Tutorial 2 Answers Comp319 Software Engineering

Question 1

Why does Brookes recommend such a large amount of the schedule (50%) is dedicated to testing and debugging?

Brooke's experience taught him that the testing and debug phases are the ones most likely to be underestimated. Failure to schedule for testing will lead to the bad news about overruns coming at the end of the project. One of the reasons for this is that the amount of testing required is very dependent on the quality of the code and modification of code to fix bugs very often leads to new bugs being introduced (often regression bugs). At the point of testing the project is usually fully staffed, this results in very high levels of increased costs.

What happens often in practise is that the schedule is partly fulfilled by shipping the product before the testing has been properly tested, this can have other impacts on the reputation of the organisation as possible financial/reputational loss the customer.

Question 2

What are the alternatives when a project is lagging behind?
Re-staffing
Assume that the underestimation relates to only the task completed, try and add enough staff to complete the project on time, assuming adding new staff can complete the work proportionately. i.e. double head count, reduces time to complete by 1/2.

Assume that the underestimation relates to all of the task, try and add enough staff to complete the project on time.

Example

A project is scheduled to take 3 months with 10 staff and is broken into 3 separate phases. This means the total effort is estimated to be 30 staff months.

Phase 1 is runs 50% over-schedule (about 6 weeks to complete) 1.5 months.

Assumption 1, only task 1 is under estimated , this means the remaining work is 20 staff months. The remaining time is 1.5 months, so the number of staff required is 20/1.5 = 13.33 i.e. 14 staff.

Assumption 2, the whole project is under-estimated, this means the total effort remaining 20 x 1.5 = 30 staff months. To complete 30 staff months in 1.5 months 30/1.5 = 20 staff, i.e. double the number of staff.

In general re-staffing might help and might not, in general it very much depends on the partitioning of the tasks to complete and the interdependency. In general re-staffing will only help projects that are under-staffed in the first place.

Re-scheduling
Re-writing the schedule and changing the deadline to take into account the under-estimation. So to put enough time in the schedule to make sure the work can be done properly.

Trimming the task, reducing the scope
In practise this is very common, this is hard however if the priority of the requirements has not been defined clearly by all stakeholders, and very often many parties have competing interests in the final product.

Question 3

Why does simply adding extra staff to a project that is over-running not always improve its delivery time?

New staff will have to be trained up to understand the system, this will take the time of the staff for are already developing the code. Re-partitioning any tasks will also require time from the original team. There is always a limit to the amount of partitioning that is possible with any system, and in practise there is a base limit to how many new staff can help with development. If resources are very plentiful (for example in the case of some open source projects) it is possible to improve efficiency by the application of developers as pair programmers, this may reduce the number of bugs and allow 2 programmers to work on the same piece of code, due to shared understanding.

Question 4

Which phases of development are more/less sensitive to the addition of extra staff?

Specification and analysis can be broken into parts for at least the detailed documentation phases. So for example if part of the specification was compliance with the Payment Card Industry Data Security Standard, this could be added by one member of staff, which another was working on other aspect of the core requirements. Design itself can be broken between data design and code design, however most design is data driven any. Aspects of the design could be partitioned based on the major sub-systems of the code.

Implementation in theory can be partitioned class by class, but again it may be difficult to progress one 1 class until another is completed. One possibility is for programmers to write stubs for their classes to assist other programmers in their development.

Finally the testing and debug phase. In general testing is an ideal task for partitioning and can be farmed out to many individuals. So often all stakeholders, developers, project leaders and customers can be involved in testing and ideally should be. The constraint however can be how many developers will be knowledgeable to debug a particular element, this is where techniques such as pair programming are particularly useful, as they share the understanding of the code between developers.

Question 5

Comment on each of the following tasks, how many staff do you think you could add to reduce their timescales? (For each example comment on the specification, design, implementation and testing phases). How easy is it to partition the tasks. Comment on the efficiency of the solution.

a) Specifying, developing and testing a language editor that would work for the following languages: Java, C#, Javascript and C++. It would have features such as version control and automatic saving and change tracking.

Specification could split into core functionality and separate elements for each language. So in theory could be split into Java module, C# module, Javascript module and C++ module, version control module, automatic saving code and change tracking code. This could be split 7 ways in theory, for development, using pair programming you could use 14 staff to develop, 1 pair for each module. This might not be the most efficient use of resources. For example it would probably make more sense to write a generic module for tokenizing and grammar checking and use a different template for each language. So some decreases in timescale might involve poorer decisions on implementation. In testing it is possible to throw as people skilled people at the problem as you can, so testing is resource sensitive. However once testing uncovers bugs, you need people who understand the code to debug it, the use of pair programming might help, since each piece of module code could be debugged by 2 programmers.

b) Implementing an encryption algorithm from a standard specification in both C#, C and Java. No significant specification phase.
The code could be written in 1 of the languages and then ported into the others.
The code could be split 3 ways and written independently.
Writing the code independently would perhaps lead to a shorter timescale, but could result in lower quality, since simply porting the code would give less chance for logical errors to be introduced in the interpretation of the specification.

c) Porting a class library containing 100 class definitions from Java to C#
Could theory be split across 100 programmers, but this would depend on coupling between the classes. In the testing phase it might be hard to co-ordinate so many developers together due to the large number of paths of communication (in theory up to 4950).

d) Developing a game of chess which will be web based and use html5 and javascript and be able to provide user to user chess as well as user against the computer chess. It should also have a login system as well as a high score facility.
Split the code into chess logic, HTML5 presentation and server side login and then use 3 analyst programmers to complete the work as sub-modules to be integrated later. Testing such a system properly may present difficulties, since developers may not have the proficiency in chess to be able to test it properly, even if they can develop a game engine which performs rule based position analysis.