Course Introduction and Overview of Software Engineering

Richard N. Taylor
Informatics 211
Fall 2007

Software Engineering

- “*A discipline that deals with the building of software systems which are so large that they are built by a team or teams of engineers*.” [Ghezzi, Jazayeri, Mandrioli]

- “*Multi-person construction of multi-version software*.” [Parnas]

Software Engineering

- "*A discipline whose aim is the production of fault-free software, delivered on-time and within budget, that satisfies the user's needs. Furthermore, the software must be easy to modify when the user's needs change*." [Schach]
- "*Difficult*." [van der Hoek]

Software Engineering

- “It’s where you actually get to design big stuff and be creative.” [Taylor]

Design, Science, Engineering, Management, Human Factors

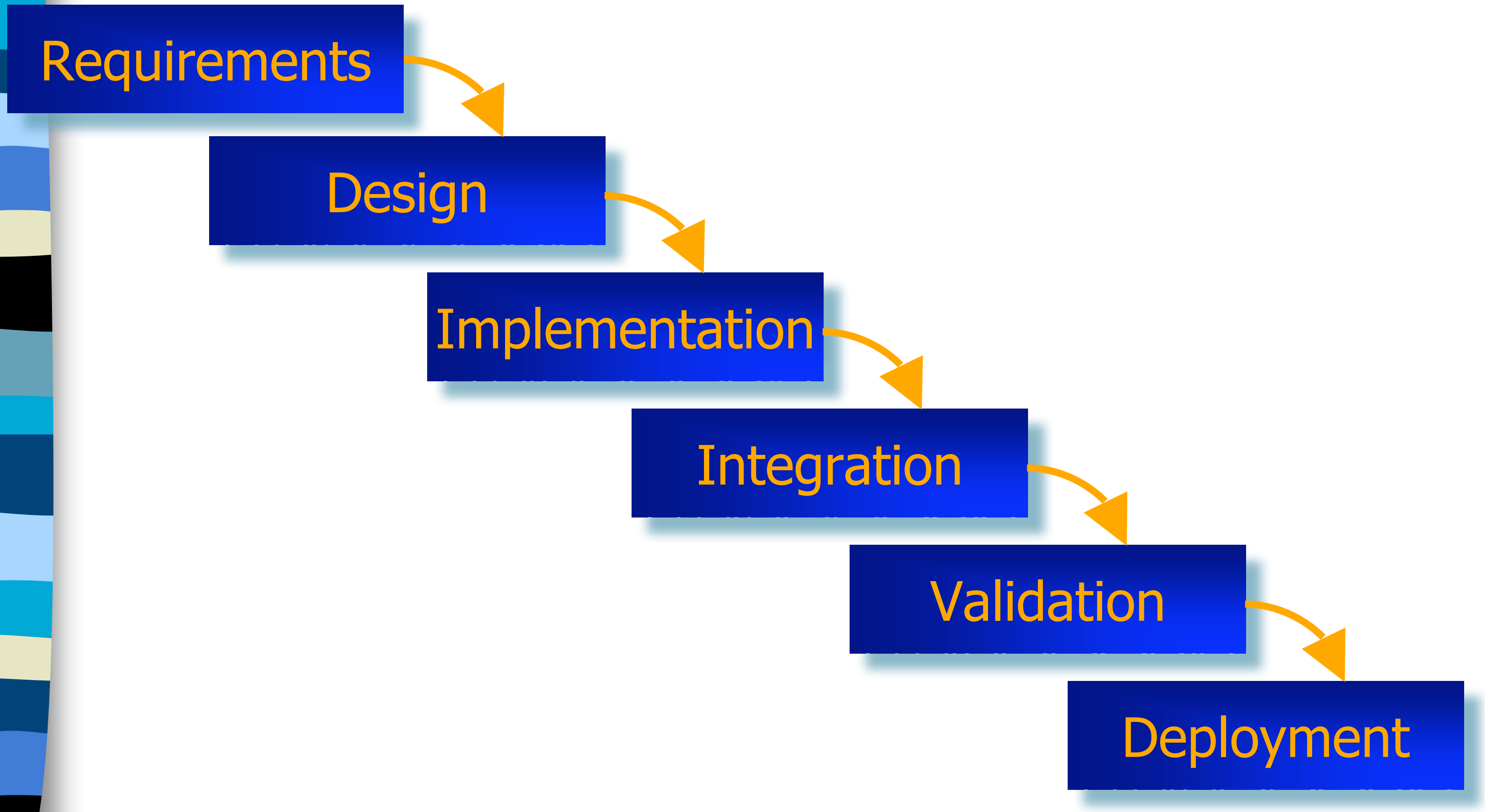
- Design: concepts, methods, techniques
- Science: empirical studies; theories characterizing aggregate system behavior (e.g. reliability)
- Management: organizing teams, directing activities, correcting problems
- Human factors: user task understanding and modeling; ergonomics in user interface design
- Engineering: tradeoffs, canonical solutions to typical problems
  - Tradeoffs and representative qualities
    - Pick any two:
      - Good, fast, cheap

Common Software Engineering Principles (CW* list)

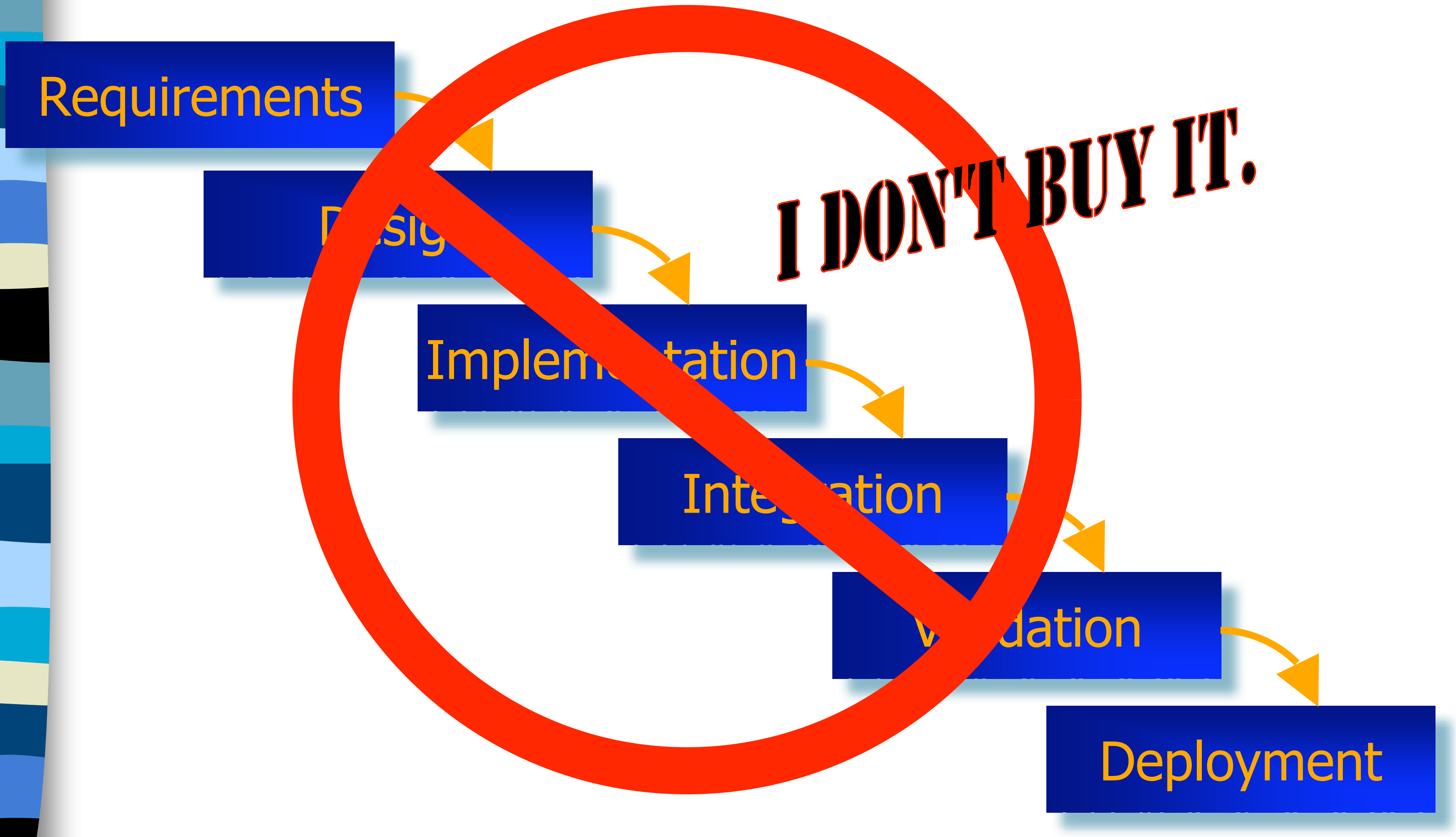
- Rigor and Formality
- Separation of Concerns
- Modularity and Decomposition
- Abstraction
- Anticipation of Change
- Generality
- Incrementality
- Reliability

*Conventional Wisdom

Software Lifecycle Context (Waterfall Model) (Old CW)



Software Lifecycle Context (Waterfall Model)



The Mythical Man-Month by Fred Brooks (I)

- Published 1975, Republished 1995
- Experience managing the development of OS/360 in 1964-65
- Central Argument
  - Large programming projects suffer management problems different in kind than small ones, due to division of labor.
  - Critical need is the preservation of the conceptual integrity of the product itself.

The Mythical Man-Month by Fred Brooks (II)

- Central Conclusions
  - Conceptual integrity achieved through exceptional designer
  - Implementation achieved through well-managed effort
  - Brooks’s Law: Adding personnel to a late project makes it later

No Silver Bullet by Fred Brooks

- **Essence:** the difficulties inherent in the nature of the software
- **Accidents:** those difficulties that today attend its production but that are not inherent
- Solution (?): Grow Great Designers

Accidental Difficulties

- Solutions exist
  - Possibly waiting to be discovered
- Past productivity increases result of overcoming
  - Inadequate programming constructs & abstractions
    - Remedied by high-level programming languages
    - Increased productivity by factor of five
    - Complexity was never inherent in program at all

Accidental Difficulties (cont’d)

- Past productivity increases result of overcoming (cont’d)
  - Viewing results of programming decisions took long time
    - Remedied by time–sharing
    - Turnaround time approaching limit of human perception
  - Difficulty of using heterogeneous programs
    - Addressed by integrated software development environments
    - Support task that was conceptually always possible

Essential Difficulties

- Only partial solutions exist for them, if any
- Cannot be abstracted away
  - Complexity
  - Conformity
  - Changeability
  - Intangibility

Complexity

- No two software parts are alike
  - If they are, they are abstracted away into one
- Complexity grows non-linearly with size
  - E.g., it is impossible to enumerate all states of program
  - Except perhaps “toy” programs

Conformity

- Software is required to conform to its
  - Operating environment
  - Hardware
- Often “last kid on block”
- Perceived as most conformable

Changeability

- Change originates with
  - New applications, users, machines, standards, laws
  - Hardware problems
- Software is viewed as infinitely malleable

Intangibility

- Software is not embedded in space
  - Often no constraining physical laws
- No obvious representation
  - E.g., familiar geometric shapes

Pewter Bullets

- Ada, C++, Java and other high–level languages
- Object-oriented design/analysis/programming
- Artificial Intelligence
- Automatic Programming
- Graphical Programming
- Program Verification
- Environments & tools
- Workstations

Promising Attacks On Complexity (In 1987)

- Buy vs. Build
- Requirements refinement & rapid prototyping
  - Hardest part is deciding what to build (or buy?)
  - Must show product to customer to get complete spec.
  - Need for iterative feedback

Promising Attacks On Complexity (cont'd)

- Incremental/Evolutionary/Spiral Development
  - Grow systems, don't build them
  - Good for morale
  - Easy backtracking
  - Early prototypes
- Great designers
  - Good design can be taught; great design cannot
  - Nurture great designers

Software Architecture (and Architects)

- Software Engineers have always employed software architectures
  - Very often without realizing it!
- Address issues identified by researchers and practitioners
  - Essential software engineering difficulties
  - Unique characteristics of programming-in-the-large
  - Need for software reuse
- Many ideas originated in other (non-computing) domains

Primacy of Design

- Software engineers collect requirements, code, test, integrate, configure, etc.
- An architecture-centric approach to software engineering places an emphasis on design
  - Design pervades the engineering activity from the very beginning
- But how do we go about the task of architectural design?

The Software Industry Today

Software Engineering is in Transition

- Component-Based Engineering and Integration
- Technological Heterogeneity
- Enterprise Heterogeneity
- Greater potential for Dynamic Evolution
- Internet-Scale Deployment

- Many competing standards
- Much conflicting terminology

Research

- Topics
  - "(The) Future of Software Engineering 2007"
- Publication venues
  - Journals
  - Conferences
- "Variance of opinions"

Publication Venues

- Good ones
  - ICSE
  - FSE
  - ASE
  - ACM TOSEM and IEEE-TSE
- Questionable ones: caveat reader
  - COMPSAC
  - (JSS)

Future of SE…

- Process
- Requirements engineering
- Reverse engineering
- Testing
- Maintenance and Evolution
- Software architecture
- OO Modeling
- SE and Middleware
- Tools and environments
- Configuration management
- Databases and SE
- SE Education
- Software analysis
- Formal specification
- Mathematical foundations
- Reliability and Dependability
- Performance
- SE for Safety
- SE for security
- SE for mobility
- SE & the Internet
- Software economics
- Empirical studies of SE
- Software metrics