Outline for June 3, 2004

Reading: Chapters 18, 22.1–22.5, 22.7

Discussion Problem

Ever wonder why people need to write program specifications and requirements carefully? The answer is that those specifications seem to live forever. Here's an example:

The United States Standard railroad gauge (distance between the rails) is 4 feet, 8.5 inches. That's an exceedingly odd number. Why was that gauge used? Because that's the way they built them in England, and the United States railroads were built by English expatriates. Why did the English people build them like that? Because the first rail lines were built by the same people who built the pre-railroad tramways, and that's the gauge they used.

So, why did "they" use that gauge? Because the people who built the tramways used the same jigs and tools that they used for building wagons, which used that wheel spacing. Okay! Why did the wagons use that odd wheel spacing? Well, if they tried to use any other spacing the wagons would break on some of the old, long distance roads, because that's the spacing of the old wheel ruts.

So who built these old rutted roads? The first long distance roads in Europe were built by Imperial Rome for the benefit of their legions. The roads have been used ever since. And the ruts? The initial ruts, which everyone else had to match for fear of destroying their wagons, were first made by Roman war chariots. Since the chariots were made for or by Imperial Rome, they were all alike in the matter of wheel spacing.

Thus, we have the answer to the original questions. The United States standard railroad gauge of 4 feet, 8.5 inches derives from the original specification for an Imperial Roman army war chariot. Specifications and bureaucracies live forever. So, the next time you are handed a specification and wonder what horse's rear came up with it, you may be exactly right, because the Imperial Roman chariots were made to be just wide enough to accommodate the back-ends of two war horses.

— source unknown, but circulated on USENET and other message systems

Outline for the Day

1. Assurance
    a. Waterfall Life Cycle Model
        i. Requirements definition and analysis
        ii. System and software design
        iii. Implementation and unit testing
        iv. Integration and system testing
        v. Operation and maintenance
    b. Other models
        i. Exploratory programming
        ii. Prototyping
        iii. Formal transformation
        iv. System assembly from re-usable components
        v. Extreme programming
2. Malicious logic
    a. Quickly review Trojan horses, viruses, bacteria; include animal and Thompson's compiler trick
    b. Logic Bombs, Worms (Schoch and Hupp)
3. Ideal: program to detect malicious logic
    a. Can be shown: not possible to be precise in most general case
    b. Can detect all such programs if willing to accept false positives
    c. Can constrain case enough to locate specific malicious logic
    d. Can use:

i. Type chcking (data vs. instructions)
ii. Limiting rights (sandboxing)
iii. Limiting sharing
iv. Preventing or detecting changes to files
v. Prevent code from acting beyond specification (proof carrying code)
vi. Check statistical characteristics of programs (more authors than known, constructs in object files not corresponding to anything in the source)