

## Why SE?

## What are we building?

### Applications:

Barcode Scanner: 10-50Kloc

Car Shifter: 20KLoc

Cellphone: 30 Kloc

ATM Network 600 Kloc

B2 Bomber: 3.5 Mloc

Shuttle:  $10^7$  Loc

M\$ Office :  $10^7$  Loc

Telephone Switch:  $10^7 \cdot 10^8$

### Boeing 777:

- 2.5 Mlocs of custom code, 1.5 Million lines of COTS.
- 79 different suppliers, each supplying part of the system.
- Specifications of the software, done through SCD (software control diagrams). 100 pages for "simplest systems" to 10,000 pages for most complex.
- Interface descriptions for each system placed in database. 40,000 data items, and 3,000 analog systems. Tool built to automatically find discrepancies. 50,000 found and removed!
- System was written in ADA.

## Why is SE hard?

- **Different kinds of problems:**
  - we are used to problems where there are pre-stated specifications (e.g., "write a program to find the shortest path..."), where it makes sense to talk about the correctness of a solution
  - **real world problems:** (e.g., "software to help people control nuclear reactors"): "acceptability" is defined by user satisfaction (validation vs verification). Implies *evolution* is intrinsic
  - **wicked problems:**
    - define and solve *concurrently*
    - no unique definition or solution
    - always room for improvement in def'n and sol'n
    - new problem, not previously encountered
    - many *stakeholders*, with different goals

## Essential Difficulties of SE: [Brooks]

- **COMPLEXITY:**
  - software is more complex for its size than any other human construct; no two parts are alike (vs. car, microchips,...);
  - science advances by simplifying, while software cannot ignore/simplify details of real world
- **CONFORMITY:**
  - among {hardware, software, people, organizations} it is software which is chosen to *bend/adapt* because it is more malleable, last to arrive on the scene, usually only one developed on site,...
- **CHANGEABILITY:**
  - once delivered, most engineered products (hardware, cars, buildings) are rarely changed because the cost to change would be a large fraction of the cost to make. The (unfortunate) *perception* is that software is cheap to change. And pressure to change comes from successful use, and aging hardware platform.
- **INVISIBILITY:**
  - since it has no physical reality, software is not properly visualizable with diagrams, etc. in the way in which houses, circuits, etc are.

## What will (not) make a big difference: [Brooks1987]

- **Minor**
  - Ada or Java or C# or Python or ...
  - Object-Oriented Programming
  - Artificial Intelligence
  - Automatic Programming
  - Graphical programming
  - Program verification
  - Environments and tools
  - Workstations
- **Major**
  - Buy vs build
  - Requirements refinement and prototyping
  - Incremental development
  - Great designers (Unix, Pascal, Smalltalk vs Cobol, PL/I, Ada, MS-DOS)

## Software Engineering

- **Engineering:** application of a systematic, disciplined, quantifiable approach to structures, machines, products, systems, or processes
  - **Software Engineering:** that form of engineering that applies
    - a systematic disciplined quantifiable approach
    - the principles of computer science, design, engineering, management, mathematics, psychology, sociology, and other disciplines as necessary
    - and sometimes just plain inventionto creating, developing, operating and maintaining cost-effective, reliably correct, high-quality solutions to *software problems* [Berry 92]
- SE* requires the *identification of a problem*, a *computer* to carry execute a software product, and a *user environment* (composed of people, tools, methodologies, etc.)

## SE: Processes, Models, and Tools

- **Processes:**

Systematic ways of organizing teams and tasks so that there is a clear, traceable path from customer requirements to the final product. (e.g.: Waterfall, Prototyping, Spiral etc.) Processes help organize and co-ordinate teams, prepare documentation, reduce bugs, manage risk, increase productivity, etc.
- **Models:**

Well-defined formal or informal languages and techniques for organizing and communicating arguments and decisions about software. e.g:

  - specification languages (Z, etc),
  - design models (UML, etc)

Models help stake-holders communicate: customers with developers, designers and developers, developers and testers etc. If they are formal, they also can help support automation
- **Tools:**

Programs which automate or otherwise support software development tasks: e.g.,

  - Eclipse, Make, CVS, etc.

Tools increase productivity, quality and can reduce costs

## Software Lifecycle

1. **Problem statement**
  - needs analysis
  - requirements specification: functional, non-functional
2. **Design**
  - architectural
  - detailed
  - (communication, database)
3. **Implementation**
  - coding
  - testing:
    - module
    - integration
  - documentation
4. **Maintenance**
  - corrective
  - adaptive
  - enhancement

*(We'll see later -- when we have a bit of time -- that these are not always carried out sequentially)*

## Pop Quiz

### Where does time go?

- Development vs Maintenance: *<45 vs >50*
- In development: *40,20,40*
- In maintenance *20,20,60*

### How do programmers spend their time

- working alone *30*
- interacting *50*
- non-productive *20*

## Why is change a problem?

- **impact of when change is made (relative cost of accepting it)**
  - req 0.1
  - desn 0.5
  - code 1
  - unit test 2
  - accept test 5
  - maint 20
- **hard to predict code use**
  - To develop a compiler, in 250K of Fortran code, how many statements are of the form *68%*
    - .. A=B; *24%*
    - .. A=B # C; *4%*
    - .. A=B#C#D; *4%*
    - .. rest
- **how good are changes?**
  - twice as likely to be incorrect as ordinary lines of code