The Nature of Software

• The Current State of Affairs: the “software crisis” or the “software challenge”
  – Studies show that software is expensive, faulty, unused, dominated by maintenance
  – Software requirements are rapidly outpacing software technology

• The need: The ability to consistently fashion software to solve appropriate problems in an orderly, predictable manner
  – An approach to the overall problem
  – Techniques to apply in each area of the approach
  – Technology transition and funding
System Engineering

• Software system is a system component
• Software engineering is part of system engineering
  – requirements of software vs. system

Telephone switching system =
  computers, telephone lines, telephones, satellites,
  software to control these and other components

  – Software engineer involved in system requirements analysis

• Understand application area
• Engineering requires compromise
What's Unique about Software?

• Software is malleable – we can modify the product itself
• Software construction is human-intensive
• Software is intangible
• Software application horizons expand with hardware capabilities
• Software problems are unprecedented in complexity
• Software solutions require unusual rigor
• Software has discontinuous operational nature
Software Qualities

- Acceptance standards for software are not yet clear
- Qualities are *goals* in practice of software engineering
- External qualities are visible to the user:
  - reliability, efficiency, usability
- Internal qualities are the concern of the developers:
  - verifiability, maintainability, extensibility
- Product qualities concern all developed artifacts:
  - maintainability, understandability, performance
- Process qualities deal with the development activity:
  - maintainability, productivity, timeliness, visibility

Internal qualities help developers achieve external qualities
Products are developed through process
Correctness

• Software is *functionally correct* if it behaves according to the functional requirements specification
  – correctness is a mathematical property
  – requires that a specification be available
  – must be possible to unambiguously determine whether the software meets the specification

• Software is *behaviorally correct* if it satisfies all specified behavioral requirements
  – different required behaviors may be specified in different paradigms or different languages

• Correctness is the ideal quality
Reliability

- Software is *reliable* if the user can depend on it
  - reliability is a statistical property
  - the probability that software will operate as expected over a specified period of time
  - we *expect* unreliable software, whereas most engineering products are expected to be reliable

- Reliability vs. Correctness
  - reliability is relative, while correctness is absolute
  - given a “correct” specification, correct software is reliable, but not vice versa
    - in practice, correct software may not operate as expected or desired
Robustness

• Software is *robust* if it behaves “reasonably” even in circumstances that were not anticipated in the requirements specification
  – robustness is a subjective property

• Robustness vs. Correctness
  – software may be correct but not be robust
  – if we could precisely define “reasonable” behavior, robustness would be equivalent to correctness (or reliability)
  – a specified requirement is an issue of correctness, an unspecified one is an issue of robustness
Performance

• Software *performance* is equated with efficiency; software is efficient if it uses available resources economically

• Performance can be assessed by complexity analysis, measurement, model analysis, and model simulation
  – performance is often addressed after an initial version addressing functionality

• Performance affects usability and scalability
Usability

• Software is *usable* if its end users find it easy to use
  – usability is an extremely subjective property
  – usability includes effort required to learn, operate, prepare input, and interpret output

• Usability refers to the human-machine interface for non-embedded systems, but to the ease of configuring the system to the environment for embedded software systems
  – usability depends on the consistency of its user and operator interfaces

• Usability may be achieved through standard user interfaces
Understandability

• Software is *understandable* if it is easy for developers to understand the produced artifacts
  – understandability is an internal product quality
  – some tasks are inherently more complex than others
  – understandability is enhanced by modularity, discipline, and standards

• External understandability deals with predictability (and hence reliability and robustness) and is also a component of usability
Verifiability

- Software is *verifiable* if satisfaction of desired properties can be easily determined
  - verifiability is an internal quality
  - verification can be performed by formal analysis or by testing
  - verifiability can be improved by monitoring the desired properties
  - verifiability is also enhanced by modularity, discipline, and standards

- Verifiability is affected by many other qualities – e.g., understandability, reliability, and visibility
Maintainability

- **Software is *maintainable*** if it can be modified easily after a version release (internal or external)
  - improvements rather than upkeep as in other engineered products
  - evidence shows that maintenance costs exceed 60% of total software costs
- **Corrective** maintenance: removal of residual faults, or “bugs”, in software after delivery (~20%)
- **Adaptive** maintenance: adjusting software to changes in application environment (~30%)
- **Perfective** maintenance: changing software to improve qualities (~50%)
Repairability

- Software is *repairable* if it allows defect correction with limited effort
  - in other disciplines, repairability is achieved by making fault-prone components accessible and products are often repaired by component replacement
  - repairability is enhanced by modularity and abstraction

- Repairability addresses corrective maintenance

- Repairability affects reliability, while the need for repairability decreases with increased reliability
Evolvability

• Software is *evolvable* if it facilitates addition of functionality or modification of existing functions
  – the malleable nature of software makes evolvability of implementation too easy
  – evolution should start at the design (or even requirements) with a feasibility study and proceed in an organized fashion
  – evolvability is also enhanced by modularity and abstraction

• Evolvability addresses adaptive and perfective maintenance

• Successful software is quite long lived and can evolve gracefully
Reusability

• Software is *reusable* if it can be used, perhaps with minor modification, to construct another product
  – reusability must be planned for
  – reusability can occur at all levels, from people to process, from requirements to code
  – trend is to develop new applications by assembling ready-made, OTS components

• Reusability is akin to evolvability
Portability

- Software is *portable* if it can run in different environments with little or no effort
  - hardware or software platform
  - portability is enhanced by assuming minimal environment capabilities or by isolating environment-dependent components
  - tradeoffs between attaining portability and using full features, so design software to adapt to environment
  - portability has gained importance as software costs far outweigh hardware costs
Interoperability

• Software is *interoperable* if it can coexist and cooperate with other systems
  – interoperable systems should be easily integrated
  – interoperability is enhanced by defining standard interfaces in application domains
  – an open system is a collection of independently-written applications that cooperate and function as an integrated system
  – trend is to release system with specification of “open interfaces”
Productivity

- *Productivity* measures the performance of development process
  - productivity offers many tradeoffs, such as personnel specialization to software reuse
  - development organization affects productivity
  - modern software engineering techniques and tools attempt to increase productivity

US industry places too much emphasis on productivity and not enough emphasis on other qualities
Timeliness

• *Timeliness* measures the ability to deliver software on time
  – timeliness requires careful scheduling, accurate work estimation, clearly specified and verifiable milestones
  – timeliness is addressed by cost estimation models
  – timeliness can be achieved through *incremental delivery* of useful system subsets

Timeliness is of little use if software does not satisfy other qualities
Visibility

• Software is *visible* if all steps and its current status are documented clearly and can be easily accessed
  – visibility allows engineers to weigh the impact of their actions and guide decisions
  – visibility facilitates teamwork
  – visibility enables managers to assess progress
  – visibility is not only an internal quality but also *external*
  – visibility is a process quality, but also requires visibility of intermediate products
Process Qualities

- Process is reliable if it consistently leads to high-quality products
- Process is robust if it can accommodate unanticipated changes in tools and environment or unanticipated use by developers
- Process performance is productivity
- Process is verifiable if we can determine if the process meets the development requirements
- Process is evolvable if it can accommodate new management and organizational techniques
- Process reusability is use of methodologies and lifecycle models for building different products
Application Domains with Specialized Quality Requirements

- **Information Systems**: data storage and retrieval
  - *data integrity* deals with corruption on malfunction
  - *security* concerns protection from unauthorized use
  - *data availability* involves length of data unavailability
  - *transaction performance* is number of transaction per time unit

- **Distributed Systems**: [semi-]independent computers connected by a communication network
  - *fault tolerance* is the ability to tolerate faults resulting from partitioning the network or failure of an individual node

- **Scientific applications**: computation-oriented
  - *accuracy* is the closeness of results to correct precision
Application Domains with Specialized Quality Requirements

• Real-Time Systems: must respond to events within precisely defined, strict time periods
  – *real-time response* deals with verifiable response time, not necessarily fast response

• Embedded Systems: software is one of many system components with interfaces to other components (as opposed to end user)
  – *safety* is concerned with operation without unacceptable risk

• Most real-time and embedded systems control safety-critical applications
Measurement and Improvement

- Qualities must be measurable
- Measurement requires that qualities be precisely defined
- Improvement requires accurate measurement
- Metrics and their relation to improvement are needed

Empirically-guided software process improvement