

**Brooks Study Questions**

1. True\_\_False\_\_ (Brooks Chap 9): Representation (i.e., data and its structure) is the essence of programming!
2. True\_\_False\_\_ (Brooks Chap 13): In good top-down design (TDD), clarity of structure and representation makes the precise statement of requirements and functions of the modules more difficult.
3. True\_\_False\_\_ (Brooks Chap 13): In good TDD, suppression of detail makes flaws in the structure more difficult to detect and locate.
4. True\_\_False\_\_ (Brooks Chap 13): In good TDD, if the design can be tested at each level of refinement (i.e., steps), then testing can start earlier and focus on the lowest level of detail before the step is complete.
5. True\_\_False\_\_ (Brooks Chap 13): In good TDD, partitioning and independence of modules avoids system bugs.
6. True\_\_False\_\_ (Brooks Chap 14): In reducing role conflicts, when the manager knows his boss will accept status reports without panic or preemption, he comes to give honest appraisal.
7. True\_\_False\_\_ (Brooks Chap 14): Brooks states that schedule slippage in a project is a disaster due to termites [not tornadoes]. Yet day-by-day slippage is easy to recognize, prevent and makeup.
8. True\_\_False\_\_ (Brooks Chap 14): Brooks also states that estimates never really change significantly as the start time draws near no matter how wrong they turn out to be.
9. True\_\_False\_\_ (Brooks Chap 14): Schedule milestones should be flexible, and define points in time that may not necessarily correspond to measurable events.
10. True\_\_False\_\_ (Brooks Chap 14): Critical path schedules tell which slips matter, show who waits for what, and are most valuable to their users while in preparation.
11. True\_\_False\_\_ (Brooks Chap 14): Every boss needs to know 2 kinds of information a) exceptions to plan that require action, and b) status picture for education.
12. True\_\_False\_\_ (Brooks Chap 15): In a useful prose description of a program, the following elements are important: a) purpose, b) environment, c) domain and range, d) functions realized and algorithms used, and e) I/O formats, as well as others.
13. True\_\_False\_\_ (Brooks Chap 15): In a useful prose description of a program, the following elements are also important: a) operating instructions, b) options, c) exception handling, d) running time, and e) accuracy and checking, as well as others.
14. True\_\_False\_\_ (Brooks Chap 15): There are two faces of a computer program a) the face of the programmer and b) the face of the user.
15. True\_\_False\_\_ (Brooks Chap 15): 3 levels of documentation are required for 3 different types of software users as follows a) Casual user, b) Military user and c) Commercial user.
16. True\_\_False\_\_ (Brooks Chap 15): Self documenting programs are created while the code is being composed and minimizes extra work.
17. True\_\_False\_\_ (Brooks Chap 16): Brooks describes two definitions of AI as (1) The use of computers to solve problems that previously could only be solved by applying human intelligence, and (2) The use of a specific set of programming techniques known as heuristic or rule based programming.
18. True\_\_False\_\_ (Sommerville): The principle value of using formal specification techniques in the software process is that it forces an analysis of the system requirements at an early stage. Correcting errors at this stage is cheaper than modifying a delivered system.

## Brooks Study Questions

19. True\_\_False\_\_ (Brooks Chap 16): One of the most touted recent developments is the programming language Ada, a special-purpose, high-level language of the 1980s. Ada reflects evolutionary improvements in language concepts and embodies features to encourage modern design and modularization concepts.
  20. True\_\_False\_\_ (Brooks Chap 17): Harel offers a silver bullet called the Vanilla Framework and Brooks argues that software structure is embedded in 3-space, so there is at least one natural mapping from a conceptual design to a 3-dimensional diagram.
  21. True\_\_False\_\_ (Brooks Chap 17): NSB asserts and argues that software engineering developments will produce an order-of-magnitude improvement in programming productivity within ten years.
  22. True\_\_False\_\_ (Brooks Chap 18): All repairs (corrective maintenance) tend to destroy structure, to increase entropy and disorder of the system.
  23. True\_\_False\_\_ (Brooks Chap 18): One of Brooks propositions is that more programming projects have gone awry for lack of calendar time than for all other causes.
  24. True\_\_False\_\_ (Brooks Chap 19): Brooks claims that software robustness and software productivity have been hurt by the *buy and build shrink-wrapped packages as components* trends of recent years.
  25. True\_\_False\_\_ (Brooks Chap 19): WIMP stands for Why Implement Mega Programs?
  26. True\_\_False\_\_ (Brooks Chap 19): Brooks claims that Apple capitalized on the notion of incremental transition from a novice to a power user.
  27. True\_\_False\_\_ (Brooks Chap 19): One of the central issues (concerns) of software engineering today is how to maintain intellectual control over complexity in large doses.
  28. True\_\_False\_\_ (Brooks Chap 12): The target machine is always used in the development of software systems to provide the services used in building the system.
  29. True\_\_False\_\_ (Brooks Chap 12): The chief reasons for using a high level language are productivity and debugging speed.
  30. True\_\_False\_\_ (Brooks Chap 10): The two best reasons for having formal documents are to capture decisions (e.g., design trade-offs), and to communicate the decisions to other stakeholders.
  31. True\_\_False\_\_ (Brooks Chap 10): Documents for a software project should cover objectives, product specifications, schedule, budget and space allocation and organization.
- 

These questions are important and will be useful in preparing for the exam:

32. (Brooks Chap 1) What do you think Brooks is getting at with the proverb “A ship on the beach is a lighthouse to the sea.” ?

The failures of others act as the glaring light leading us down a different path toward success.

33. (Brooks Chap 1) What is the Tar Pit?

The Tar pit is a metaphor for *large system programming* (also known as Software Engineering). A place where many a fierce dinosaur (i.e., software engineer/software project manager) has struggled only to sink further into the craws of the quick-sand tar pit. Why you may ask?

## Brooks Study Questions

Because software presents a wicked problem. This story provides the perfect introduction to the later article "...no silver bullet."

34. (Brooks Chap 2) This chapter discusses the "Mythical Man-Month." What is a man-month and why is it so mythical? Also explain the point he makes with regards to Figures 2.5- 2.7.

A man-month is a unit of labor equal to the amount of work one man can do in one month. The myth is that a month and a man are somewhat interchangeable in that the total number of man-months is all that really matters. For example, 4 men working for one month is equivalent to 1 man working for 4 months or 2 men working for 2 months. This is not the case, by observation. Adding more men to a late project actually makes the project later and this effect only gets worse as people are added.

35. (Brooks Chap 3) This is one of my favorite chapters by Brooks. Please read it carefully because it should give you much insight into your project's evolution and success. How does Mills describe the chief programmer? What is the role of the toolsmith? What does Brooks mean by *uno animo*?

The chief programmer (a.k.a. "Surgeon") will personally define the functional and performance specifications, writes the code, documentation, and tests the system. He has great talent, 10 years of experience, and system and application experience. The toolsmith provides whatever tools the "surgeon" needs. "Ten people of them, seven are professionals, they are all working on the same problem, the system is the product of one mind." Uno Animo = One Mind.

36. (Brooks Chap 4): Conceptual integrity does require that a system reflect a single philosophy and that the specification as seen by the user flow from a few minds. Because of the real division of labor into **architecture**, **implementation**, and realization, however, this does not imply that a system so designed will take longer to build. Experience shows the opposite, that the integral system goes together **faster** and takes **less** time to test.

37. (Brooks Chap 3 and 4): In these articles the author comments on the notion of conceptual integrity. How is conceptual integrity achieved? There are three basic ideas (BE BRIEF).

(a) \_\_\_\_\_

Separation of architectural *effort* from implementation (is a very powerful way of getting conceptual integrity on very large projects).

(b) \_\_\_\_\_

Surgical team: Surgeon and copilot are each cognizant of all the design and code (which ensures the conceptual integrity of the work; the lack of division of the problem and the superior-subordinate relationship make it possible for the surgical team to act *uno animo*).

(c) \_\_\_\_\_

In the surgical team, there are no differences of interest and differences of judgement are settled by the surgeon unilaterally which enables the team to act in *uno animo*.

38. (Brooks Chap 5) In your own words (please) what is the second-system effect?

The second-system effect is the heavy gold-plating of your second design after the very cautious and successful first design. It usually results in a product that is too expensive and contains featured not really used.

**CS 422 Software Engineering Principles**  
**Brooks Study Questions**

**Name:** \_\_\_\_\_

39. (Brooks Chap 6): The manual is the external **specification** of the product. (Hint, the style must be precise, full, and accurately detailed.)

40. (Brooks Chap 6): Brooks describes the merits and weaknesses of formal definitions. Name two strengths and two weaknesses:

Strengths:

Weaknesses:

1. \_\_\_\_\_  
 \_\_\_\_\_

1. \_\_\_\_\_  
 \_\_\_\_\_

**Strengths: Precise and unambiguous and more verifiable**

**Weaknesses: Lack comprehensibility and lack of trained practitioners**

2. \_\_\_\_\_  
 \_\_\_\_\_

2. \_\_\_\_\_  
 \_\_\_\_\_

**Strengths: Complete: gaps are more conspicuous and Consistent: ability to reason**

**Weaknesses: Cost to project initially is higher and difficult to learn**

41. (Brooks Chap 7): The tower of Babel was perhaps the first engineering fiasco, but it wasn't the last. Communication and its consequent, **organization**, are critical for success. The techniques of communication and organization demand from the manager much thought and as much experienced competence as the software technology itself.

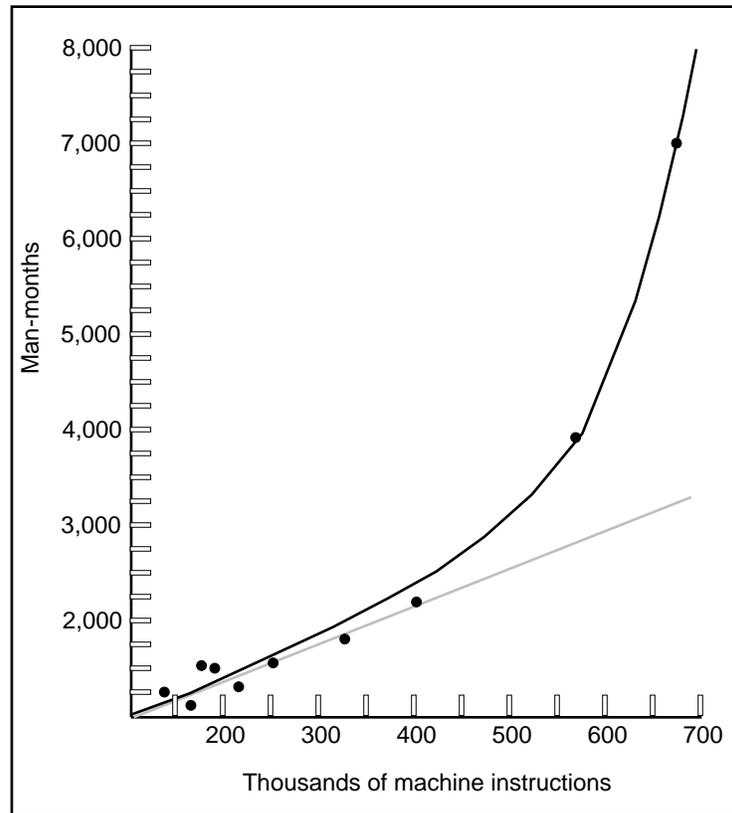
42. (Brooks Chap 7): The means by which communication is obviated are division of labor and specialization of function. The tree-like structure of organizations reflects the diminishing need for detailed communication when division and specialization of labor are applied. Brooks considers a tree-like programming organization, and examines the essentials which any sub-tree must have in order to be effective. He lists 6 *essentials*. Choose six from the list below.

a. \_\_\_\_      b. \_\_\_\_      c. \_\_\_\_      d. \_\_\_\_      e. \_\_\_\_      f. \_\_\_\_

**1, 2, 4, 8, 10, 12 (in any order)**

- |   |  |
|---|--|
| (1) Division of labor                     | (7) Separation of architecture from implementation |
| (2) Mission                               | (8) Technical director or architect                |
| (3) Trained practitioners                 | (9) Concrete milestones                            |
| (4) Schedule                              | (10) Interface definition among the parts          |
| (5) Precise and unambiguous specification | (11) Conceptual integrity                          |
| (6) Supplier                              | (12) Producer                                      |

43. (Brooks Chap 8): The following graph (Figure 8.1) is related to an equation that is used to determine the effort needed to complete a given software project. This graph show the general relationship described by the data of Portman, Aron, Harr, and OS/360.



(Fig. 8.1 in Brooks) Programming effort as a function of program size.

The following questions are generally based on this relationship.

a. Figure 8.1 shows the equation:

$$\text{Effort} = (\text{constant}) * (\text{\# of instructions})^{1.5} \text{ \{fill in the blank\}.}$$

- b. The 4 data sets which are discussed (i.e., presented by Portman, Aron, Harr, OS/360) all confirm striking differences in **productivity** related to the complexity and difficulty of the task itself.
- c. Corbato of MIT's Project MAC reported a mean productivity of 1200 lines of debugged PL/1 statements per man-year on the MULTICS system. That data seem to be comparable in terms of kind of effort included. However, Corbato's numbers are in *lines* per man-year, not *words*! Each statement in his system corresponds to about 3-5 words of handwritten code! This suggests two conclusions:
- Programming seems **constant** in terms of elementary statements, a conclusion that is reasonable in terms of the thought a statement requires and the errors it may include.
  - Programming productivity may be increased as much as five times when a suitable **high level** language is used.

44. (Brooks Chap 9): Since size is such a large part of the user cost of a programming system product, the builder must set size targets, **control** size, and devise size **reduction** techniques, just as the hardware builder sets component count targets, **controls** component count, and devises count **reduction** techniques. Brooks continues to relate a OS/360 experience and concludes the following moral: Set total size budgets as well as **resident space** budgets; set budgets on backing-store accesses as well as on size. The second moral is subsequently described which calls for defining exactly what a module **must do** (two words) when you specify how big it is.

45. (Brooks Chap 15), Test cases fall into 3 parts of the input data domain. *The first type are the mainline cases that exercise chief functions.* The other 2 types of test cases, by their nature, are *barely Legitimate*. Name & describe the other two types in terms of their purpose (or intent).

(Second Type) *Barely legitimate* cases that probe the edge of the input data domain, ensuring that largest possible values, smallest possible values, and all kinds of valid exceptions

(Third Type) *Barely legitimate* cases that probe the domain boundary from the other side, ensuring that invalid inputs raise proper diagnostic messages.

46. (Brooks Chap 16): What is the silver bullet that Brooks is talking about in the article "No Silver Bullet: Essence and Accidents of Software Engineering."

This bullet is the proverbial solution to end all problems known to be the most difficult hurdles in the field of software engineering (i.e., cost, quality and timely delivery [or productivity]). One example may be, perhaps the silver bullet is to be found by eliminating the errors at the source, in the system-design phase?

47. (Brooks Chap 16): Based on the article "No Silver Bullet: Essence and Accidents of Software Engineering," describe the meaning of the following items:

(a) Expert Systems \_\_\_\_\_

An expert System is a program that contains a generalized inference engine and a rule base, takes input data and assumptions, explores the inferences derivable from the rule base, yields conclusions and advice, and offers to explain its results by retracing its reasoning for the user.

(b) Automatic Programming \_\_\_\_\_

The generation of a program for solving a problem from a statement of the problem specifications. Most successful so-called systems have the following favorable properties: (1) problems are readily characterized by relatively few parameters, (2) There are many known methods of solution to provide a library of alternatives, and (3) extensive analysis has led to explicit rules for selecting solution techniques, given problem parameters.

(c) Graphical Programming \_\_\_\_\_

The application of computer graphics to software design (e.g., a mistaken assumption for instance is that flowcharts are the ideal program design medium and thus providing powerful facilities for constructing them).

(d) Program Verification \_\_\_\_\_

Does not mean error-proof programs (i.e., mathematical proofs also can be faulty). This powerful notion is concerned with the development of formal (e.g., algebraic) specification of

**Brooks Study Questions**

the program and then proving that the program meets its specification. The hardest part of the software task is arriving at a complete and consistent specification and much of the essence of building a program is in fact the debugging of the specification.

48. (Brooks Chap 11): The fundamental problem with program maintenance is that fixing the defect has a substantial (20-50%) chance of introducing another. So the whole process is two steps forward and one step back. How can we begin to avoid such side affects?

Page 122 Methods of designing programs so as to eliminate or at least illuminate side effects can have an immense payoff in maintenance costs. So can methods of implementing designs with fewer people, few interfaces, and hence fewer bugs.

49. (Brooks Chap 11): Systems program building is entropy-decreasing process, hence metastable. What is program maintenance in contrast to this thesis?

Page 123 Program maintenance is an entropy-increasing process, and even its most skillful execution only delays the subsidence of the system into unfixable obsolescence.

50. (Brooks Chap 20): What are the three main questions that Brooks addresses here that have been given to him by his readers 20 years after?

Page 255 What do you now think was wrong when written?

What is now obsolete?

What is really new in the software engineering world?

51. (Brooks Chap 20): What example does Brooks give about enforcing architecture to provide conceptual integrity across an application?

Page 260 He gives a metaphoric example of Windows (WIMP interface).

52. (Brooks Chap 20): What is the problem with Microsoft's "Build Every Night" approach?

Page 270 It is really hard. You have to devote lots of resources, but it is a developed process, a tracked and known process..

53. (Brooks Chap 20): What are the distinctive concerns of software engineering today (as set forth in Chapter 1) but clearly enumerated in Chapter 20?

Page 288 How to design and build a set of programs into a system

How to design and build a program or a system into a robust, tested, documented, supported product

How to maintain intellectual control over complexity in large doses.

37. How true is Brooks law? \_\_\_\_\_.

Page 274 "Adding more people to a late project always makes it more costly, but it does not always cause it to be completed later."