MACIASZEK, L.A. (2007):
Requirements Analysis and System Design, 3rd ed.
Addison Wesley, Harlow England
ISBN 978-0-321-44036-5

Chapter 1
Software Process

Topics

- The nature of software development
- System planning
- Systems for three management levels
- The software development lifecycle
- Development models and methods
- Problem statements for case studies (separate set of slides)

4. The software development lifecycle

- strategic
- tactical
- operational

Software development lifecycle

- The **lifecycle** identifies:
  - the applied modeling approach
  - the exact phases along which the software product is transformed – from initial inception to phasing it out
  - the method (methodology) and associated development process

Development approach

- Software has become much more **interactive**
  - event-driven
  - user in control
  - software objects service random and unpredictable events
- Conventional software has been well served by the so-called **structured approach**
- Modern interactive GUI systems require object programming and the **object approach** is the best way to design such systems

Structured approach

- Popularized in 1980s
- Based on:
  - DFD (data flow diagrams) for process modeling
  - ERD (entity relationship diagrams) for data modeling
- *Process-centric*
  - Brakes system down to manageable units in the activity called *functional decomposition*
- Not well aligned with modern software engineering:
  - *sequential and transformational* approach rather than iterative and incremental
  - tends to deliver inflexible solutions that satisfy the set of identified business functions
  - assumes development from scratch, and it does not support the reuse of pre-existing components

Object-oriented approach

- Popular since 1990s
- *Data-centric* – it evolves around class models
  - But the growing significance of *use cases* in UML shifts the emphasis slightly from data to processes
- Matches the *event-driven* programming demanded by interactive GUI-based applications
- Addresses the needs of *emerging applications*, such as *workgroup computing* and *multimedia systems*
- Good at fighting *application backlogs* using *object wrapping* and similar techniques
- Aligns with the *iterative and incremental* process

Problems related to OO development

- The *semantic gap* between the object-oriented modeling artifacts and the implementation of the data-centric artifacts with relational database technology can be significant
- *Project management* is more difficult
  - In object-oriented development through “elaboration” there are no clear boundaries between phases, and project documentation evolves continuously
- Object solutions are significantly more *complex* than old-style structured systems
  - The *complexity* results from the need for extensive inter-object communication (”complexity in the wires”)
  - Good *architectural design* a necessity

Lifecycle phases

- Business Analysis
  - functional and non-functional requirements
- System Design
  - architectural design
  - detailed design
- Implementation
  - coding
  - round-trip engineering
- Integration and Deployment
- Operation and Maintenance

Business analysis

- Also **requirements analysis**
- Activity of determining and specifying customer requirements
  - business analyst determines requirements
  - system analyst specifies (or models) requirements
- Business analysis is linked to **business process reengineering** (BPR)
  - aim of BPR is to propose new ways of conducting business and gaining competitive advantage
- Business analysis becomes increasingly an act of **requirements engineering**

Requirements determination

- **Requirement** – ‘a statement of a system service or constraint’
- **Service statement**
  - a business rule that must be obeyed at all times (e.g. ‘fortnightly salaries are paid on Wednesdays’)
  - a computation that the system must carry out (e.g. ‘calculate salesperson commission based on the sales in the last fortnight using a particular formula’)
- **Constraint statement**
  - a restriction on the system’s behavior (‘only direct managers can see the salary information of their staff’)
  - a restriction on the system’s development (‘we must use Sybase development tools’)

Requirements specification

- Begins when the developers start modeling the requirements using a particular method (such as UML)
  - CASE tool is used to enter, analyze and document the models
  - Requirements Document is enriched with graphical models and CASE-generated reports
    - Specifications document (the specs in the jargon) replaces the requirements document
- Most important specification techniques
  - class diagrams
  - use case diagrams
- Ideally, the specification models should be independent from the hardware/software platform on which the system is to be deployed

Architectural design

- The description of the system in terms of its modules (components)
- Concerned with
  - selection of a solution strategy
    - to resolve client (user interface) and server (database) issues as well as any middleware needed to ‘glue’ client and server processes
  - modularization of the system
    - relatively independent from a solution strategy but the detailed design of components must conform to a selected client/server solution
- Client/server models are frequently extended to provide a three-tier architecture where application logic constitutes a separate layer
- Good architectural design produces adaptive (supportable) systems, i.e. systems that are understandable, maintainable, and scalable (extensible)

Detailed design

- Description of the internal workings of each software component
- Develops detailed algorithms and data structures for each component
- Dependent on the underlying implementation platform
  - client
  - server
  - middleware

Implementation

- Involves
  - installation of purchased software
  - coding of custom-written software
  - other important activities, such as loading of test and production databases, testing, user training, hardware issues, etc.
- Client programs
- Server programs

Integration and deployment

- **Module integration** can take more time and effort than any one of the earlier lifecycle phases, including implementation
  - 'The whole is more than the sum of the parts' (Aristotle)
  - must be carefully planned from the very beginning of the software lifecycle
- **Application integration** – integrating disparate (*stovepipe*) applications into a unified enterprise application through which all data and processes are shared
- **Deployment** must be carefully managed and allow, if at all possible, fallback to the old solution, if problems encountered

Operation and maintenance

- **Operation** signifies change over from the existing business solution, whether in software or not
- **Maintenance** is not only an inherent part of the software lifecycle – it accounts for most of it as far as IT personnel time and effort is concerned
  - Housekeeping
  - Adaptive maintenance
  - Perfective maintenance
- **Phasing out** would normally happen due to reasons that have little to do with the usefulness of the software

Activities spanning the lifecycle

- Project planning
- Metrics
- Testing

Project planning

- If you can’t plan it, you can’t do it
- Activity of estimating the project’s deliverables, costs, time, risks, milestones, and resource requirements
- Includes the selection of development methods, processes, tools, standards, team organization, etc.
- A moving target
- Typical constraints are time and money

Metrics

- Measuring development time and effort
- Without measuring the past, the organization is not able to plan accurately for the future
- Metrics are usually discussed in the context of software quality and complexity – they apply to the quality and complexity of the software product
- Equally important application of metrics is measuring the development models (development products) at different phases of the lifecycle → to assess the effectiveness of the process and to improve the quality of work at various lifecycle phases

Testing

- **Test cases** should be defined for each functional module (use case) described in the requirements document
- Desktop testing by developers not sufficient
- Methodical testing by Software Quality Assurance (SQA) group necessary
- Requirements, specifications and any documents (including program source code) can be tested in formal reviews (so-called walkthroughs and inspections)
- Execution-based testing:
  - Testing to specs (black-box testing)
  - Testing to code (white-box or glass-box testing)

Review Quiz 1.4

1. Which software development approach, structured or object-oriented, takes advantage of the activity of functional decomposition?
2. What is another name for business analysis?
3. Which development phase is largely responsible for producing/delivering an adaptive system?
4. The notion of a stub is associated with what development phase?
5. Which activities span the development lifecycle and are not, therefore, distinct lifecycle phases?