Object-Oriented Software Analysis and Design

School of Computer Science
University of Windsor

What is the UP?

- A **software development process** describes an approach to building, deploying, and possibly maintaining software.
- The **Unified Process (UP)** has emerged as a popular iterative software development process for building object-oriented systems.
- In particular, the **Rational Unified Process (RUP)**, a detailed refinement of the Unified Process, has been widely adopted.
- The UP is very flexible and open, and encourages including skillful practices from other iterative methods, such as from Extreme Programming (XP), Scrum, and so forth.

  For example,

  XP's test-driven development, refactoring and continuous integration practices can fit within a UP project. So can Scrum's common project room ("war room") and daily Scrum meeting practice.

What is the UP? (contd.)

- A **key practice** in both the UP and most other modern methods is **iterative development**.
- In this lifecycle approach, development is organized into a series of short, fixed-length (for example, three-week) mini-projects called iterations.
- The outcome of each is a tested, integrated, and executable partial system.

What is the UP? (contd.)

- Each iteration includes its own requirements analysis, design, implementation, and testing activities.
- The system grows incrementally over time, iteration by iteration, and thus this approach is also known as **iterative and incremental development**.
- Because feedback and adaptation evolve the specifications and design, it is also known as **iterative and evolutionary development**.

What is the UP? (contd.)

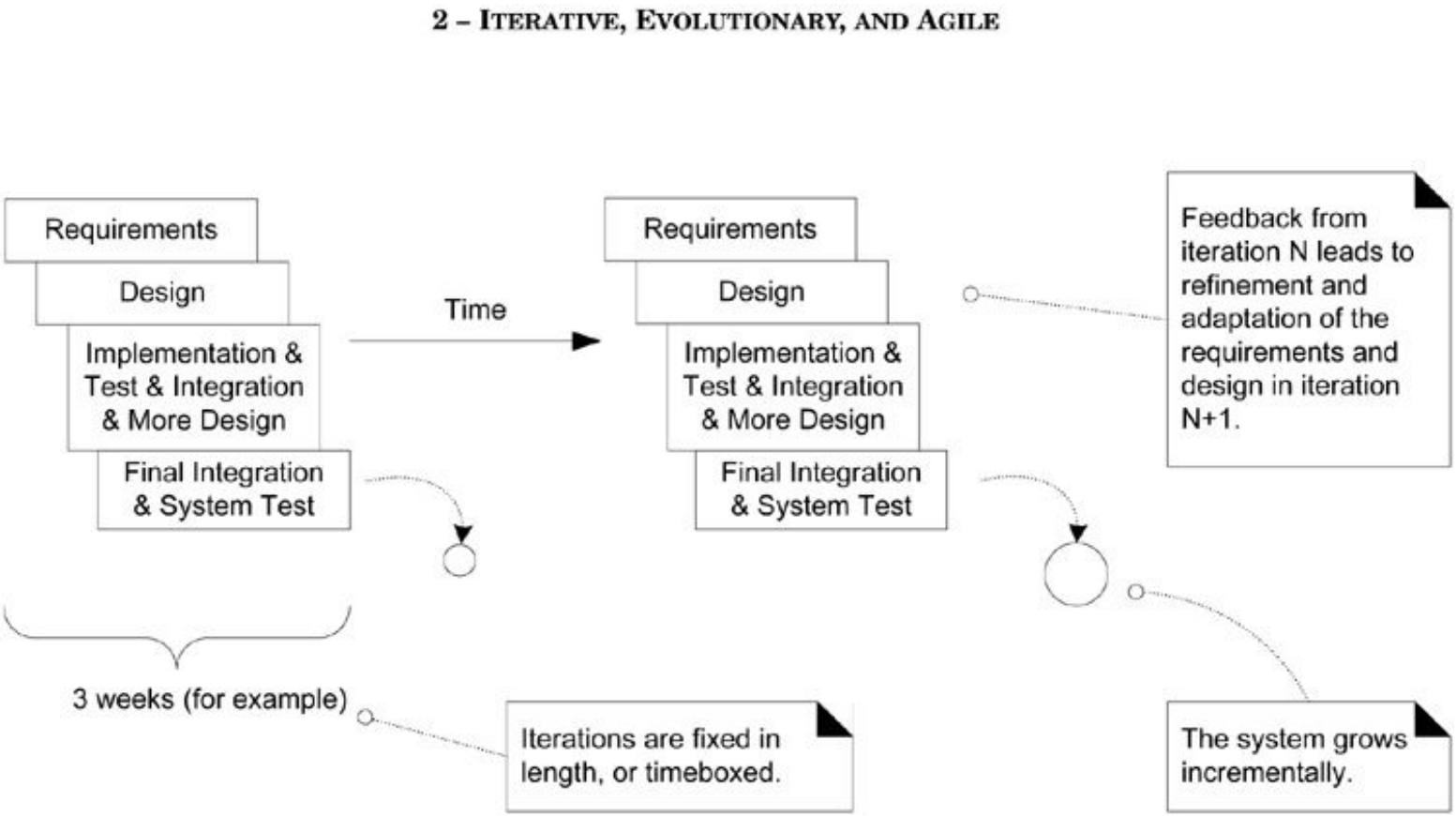


Figure: *Iterative and evolutionary development.*

What is the UP? (contd.)

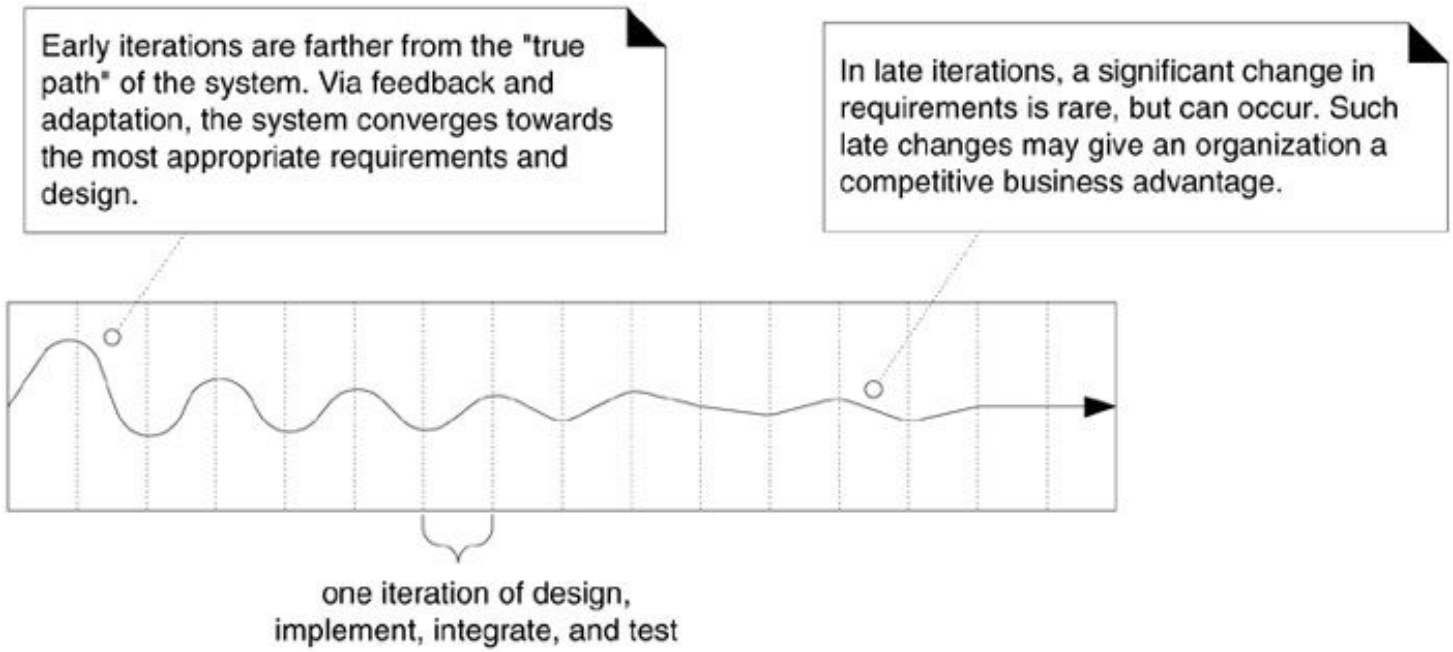


Figure: *Iterative feedback and evolution leads towards the desired system. The requirements and design instability lowers over time.*

Are There Benefits to Iterative Development?

- less project failure, better productivity, and lower defect rates; shown by research into iterative and evolutionary methods
- early rather than late mitigation of high risks (technical, requirements, objectives, usability, and so forth)
- early visible progress
- early feedback, user engagement, and adaptation, leading to a refined system that more closely meets the real needs of the stakeholders
- managed complexity; the team is not overwhelmed by "analysis paralysis" or very long and complex steps
- the learning within an iteration can be methodically used to improve the development process itself, iteration by iteration

How Long Should an Iteration Be? What is Iteration Timeboxing?

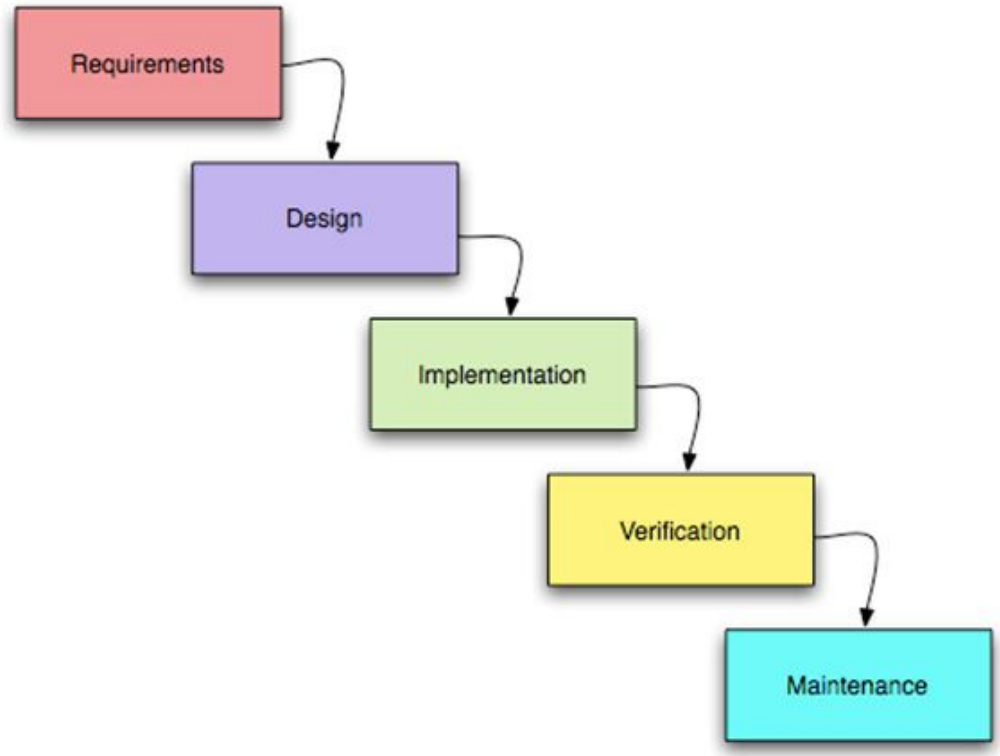
- Most iterative methods recommend an iteration length **between two and six weeks**.
- Small steps, rapid feedback, and adaptation are central ideas in iterative development; long iterations subvert the core motivation for iterative development and increase project risk.
- A key idea is that iterations are **timeboxed**, or fixed in length.

  For example,

  if the next iteration is chosen to be three weeks long, then the partial system must be integrated, tested, and stabilized by the scheduled date—date slippage is illegal.

What About the Waterfall Lifecycle?

- In a waterfall (or sequential) lifecycle process there is an attempt to define (in detail) all or most of the requirements before programming. And often, to create a thorough design before programming.



What About the Waterfall Lifecycle? (contd.)

- It is strongly associated with high rates of failure, lower productivity, and higher defect rates (than iterative projects).
  - On average, 45% of the features in waterfall requirements are never used, and
  - early waterfall schedules and estimates vary up to 400% from the final actuals.
- In hindsight, we now know that waterfall advice was based on speculation and hearsay, rather than evidence-based practices.
- In contrast, iterative and evolutionary practices are backed by evidencestudies show they are less failure prone, and associated with better productivity and defect rates.

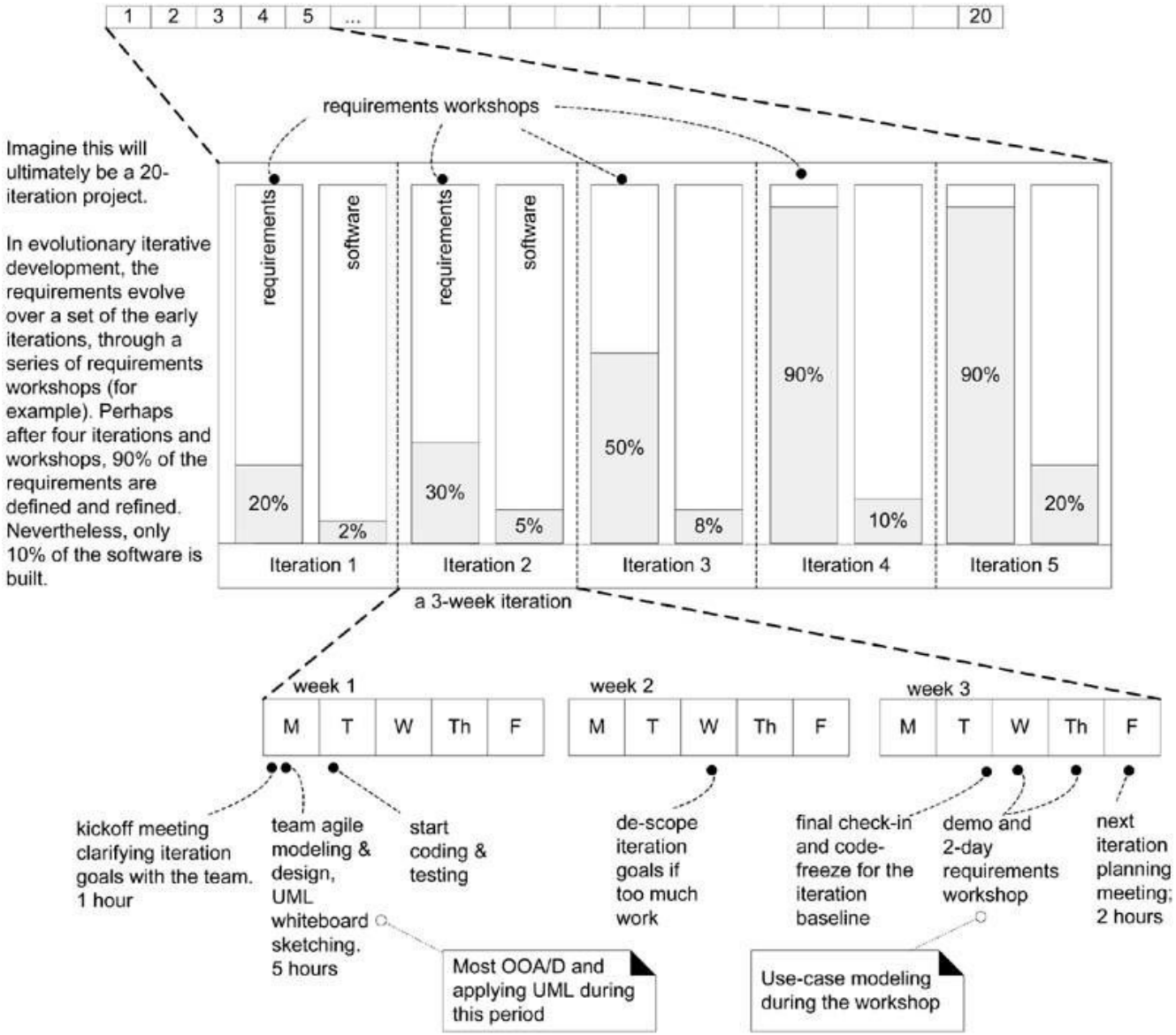
Why is the Waterfall so Failure-Prone? (contd.)

- Any analysis, modeling, development, or management practice based on the assumption that things are long-term stable (i.e., the waterfall) is fundamentally flawed.
- Change is the constant on software projects. Iterative and evolutionary methods assume and embrace change and adaptation of partial and evolving specifications, models, and plans based on feedback.

The Need for Feedback and Adaptation

- Feedback from early development, programmers trying to read specifications, and client demos to refine the requirements.
- Feedback from tests and developers to refine the design or models.
- Feedback from the progress of the team tackling early features to refine the schedule and estimates.
- Feedback from the client and marketplace to re-prioritize the features to tackle in the next iteration.

How to do Iterative and Evolutionary Analysis and Design?



Risk-Driven and Client-Driven Iterative Planning

- The UP (and most new methods) encourage a combination of **risk-driven** and **client-driven** iterative planning.
- This means that the goals of the early iterations are chosen to
  1. identify and drive down the highest risks, and
  2. build visible features that the client cares most about.
- **Risk-driven iterative development** includes more specifically the practice of **architecture-centric** iterative development, meaning that early iterations focus on building, testing, and stabilizing the core architecture. Because not having a solid architecture is a common high risk.

What are Agile Methods and Attitudes?

- Agile development methods usually apply
  - timeboxed iterative and evolutionary development,
  - employ adaptive planning,
  - promote incremental delivery, and
  - include other values and practices that encourage agility–rapid and flexible response to change.
- Agile methods can be defined as,
  - short timeboxed iterations with evolutionary refinement of plans, requirements, and design is a basic practice the methods share.
  - In addition, they promote practices and principles that reflect an agile sensibility of simplicity, lightness, communication, self-organizing teams, and more.

The Agile Manifesto

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

The Agile Principles

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale.

4. Business people and developers must work together daily throughout the project.

5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The Agile Principles

6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development.
9. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

The Agile Principles

10. Continuous attention to technical excellence and good design enhances agility.

11. Simplicity the art of maximizing the amount of work not done is essential.

12. The best architectures, requirements, and designs emerge from self-organizing teams.

13. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

What is Agile Modeling?

- The purpose of modeling (sketching UML, ...) is primarily to understand, not to document.
- The very act of modeling can and should provide a way to better understand the problem or solution space.
- The purpose of "doing UML" (which should really mean "doing OOA/D") is not for a designer to create many detailed UML diagrams that are handed off to a programmer, but rather to quickly explore (more quickly than with code) alternatives and the path to a good OO design.

What is Agile Modeling? (contd.)

- The purpose of modeling and models is primarily to support understanding and communication, not documentation.
- Don't model or apply the UML to all or most of the software design. Model and apply the UML for the smaller percentage of unusual, difficult, tricky parts of the design space.
- Use the simplest tool possible.

What is Agile Modeling? (contd.)

- Don't model alone, model in pairs (or triads) at the whiteboard, in the awareness that the purpose of modeling is to discover, understand, and share that understanding. Rotate the pen sketching across the members so that all participate.
- Create models in parallel. **For example**, on one whiteboard start sketching a dynamic-view UML interaction diagram, and on another whiteboard, start sketching the complementary static-view UML class diagram. Develop the two models (two views) together, switching back and forth.
- Use "good enough" simple notation while sketching with a pen on whiteboards. Exact UML details aren't important, as long as the modelers understand each other. Stick to simple, frequently used UML elements.

What is Agile Modeling? (contd.)

- Know that all models will be inaccurate, and the final code or design different - sometimes dramatically different than the model.
- Developers themselves should do the OO design modeling, for themselves, not to create diagrams that are given to other programmers to implement - an example of un-agile waterfall-oriented practices.

What is Agile UP?

- The UP was not meant by its creators to be heavy or un-agile, although its large optional set of activities and artifacts have understandably led some to that impression.
- Rather, it was meant to be adopted and applied in the spirit of adaptability and lightness - an agile UP.

What is Agile UP? (contd.)

- Some examples of how this applies:
  - Prefer a small set of UP activities and artifacts. Since the UP is iterative and evolutionary, requirements and designs are not completed before implementation. They adaptively emerge through a series of iterations, based on feedback.
  - Apply the UML with agile modeling practices
  - There isn't a detailed plan for the entire project. There is a high-level plan (called the Phase Plan) that estimates the project end date and other major milestones, but it does not detail the fine-grained steps to those milestones.
  - A detailed plan (called the Iteration Plan) only plans with greater detail one iteration in advance. Detailed planning is done adaptively from iteration to iteration.

Are There Other Critical UP Practices?

- tackle high-risk and high-value issues in early iterations
- continuously engage users for evaluation, feedback, and requirements
- build a cohesive, core architecture in early iterations
- continuously verify quality; test early, often, and realistically
- apply use cases where appropriate
- do some visual modeling (with the UML)
- carefully manage requirements
- practice change request and configuration management

What are the UP Phases?

A UP project organizes the work and iterations across four major phases:

- **Inception**—approximate vision, business case, scope, vague estimates.
- **Elaboration**—refined vision, iterative implementation of the core architecture, resolution of high risks, identification of most requirements and scope, more realistic estimates.
- **Construction**—iterative implementation of the remaining lower risk and easier elements, and preparation for deployment.
- **Transition**—beta tests, deployment.

The UP phases (contd.)

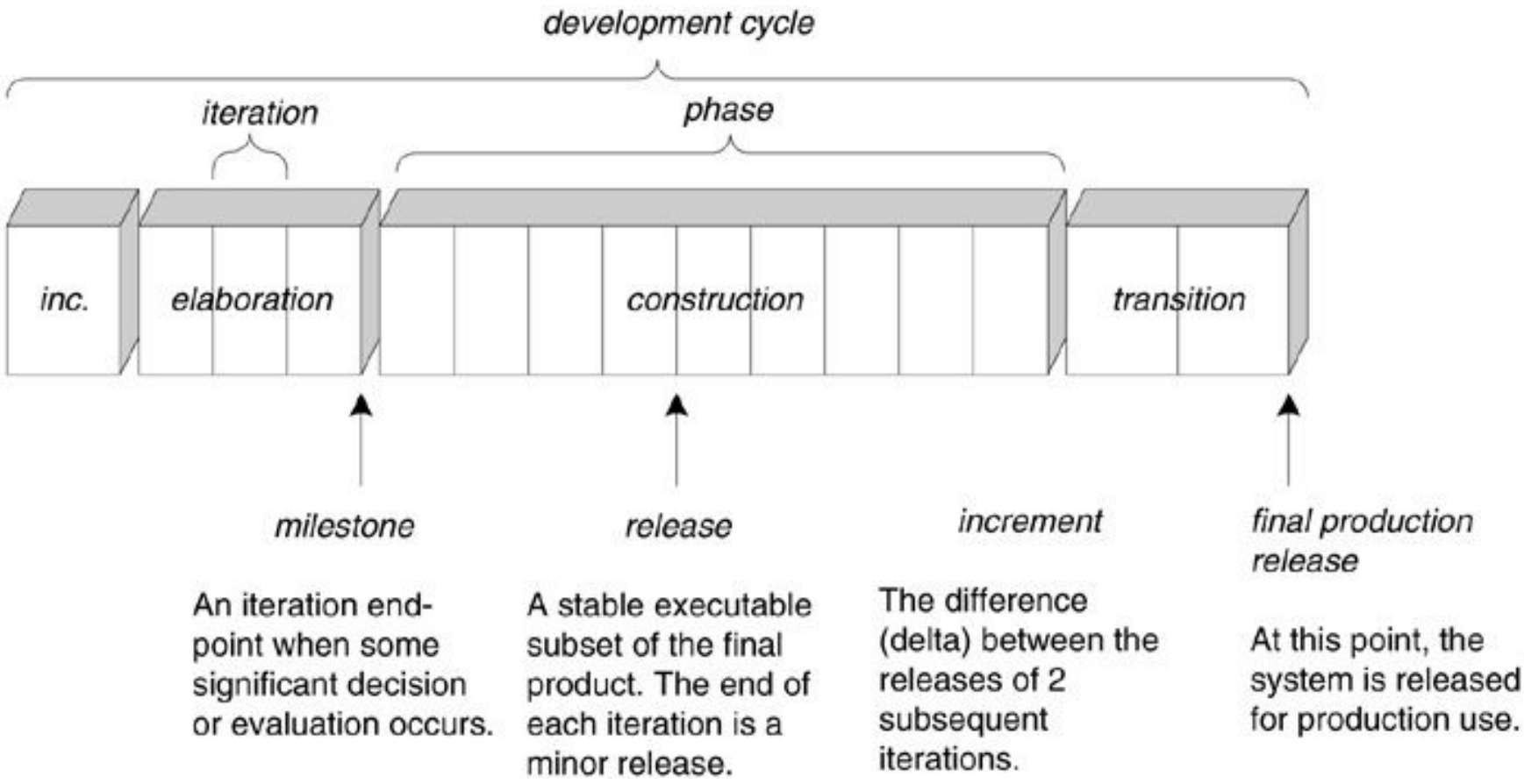


Figure: *Schedule-oriented terms in the UP.*

The UP phases (contd.)

- Inception is not a requirements phase; rather, it is a feasibility phase, where just enough investigation is done to support a decision to continue or stop.
- Similarly, elaboration is not the requirements or design phase; rather, it is a phase where the core architecture is iteratively implemented, and high-risk issues are mitigated.

The UP Disciplines

- The UP describes work activities, such as writing a use case, within disciplines—a set of activities in a subject area, such as the activities in the requirement analysis.
- In the UP, an artifact is the general term for any work product: code, Web graphics, database schema, text documents, diagrams, models, and so on.

The UP Disciplines (contd.)

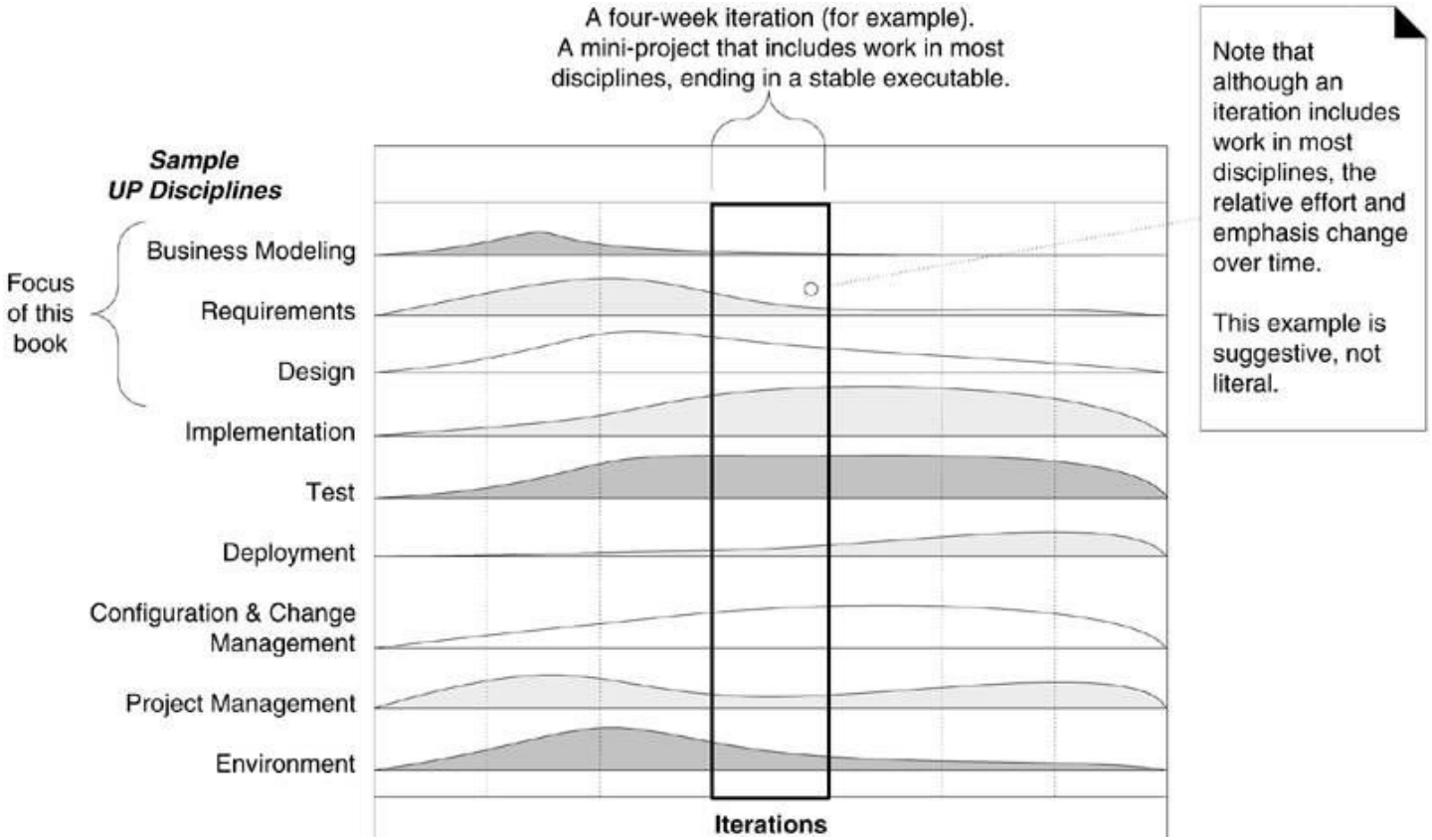


Figure: *UP disciplines*

Are There Optional Artifacts or Practices in the UP?

- Yes! Almost everything is optional. That said, some UP practices and principles are invariant, such as iterative and risk-driven development, and continuous verification of quality.
- However, a key insight into the UP is that all activities and artifacts (models, diagrams, documents, ...) are optional well, maybe not the code!

What is the Development Case?

- The choice of practices and UP artifacts for a project may be written up in a short document called the Development Case (an artifact in the Environment discipline).

What is the Development Case? (contd.)

Table 2.1. Sample Development Case. s - start; r - refine

Discipline	Practice	Artifact	Incep.	Elab.	Const.	Trans.
		Iteration →	I1	E1..En	C1..Cn	T1..T2
Business Modeling	agile modeling req. workshop	Domain Model		s		
Requirements	req. workshop vision box exercise dot voting	Use-Case Model	s	r		
		Vision	s	r		
		Supplementary Specification	s	r		
		Glossary	s	r		
Design	agile modeling test-driven dev.	Design Model		s	r	
		SW Architecture Document		s		
		Data Model		s	r	
Implementation	test-driven dev. pair programming continuous integration coding standards	...				
Project Management	agile PM daily Scrum meeting	...				
...						

Figure: *Sample Development Case. s - start; r - refine*

You Know You Didn't Understand Iterative Development or the UP When...

- You try to define most of the requirements before starting design or implementation. Similarly, you try to define most of the design before starting implementation; you try to fully define and commit to an architecture before iterative programming and testing.
- You spend days or weeks in UML modeling before programming, or you think UML diagramming and design activities are a time to fully and accurately define designs and models in great detail. And you regard programming as a simple mechanical translation of these into code.
- You think that inception = requirements, elaboration = design, and construction = implementation (that is, superimposing the waterfall on the UP).

You Know You Didn't Understand Iterative Development or the UP When... (contd.)

- You think that the purpose of elaboration is to fully and carefully define models, which are translated into code during construction.
- You believe that a suitable iteration length is three months long, rather than three weeks long.
- You think that adopting the UP means to do many of the possible activities and create many documents, and you think of or experience the UP as a formal, fussy process with many steps to be followed.
- You try to plan a project in detail from start to finish; you try to speculatively predict all the iterations, and what should happen in each one.

It's Quiz Time

1. In the iterative development or the UP, the purpose of elaboration is to fully and carefully define models, which are translated into code during construction. (True or False)
2. Agile processes promote sustainable development. (True or False)
3. Which one is not an UP phase?
   - 3.1 Transition
   - 3.2 Elaboration
   - 3.3 Construction
   - 3.4 Deployment