

# Testing Software Documentation

---

# The good old days

---

- ❑ Software documentation was:
  - a `readme.txt` file copied onto the software's floppy disk
  - a 1 page insert put into the shrink-wrapped package containing the software
  - comments in the source code!
  - Unix man pages are still in vogue, however ...
- ❑ Testers ran a spell checker on the file and that was about the extent of testing the documentation.

# Today ...

---

- Much of the non-code is the software documentation, which requires much effort to produce.
- Documentation is now a major part of a software system.
  - It might exceed the amount of source code!
  - It might be integrated into the software (e.g., help system)
- Testers have to cover the code and the documentation.
  - Assuring that the documentation is correct is part of a software tester's job.

# Classes of software documentation

---

- ☐ Packaging text and graphics
- ☐ Marketing material, ads, and other inserts
- ☐ Warranty/registration
- ☐ End User License Agreement (EULA)
- ☐ Labels and stickers
- ☐ Installation and setup instructions
- ☐ User's manual
- ☐ Online help
- ☐ Tutorials, wizards, and computer-based training
- ☐ Samples, examples, and templates
- ☐ Error messages

# Importance of documentation testing

---

- ❑ Improves usability

- Not all software was written for the Mac :-)

- ❑ Improves reliability

- Testing the documentation is part of black-box testing.
  - A bug in the user manual is like a bug in the software.

- ❑ Lowers support cost

- The exercise of writing the documentation helped debug the system.

# Documentation testing checklist

---

- ☐ Audience:
  - E.g., make sure documentation is not too novice or too advanced.
- ☐ Terminology:
  - Is it suitable for the audience?
  - Terms used consistently?
  - Abbreviations for acronyms?
- ☐ Content and subject matter:
  - Appropriate subjects covered?
  - No subjects missing?
  - Proper depth?
  - Missing features described accidentally?

# Documentation testing checklist (cont'd)

---

- ☐ Just the facts:
  - All information technically correct?
  - Correct table of contents, index, chapter references?
  - Correct website URLs, phone numbers?
- ☐ Step by step:
  - Any missing steps?
  - Compared tester results to those shown in the documentation?
- ☐ Figures and screen captures:
  - Accurate and precise?
  - Are they from the latest version of the software?
  - Are the figure captions correct?
- ☐ Samples and examples:
  - Do all the examples work as advertised?
- ☐ Spelling and grammar

# Auto-generated code documents

---

- Tools such as:

- **Doxygen**
- Javadoc
- ROBODoc
- POD
- TwinText

```
/**
 * The time class represents a moment of time.
 *
 * \author John Doe
 */
class Time {
/**
 * Constructor that sets the time to a given
 * value.
 * \param timemillis is a number of milliseconds
 * passed since Jan 1. 1970
 */
Time(int timemillis) {
```

can be used to auto-generate the code documents from source code comments and create HTML reference manuals.

- Code documents can be organized into a *reference guide* style that enables programmers to quickly look up functions or classes.
- Comprehensive survey of code documentation tools:
  - [http://en.wikipedia.org/wiki/Comparison\\_of\\_documentation\\_generators](http://en.wikipedia.org/wiki/Comparison_of_documentation_generators)



# Discussion ...

---

- ☐ Who should write software documentation?
- ☐ Why is documentation a second-class citizen compared to code?
- ☐ Why is keeping code/executables and documentation consistent difficult?
  - Is the problem inherent or due to sloppy software engineering?