Systems Engineering

Chapter 2:

Software Systems Engineering

Outline

§ 1. Nature of Software
§ 2. Concept Development
§ 3. Engineering Development

§1. Nature of Software

- System software
  - Services for other software
  - Operating systems and middleware
- Embedded software
  - Resides within a larger system and implements specific functions
  - Satellites, defense systems, energy systems
- Application software
  - Stand-alone program that solves a specific need
  - Text editors, media players

Types of Software Systems

- Software-embedded systems
  - Hybrid combination of hardware, software, and people
  - Hardware performs principal actions, software is supporting
  - Vehicles, computer-controlled manufacturing machinery
- Software-intensive systems
  - Network of computers and users (any information system)
  - Software on computers perform virtually all of the system functionality, in support of human operators
  - WWW is an extreme case
- Data-intensive computing systems
  - Large-scale computing resources
  - Supercomputers, parallel computing
  - Weather analysis

Sources of Software

1. Outsourcing
2. IT service companies
3. Packaged software producers
4. Vendors of enterprise-wide solution software
5. Cloud computing
6. Open source
7. In-House

Outsourcing

- Practice when one organization develops or runs a computer application for another organization
- Extreme 1: a company develops and runs your application on its computers
  - You only supply input and take output
  - Payroll systems
- Extreme 2: a hired company runs your applications at your site on your computers
- Big business
  - Cost-effective solution
  - India, China, Latin America
  - Nearshoring: no more than one time zone away

IT service companies

- Development of custom information systems for internal use
- Your company
  - needs an information system, but
  - does not have the expertise/personnel to develop
- IBM, HP

Packaged software producers

- Packaged software aims at many market segments
  - General, broad-based packages
  - Narrow, niche packages
- Software runs on many different platforms
- Software is difficult for modification to meet specific needs of a particular organization
  - Typical rule: 70% need satisfaction
- Microsoft

Enterprise-wide solutions

- Complete software solutions that support all operations and business processes of the organization
- Enterprise resource planning (ERP) systems
  - All parts of a business process is integrated in a unified information system
- Integrated modules:
  - Each supports an individual business function, e.g., accounting, manufacturing, or human resources
  - Single repository of data for all aspects of business processes
- Oracle, SAP, 1C

Cloud computing

- Renting (licensing) applications from third-party providers who run the applications at remote sites
  - Access to the applications through the Internet (or virtual private networks)
  - Apps provider buys, installs, maintains, and upgrades the applications
  - Users pay on a per-use basis or they license the software for fixed period
  - Users do not have to invest in the hardware and software resources
- Benefits for your company:
  - Freeing internal IT staff
  - Gaining access to applications faster than via internal development
  - Achieving lower-cost access to corporate-quality applications
- Concerns: reliability, security, compliance with government regulations
- Google apps

Open source

- Not just a final product but the source code itself
- Development by community, not by a particular company
- Money made with open source
  - Provision of maintenance and other services
  - One version is free, a featured version is commercial
- Examples:
  - Linux (~15% servers, ~2% desktops), mySQL, Firefox
  - SourceForge.net

In-house

- Development of an information system from scratch for internal corporate use
- Now, a small fraction of efforts of internal IT departments
- Higher maintenance burden than other development options
- Hybrid solutions:
  in-house + purchased components

Software Reuse

- The use of previously developed software resources in new applications
- Great savings: Less code writing and testing
  - Object-oriented development
  - Component-based development
  - Corporate code repository
- Approaches: Table 2-3 from [Hoffer et.al 2011], p.42
  - Ad hoc: individual driven
  - Facilitated: encouraged by organization
  - Managed: mandated by organization
  - Designed: development of reusable code

Software Development Life Cycle Models

- Fundamental phases
- Linear model
- Incremental model
- Evolutionary model
- Agile development

§2. Concept Development

- Activities that define the requirements and architecture of the software elements of the system
- Analysis and Design
- Demarcation line between analysis and design may vary substantially

Needs Analysis

- A feasible development approach is available
- The system is worth the effort to develop and produce it
- Software-intensive systems:
  - Software automates functions that have been performed by people or hardware
  - At less cost, in less time, and more accurately
- Tradeoff the gains in performance and costs against the effort to develop and deploy the new system
- Extensive market analysis

Requirements Analysis

- Software-Embedded system components
  - Component-aware requirements at the system and subsystem levels
  - Allocation to computer system configuration items
- Software-intensive systems
  - Formulation of the overall system requirements is subject to analysis and participation by software systems engineers
  - Customers with little understanding of what software automation is capable of doing
  - Prototyping, RAD, evolutionary development – an early version of the system for experimentation by users

Requirements Generation Process

- Iterations of 4 critical steps:
  - Elicitation (language barrier between users and developers)
  - Analysis and Negotiation (necessity, consistency, completeness, feasibility)
  - Documentation
  - Validation (every requirement is consistent, coherent, unambiguous)
- Use cases (understanding of sequence of events and activities that need to be performed)
- Interface requirements (association of each input and output with requirements)

System Architecture

- Partition into relatively independent subsystems that may be designed, developed, produced, and tested as separate system building blocks
- Achievement of a high degree of "Modularity"
- Binding: **loose** and **tight** (grouping of closely related blocks)
- Coupling: **tight** and **loose** (interactions between blocks are minimized)
- Architecture modeling
  - Structured analysis and design
  - Object-oriented analysis and design

Structured Analysis and Design

- Functional flow block diagram (FFBD)
- Data flow diagram (DFD)
- Entity relation diagram (ERD)
- State transition diagram (STD)
- Data dictionary

Object-Oriented Analysis and Design

- Class encapsulates data and functions that operate on them
- Self-contained, robust, reusable program building blocks
- Arranging related classes into groups (subsystems or packages)
- Defining all of the relations/responsibilities within and among the groups
- UML: unified modeling language

§3. Engineering Development

- Coding and unit testing
- Software integration and test
- Software engineering management

Program Code

- Code and program structure
  - Reflection of the architecture
  - Commenting
  - Self-documentation
  - Self-testing
  - Accompanied docs (README, CHANGELOG, NEWS, …)
- Programming languages
  - Fourth-generation languages (4GL): coupled with database and structured query language (SQL)
  - Special-purpose languages: mimic the problem domain where possible

Programming Support Tools

- Editors
- Debuggers
- Compilers
- Linkers and loaders
- IDE, SDK
- Prototyping tools

Software product design

- Hardware: transformation of development prototype into reliable, maintainable, and producible units
- Software: no "production" process
- Software product = usable by others
- Critical characteristics:
  - Maintainability
  - User interface
  - Performance and other non-functional requirements
- TIME(Software product) ~ 3*TIME(working program)

Unit Testing

- Focused on individual software components
- By programmers themselves
- "white box" tests
  - Known internal details
  - Exercise critical parts
- Automation (tools available in many IDEs)
- Self-testing (built into the code)
- Regularly running

Software Integration and Test

- A large fraction of the entire development effort
- Verification
  - Process of determining whether the software implements the functionality and features correctly and accurately
- Validation
  - Process of determining whether the software satisfies the users' or customers' needs
  - Whether we implemented the right product
- Testing is a primary method for verification and validation

Integration Testing

- Performed on a partially assembled system
- System components are progressively linked together
- "Black box" tests
- Regression testing
  - Repeating a selected fraction of tests to ensure the discovery of newly created discrepancies
  - Careful selectivity of the test cases to be repeated

System Testing

- Validation tests
- Alpha testing
  - In a controlled environment at the developer's side
- Beta testing
  - At a customer's side, without the developers

Software Engineering Management

- CASE tools
- Requirements management tools
- Software metrics tools
- Integrated development support tools

Software Configuration Management (CM)

Difference between hardware and software

- Software abstractness and lack of well-defined components makes it difficult to understand
- Software has more interfaces; their penetration is deeper and hence is difficult to trace
- Any change may propagate deep into the system
- Any change may require retesting the whole system
- When a software system fails, it often breaks down abruptly
- The flexibility of software renders making a software change deceptively easy

Quantitative Measurements

- Project metrics
  - Success of project management
  - Human resources
- Process metrics
  - Correspondence to process model/standard
  - Capability maturity assessment
  - Capability Maturity Model (CMM): 5 maturity levels
- Technical metrics
  - Assessing quality of the product
  - Code, Documentation, Testing

Materials for seminar

- Presentation on recent models of software development
- Family of agile development methods
  - Extreme programming
  - Scrum
  - …