Questão B14

"Quanto mais cedo se encontrar o problema no software, melhor. E há dois tipos de problemas de software. Um deles são os *bugs*, que são problemas de implementação. O outro são falhas de software -- problemas de arquitetura, no desenho. As pessoas prestam muita atenção aos bugs e não o suficiente às falhas." In: Gary McGraw, https://www.computerworld.com/article/2563708/q-a--quality-software-means-more-secure-software.html

(Original English text: "The earlier you find the software problem, the better. And there are two kinds of software problems. One is bugs, which are implementation problems. The other is software flaws -- architectural problems in the design. People pay too much attention to bugs and not enough on flaws.")

Explique como é que uma estratégia de garantia de qualidade do software pode endereçar as preocupações expressas pelo autor.

McGraw expresses that there are two types of software problems, ones related to the more technical implementation of the software, and ones that pertain to fundamental design flaws in the core design and architecture of the software. He also focuses on the problem that developers commonly pay more attention to bugs rather than design flaws. To advocate as to why software quality assurance strategies can tackle these issues, we should first start by defining what **software quality** is. SQ is the degree to which a software product satisfies stated and implied needs. As such, **SQA strategies** are the plans, the strategic vision, set into motion by the software's development team to assure that their product meets the desired quality standards. They consist in a set of rules and practices that will command the entire development process from inception to completion. It should define factors such as the technology stack, requirement collection (with user-stories, personas, …), non-functional requirements, CI/CD practices, design and code standards, testing strategy, process automation, DoD, quality metrics and so on. They are basically guidelines that the entire team follows to maintain a certain level of quality and to know how to build the product.

McGraw professes that the sooner we find problems in our systems the better, and truly, by defining an appropriate SQA strategy, which should include plans for appropriate testing as well as requirements gathering, teams can better brace and more quickly detect them. Bugs-wise including a testing plan in the SQA strategy forces developers to drop the *"I'll fix it later"* mentality in favour of a test-driven mindset leading to them writing, not just code, but code that is *assured* and has been tested to work. An SQA strategy does not prevent the creation of mistakes, but rather it shortens the time considerably between their appearance, discovery, and treatment. This applies not only to code, but also to problems related to the core design architecture. SQA strategies force developers to think about their product's personas, it forces them to think about the user stories, it forces them to think about the product **not just in terms of code**. With this, design decisions that go against the product specifications, set by thinking about the end-user's requirements, (alongside more technical errors) are more easily and quickly found, and more hastily and cheaply treated.

A good SQA strategy is going to force developers to care not only about **verification**, as in, whether the system is working and being built with good practices and without bugs, but also about **validation**, whether the system is being built in accordance to the user's needs and requirements. This ultimately prevents McGraw's fear of developers focusing too much on bugs whilst neglecting the validity issues caused by having fundamental problems in the design architecture, which go against what the product's value for the users. It should also be noted that without an SQA strategy, teams could very well focus on either verification or validation or even outright neglect both. In the end, what developers want to, is to build as robust and valuable a product as possible with **enough quality** to make it successful whilst saving as much money as possible. Trying to create the product to these specifications, without defining what *"enough quality"* means would be nigh impossible, and this is something that SQA strategies also aim to help with. They allow us to organize our work by keeping in mind metrics that measure the quality we need to aim for, they prevent developers from focusing too much on the technical aspects whilst neglecting the more design oriented issues, they help us detect problems as fast as possible, by defining appropriate testing plans, thus avoiding the high costs of performing problem-fixes late in development and maintenance. SQA strategies are paramount to nowadays software development for a good reason and not defining one would be comparable to trying to drive through an unknown area without a map. You **might** still reach your destination, but you will end up taking more wrong turns, burning more fuel, making the trip more expensive. And in the end, you might realize you ended up in the wrong place.