Test 2018
(https://drive.google.com/drive/folders/1Nuu-EE5vSVuRSLmiPpquI5JuLH4t6GPj?fbclid=IwAR16nIobORT3OtsPKFxV6R0ubHyWtv_BO-HiEFmOd0nF6fjrK01b8AOpy7A)

1. IEEE defines software engineering as "the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software".
Which concerns are in your opinion the most important in software engineering? Why?

d) defining and enforcing good values, principles and practices capable to produce good software.

If good values, principles and practices are being applied, the end product is sure to be good. These points allow for better communication, understanding and better team work. No matter what developing technique, team or product being developed, if the developers have been taught the same practices, they are sure to understand each other's work and work well together, result in a better end product.

2. RUP is not a prescriptive software development methodology, but instead a framework. What do you think was the motivation from its authors?
Elaborate what can vary between instances of RUP.

d) Each software development project has its own intricacies. Being a framework enables RUP to be adapted to several projects without imposing strict rules that

would become inefficient for some contexts.

The RUP is a framework, so it only describes the phases a software development process project should undergo. What can vary between instances of RUP are how each phase is implemented. The goal with RUP is to guide the developers on which steps that must be taken but leave up to them on how they should take those steps.

3. eXtreme Programming is praised by its software development efficiency.
Pair Programming is one of the practices contributing to said efficiency.
What does pair Programming consist of? Briefly detail the practice using your own words.

c) One developer working on a feature, while being continuously reviewed by another, often trading their roles.

Pair Programming consists on two programmers working together at one machine.
Driver enters code while navigator critiques it. They periodically switch roles and pair. This servers as an informal review process, helps developing collective ownership and spread knowledge and Improves quality (less defects, better design), whilst maintaining (or improving) productivity.

4. No scrum for us

5. Requirements documents have different target audiences, from customers to engineers. What should a good software requirements document include?

c) the functional and nonfunctional requirements in detail.

The functional and nonfunctional requirements are the most fundamental to be included in a software requirements document since those are the actual specification/characteristics for the product being developed. Although system models are quite useful and can be included too, the functional and nonfunctional requirements are what's indispensable since they allow both customer and developing team to be sure of what is being agreed to, concerning what the system should do.

6. software engineering has been a research subject for the past decades. Focusing on software architectures, engineers have learned much about how to build good software. What technique/approach is typically adopted to share such knowledge between engineers? Describe why is it a good technique/approach?

c) Design Patterns.

Patterns are a means of commonly used in software engineering of representing, sharing and reusing knowledge. A pattern describes a proven solution for a recurring problem. A good design pattern is a stylized description of a good architectural design practice, which has been tried and tested in different

environments.

7. Legacy systems are older software systems, developed using obsolete software and hardware technologies, that remain useful for a business. It is often cheaper and less risky to maintain a legacy system than to develop and a replacement system using modern technology. What do you think is the main reason for this?

c) The business value of the legacy system still outweights the technology value.

The business value of a legacy system and the quality of
the application should be assessed to help decide if a
system should be replaced, transformed or maintained.
Training the employees to work with a new system can be very difficult.
Also, developing a software from scratch can be extremely difficult and expensive. So if the legacy system works and doesn't have major problems it doesn't make sense to replace it in one go.

8. Test-driven development promotes smaller, terse and more balanced codebases. What activities/benefits do you think contribute the most to this outcome?

a) Test-first and refracturing.

A Test-driven Development and refracturing are usually related.
They promote a simple tested design.
A test-first methodology helps ensure the developer will create only the necessary code to pass the tests. This way there's a smaller chance the programmer will make unnecessary code and this make a larger codebase.
With this in mind, at each new test, the code addition should be minimal, so refactoring is encourage as a way to, at each step, review what has been done and possibly restructure the code in a better way according to well known refactor techniques, contributing again to a smaller and more balanced codebase.

Other Questions

5. Requirements engineering refers to the process of defining, documenting and maintaining requirements in the software engineering process. There are different kinds os activities involved, and different types of requirements.

Other important characteristics of the process of requirements engineering are:

c) Requirements engineering is very valued in waterfall-like processes, being always the first phase of the development process. More recent processes assume the importance of requirements engineering along all the process, even at usability studies phases.

The first two stages of a waterfall-like process are dedicated to requirements engineering (system and Software requirements). In the case of more recent

processes like agile methods, they usually use incremental requirements engineering
and may express requirements as user stories.

The A option is wrong. The flow of requirements engineering is:
1. Requirements elicitation; 2. Requirements analysis and negotiation; 3. Requirements specification; 4. Requirements validation and 5: Requirements management. So the last step is management not validation.

The B option is wrong because both system and user requirements are functional.

The D option is wrong because engineering requirements are not ignore in agile methods. Rather the system requirements are considered a waste of time since they
change so quickly. Software requirements are considered very important and are usually expressed via user stories through the development.

4. No scrum --

4. No scrum --

7.

It is common knowledge in Software Engineering that "software must evolve or it will die". Evolving software is thus a key practice to ensure software longevity. What do you think is a major factor of entropy to evolving a system?

(entropy = chaos = change)

b) The change proposals.

The Software is always evolving: always changing in response to changing customer needs. This stage will cause a major entropy to an evolving system since requirements are in constant change and the project and its team must adapt frequently to (sometimes drastic) changes.

8. Acceptance testing is a user testing process where the aim is to decide if the software is good enough to be deployed and used in its operation environment. But doing acceptance testing is not easy and straightforward, mainly because:
c) It must cover the needs of all stakeholders.

Acceptance Tests are specified by the customer to check overall system functioning. Since they are specified by the customer is hard to check if they cover the needs of all stakeholders.

2. RUP is composed by several engineering disciplines. Analysis and design is one of those disciplines. What roles are required by this discipline? Elaborate what they use as input and what is the output of their work.

d) Software architects and designers.

The architects perform an architecture an architectural analysis reviewed by the designers through a use-case analysis. Then the architects use that input to do the architecture design, describe concurrency and distribution.
The architects review the architecture design and send it to the designers that implement the subsystem and class design. AFter this the use-case design is developed based on the subsystem and class design and the design is reviewed by the team.

3. eXtreme Programming is an iterative methodology that values simplicity and
adaptability to change. Despite its advantages, it is not ideal for all software

projects. Choose which of the following projects you believe XP would not
applicable,
justifying your choice.

c) A research project for an heart implant.
d) A large scale micro-service cloud application.

eXtreme Programming is an agile method. Agile methods may be
problematic for systems that require pre-delivery analysis
(e.g. critical systems) or systems developed by several
teams.
Since option d) is for a large scale, there are several teams working
together. If there isn't a strict plan, each team won't know exactly what
they are supposed to develop and critical problems can occur.
The software for option c) must be extremely reliable (it is a critical
system) since it is for an hear implant. Any bug in the system would cause
catastrophical consequences in the end-user/stakeholders.

4. The agile manifesto defends:
Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

d) A less bureaucratic approach to develop software by bringing clients
and development team to work better and together.

Agile methods focuses more on the people rather than the
process and tools. In agile processes, planning is incremental and
it is easier to change the process to reflect changing customer
requirements. The clients can give feedback to the developers
during the developing phase, and the changes they request can be implemented
easier than in plan-driven approaches. This creates a positive culture
between both parties resulting in a better software.

1. Software is getting extremely pervasive in our life, and at the same
time is getting harder to build with quality and efficiency. Since many
years ago, the computing power had outpaced the ability of software
engineers to effectively use those capabilities, resulting in many problems.
Which are in your opinion the problems that were never solved up until
our days?

a) Projects running over-budget, over-time and unmanageable.

Still is this day and age the time and resource variables are one of the
most difficult to adjust.
It is very difficult to adjust the staffing:
â□¢ Staffing increases have long lead times.
â□¢ Increased intensity has diminishing returns.
â□¢ Team culture requires some degree of stability.

The time variable is the most painful variable to adjust:
â□¢ Early commitments are usually date-based.
â□¢ Target dates are often the most important
objective.
â□¢ Within a date boundary, thereâ□□s only so much
time.

Usually it is very difficult to adjust components that are directly
connected to the people aspect of a software development project. People
are unpredictable and unique. A style that works for a person doesn't work

for another.

2. The Rational Unified Process (RUP) was introduced in the early 2000s. Choose the best description for RUP and shortly elaborate about it in your words.
a) RUP is an interactive software development process framework.

RUP is a software development process framework which is divided in cycles, phases and iterations. The functionality of the system is delivered after a series of releases of increasing completeness. In each iteration, different requirements are selected to be developed, providing new releases every iteration. Therefore, we can conclude that RUP is an interactive Development process framework.

7. What is the main cause for software degradation?
c) Poor Refracturing.

Bad refactoring leads to an unmaintainable code base. Changing our adding new features is difficult because the code may be duplicated and overall with a bad design leading to bugs and broke code.

9. Software project management is concerned with activities to ensure that software is delivered on time, on budget and in accordance with the requirements of the organizations developing and procuring software. One of the activities of project management is planning the software releases, which features to deliver and when.

Which of the following do you consider the best practice to plan a software project?

b) The developers estimate the time to implement a set of features.

The team reestimates the work effort based on the more detailed assessment and adjusts the features planned for the iteration if necessary.
By doing that, the manager can more accurately estimate the required time for the next release and plan out with the team the feasibility of said features.

5. Requirements elicitation and analysis include discovery, organization, and specification of requirements. It seems a paradox, but it is common to say that stakeholders don't know what they really want or don't know how to express it, and usually do not agree, creating requirements conflicts. Which of the following practices do you consider to be important to do for an effective requirements elicitation?

c) Specify the requirements as use cases, or user stories, to illustrate all possible iterations with the system.

Establishing the system requirements involves technical staff working with costumers or other stakeholders to find which serveries/functionalities should the system provide, the application domain, the work environment and operational constraints. Use cases or user stories are easy to comprehend by the stakeholders and they can give their feed back. This will avoid conflicts provided by language or technical formats not commonly used by stakeholders.

8.