Seguridad para IS

The ACM & IEEE curriculum guidelines for SE includes three main pillars in the area of security, which are: “Security fundamentals”, “Computer and network security” and “Developing secure software”, which in total are only 20 hours of the curriculum. In this very same, is established that this academic training must satisfy two distinct but related components, as a standalone knowledge area: “It deals with the protection of information, systems, and networks” and also, as a crosscutting concern: “It provides a focus on how security must be incorporated into all parts of the software development life cycle.”.

So, I would say that at the very least, the curriculum of a software engineer must include security concepts that allow him to develop secure systems and the ability to spot areas where a system has security deficiencies. The ones I would consider the most relevant are: network related, like encryption, authentication, secure database access; and secure design, like verification, validation, and secure design principles and patterns in general; thus satisfying both components mentioned previously.

References:

ACM & IEEE (2015). Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. Retrieved from class materials (ACMIEEECurruculaSE2014.pdf).

Políglotas

Since nowadays there is a constant emergence of new programming languages, I think a fundamental skill of a programmer is to be able to easily adapt to new languages, so the strategy I would use is not to focus on the syntax the languages, but to study and identify universal patterns of programing, as well as spotting the peculiarities of the language that I currently using that would not apply in others, so that you do not make the mistake of using this new language incorrectly, similar to what Hunter M. (2020) mentions “To become fluent in a new programming language, you can’t code with it as if it was a language you already know. If you do, the likelihood is that the language won’t necessarily understand what you’re asking it to do, or it will do what you’re asking in a very inefficient manner. In other words, don’t repeat my mistake — use the new language in the way it was intended”.

From there I would read the documentation on how to carry out basic tasks with this new language, learn about its peculiarities and how it is generally worked with, and spot the similarities with your previous knowledge, that is, recognizing patterns of things you have done before. And finally start very small projects that I know are within my reach, for example, a project that I already I have done before in the language I already known. All this with the purpose of familiarizing myself with the language and making this process as smooth and efficient as possible.

References:

Hunter M. (2020). How to Become Fluent in Multiple Programming Languages. Towards Data Science. Retrieved from https://towardsdatascience.com/how-to-become-fluent-in-multiple-programming-languages-9f473c146b90

CU vs US

“Although there are some similarities between User Stories and Use Cases, User Stories and Use Cases are not interchangeable; both User Stories and Use Cases identify users and they both describe goal, but they serve different purposes”. So, knowing that a Use Case Specification is simply the one which provides textual details of a Use Case (IBM, 2021) and putting aside the obvious difference (the way they're structured), the differences I would highlight is their purpose and extension, Use Cases Specification focus on the functionality of a feature in addition to having a greater amount of detail, while User Stories focus on justifying the existence of a feature, defining the for what, the for whom and the why of this functionality, as well as regularly being shorter in content.

So, we can say that Use Cases Specification is more recommended for instances where a greater amount of detail and a more structured approach is required; and on the other hand, User Stories stand out more where more discussion and flexibility is required, in addition to being more suitable for shorter and simpler requirements that could be completed in a single sprint (since they are much shorter than a Use Case Specification).

References:

Visual Paradigm. User Story vs Use Case. Retrieved from https://www.visual-paradigm.com/guide/agile-software-development/user-story-vs-use-case/

IBM (2021). Descripción de especificación de caso de uso. Retrieved from https://www.ibm.com/docs/es/elm/6.0.2?topic=cases-use-case-specification-outline

Métodos y Técnicas de Pruebas

One of the most effective techniques to include tests in the development process is known as "Unit testing" which basically consists of the testing of individual units or components of a software, this with the purpose of validating that each of this performs as expected (Thomas Hamilton, 2022). This is technique is very effective and flexible since a “unit” to be tested may be an individual function, method, procedure, module, etc., which in any of these would notify us if any changes made to the code base causes unexpected results.

As to which stage tests can be considered, Software Testing Help (2022) tells us that “Software testing should start early in the Software Development Life Cycle. This helps to capture and eliminate defects in the early stages of SDLC i.e requirement gathering and design phases. An early start to testing helps to reduce the number of defects and ultimately the rework cost in the end”, so personally I think that unit testing and testing in general should be done as early as possible in development (specially in big code bases or teams), and unit testing couldn't be a better example of the importance of this, since this being the test of the individual components that in conjunction make a system work, we cannot expect a system to work if its components are not working properly at all or they have problems in certain conditions that were not considered.

References:

Thomas Hamilton (2022). Unit Testing Tutorial – What is, Types & Test Example. Guru99. Retrieved from https://www.guru99.com/unit-testing-guide.html

Software Testing Help (2022). What is Early Testing: Test Early, Test Often BUT How? (A Practical Guide). Retrieved from https://www.softwaretestinghelp.com/early-testing/