# ECE 321 Software Requirements Engineering

Lecture 2: Software life cycles and software attributes

# Outline

- **Software life cycles**
  - Introduction
  - Models: waterfall, incremental, spiral
- **Software quality**
  - Influence software development

# The software life cycle

- Organizes the development of a software product
  - Series of steps/phases
  - Each phase results in development of part of the system, or something associated with the system
  - Each phase can last days to years
- Includes information about
  - Process stages
  - Overall process
  - Intermediate products
  - Stakeholders

# What is a stakeholder?

*“A stakeholder is a person holding a large and sharp stake...*
*If you don’t look after your stakeholders, you know where the stake will end up.”*

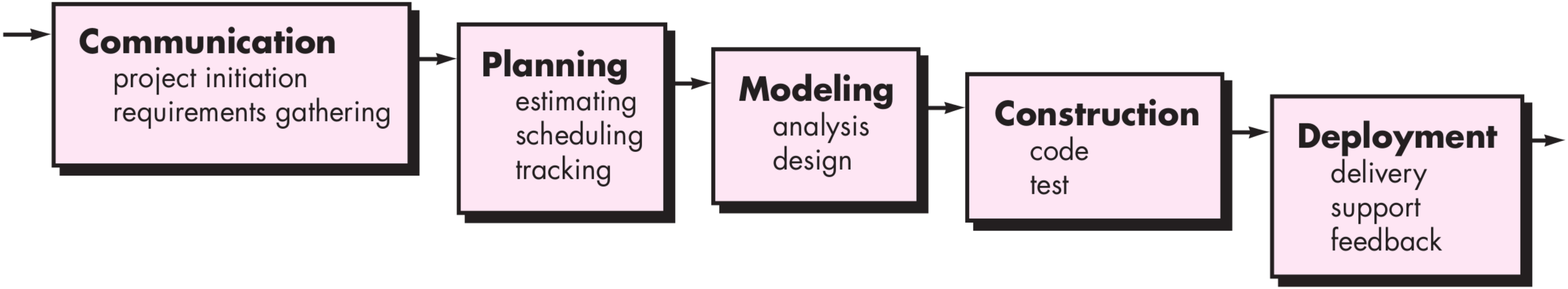
# Examples of stakeholders

- Project sponsors / customers
- End-users
- Domain experts
- Software engineers
- System / network administrators
- ...

# Incremental vs. iterative models

- **Iterative** (**waterfall** model)
  - Re-do project in each stage
    - Sequential phases
- Incremental (**evolutionary**, **spiral** models)
  - Add to project in each stage, at each stage:
    - Identify risks
    - Brainstorm to reduce or eliminate these risks
    - Form a concrete plan with specific artifacts
    - Carry out the plan

# Waterfall model



- Each phase
  - Has well defined start and end points
  - Has deliverables for the next phase
  - Generates results that ‘flow’ into the next phase

# The waterfall model consists of 5 phases

- **Communication**
  - Analyzing and specifying requirements
    - Main result: Software Requirement Specification document
    - End-user is consulted to define the document
- **Planning**
- **Modeling / design**
- **Construction (code & testing)**
- **Deployment**

# Strengths of the waterfall model

- Excellent for discipline, visibility and control
- Encourages extensive well-written documentation
- Easy for management to understand and monitor
- Good when:
  - Requirements are stable: everybody knows exactly what to do
  - Solutions are well understood
  - People make little or no mistakes
  - Customer can wait until you are ready: no need for prototypes

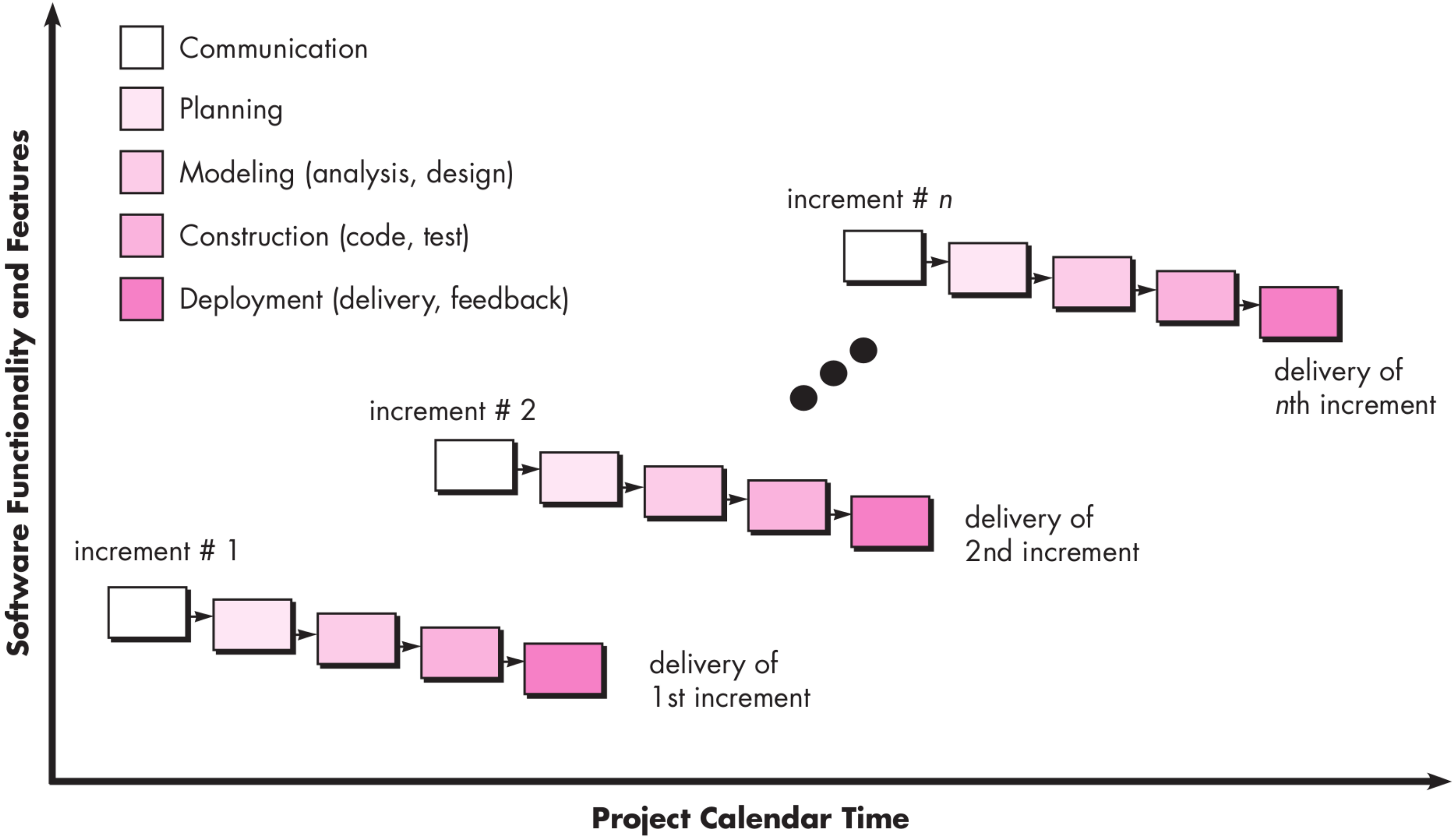
# Weaknesses of the waterfall model

- Concurrent development is difficult, e.g., requirement analysis and system test design are often performed together
- Results in postponing decisions
  - Difficult tasks are performed later in the process
- No place for corrections and feedback loops
- Discourages prototyping
- May cause disconnection with end-user
- Large investment before a product emerges

# Incremental model

- Development cycle is divided into smaller “incremental” cycles
- Each cycle (increment/iteration) adds something to the product
  - Each cycle uses waterfall-based model to add some functionality
  - Each cycle includes planning for the next cycle
  - Users can access a product at the end of each cycle

# The incremental model visualized



# Strengths of the incremental model

- Address the problem of changes in requirements
  - Incorporates early and ongoing involvement of the end-user in the development process
- Provides improved visibility
  - Smaller, more manageable pieces hence better visibility of risks
  - More flexibility
- Increases productivity and motivation
  - Early results will boost morale of all stakeholders

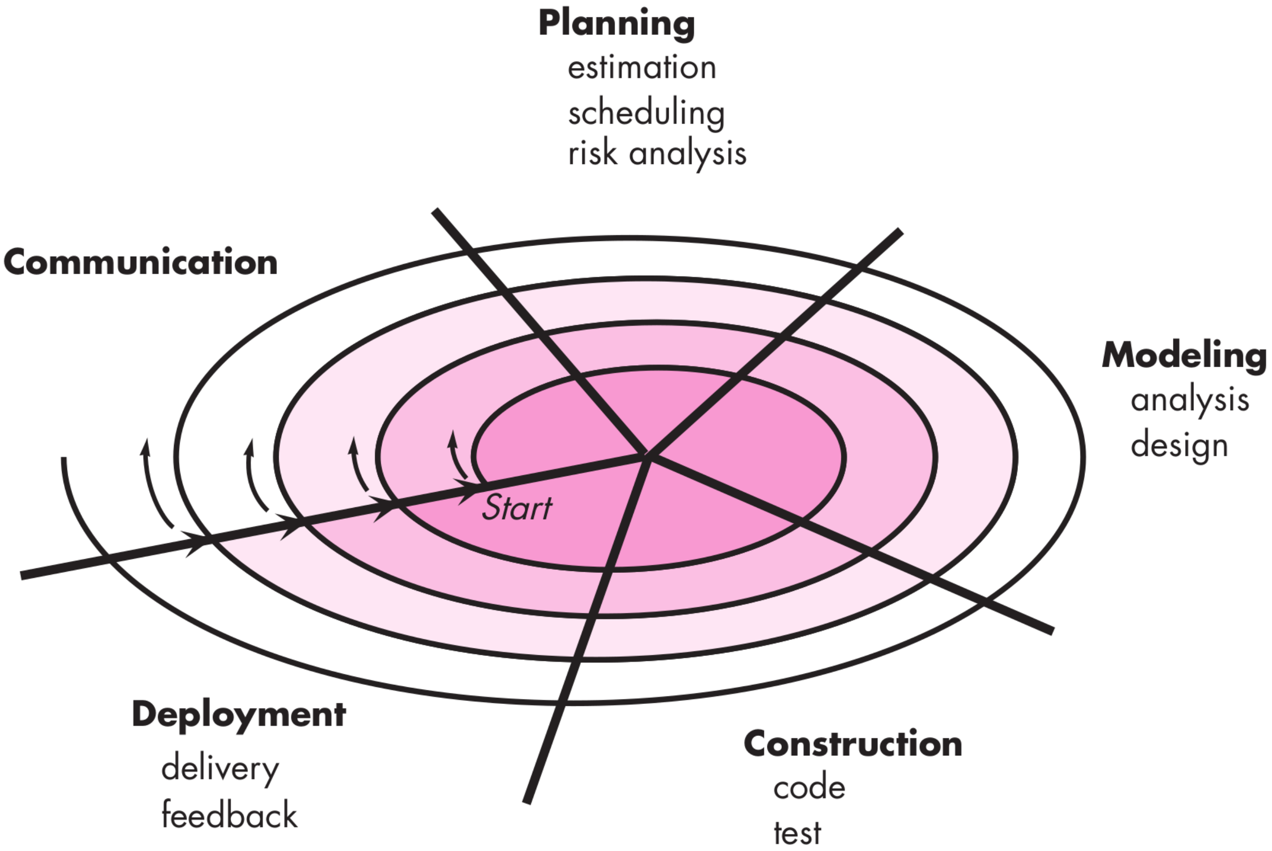
# Incremental vs. evolutionary model

- They are similar, BUT
  - Evolutionary implies that the requirements evolve during the life cycle
  - Evolutionary models do not know upfront how the end-product will look

# Spiral model

- Some functionality is more risky than other
- The spiral model is driven by risk analysis
  - Implement one risky functionality per cycle and ask for feedback on the prototype
  - Plan rest of the product based on the feedback

# The spiral model visualized



# Summary of development models

- **Waterfall**
  - Notion of stages
- **Incremental**
  - Notion of changing requirements and necessity to consult the user during the entire process
- **Spiral**
  - Prototyping & risk analysis

# What is software quality?

- External vs. Internal
  - External: visible to the end user
  - Internal: visible to the developers
  - Boundary is not always clear!
- Product vs. Process
  - The quality of a process can impact the quality of a product
  - Some quality attributes apply to both process and product (e.g., efficiency)

# Software quality attributes

| Discussed today | Other |
|---|---|
| Correctness | User friendliness |
| Reliability | Reparability |
| Robustness | Evolvability |
| Performance | Understandability |
| Maintainability | Productivity |
| Portability | Timeliness |
| Security | Visibility |
| Availability | |
| Flexibility | |
| Interoperability | |
| Usability | |
| Reusability | |
| Testability | |

# Software correctness

- Does the software do what it is supposed to do?
- A system is functionally correct if it behaves according to its functional requirements
- Assessment: testing and verification
  - Testing: experimental
  - Verification: formal

# How to improve correctness?

- Use a structured and formal approach to develop the requirement specification
- Use appropriate tools
  - e.g., high-level languages that support static analysis
- Use standard algorithms and modules

# Software reliability

- Software is reliable when a user can depend on it
  - Formally defined as probabilities that the software will operate as expected
    - Percentage of correctly completed operations
    - Average length of time the system runs before failing
- Note the contrast with other engineering disciplines!
  - Software is expected to have bugs

# Software reliability vs. correctness

- Correctness is absolute
  - Even if ONE requirement is not satisfied, the system is incorrect
- Reliability is relative
  - A system can be reliable but not correct

# Software robustness

- Software is robust if it behaves ‘reasonably’ in situations not specified in its requirements
  - Incorrect input, hardware failure, loss of power
- Related to correctness
  - If ALL exceptional situations are in the specification, then robustness = correctness

# Examples of robustness requirements

- **Text editor**
  - If the editor fails before the user saves the file, the editor should be able te recover all changes
- **Modeling tool**
  - All model parameters should have default values which will be used when current values are missing

# Software performance

- Does the software make efficient use of its computing resources?
  - CPU, memory, disk space, etc.
- How quickly does it perform its operations?
- Can be studied through models or measurements

# Examples of software performance requirements

- **Compiler**
  - The interpreter shall parse at least 5000 error-free statements per minute
- **Web site**
  - Every page shall download in 15 seconds or less on a 3G connection

# Software maintainability

- How easy is it to modify the software or fix a bug?
- Example requirement
  - Existing reports shall be modified so that they comply with new regulations within 20 labour hours or less
- Maintainability is difficult to quantify!

# Software portability

- Can the system run in different environments in terms of hardware and operating system?
- Can be implemented by
  - Portable programming languages
  - Particular tools, like compilers
- Example requirement
  - use software in Linux and Windows

# Software security

- Concerns unauthorized access and other forms of system misuse
  - Usage of cryptographic techniques
  - Keeping logs
  - Constraining communication between different parts of the system
  - Checking data integrity for critical variables
- Example requirement
  - The communication between user console and data storage shall be protected by a symmetric encryption algorithm that uses at least 128-bit long keys

# Software availability

- Concerns the ability to guarantee planned uptime
  - Hardware and software fault tolerance
  - Scheduled maintenance
  - Checkpoints
  - Recovery
  - Restart
- Example requirement
  - The system shall at least be 99.5 percent available on weekdays between 6am and midnight

# Software flexibility

- How easy is it to add new functionality?
  - Could include adaptation to a new OS or hardware
- Example requirement
  - A programmer with at least one year of experience with this product shall be able to add a new printer driver with no more than four hours of labour

# Software interoperability

- Concerns capability of exchanging data or services with other systems
- Example requirement
  - The product shall be able to import files in CSV format

# Software usability

- Ease of use of the product
- Example requirement
  - A trained user shall be able to submit a complete request in an average of two and a maximum of six minutes

# Software reusability

- Can we reuse a software component in other applications?
  - Usually concerns libraries or general objects
  - Improved by high-level programming languages (e.g., standard libraries)
- Example requirement
  - The printer driver shall be reusable at the object code level in other applications that use the ANSII C standard

# Software testability

- Concerns the ease with which software components or the integrated product can be tested
  - For example, can we isolate a component?
- Example requirement
  - Each function shall have at least one unit test

# What to keep in mind about software quality attributes

- There are many different ones!
- Usually a project uses only a few
  - Depending on the type of project
  - e.g., a website and a car have very different types of requirements