What is Software Engineering About?

Stuart Faulk

CIS 422/522

The “Software Crisis”

- **Have been in “crisis” since the advent of “big” software (roughly 1965)**
- **What we want for software development**
  - Low risk, predictability
  - Lower costs and proportionate costs
  - Faster turnaround
- **What we have:**
  - High risk, high failure rate
  - Poor delivered quality
  - Unpredictable schedule, cost, effort
- **Characterized by lack of *control* (inability plan the work, work the plan)**

Symptoms of the Crisis

- Two of every eight large software project is cancelled
- Average projects overshoot schedule by 50%, large project do much worse
- 75% of large systems are failures in the sense that they do not operate as intended
- 60% of them fail to deliver a single working line of code
- E.g., Ariane 5, Therac 25, Mars Lander, DFW Airport, FAA ATC etc., etc. (See examples in Text)

Discussion Context

- **Focus large, complex systems**
  - Multi-person: many developers, many stakeholders
  - Multi-version: intentional and unintentional evolution
- **Quantitatively distinct from small developments**
  - Complexity of software (e.g. rises non-linearly with size)
  - Complexity of communication rises exponentially
- **Qualitatively distinct from small developments**
  - Multi-person introduces need for organizational functions (management, accounting, marketing), policies, oversight, etc.
  - More stakeholders and more kinds of stakeholders
- **Rule of thumb: project starts to be “large” when group developing a single product can’t fit around a table.**

Software “Industry” is Pre-Industrial

Pre-Industrial

- **Craftsman builds product**
  - Builds one product at a time
  - Each product is unique, parts are not interchangeable
  - Quality depends on craftsman’s skill – product of training, experience
  - Many opportunities for error
- **Focus on individual products**
  - Customization is easy
- **Scaling is difficult**
  - Parts are not interchangeable
  - No economy of scale
  - **Control problems rise *exponentially* with product size!**

Post-Industrial

- **Products produced by machines**
  - Quality depends on machines & manufacturing process
  - Production requires little training or experience
- **Focus on developing the <u>means of production</u>**
  - Craftsman builds means to build product (tools, factory)
  - Customization is difficult
- **Easily scales**
  - Parts are interchangeable
  - Products are alike
  - Economies of scale apply

Industrial Model Distinguished by its ***Process***

Pre-Industrial

Build Product
(by hand)

Product

Post-Industrial

Develop Means of Production
- Design Manufacturing process
- Design Factory
- Build Factory

Factory
(Means of Production)

Build Products

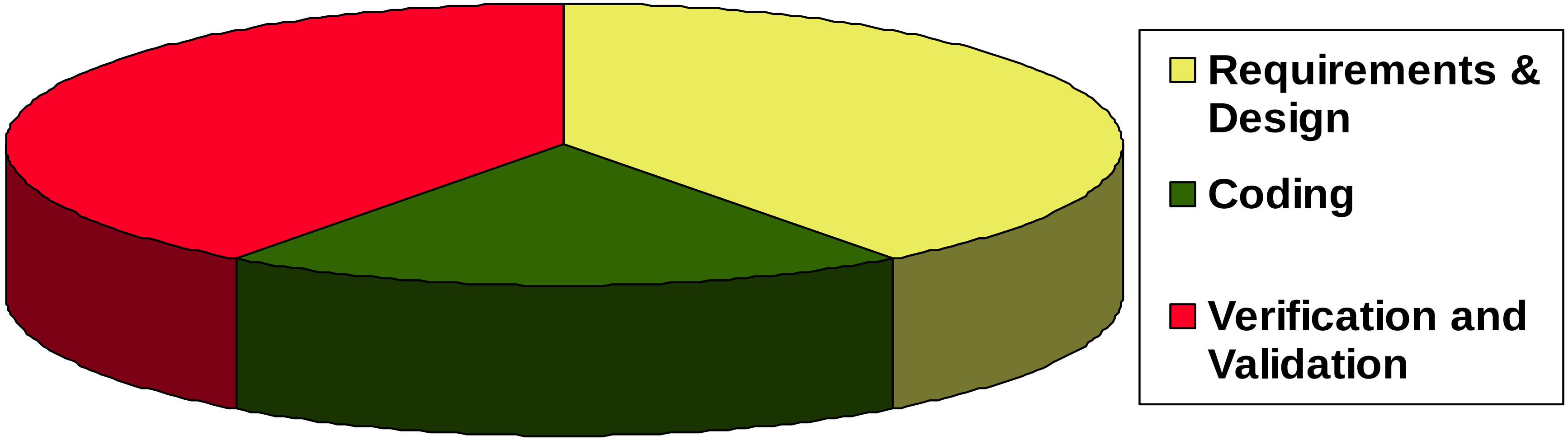
Product

Historical note: Eli Whitney's vision and invention transformed the product development process.

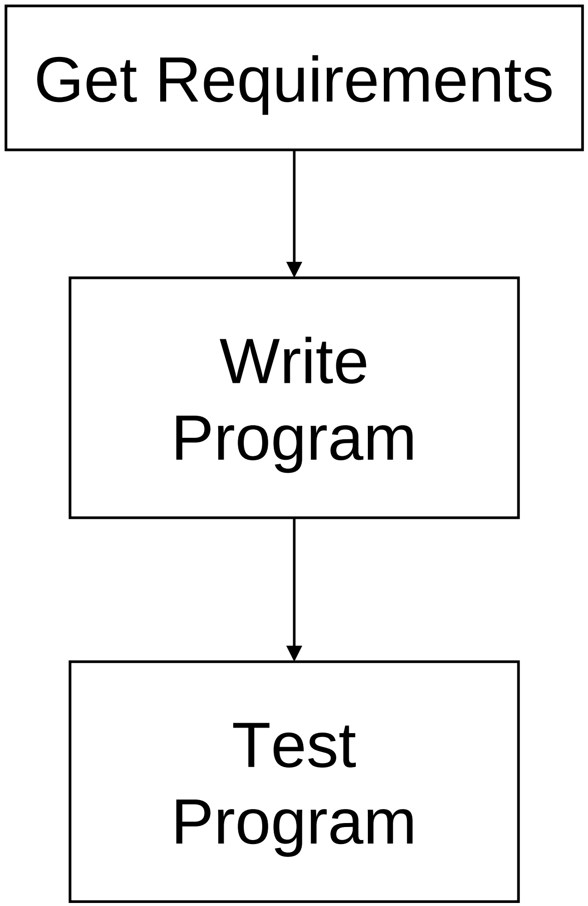
Implications

- **Small system development is driven by technical issues (I.e., programming)**
- **Large system development is dominated by organizational and control issues**
  - Managing complexity, communication, coordination, etc.
  - Projects fail when these issues are inadequately addressed
- **Lesson #1**: **programming ≠ software engineering**
  - Techniques that work for small systems fail utterly when scaled up
  - Programming alone won't get you through real developments **or even this course**

40-20-40 Rule



Programming View



Insert

Origins of SE

- **Term "software engineering" was coined at 1968 NATO conference**:

  "Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines."

- **Response to "software crisis" manifest by systems that**
  - Failed to provide desired customer functionality
  - Lacked critical qualities (e.g., performance, safety, reliability)
  - Overran budget and schedule (hugely)
  - Were difficult to change or maintain
- **Desire for SE to be more like other engineering disciplines**
  - Analytical, predictable, manageable
  - State as an aspiration, not statement of existing condition

Has anything changed?

- **Incorrect to conclude that no progress has been made**
  - Substantial improvements in programming languages, tool
  - Better understanding and control of processes
- **But the problems have also changed**
  - Large developments now are orders of magnitude more code than in 1968
  - Improved capabilities are overcome by larger problems, greater complexity
- **Note: “software crisis” is a euphemism for “state of the practice”**

What hasn't changed?

- Still not an engineering discipline in classic sense
  - Implies use of applied mathematics and systematic methods to develop and assess product properties
  - These tools are immature where they exist at all
  - Software "engineering" is not taught, licensed, regulated, ore recognized as an engineering discipline (e.g., by engineers)

What hasn't changed?

- But we often don't apply what we know
  - Existing methods, models often not understood or used in industry
  - Little attention is given to process or products other than code
  - Quality of products depends on qualities of the individuals rather than qualities of engineering practices
- Development continues to be characterized by **lack of *control*** (inability plan the work, work the plan)

View of SE in this Course

- The <u>purpose of software engineering</u> is to *gain* and *maintain* intellectual and managerial control over the products and processes of software development.
  - "Intellectual control" means that we are able make rational choices based on an understanding of the downstream effects of those choices (e.g., on system properties).
  - Managerial control means we control development *resources* (budget, schedule, personnel).

Control is the Goal

- **Both are necessary for success!**
- **Intellectual control implies**
  - We understand what we are trying to achieve
  - Can distinguish good choices from bad
  - We can reliably and predictably achieve what we want
- **Managerial control implies**
  - We make accurate estimations
  - We deliver on schedule and within budget
- Assertion: Managerial control is not really possible without intellectual control (no matter what the Harvard School of Business says)

Course Approach

- **Will learn methods for acquiring and maintaining control of software projects**
- **Managerial control**
  - Planning and controlling development
  - Process models addressing development issues (e.g. risk, time to market)
  - People management and team organization
- **Intellectual control**
  - Methods for software requirements, architecture, design, test
  - Notations, verification & validation
- **Caveat: we can really only scratch the surface (but it's important)**

Assignment

- Reading:
  - Text: Chapter 5
- Project: prepare for first project meeting (team assignments Friday)
  - Begin considering how you will approach the problem
  - Think about what role you want to play