Introduction to Software Development Methods

Prof. Robert B. France
Dept. of Computer Science
Colorado State University

Evolution of Programming

- Fifties-early sixties: Early programming concerned with putting together instructions to solve problems with well-known solutions
  - Focus on expressing solution in programming language
  - Programs written by end-users
- Sixties: Growth in demand for software led to users delegating programming tasks to programmers
  - Focus on programming-in-the-small concerns
- Late sixties onwards: Increasing complexity of computer-based systems led to team-based development of software
  - Focus on programming-in-the-large concerns
- 2000 onwards: Increasing complexity, pervasiveness, and criticality of software systems leads to calls for new generation of software development techniques.
  - Focus on mega-programming concerns and use of domain-specific techniques.

Custom vs. Generic Software

- Custom software
  - Software developed to meet specific needs of an organization
  - In-house development: software developed by an organization’s IT team
  - Contract development: software developed by a contractor
  - Examples: space-shuttle software, flight management systems, air-traffic control systems, websites
- Generic software
  - Developed to be sold on the open market
  - Perform functions that many people need.
  - Sometimes referred to as COTS or shrink-wrapped software
  - Examples: Word processors, spreadsheets, Enterprise Resource Planning applications.

Classes of Software

- Embedded systems
  - Embedded software interfaces with hardware or other software systems (e.g., software in automobiles, washing machines, cell phones)
- Distributed systems
  - In a distributed system, software is spread across a number of processors (e.g., communication systems, transportation monitoring systems)
- Real-time systems
  - The correctness of information produced by real-time systems is dependent on the time taken to produce the information (e.g., anti-lock braking systems, cruise-control systems, flight management systems)
- Business systems
  - Software used to support business operations (e.g., ERP, payroll systems, order management systems)
- Reactive systems
  - Software that affects its environment in response to changes in the environment (e.g., home climate control systems)
Software Development Stakeholders (Roles)

- Users
  - Those who use the software
- Customers
  - Those who pay for the software
- Software developers
  - Architects
  - Programmers
  - testers
- Development Project Managers
- All of the above can be played by a single person

Modern Software Systems

- The internet has spawned a new generation of distributed software systems
  - E-commerce
  - Business-to-business systems
  - Web-based service providers
- New classes of software
  - Client-server
  - N-tier architectures
  - Web applications
  - Web services
  - Service Oriented Architectures (SOA)

What makes software development difficult?

- Novelty of application
  - Cannot rely on previous experience
- Software is intangible
  - Hard to understand development effort
- Change is more difficult than it appears
  - People make changes without fully understanding them
- Problem is not always well-defined
  - Stakeholders may not have a clear idea of the functionality they desire
- Desired qualities may be at odds with each other
  - Developers need to make trade-offs

What makes software development difficult? (2)

- Development is labor-intensive
  - Difficult to automate development tasks
- Untrained people can hack something together
  - Quality problems are hard to notice
Factors Behind the Emergence of SE

- Inability to predict time, effort, and costs.
- Inability to deliver quality software.
- Changes in the ratio of hardware and software costs.
- Increasingly important role of maintenance.
- Increased demand for more complex software.

Resulting state of affairs

- Much software has poor design and is getting worse.
- Demand for software is high and rising.
- We are in a perpetual ‘software crisis’.
- We have to learn to ‘engineer’ software.

Next Generation Development Techniques and Notations?

- UML (Unified Modeling Language)
- XML
- Adaptive programming
- AOP (Aspect-oriented programming)
- Agile processes
  - XP (eXtreme-Programming)
  - Scrum
- Crystal methods
- Model Driven Development
  - MDA (Model Driven Architecture)
  - MIC (Model Integrated Computing)
  - AOM (Aspect-Oriented Modeling)
- Domain-specific development

Software Engineering or Science?

- Scientific view: study of phenomena that affect the production of software.
- Engineering view: concerned with developing quality software solutions efficiently and economically.
- Software is created to solve a problem.
Software Engineering Definition

- SE is the application of a systematic, scientifically-based approach by which the capabilities of computers are made useful to humans via software.
- Emphasizes both scientific and engineering aspects of SE.

Problem Solving View

- SE is “the discipline of resolving problems with software solutions” (B. Blum).
- Solving customers’ problems
  - Sometimes the software solution can be acquired rather than developed
  - Focus should be on providing only those features needed to solve the problem (unnecessary features add to complexity)
  - Identifying and understanding the problem requires communication among software engineers and stakeholders

Problem Solving Aids

- Tool: an automated system that supports the application of a method/technique
- Procedure: pre-selected methods and tools that, in concert, promote a certain problem solving approach.

The case for engineering software

- Software systems are becoming more complex, more pervasive and more mission-critical
- Software engineering techniques are needed because complex systems cannot be completely understood by one person
- Teamwork and co-ordination are required
- Key challenge: Dividing up the work (decomposition) and ensuring that the parts of the system work properly together (synthesis)
- The end-product that is produced must be of sufficient quality
Software System Problems (Examples)
- Colorado human resource problem
- IRS document processing system
- Military “mishaps”
- DIA automated baggage system
- Therac-25 machine
- SDI (Star Wars)
- Ariane Rocket

A Software Engineering Profession?
- The term Software Engineering was coined in 1968
  - Realization that the principles of engineering should be applied to software development
- Engineering is a licensed profession
  - In order to protect the public
- Engineers design artifacts following well accepted practices which involve the application of science, mathematics and economics
- Ethical practice is also a key tenet of the profession
- Are we there yet in the software development community?

SE Objective
- To produce quality software that solves a problem within budget and on time.
- Tools, techniques/methods, and procedures are used to:
  - control complexity
  - enhance software quality
  - improve development efficiency

Assessing Software “Goodness”
- Software engineers use tools and techniques to enhance their ability to develop “good” software.
- SEEnrs must be able to demonstrate that software has desired qualities.
- SEEnrs must understand the nature of software quality.
What is “Quality”? 

No definitive answer! 
Notion of quality differs across projects, products, and stakeholders.

Customer: 
solves problems at an acceptable cost in terms of money paid and resources used

User: 
easy to learn; efficient to use; helps get work done

Developer: 
 easy to design; easy to maintain; easy to reuse its parts

Development manager:
sells more and pleases customers while costing less to develop and maintain

Quality Attributes

- IEEE: Software quality is the totality of features and characteristics (attributes) of a product or service that bears on its ability to satisfy given needs.
- Example attributes: correctness, usability, efficiency
- Relative importance of attributes can vary across stakeholders
- Programmers seek correctness and efficiency, testers prefer software that is easier to test, maintainers prefer software that is easy to understand and users desire software that is easy to learn and that makes them more productive
- Trade-offs are made when improving an attribute has a detrimental effect on another
- Efficiency vs. understandability; reliability vs. efficiency; usability vs. efficiency; security vs. availability

Quality Objectives

- Setting objectives for quality is a key engineering activity
  - You then design to meet the objectives
  - Avoids ‘over-engineering’ which wastes money
- Optimizing is also sometimes necessary
  - E.g. obtain the highest possible reliability using a fixed budget

Quality Perspectives

- Transcendent: concerns innate excellence (an ideal)
- User-based: concerns “fitness for use”
- Product-based: attributes of software and development process
- Manufacturing-based: conformance to specifications
- Value-based: determined by amount customer is willing to pay for product
Product Quality Attributes

- **Evolvability**: Is the software easy to extend? Is it easy to adapt?
- **Dependability**: Does the software behave in a manner that does not jeopardize its critical mission? Does it protect sensitive information? Does it behave safely?
  - Software should be safe, secure, robust, and reliable

Product Attributes (2)

- **Efficiency**: Will the software perform in a reasonable amount of time and use a reasonable amount of resources?
- **Usability**: Is the software easy to learn and use?
- **Correctness**: Does the software behave as stated in its requirements specification?
- **Robustness**: Does the software behave well in situations not covered in specifications?
- **Reliability**: Does the software produce the desired result consistently over a period of time?

Process Attributes

- **Visibility**: Activities provide clear indications of progress.
- **Understandability**: Activities and their order of execution are well-defined.
- **Supportability**: Automated support for activities is available.
- **Usability**: Process is acceptable to and usable by developers.

Quality by Domain

- **Real-time systems (e.g., autonomous vehicles)**
  - Response time constraints
  - Reliability
  - Safety (absence of undesirable behaviors that can cause hazards)
  - Usability
- **Embedded systems (e.g., cell phone software)**
  - Reliability
  - Efficiency of performance
  - Robustness
Quality by Domain
- Information systems (e.g., Banking systems)
  - Data Integrity
  - Usability
  - Robustness
  - Evolvability
- Distributed systems (e.g., online auction systems)
  - Fault tolerance
  - Reliability
  - Availability
  - Data integrity
  - Transparency

Essence of SE
- Concerned with programming-in-the-large and programming-in-the-small issues.
- Complexity of problems and solutions requires team development.
- Central theme is controlling complexity.

Essence of SE (2)
- Software evolves (change is inevitable for useful systems).
- Efficiency of software process is important.
- Software has to effectively support its direct and indirect users.

Fundamental SE Concepts
- Abstraction
- Analysis/Design Methods and Notations
- User Interface Design
- Software Architecture
- Software Process
- Reuse
- Measurement
- Tools
Physical vs. Software Engineering

- In SE cost of construction is incurred during development; manufacturing costs are comparatively inexpensive.
- Software has no physical parts, thus reliability is determined by errors in design and construction.

Visibility is low - difficult to visualize progress.
- Software is not continuous in nature.
  - Small errors can have disastrous effects.
  - Small changes in specification do not often translate to small changes in implementation.

SE Findings

- Software development tasks are often devoted to maintaining existing systems
- The later errors are identified the more expensive they are to correct.
- Failure to define correct requirements account for the greatest number of errors.

SE Findings (2)

- Programming is less error prone than design.
- Adding manpower to a late project makes it later.
- As software evolves its complexity increases unless it is properly maintained.
SE Myths

- Software development is primarily concerned with programming.
- Software change is easily accommodated because software is flexible.
- A developer’s task is finished when software is delivered to customer.
- Success of a project is determined solely by product quality.

SE Myths (2)

- A developer need only show that a program is correct w.r.t. a specification to validate its function.
- Use of state-of-the-art technology will solve all development problems.

SE Myths (3)

- All programmers are equal in ability.
- Developing and documenting standards will solve development problems.
- Development difficulty depends only on functional complexity.
- Software errors are unavoidable.

Software Engineering Concerns