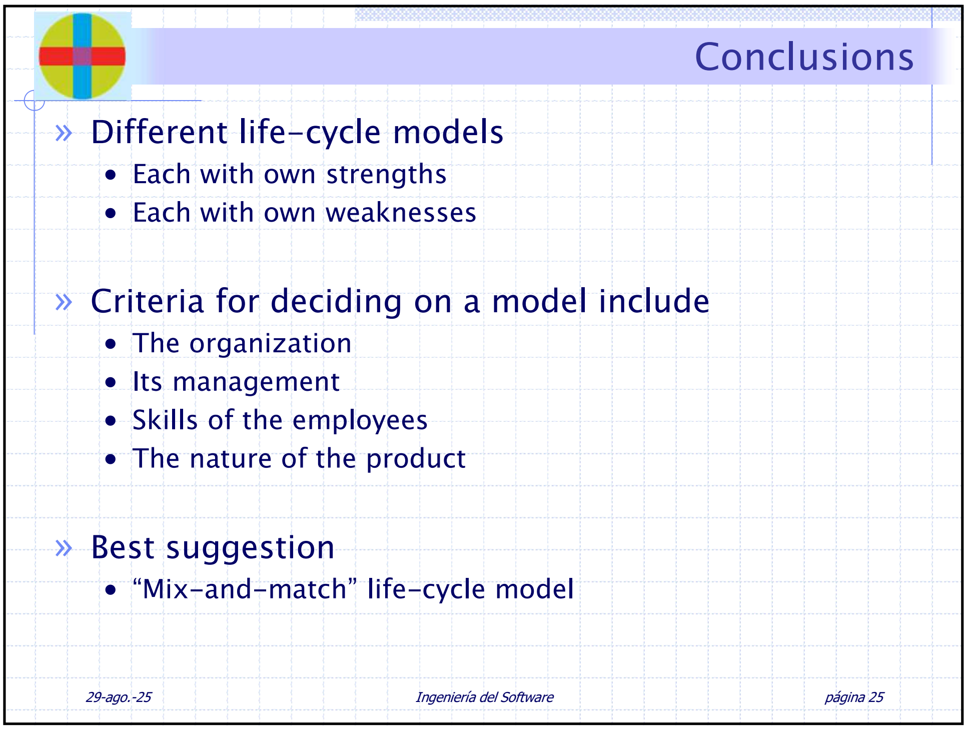
Next Iteration

Copyright 2000 J. Donvan Wells



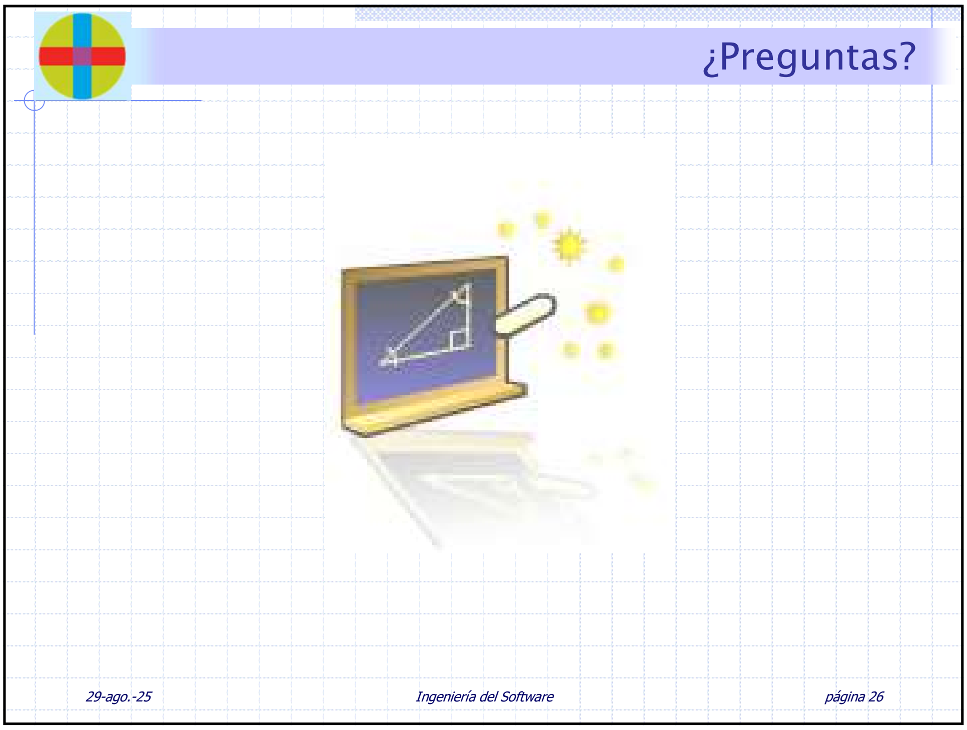
XP values, principles and practices

- **4 Values**:
  - Simplicity
  - Communication
  - Feedback
  - Courage
- **5 Basic Principles**:
  - Rapid feedback
  - Assume simplicity
  - Incremental change
  - Embrace change
  - Quality work
- The **12 practices** of XP:
  - Planning game
  - Small releases
  - Metaphor
  - Simple design
  - Testing
  - Refactoring
  - Pair programming
  - Collective ownership
  - Continuous integration
  - 40-hour week
  - On-site customer
  - Coding standards



Conclusions

- Different life-cycle models
  - Each with own strengths
  - Each with own weaknesses
- Criteria for deciding on a model include
  - The organization
  - Its management
  - Skills of the employees
  - The nature of the product
- Best suggestion
  - “Mix-and-match” life-cycle model



¿Preguntas?