Chapter 1

Software Projects

© Pearson Education Limited 2005

Topics

- Software projects
- Software development lifecycle
- Software engineering quintessence
- Lifecycle phases – from requirements to maintenance

- Lifecycle models
  - Waterfall
  - Iterative
The theme

“Software engineering is the field of computer science that deals with the building of software systems that are so large or so complex that they are built by a team or teams of engineers.”

Ghezzi et al.

“Enterprise applications often have complex data – and lots of it – to work on, together with business rules that fail all tests of logical reasoning.”

Martin Fowler

The Standish Group’s “Chaos Report”

• in 1994 reported that only 16.2% of software projects were completed on-time and on-budget
  – 9% for large and complex projects
• in 2003, 34% of projects completed on-time and on-budget

Although the improvement is significant, it is dismal when compared with traditional engineering disciplines, such as architecture or electrical engineering.

Software development

■ Software project
  • planned undertaking intended to deliver software product or service
  • completed over a period of time

■ Software lifecycle
  • the changes that happen in the “life” of a software product
  • lifecycle phases – identifiable phases between the product’s “birth” and its eventual “death”

■ Iterative and incremental development
  • the product is phased in – introduced in stages

■ Stepwise retirement
  • the product is phased out – gradually stopped being used
Lifecycle phases

Quintessence of SE

- enterprise information system is more than a software system
- software process is part of business process
- software engineering is different from traditional engineering
- software engineering is more than programming
- software engineering is about modeling
- enterprise information systems are complex
Software system is less than EIS

- Information system is a social system

Software process is part of business process

- The result of a software process is software
  - efficiency
- The result of a business process is business
  - effectiveness

Strategic management

Knowledge Systems

Knowledge Bases

OLAP (OnLine Analytical Processing) Systems

OLTP (OnLine Transaction Processing) Systems

Data Warehouses

Databases
Software is immaterial
- classical mathematical models apply to some but not to all aspects of software
- software is defined in fuzzy terms – “good”, “bad”, “acceptable”, “satisfying user requirements”, etc. (as in the service sector)

Software engineering may be tackling “fuzzy” problems but this does not mean that it must be less rigorous or not provable
- rigor is not the same as formality
- a software process may be rigorous even though not formally proven by mathematical laws

Software must be supportable

(Application) software is not manufactured – it is implemented (to fit its environment)

Software must be supportable

(Application) software is not manufactured – it is implemented (to fit its environment)

Software engineering applies to complex problems that cannot be solved by programming alone
- Complex systems must be designed before they can be programmed

Before a system can be designed, a software engineer must understand its requirements

Software engineer integrates components, programmers codes them

Software engineering is a team activity
- Teams must be managed
**SE is about modeling**

- **Models** are abstractions from reality
- Abstraction applies to software products and to software processes
  - **Software process model** is an abstract representation of the software process
  - **Software product model** is an abstract representation of a software product
- The **functional paradigm** breaks a complex system down to manageable units using the approach known as functional decomposition
- The **object-oriented paradigm** breaks a system down into packages/components of classes linked by various relationships

**Software system is complex**

- Hierarchical structures reduce complexity (Herb Simon, 1962)
  - complex – made up of a large number of parts that interact in a non-simple way
- A structure is stable if cohesion is strong and coupling low (Larry Constantine, 1974)
  - cohesion – intra-module communication
  - coupling – inter-module interaction
- Only what is hidden can be changed without risk (David Parnas, 1972)
- Separation of concerns leads to standard architectures (Ernst Denert, 1991)
- An evolving system increases its complexity unless work is done to reduce it (Meir Lehman)
Size and complexity

- **Legacy systems**
  - Monolithic, processing sequential and predictable
  - Complexity = size

- **Object systems**
  - Distributed, processing random and unpredictable
  - Complexity in wires
    - “cost of glue code is three times cost of application code” (Endres, Rombach, 2003)

Object systems → new legacy systems?

- **Unsupportable system → legacy system**
  - software systems do not wear out; they only lose relevance

- **Supportability = understandability + maintainability + scalability**

- **Properties of complex systems that are supportable:**
  - Take the form of hierarchy and composition of objects
  - Intra-linkages of components stronger than inter-linkages
  - Dynamic links legalized as static associations
  - Complex systems that work are result of simple systems that worked (evolution)
  - “Evolution has a preference for hierarchical systems because they are more stable when interrupted” (Endres, Rombach, 2003)
Lifecycle phases

- requirements analysis
- system design
- implementation
- integration and deployment
- operation and maintenance

Requirements analysis

“User requirements are statements, in a natural language plus diagrams, of what services the system is expected to provide and the constraints under which it must operate” (Sommerville)

- Requirements determination – one of the greatest challenges
- Requirements specification – Unified Modeling Language (UML)
- Computer Assisted Software Engineering (CASE)
- Requirements document
  - system services (what the system does)
  - system constraints (how the system is constrained)
- Software Quality Assurance (SQA)
  - walkthroughs and inspections
System design

- **A software design** is a description of the structure of the software to be implemented, the data which is part of the system, the interfaces between system components and, sometimes, the algorithms used." (Sommerville)
- In practice the distinction between analysis and design is blurred
  - lifecycle models are iterative with increments
  - the same modeling language (UML) is used for analysis and design

  - **Detailed design**
  - **Architectural design**
  - **Traceability management**

Implementation

- **Implementation** is mostly programming, but → the designs are “underspecified” → extra designing before coding
- A programmer is a **component engineer**
- A programmer is a **roundtrip engineer**
  - **Integrated Development Environments** (IDEs)
- **Debugging**
- **Testing**
  - code reviews (walkthroughs and inspection)
  - execution-based
    - testing to specs (black-box testing)
    - testing to code (white-box testing)
Integration and deployment

Integration assembles the application from the set of components previously implemented and tested
• difficult to disassociate it from
  – implementation (continuous integration of the agile development)
  – testing (integration testing)
• driven by the architectural design of the system

Deployment is the handing over of the system to customers for production use
• software is deployed in releases
• system testing and acceptance testing prior to release
• user training
• user documentation

Testing

Stub – a piece of code that simulates the behavior of yet-to-be-developed component
• in top-down testing
  • circular dependencies a problem, but big-bang testing not a solution

Driver – a piece of code to “drive” the integration so that the integrated increment (build) gets the data and other context that would be normally given to it by yet-to-be-developed component
• in bottom-up testing

Test harness or test scaffolding – supporting test software consisting of stubs and drivers (useful only during integration)
**Operation and maintenance**

- **Operation** signifies the lifecycle phase when the software product is used in day-to-day operations and the previous system (manual or automated) is phased out.
- Operation coincides with the start of product **maintenance**
  - corrective (housekeeping)
  - adaptive
  - perfective

**Legacy systems**

**Lifecycle differs between projects**

- software engineering experience, skills and knowledge of the development team
- business experience and knowledge
- kind of application domain
- business environment changes
- internal business changes
- project size
Waterfall lifecycle with feedback

Waterfall with feedback, overlaps, and prototypes
Iterative lifecycle with increments

- **Iteration** in software development is a repetition of some process with an objective to enrich the software product.
- Iterative lifecycle assumes **increments** – an improved or extended version of the product at the end of each iteration.
- Iterative lifecycle assumes **builds** – executable code that is a deliverable of an iteration.
- Iterative lifecycle assumes **short iterations** between increments, in weeks or days, not months.

**Models:**
- spiral
- IBM Rational Unified Process (RUP)
- Model Driven Architecture (MDA)
- Agile lifecycle with short cycles
**Spiral model**

- Initial requirements gathering and project planning
- Planning following customer evaluation
- Risk analysis based on initial requirements
- Risk analysis based on customer reaction
- "go, no-go" decision
- Iteration 1 build
- Iteration 2 build
- Customer evaluation
- Engineering
- Project cost

**Rational Unified Process (RUP)**

- Requirements
- Planning
- Implementation
- Test
- Evaluation
- Business Modeling
- Initial Planning
- Analysis & Design
- Management
- Environment
- Configuration Management
- Deployment
**Model Driven Architecture (MDA)**

The MDA process involves the following steps:

1. **Requirements**
2. **Analysis**
3. **Design**
4. **Coding**
5. **Testing**

The outputs of each step are:

- **Platform Independent Model**
- **Platform Specific Model**
- **PSM bridge**
- **Code bridge**

**User** → **Product** → **Deployment**

**Agile lifecycle with short cycles**

The agile lifecycle includes:

- **User Stories**
- **Acceptance Tests**
- **Refactoring**
- **Test-Driven Development**
- **Continuous Integration**

**Iteration 1** Product

**Iteration 2** Product

**Operation and Maintenance**
13 XP practices

- whole team (no barriers between all stakeholders)
- metaphor (common language; “desktop metaphor”)
- the planning game (user stories)
- simple design
- small releases (two-week iterations)
- customer tests (acceptance tests)
- pair programming
- test-driven development (automated test first)
- design improvement (refactoring)
- collective code ownership
- continuous integration (builds many times per day)
- sustainable pace (no overtime)
- coding standards

Summary

- Software engineering is concerned with development of large software systems.
- The stages of software development process are referred to as software lifecycle phases.
- Software process is part of a business process.
- The immaterial and changeable nature of software are but two factors that make software engineering different from traditional engineering.
- Software engineering is more than programming.
- Software engineering is about modeling.
- The lifecycle phases are requirements analysis, system design, implementation, integration and deployment, and operation and maintenance.
- The lifecycle models are broadly divided into the waterfall models with feedback and the iterative models with increments.
- Waterfall models are not suitable for modern software production.
- There are four main representatives of iterative models: spiral, Rational Unified Process (RUP), Model Driven Architecture (MDA), and the agile model.