Chapter 11: Maintaining the system

1.

a. An air traffic control system is an E-system because it is subject to numerous environmental changes, e.g. changes in air traffic patterns and routes, changing understanding of weather patterns, changes in aviation law, etc. Any aspect of such a system could change.

b. An operating system for a microcomputer is also an E-system because it could change based on changes to the hardware it was written to support, or to changing user desires. Again, any aspect of the system could change.

c. A floating-point acceleration system is most likely a P-system because the implementation is a less-than-perfect solution to the problem, although the problem is well defined. In this case, the specification, the requirements, and the system itself could all change, but not the problem or the world in which the system operates.

d. A database management system is an E-system because there are numerous aspects of the real world that could motivate changes in the system. The underlying hardware, other software, customer needs and wants, etc., could all change in ways that necessitate changing some aspect of the database management system.

e. A system to find the prime factors of a number is an S-system because the problem and solution can be precisely stated, the specification can precisely and completely describe the solution, and the implementation can completely fulfill the specification. No aspect of this system is likely to change.

f. A system to find the first prime number larger than a given number is likely to be a P-system because, although the solution to the problem can be completely specified, it cannot be completely implemented, practically speaking, for arbitrarily large input numbers. Thus, the system would implement a practical abstraction, or approximation, to the solution. The specification, requirements, and system implementation could all change in order to implement the actual solution better.

2. High coupling among components usually increases the number of components that must be modified to implement a particular change, and also makes it difficult to determine which components must be changed. As a result, there are more opportunities for faults to be introduced into the system as the result of a change.

3. Nearly all aspects of system "success" depend on the system's documentation. In particular, the quality of the technical documentation generated during system development will affect how well it is maintained, and whether or not the reliability of the system will degrade over time as it is changed. Also related to maintenance, the documentation needs to be easy to update, or it will become useless over time even if it was of high quality to begin with.

5. Maintenance programming is challenging because the programmer must work, to some extent, within the style and paradigm used by the original programmer, even if that style seems unnatural, or even undesirable. Maintenance programmers must have exceptional "people skills," especially communication skills, in order to effectively communicate with original programmers to determine their intent and reasoning with respect to code that is difficult to understand. Maintenance programmers must also be able to do this in a non-judgmental and diplomatic way, even if they consider the original programmer's work to be poor. Other desirable characteristics of a maintenance programmer are good analytic code reading abilities.

6. The advantage of writing documentation during development is that it is more likely to be correct (because the writer is not relying on memory). The drawbacks are that it takes time away from development, and that often a developer has a better sense of the system as a whole (i.e. the "big picture") when they are not involved in the coding of individual components.

7. Some issues to consider:

- Can you easily tell, from the documentation, what components implement each function of the system?

- Is it easy to trace the control flow of the program from the documentation?
- When changing a particular component, could you tell from the documentation which other components are likely to need changing?
- Does a large system need these types of documentation more or less than a small system? Why?

8. It's very important to retain formal test data and test scripts for maintenance in order to perform regression test after changes have been made, and also to guide the maintainers in designing new tests.

9. When a component has a single entry and a single exit, it is much easier to trace the execution steps during testing. There is only one way to get into a program, and one way to get out. If an execution trace of a program shows that a component was at least partially executed, then the tester knows that the code at the entry point was executed. And if control did not leave the component, then the tester knows that the code at the exit point did not execute. When there are multiple entrances and exits, it is extremely difficult to trace all the different possibilities.

10. Low coupling and high cohesion in a system implies a good modular structure at the component level, which would mean that restructuring would more likely happen at a higher, and easier, level, such as the subsystem level. Other types of rejuvenation would be facilitated as well because of the modular structure. However, such a structure would most likely decrease the need for rejuvenation.

    Good exception identification and handling functions might hinder redocumentation efforts because it would be difficult to understand the intent of the exception handling code if it is not already well documented. Restructuring, however, should be facilitated if the exception handling functions are easily identified and separate from the rest of the system. Reverse engineering might have some of the same problems as redocumentation, and consequently so would reengineering.

    Extensive fault handling and tolerance functionality is usually difficult to implement in separate routines, so it is often embedded in other functions of the system. This makes all types of rejuvenation more difficult because the fault handling and tolerance strategy (which might be quite sophisticated and complicated) must be understood in order to understand any of the code.

11. The Ariane-5 software is an E-system because it is subject to change in response to many types of changes in its environment. It was such a change (in hardware) that was not responded to adequately that caused the disaster.

12. The cyclomatic number allows us to order components by cyclomatic number only. That is, we can only say that one component has more independent paths than another. We cannot, with only the cyclomatic number, say that one component has higher quality or lower complexity, in general, than another. One aspect of complexity that is not captured by the cyclomatic number is interface complexity, i.e. the complexity of the data passed into and out of a component and how many other components are related to that component.

13. Yes, because a prediction model such as Porter and Selby's is not meant to be used in this way. The model only points to components that are likely to fail. It does not imply the converse, i.e. that other components will not fail.

14. Below are outlined the functional criteria and the ways in which they contribute to ease of maintenance:

    a. **Record versions or references to them** – helps the maintainer find when and where changes have been made and to track the history of a component

    b. **Retrieve any version on demand** – same as above

    c. **Record relationships** – helps the maintainer understand relationships between components that might not be obvious from the code itself

    d. **Record relationships between versions to which the tool controls access and those to which it does not** – notifies the maintainer when there are other versions that need to be taken into consideration; otherwise such versions might be overlooked

    e. **Control security and record authorizations** – prevents uncontrolled and undocumented changes to code and other artifacts

    f. **Record changes to a file** – provides documentation of all changes

    g. **Record a version's status** – allows the maintainer to keep track of maintenance progress

h. **Assist in configuring a version** – helps the maintainer to keep track of all the detailed requirements of each configuration

i. **Relate to a project control tool** – helps in managing the maintenance project

j. **Produce reports** – same as above

k. **Control releases** – helps the maintainer to document and manage all the release details

l. **Control itself** – reduces the overhead burden in using any tool

m. **Archive and retrieve infrequently-used files** – prevents the loss of information which may at some point be useful while consuming minimal resources in the maintenance of that information