COMP 3111
SOFTWARE ENGINEERING

LECTURE 5
SOFTWARE DEVELOPMENT

SOFTWARE DEVELOPMENT OUTLINE

- ✓ Overview of Software Development
  - Nature and Types of Software
  - Types of Software Development Projects
  - Software Development Life Cycle (SDLC)
  - The Four P's in Software Development

- ➔ **Software Development Processes**
  - Monolithic
    - Waterfall
  - Iterative and Incremental
    - Code-and-Fix
    - Prototyping
    - Spiral
    - Phased-release
    - Agile
    - Unified Process (UP)

SOFTWARE DEVELOPMENT PROCESSES STAGES

- Most software development processes share the following stages:
  - gathering the system requirements
  - analyzing and designing the system
  - implementing the system
  - testing the system

- They mainly differ in how these stages are:

1. combined
2. emphasized
3. carried out

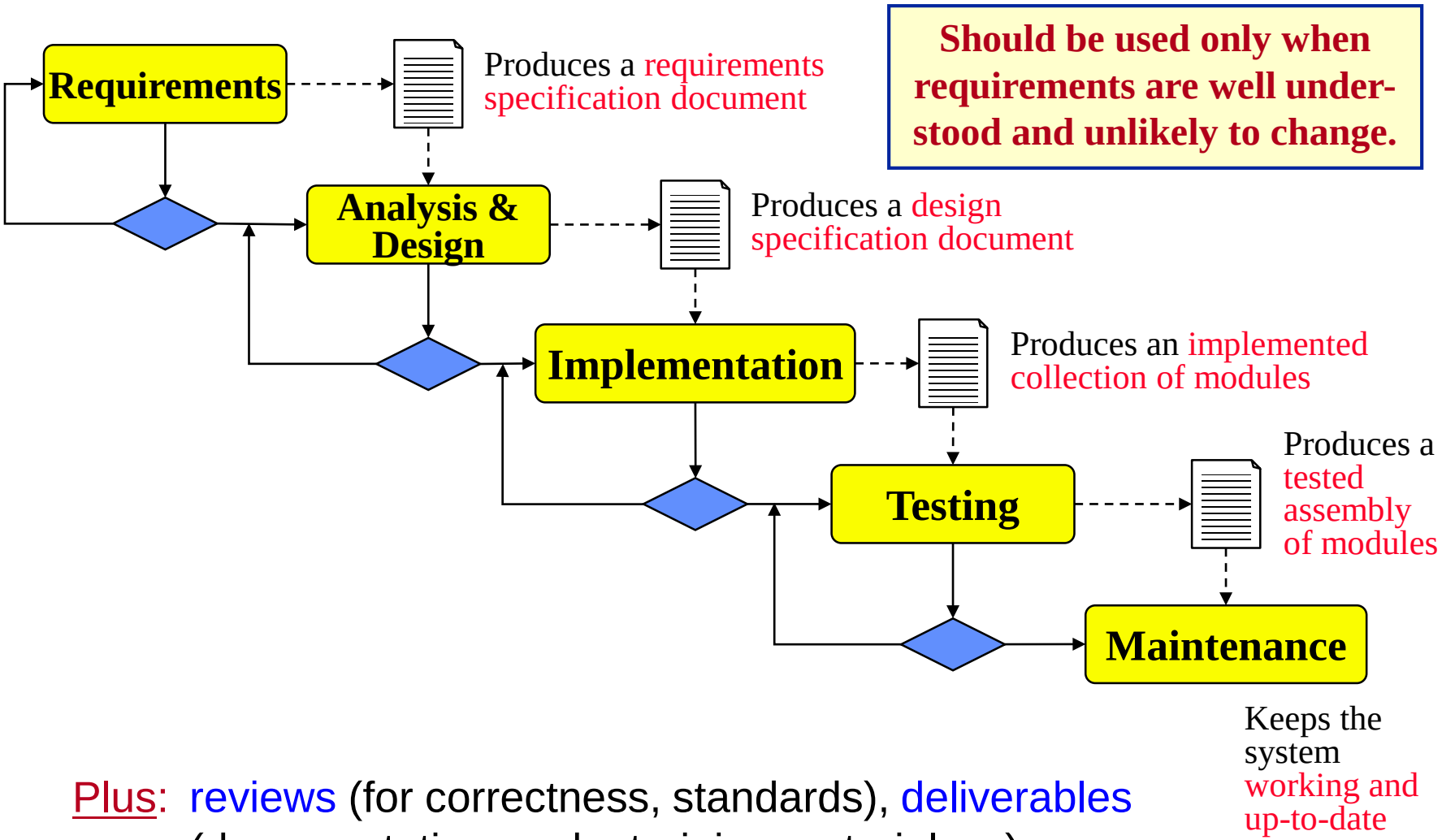
We want to understand the strengths and weaknesses of different software development processes.

SOFTWARE DEVELOPMENT OUTLINE

- ✓ Overview of Software Development
  - Nature and Types of Software
  - Types of Software Development Projects
  - Software Development Life Cycle
  - The Four P's in Software Development

- ➔ **Survey of Software Development Processes**
  - ➔ **Monolithic**
    - ➢ **Waterfall**
  - Iterative and Incremental
    - ➢ Code-and-Fix
    - ➢ Prototyping
    - ➢ Spiral
    - ➢ Phased-release
    - ➢ Agile
    - ➢ Unified Process (UP)

WATERFALL PROCESS



Plus: reviews (for correctness, standards), deliverables (documentation, code, training material, …), …

WATERFALL PROCESS: PROS & CONS

Pros

- Imposes needed discipline (rigor and formality).
- Keeps development predictable and easy to monitor.
- Enforces documentation standards and approval of documents before proceeding.
- Fits well with other engineering process models (e.g., hardware development).

Cons

- Assumes linear, sequential development is possible.
- Rigid assuming results of each phase can be frozen before proceeding to the next phase.
- Different languages/notations often used in each phase.
- Makes little provision or opportunity for user feedback, which is a source of high risk.

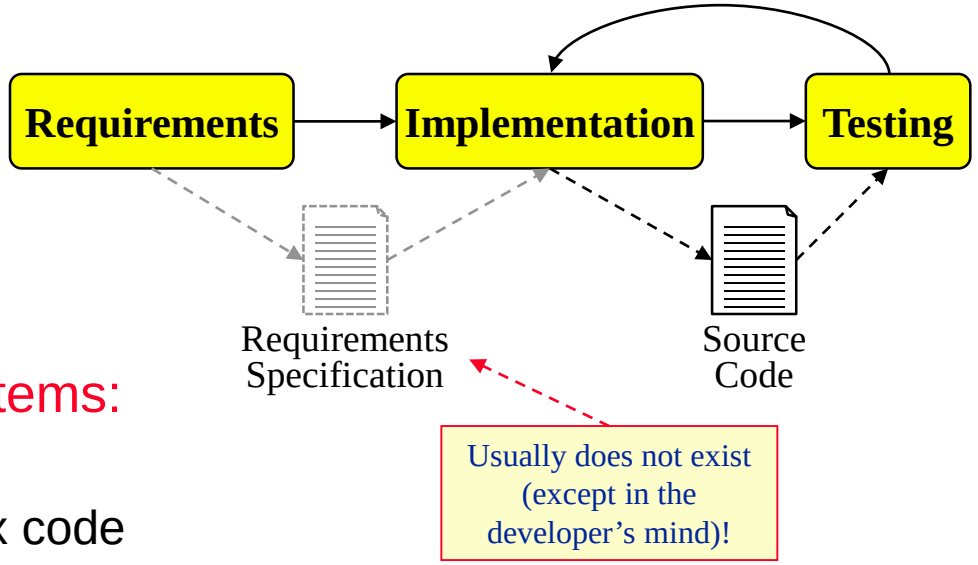
SOFTWARE DEVELOPMENT OUTLINE

- ✓ Overview of Software Development
  - Nature and Types of Software
  - Types of Software Development Projects
  - Software Development Life Cycle
  - The Four P's in Software Development

- ➔ **Survey of Software Development Processes**
  - Monolithic
    - Waterfall
  - ➔ **Iterative and Incremental**
    - **Code-and-Fix**
    - **Prototyping**
    - **Spiral**
    - **Phased-release**
    - **Agile**
    - **Unified Process (UP)**

CODE-AND-FIX PROCESS

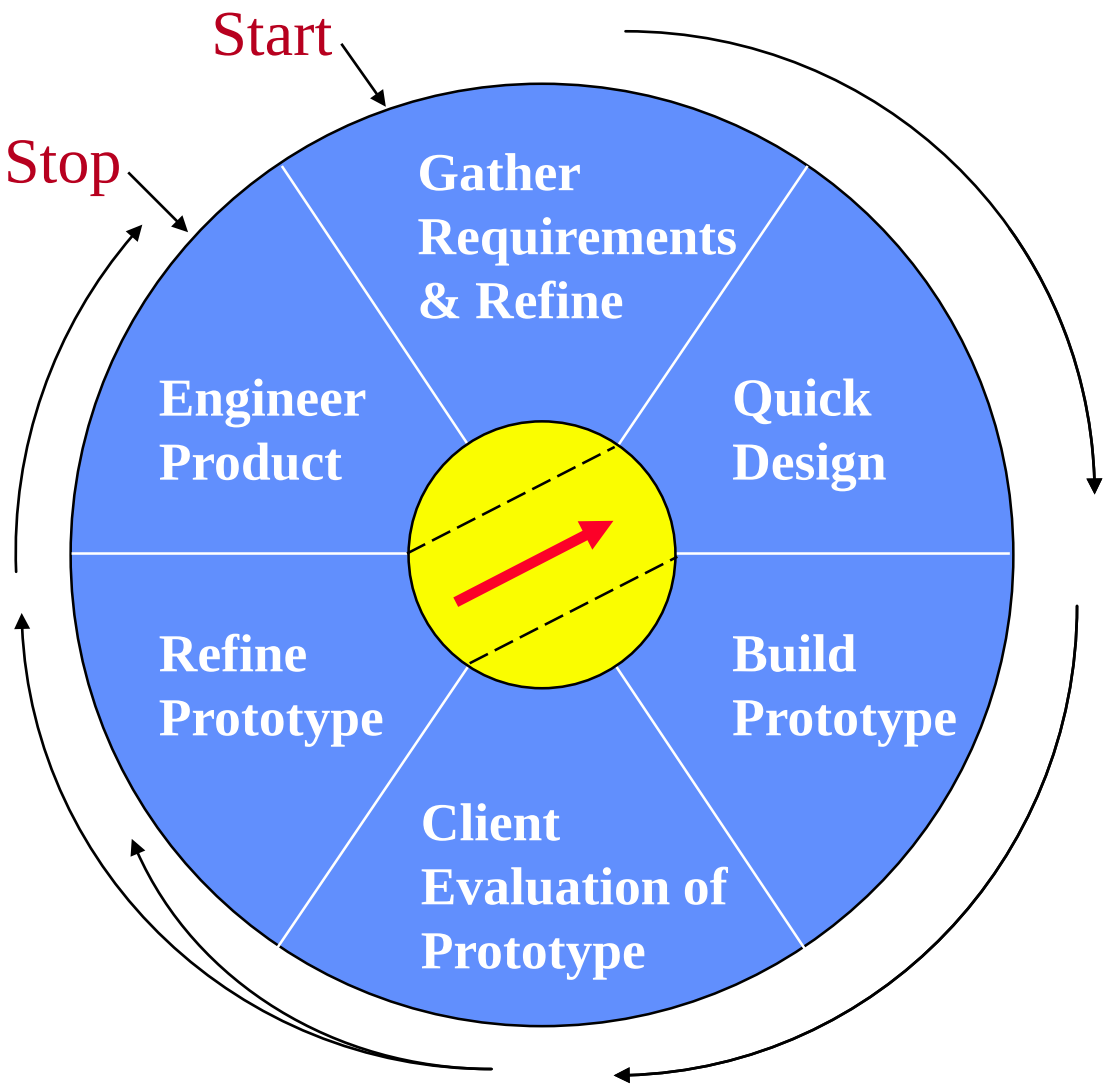
- Many changes
  - ☞ code structure often becomes messy



- Unsuitable for large systems:
  - turnover of personnel
  - difficult to understand/fix code
  - requirements can easily be unmatched

- The software development process becomes:
  - unpredictable and uncontrollable
  - over schedule, over budget and fails to meet expectations

PROTOTYPING PROCESS



- Basically a code-and-fix process, **BUT** includes client evaluation and enforces some discipline.
- Useful when requirements are vague or unknown as it allows exploration of
  - functionality needed
  - user interface

What to do with the final prototype?
(80/20 rule)

PROTOTYPING PROCESS: PROS & CONS

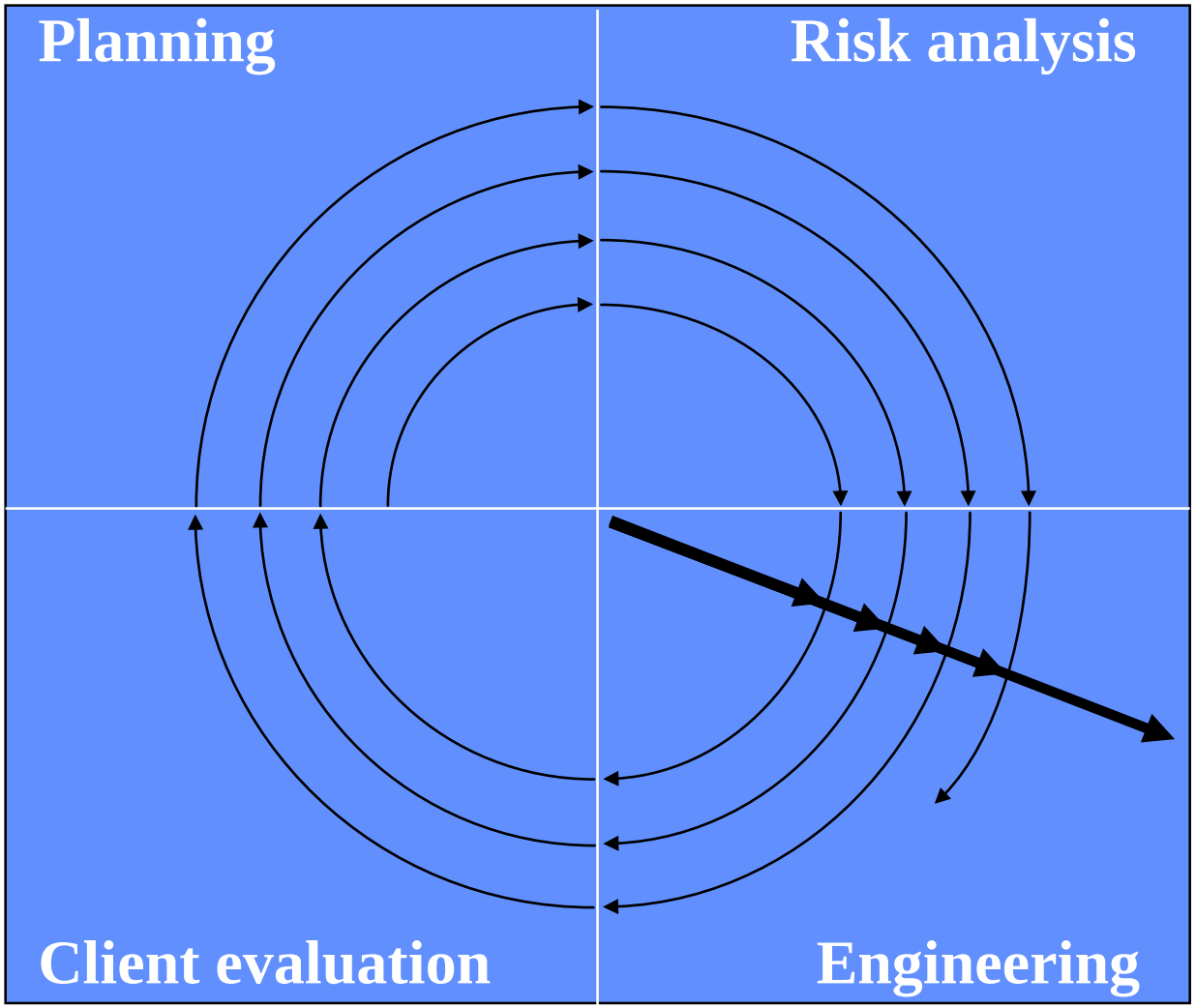
Pros

- Allows requirements to be quickly explored.
- Allows user feedback and approval to be obtained.
- Allows different solutions to be explored.

Cons

- It is not really a complete software development process.
- The process is not visible making progress hard to measure.
- Documentation is often sparse or completely absent.
- The final “product” is not a complete system.

SPIRAL PROCESS



Go, no-go decision

Toward a completed system

SPIRAL PROCESS: PROS & CONS

Pros

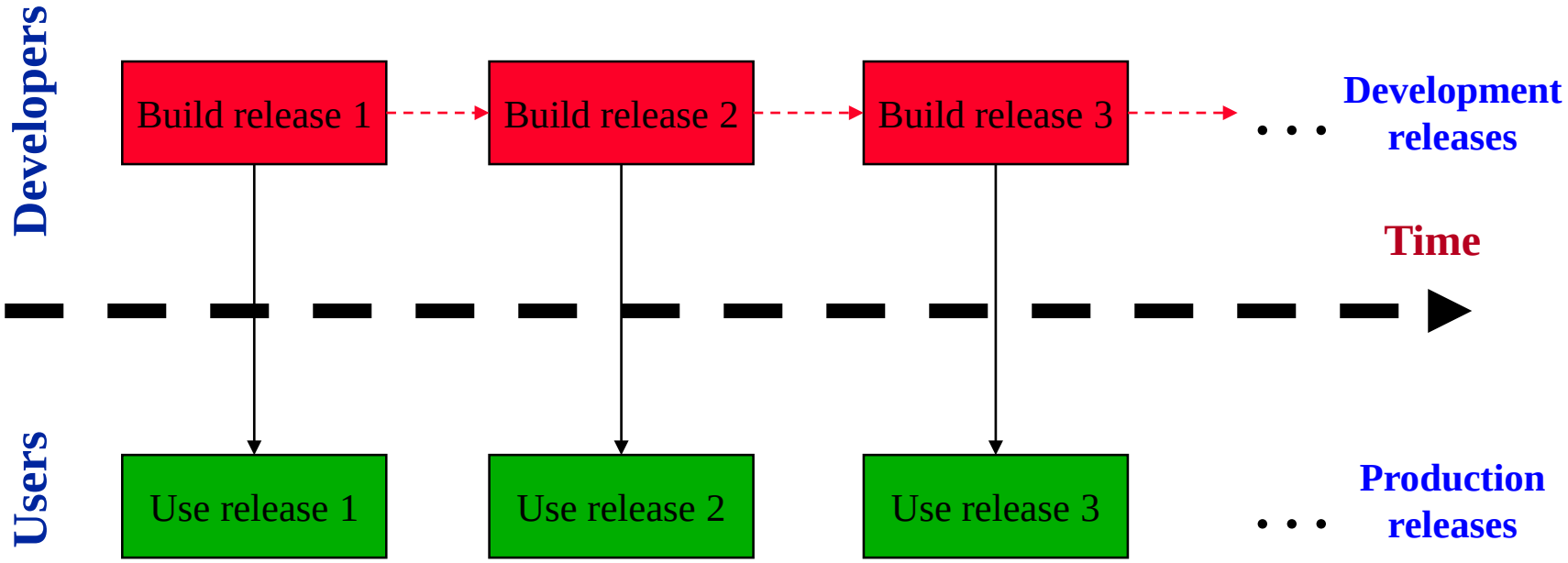
- Risk evaluation can help reduce development problems.
- Planning and client evaluation phases help the product better meet client expectations.
- Iterative and incremental planning, engineering and evaluation facilitates project management.

Cons

- Relies on expertise in risk assessment.
- Needs more elaboration of the phases (i.e., specific activities that should be performed).
- More appropriate for internal development than contract development.

PHASED-RELEASE PROCESS

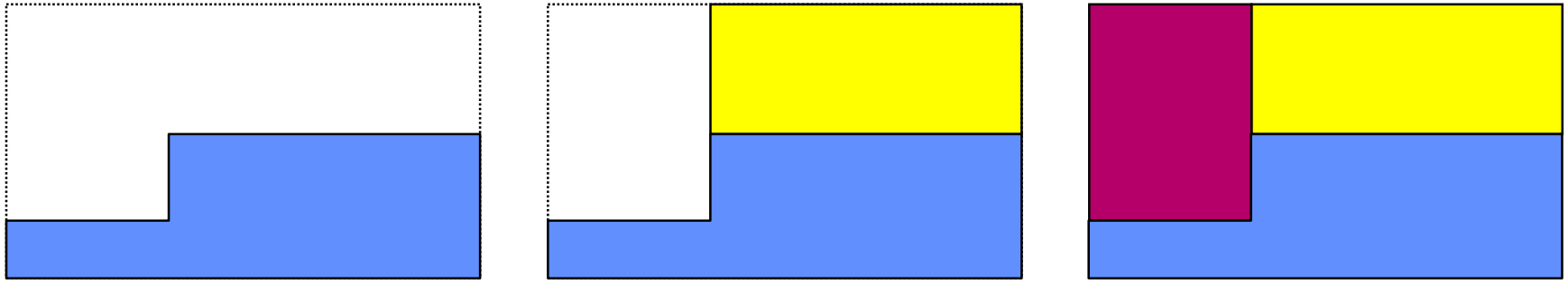
Premise: Change is inevitable, so plan for it!



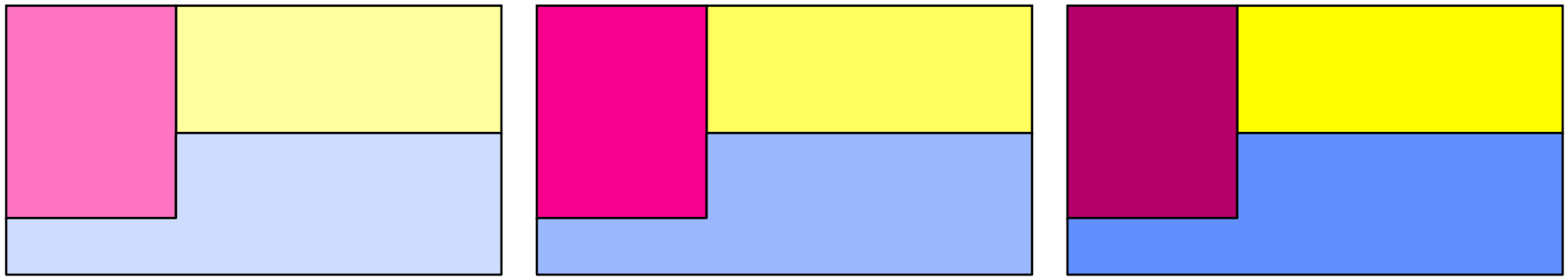
Releases are developed and used in parallel.

PHASED-RELEASE PROCESS (cont'd)

incremental development → partial system; full functionality



iterative development → full system; partial functionality



Many organizations use a combination of iterative and incremental.

PHASED-RELEASE PROCESS: PROS & CONS

Pros

- Reduces the risk of project failure.
- Promotes system modularity.
- Allows frequent releases.
- Allows appropriate expertise to be applied.
- Allows early training and feedback.

Cons

- The system pieces need to be relatively small.
- It may be hard to identify common facilities needed by all pieces.

AGILE PROCESS

- Any phased (incremental) approach where the emphasis is more towards the items on the left.

← more important	less important →
individuals and interactions	*processes and tools*
working software	*comprehensive documentation*
client involvement/collaboration	*contract negotiation*
responsiveness to change	*following a plan*

☞ **This does not imply that there is no value in the items on the right!**

AGILE PROCESS (cont'd)

- **Methods**
  - Extreme Programming (XP)
  - Continuous Integration
  - Scrum

- **Practices**
  - Planning poker → used to estimate time required to implement a feature (see http://en.wikipedia.org/wiki/Planning_poker)
  - Pair programming → used to write code for a feature
  - Test Driven Development (TDD) → used to test the code

AGILE PROCESS: EXTREME PROGRAMMING (XP)

- Requirements and analysis:
  - *developer* determines features needed
    estimates time and cost for each feature
  - *client* selects features to be included in each iteration

- Implementation (by iterations/sprints):
  - the *developer* breaks each iteration into tasks
  - for each task (where tasks can be carried out in parallel) the *developer*
    - designs test cases (*test-driven development*)
    - implements the task using *pair programming*
    - integrates the task into the current product

The major emphasis is here.

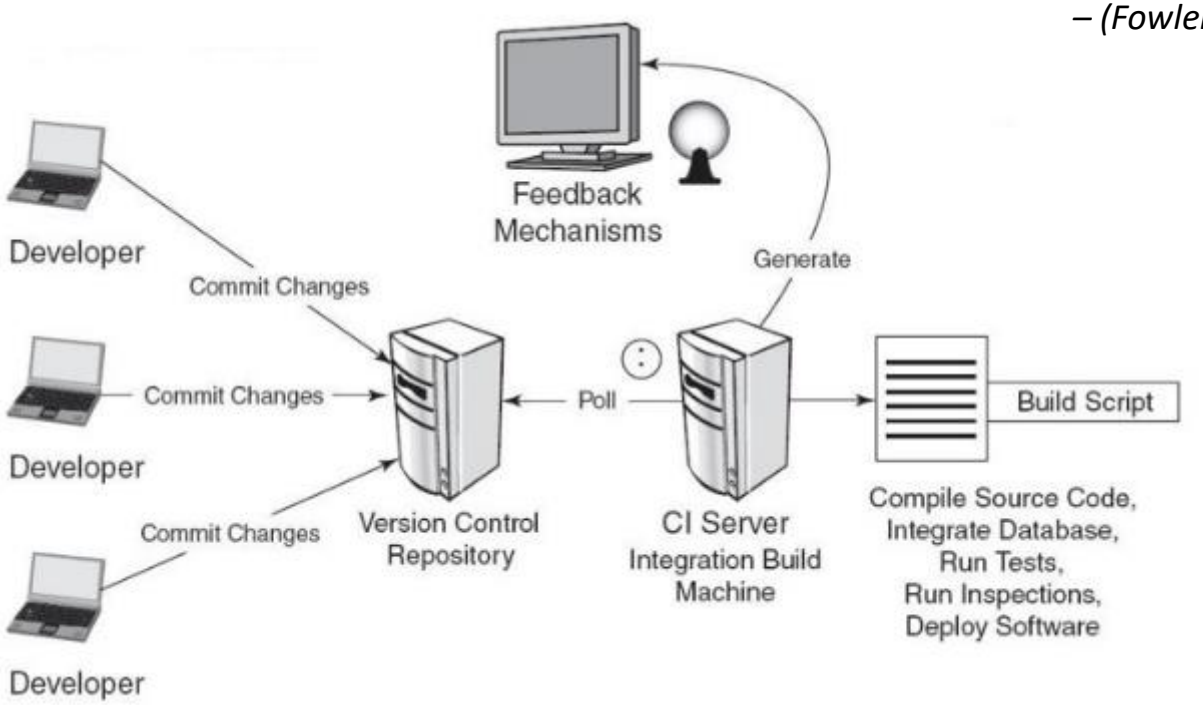
AGILE PROCESS: CONTINUOUS INTEGRATION (CI)

- Continuous Integration (CI) is an agile software development process where developers integrate code into a shared repository on a daily basis.
  - allows developers to integrate and test their code early
  - quickly catches/exposes any bug that breaks the build
  - reduce integration conflicts by doing integration continuously
  - help developers to check the progress of their development
  - building and testing are done automatically with scripts

- A CI Server is required to compile the code periodically, and reports the results to the developers.

CONTINUOUS INTEGRATION: ARCHITECTURE

The key to fixing problems quickly is finding them quickly.
– (Fowler, 2006)



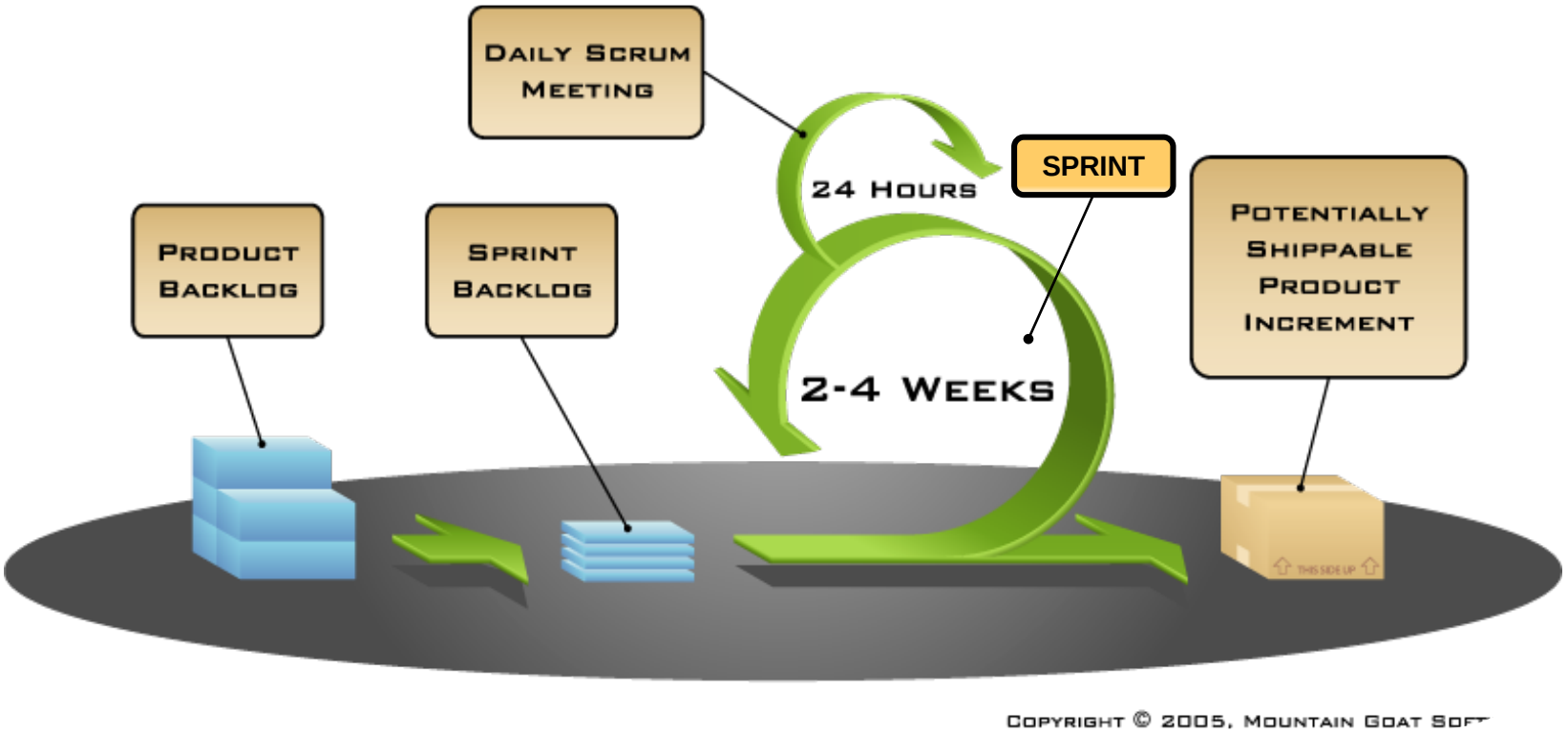
- Developers submit work to the main repository on a daily basis
- CI Server automatically runs build/test scripts, and notifies developers about failed build/tests

AGILE PROCESS: SCRUM

AGILE PROCESS: SCRUM

- Scrum is an agile software development process that mainly specifies what you should do to develop a software product.
- No specific software engineering practices are prescribed for developing the product; the team needs to decide how to do it.
- The requirements are captured as items in a "product backlog"; the product owner (client) sets the priorities for the items.
- The software product is developed in a series of iterations called "sprints".
- Teams self-organize to determine the best way to deliver the product.

SCRUM: SPRINT WORKFLOW



- The software product is designed, coded and tested during the sprints.

- The requirements are not allowed to change during a sprint.

SCRUM: FRAMEWORK

Roles

- Product owner
- ScrumMaster
- Team

Meetings

- Sprint planning
- Daily scrum meeting
- Sprint review
- Sprint retrospective

Artifacts

- Product backlog
- Sprint backlog
- Burndown charts

SCRUM: ROLES

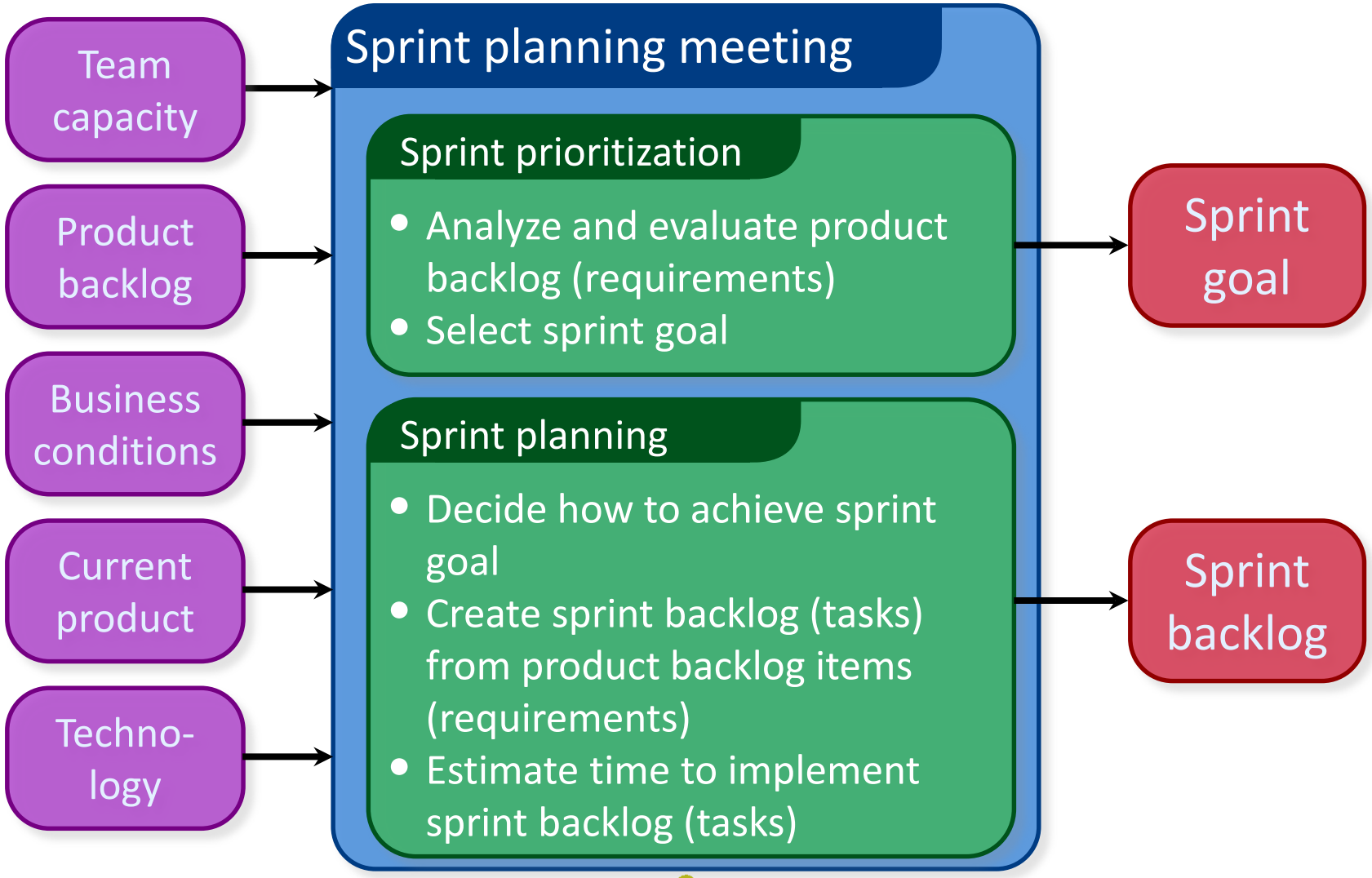
Product Owner (aka Client)

- Is the key stakeholder (represents users, client)
- Defines and prioritizes the requirements of the product.
- Adjusts requirements and priority every iteration, as needed.
- Decides on the release date and content.
- Accepts or rejects work results.

Scrum Master (aka Project Manager / Team Leader)

- Responsible for enacting Scrum values and practices.
- Ensures that the team is fully functional and productive.
- Enables close cooperation across all roles and functions.
- Removes impediments to progress and shields the team from external interferences.

SCRUM: SPRINT PLANNING MEETING



SCRUM: DAILY SCRUM MEETING

- A team meeting in which everyone answers three questions:

1. What did you do yesterday?
2. What will you do today?
3. Is anything in your way?

SCRUM: ARTIFACTS

Product Backlog

- Represents the requirements of the system (i.e., a list of all desired functionality of the system).
- Ideally expressed such that each item has value to the users or customers of the product.
- Items in the backlog are prioritized by the product owner (client) and reprioritized at the start of each sprint.

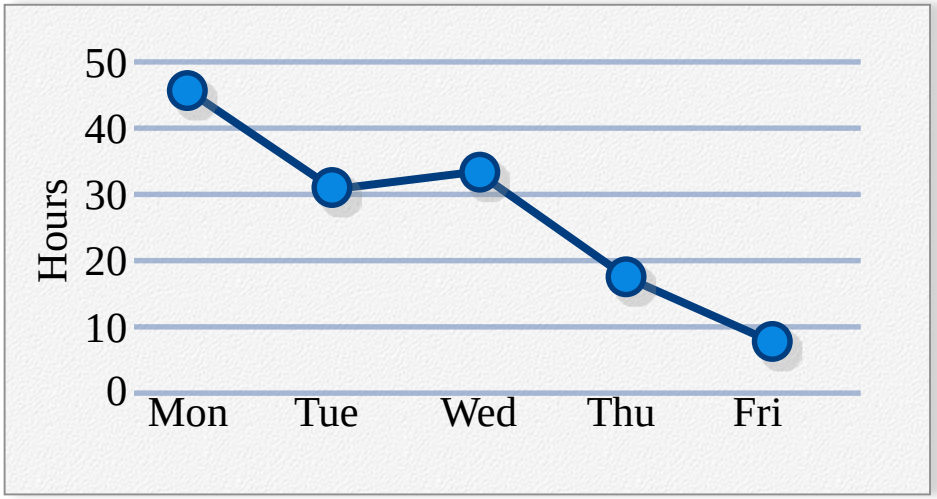
Sprint Backlog

- Contains items selected from the product backlog based on item priority and on how much the team thinks they can do in a sprint.
- A product backlog item may become several sprint backlog tasks.
- Team members select sprint backlog items to work on during the sprint.

SCRUM: ARTIFACTS (cont'd)

Burndown Chart

Tasks	Mon	Tue	Wed	Thu	Fri
Code the user interface	8	4	8		
Code the middle tier	16	12	10	7	
Test the middle tier	8	16	16	11	8
Write online help	12				



The burndown chart graphically shows the *total hours remaining* each day to complete the sprint.

AGILE PROCESS: PROS & CONS

Pros

- The development is adaptable to changing requirements (flexible).
- Immediate feedback is provided by the client/users.
- Results in faster speed-to-market.
- There are fewer defects in the final product.

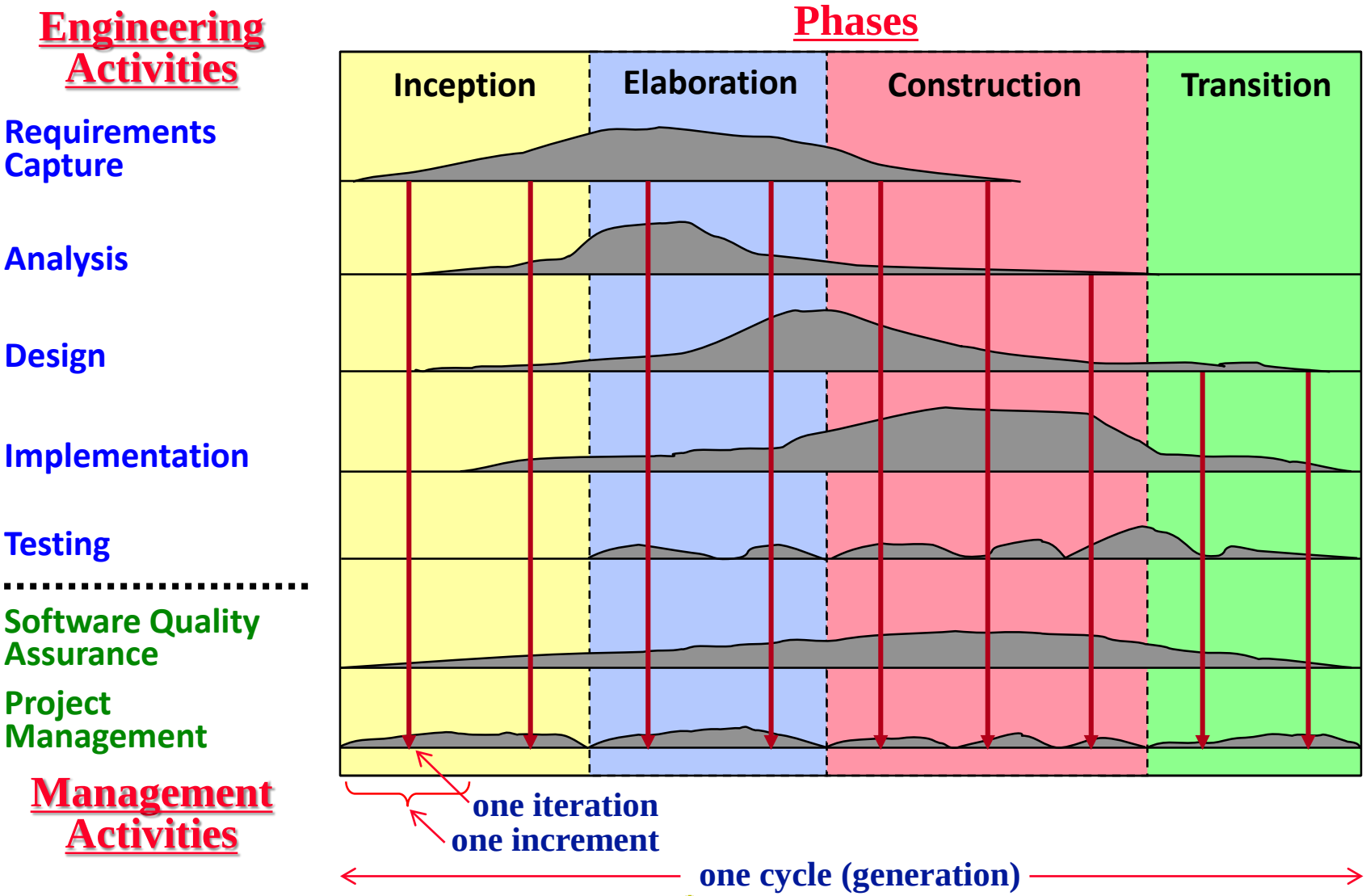
Cons

- Active user involvement and close collaboration are required.
- There is often a lack of documentation.
- There can be scope creep as the client/users add requirements.
- Daily stand-up meetings can take a toll.

SOFTWARE DEVELOPMENT PROCESS: PRINCIPLES

- **rigor and formality** (Waterfall; Spiral)
- **separation of concerns and modularity** (Waterfall; Spiral; Phased-release)
- **abstraction and generality** (Waterfall; Spiral)
- **anticipation of change** (Spiral; Phased-release; Agile)
- **incremental development** (Prototyping; Spiral; Phased-release; Agile)
- **risk assessment** (Spiral)

UNIFIED PROCESS (UP): LIFE CYCLE



UNIFIED PROCESS (UP): MAIN FEATURES

The UP selects from the best practices of previous processes to:

- **provide a generic process framework**
  - ☞ It needs to be instantiated/specialized for specific application areas, organizations, competence levels, project sizes, etc.
- **define a set of activities (workflows)**
  - ☞ The workflows transform users' requirements into a software system.
- **define a set of models (artifacts)**
  - ☞ Models range from abstract (user-level) to concrete (code).
  - ☞ Models are transformed by the workflows into other models.

Each iteration results in a working product.
Each increment establishes a system baseline.

SOFTWARE DEVELOPMENT: SUMMARY

- A software development process needs to consider both management and engineering issues.

- A software development process needs to consider the characteristics of the:
  - organization → size; access to users/client; need for formality.
  - project → small/large; vague/well-defined; novel/well-known.
  - people → availability of expertise; skill of developers.

- The *Unified Process* incorporates best practices of previous software development processes.

The Unified Process provides a *generic framework* to discuss software development activities.

COMP 3111 SYLLABUS

- ✓ **1. Introduction**
- ✓ **2. Modeling Software Systems using UML**
- ✓ **3. Software Development**
4. System Requirements Capture
5. Implementation
6. Testing
7. System Analysis and Design
8. Software Quality Assurance
9. Managing Software Development

SOFTWARE DEVELOPMENT EXERCISE