Section 08 – Lesson Plan

Discuss the 'Ariane 5' story and the conclusions from it relevant to software engineering:
(from http://www2.vuw.ac.nz/staff/stephen_marshall/SE/Failures/SE_Ariane.html)

- "Software should be assumed to be faulty until applying the currently accepted best practice methods can demonstrate that it is correct."
- The failure suggests the following lessons for other software systems:
  - (A) *Test!*
  - (B) *Try to write code so that it cannot fail.*
    - The SRI should never have sent data to the main computer that was not valid and open to be misinterpreted as flight instructions. The SRI should have simply continued to provide its last best guess if nothing else could have been done, while simultaneously indicating that a problem had occurred elsewhere.
  - (C) *Don't allow errors or exceptions to propagate in an uncontrolled manner.*
    - The possibility of an exception had been allowed for in the design, just not for this particular calculation. At the very least the particular function should have caught the failure and acted to solve the problem without taking out the entire processor.
  - (D) *When reusing code from another system, make sure that any existing requirements or assumptions remain valid.*
    - You should be able to track from requirements to code and vice-versa.
  - (E) *Reused code still needs to be tested.*
    - Yes, it's likely to have fewer bugs if it has worked before, but the reuse has changed the context, new bugs may appear or old ones be exposed.
  - (F) *Make sure that the documentation links aspects of the design with specific requirements and vice-versa*
    - … so that when requirements are changed, affected design and coding decisions can be easily identified.
  - (G) *Test!*
- Which of these practices (A)-(G) are teams currently using?
- By not following which of these practices have you personally been "bitten"?

What does code size imply?
(from Notkin's overview http://www.cs.washington.edu/education/courses/503/00sp/lecture%202%20(overview).htm)

- As the size of the software system grows, the key discipline changes

Code Size	Discipline
10^3	Mathematics
10^4	Science
10^5	Engineering
10^6	Social Science
10^7	Politics
10^8	??

  - How many of you have developed products of the following sizes: (A) 10^3 LOC; (B) 10^4 LOC; (C) More?
- Average delivered source lines per person
  - First ask students for their estimates
    - (A) 1,000 LOC/year; (B) 10,000 LOC/year; (C) 100,000 LOC/year; (D) more
  - Common estimates are that a person can deliver about 1,000 source lines per year
    - Including documentation, scaffolding, etc.

"Good enough" software

- Criteria for something to be "good enough" (by James Bach, http://www.satisfice.com/articles.shtml)
  - It has sufficient benefits.
  - It has no critical problems.
  - The benefits sufficiently outweigh the problems.
  - In the present situation, and all things considered, further improvement would be more harmful than helpful.
- Key questions to ask when doing an evaluation:
  - Good enough for whom?
  - Good enough for what?

Show and discuss the plate about "Good, Fast and Cheap" jobs:

- One can select at most two of these three qualities.
  - Why?