Chapter 11

- **Quality Concepts**

Software Quality

- In 2005, *ComputerWorld* [Hil05] lamented that
  - "bad software plagues nearly every organization that uses computers, causing lost work hours during computer downtime, lost or corrupted data, missed sales opportunities, high IT support and maintenance costs, and low customer satisfaction.
- A year later, *InfoWorld* [Fos06] wrote about the
  - "the sorry state of software quality" reporting that the quality problem had not gotten any better.
- Today, software quality remains an issue, but who is to blame?
  - Customers blame developers, arguing that sloppy practices lead to low-quality software.
  - Developers blame customers (and other stakeholders), arguing that irrational delivery dates and a continuing stream of changes force them to deliver software before it has been fully validated.

Quality

- The *American Heritage Dictionary* defines *quality* as
  - “a characteristic or attribute of something.”
- For software, two kinds of quality may be encountered:
  - Quality of design encompasses requirements, specifications, and the design of the system.
  - Quality of conformance is an issue focused primarily on implementation.
  - User satisfaction = compliant product + good quality + delivery within budget and schedule

Software Quality

- Software quality can be defined as:
  - *An effective software process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.*
- This definition has been adapted from [Bes04] and replaces a more manufacturing-oriented view presented in earlier editions of this book.

The Software Quality Dilemma

- If you produce a software system that has terrible quality, you lose because no one will want to buy it.
- If on the other hand you spend infinite time, extremely large effort, and huge sums of money to build the absolutely perfect piece of software, then it's going to take so long to complete and it will be so expensive to produce that you'll be out of business anyway.
- Either you missed the market window, or you simply exhausted all your resources.
- So people in industry try to get to that magical middle ground where the product is good enough not to be rejected right away, such as during evaluation, but also not the object of so much perfectionism and so much work that it would take too long or cost too much to complete. [Ven03]

"Good Enough" Software

- Good enough software delivers high quality functions and features that end-users desire, but at the same time it delivers other more obscure or specialized functions and features that contain known bugs.
- Arguments *against* "good enough."
  - It is true that "good enough" may work in some application domains and for a few major software companies. After all, if a company has a large marketing budget and can convince enough people to buy version 1.0, it has succeeded in locking them in.
  - If you work for a small company be wary of this philosophy. If you deliver a "good enough" (buggy) product, you risk permanent damage to your company's reputation.
  - You may never get a chance to deliver version 2.0 because bad buzz may cause your sales to plummet and your company to fold.
  - If you work in certain application domains (e.g., real time embedded software, application software that is integrated with hardware can be negligent and open your company to expensive litigation.

Questions

- 1 What is software quality?
- 2 What is the software quality dilemma?