METHODOLOGIES

Citra N., MT

Objectives

- To introduce software process models
- To describe three generic process models and when they may be used
- To describe outline process models for requirements engineering, software development, testing and evolution

- Information System is a combination of people, hardware, software, communication devices, network and data resources that processes (can be storing, retrieving, transforming information) data and information for a specific purpose.
- The operation theory is just similar to any other system, which needs inputs from user (key in instructions and commands, typing, scanning). The inputted data then will be processed (calculating, reporting) using technology devices such as computers, and produce output (printing reports, displaying results) that will be sent to another user or other system via a network and a feedback method that controls the operation.

Definition

- A methodology is a description of the steps a development team should go through in order to produce a high-quality system.
- A methodology also describes what should be produced (documents, diagrams, code, etc.) and what form the products should take (for example, content, icons, coding style).
- At every stage, a methodology specifies what we should do next, so we're not left scratching our heads, thinking 'Okay, what now?'
- A methodology helps us to produce code that is more extensible (easier to change), more reusable (applicable to other problems) and easier to debug (because it has more documentation).

Definition

- A methodology is a systematic way of doing things. It is a repeatable process that we can follow from the earliest stages of software development through to the maintenance of an installed system. As well as the process, a methodology should specify what we're expected to produce as we follow the process (and what form the products should take).
- A methodology will also include advice or techniques for resource management, planning, scheduling and other management tasks.

Benefits

- Documentation: All methodologies promote thorough documentation of every stage of the development effort, so that the finished system is not an impenetrable monolith.
- Reduced latency: Since the workflows, activities, roles and inter-dependencies are better understood, there is less opportunity for human (and other) resources to lie idle for want of something to do.
- Improved chances of delivery on time and within budget.
- Better communication between users, sales people, managers and developers: A good methodology is based on logic and common sense, so it will be easy for all participants to grasp the basics; thus, we have a more orderly development, with less scope for misunderstanding and wasted effort.

A good methodology

- Planning: Deciding what needs to be done.
- Scheduling: Mapping out when things will be done.
- Resourcing: Estimating and acquiring the human, software, hardware and other resources that are needed.
- Workflows: The subprocesses within the wider development effort (for example, designing the system architecture, modeling the problem domain and planning the development effort).
- Activities: Individual tasks within a workflow, such as testing a component, drawing a class diagram or detailing a use case, too small or indefinable to be a workflow in their own right.
- Roles: The parts played by personnel within the methodology (developer, tester or sales person).
- Artifacts: The products of the development effort: pieces of software, design documents, training plans and manuals.
- Education: Deciding how to train personnel, if necessary, to fulfill their required roles; deciding how end users (staff, customers, sales people) will learn how to use the new system. For the purposes of this book, we won't be looking at the details of an industrial methodology

Phases in methodologies (In general)

Clasical phase
Requirement

- Requirements capture is about discovering what we're going to achieve with our new piece of software and has two aspects. Business modeling involves understanding the context in which our software will operate – if we don't understand the context, we have little chance of producing something to enhance that context.
- System requirements modeling (or functional specification) means deciding what capabilities the new software will have and writing down those capabilities. We need to be clear about what our software will do and what it won't do, so that the development doesn't veer off into irrelevant areas and we know both when we've finished and whether we've been successful.

Clasical phase
Analysis

- Analysis means understanding what we're dealing with. Before we can design a solution, we need to be clear about the relevant entities, their properties and their inter-relationships. We also need to be able to verify our understanding. This can involve customers and end users, since they're likely to be subject-matter experts.

Clasical phase
Design

- In the design phase, we work out how to solve the problem. In other words, we make decisions, based on experience, estimation and intuition, about what software we will write and how we will deploy it. System design breaks the system down into logical subsystems (processes) and physical subsystems (computers and networks), decides how machines will communicate, chooses the right technologies for the job, and so on.
- In subsystem design we decide how to cut each logical subsystem into effective, efficient and feasible code.

Clasical phase
Coding

- This is where we do writing pieces of code that work together to form subsystems, which in turn collaborate to form the whole system.
- Although we would expect most of the difficult coding decisions to have been made before we reach this phase (during design), there is still plenty of scope for creativity: although the public interfaces of our software components will have been well designed, specified and documented, programmers have free to decide on the inner workings. As long as the end result is effective and correct, everyone will be happy.

Clasical phase
Testing

- When our software is complete, it must be tested against the system requirements to see if it fits the original goals.
- As well as this kind of conformance testing, it's a good idea to see if our software can be broken via its external interfaces – this helps to protect us against accidental or malicious abuse of the system when it's been deployed.

Clasical phase Implementation

- In the implementation phase, we're concerned with getting the hardware and software to the end users, along with manuals and training materials.
- This may be a complex process, involving a gradual, planned transition from the old way of working to the new.

Clasical phase Maintenance

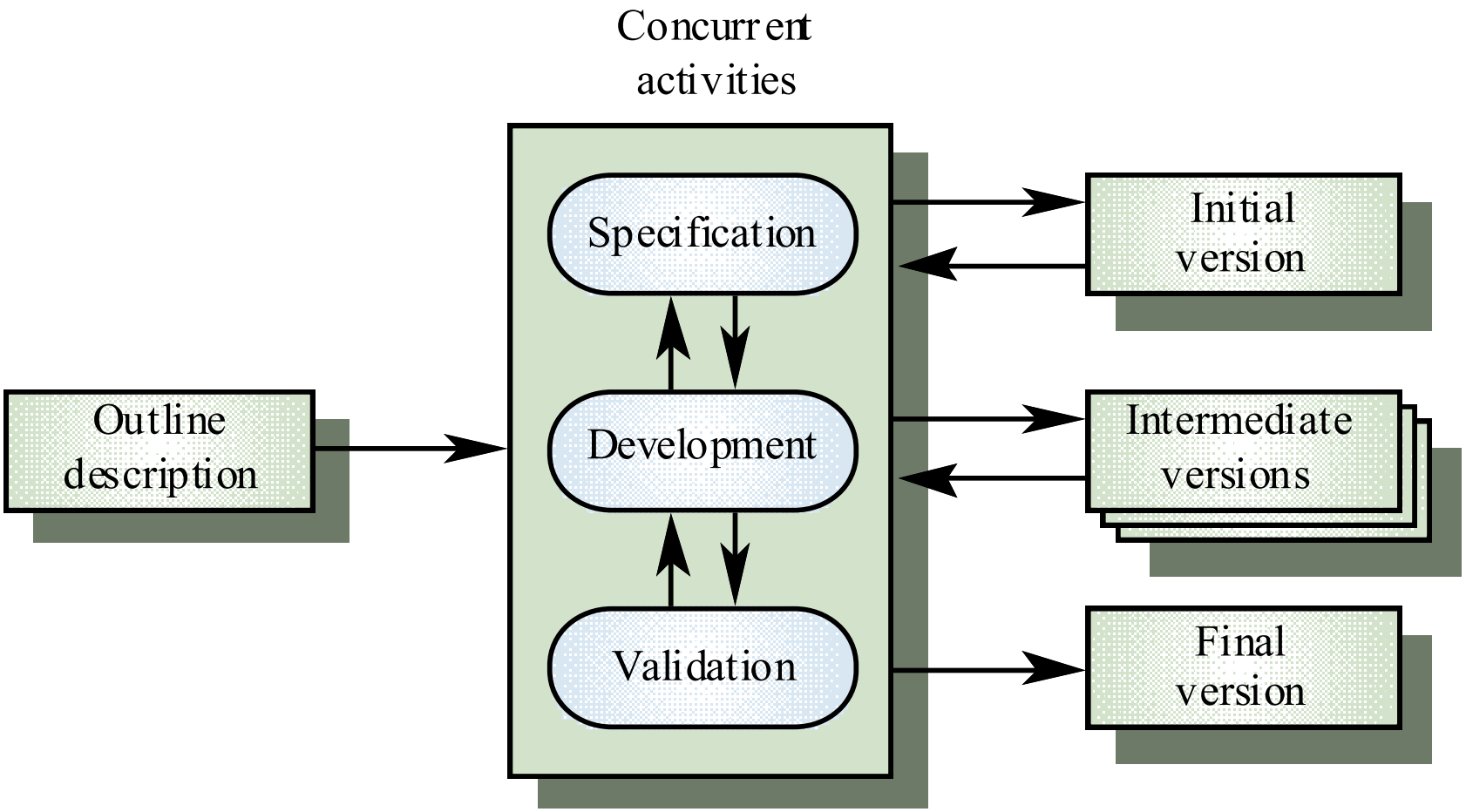
- When our system is implemented, it has only just been born. A long life stretches before it, during which it has to stand up to everyday use – this is where the real testing happens.
- As software developers, we're normally interested in maintenance because of the faults (bugs) that are found in our software. We must find the faults and remove them as quickly as possible, rolling out fixed versions of the software to keep the end users happy. As well as faults, our users may discover deficiencies (things that the system should do but doesn't) and extra requirements (things that would improve the system). From the business point of view, we would hope to fix and improve our software over time to maintain competitive advantage.

Kind of methodologies

(1) Evolutionary development

- Underlying idea
  - Give an initial implementation to the users and then refinine it through many versions based on user feedback
- Exploratory development
  - Objective is to work with customers and to evolve a final system from an initial outline specification. Should start with well-understood requirements
- Throw-away prototyping
  - Objective is to understand the system requirements. Should start with poorly understood requirements

Evolutionary development…



(2) Component-based software engineering

- Reuse occurs informally in almost all software projects
- Based on systematic reuse where systems are integrated from existing components or COTS (Commercial-off-the-shelf) systems.
- Process stages
  - Component analysis
  - Requirements modification
  - System design with reuse
  - Development and integration
- This approach is becoming increasingly used as component standards have emerged

Process iteration and hybrid models

- LARGE SYSTEMS need different approaches for different parts
- System requirements **always** evolve during a project
  - So process iteration where earlier stages are reworked is always part of the process for large systems
- Two (related) approaches
  - Incremental development
  - Spiral development

(3) Incremental development

- The development and delivery is broken down into increments with each increment delivering part of the required functionality
- User requirements are prioritised and the highest priority requirements are included in early increments
- Once the development of an increment is started, the requirements are frozen
  - But requirements for later increments can continue to evolve

(4) Extreme programming

- New approach to development based on the development and delivery of very small increments of functionality
- Relies on constant code improvement, user involvement in the development team and pairwise programming
- Good for small teams

Object oriented

OBJECT-ORIENTED METHODOLOGIES

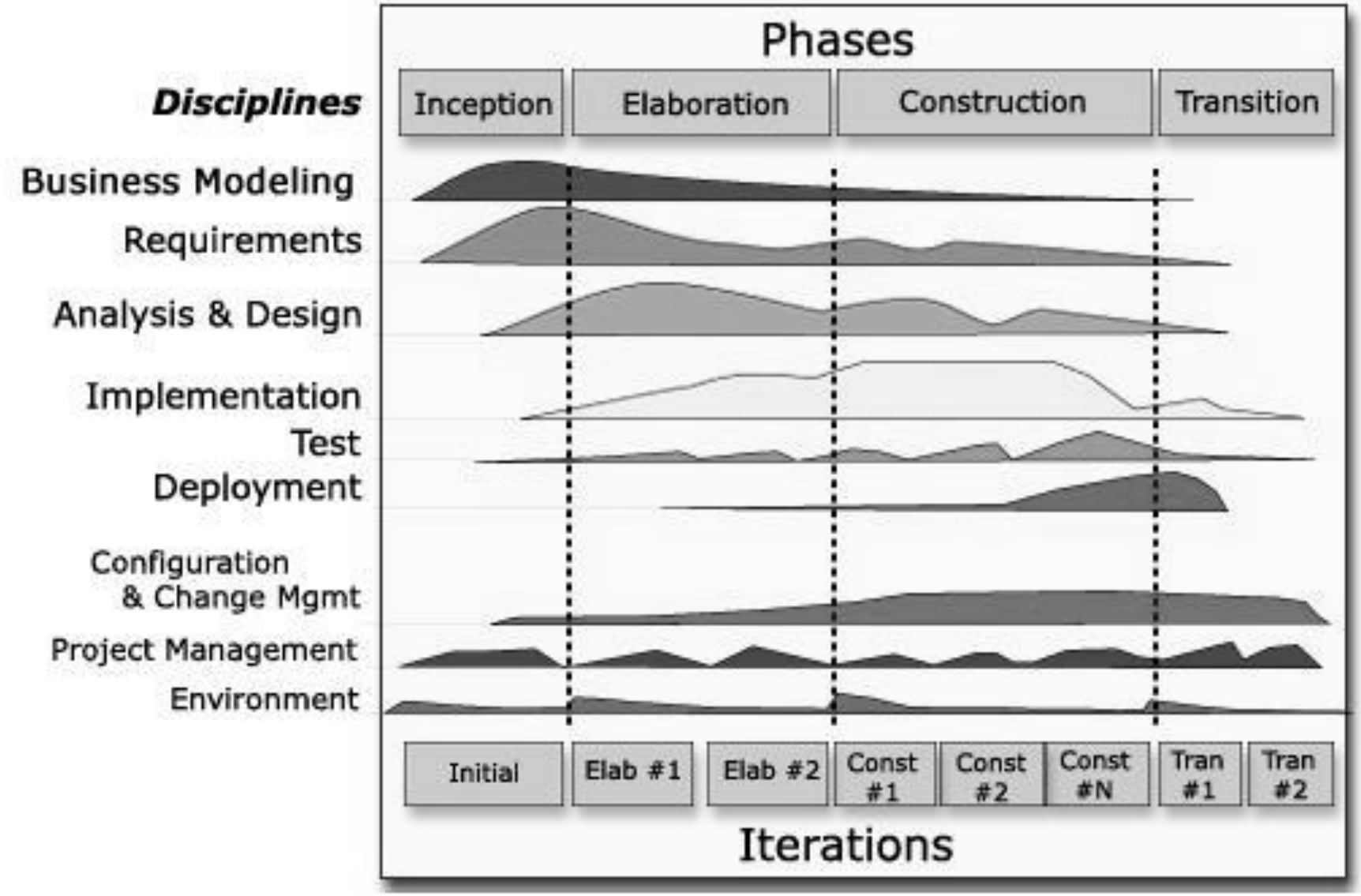
- Object-oriented methodologies tend not to be too prescriptive: the developers are given some choice about whether they use a particular type of diagram, for example. Therefore, the development team must select a methodology and agree which artifacts are to be produced, before they do any detailed planning or scheduling.

- When object-oriented programming was catching on, in the 1990s, developers invented object-oriented methodologies, better suited to an object-oriented programming style.
- These days, one of the market leading methodologies is the Rational Unified Process (RUP) [Jacobson et al. 99], owned by IBM. Roughly speaking, RUP is a convergence of Objectory, Booch and OMT.

UP Dimensions

- **First dimension** describe as horizontal represent dinamics aspects of software development, explaining about stages in UP and their *major milestone*. The stages are *Inception*, *Elaboration*, *Construction*, dan *Transition*.
- **Second dimension** describe as vertical represents statics aspects of software development, explaining main activity for each stages. Key are *who*, *what*, *how* dan *when*.

UP (Unified Process)



UP advantages

- *Improve productivity*
- *Deliver high quality system*
- *Lower maintenance cost*
- *Facilitate reuse*
- *Manage complexity*