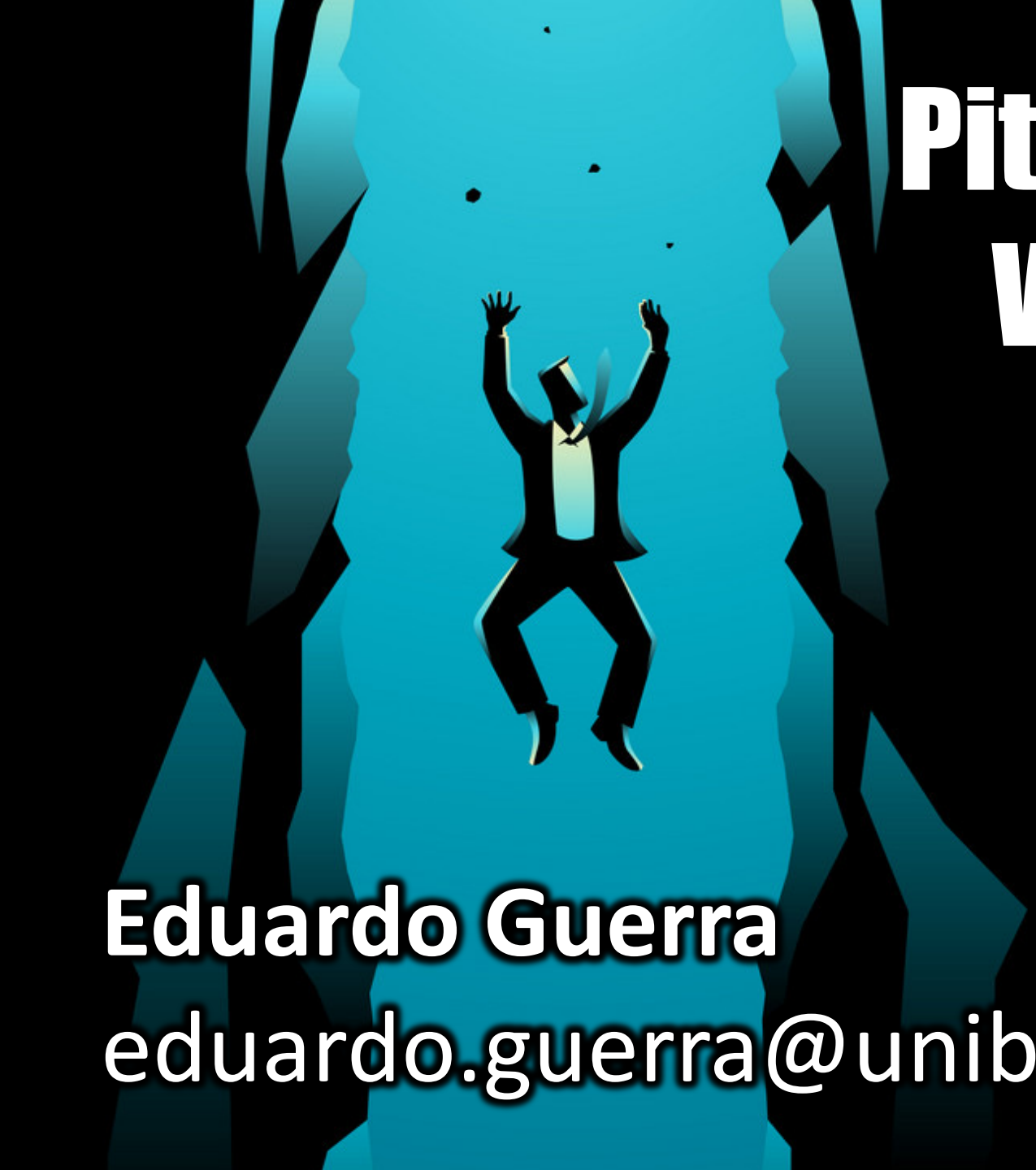
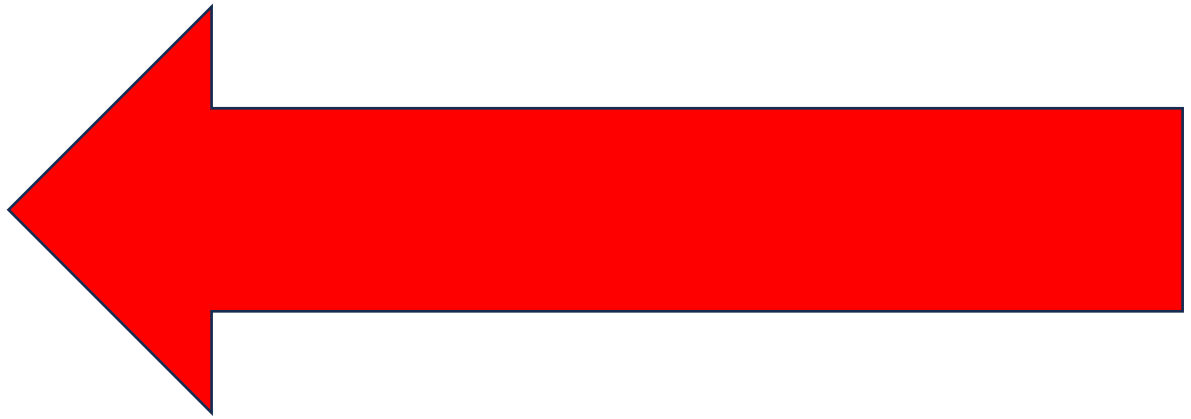


Pitfalls and Mistakes When Dealing With Non-Functional Requirements

Eduardo Guerra
eduardo.guerra@unibz.it


non-functional
requirements
extra-functional
requirements
quality
attributes

the ugly duck...



We need that duck!

An Empirical Study about the Instability and Uncertainty of Non-functional Requirements

Luiz Viviani[1], Eduardo Guerra[2][0000−0001−5555−3487], Jorge Melegati[2][0000−0003−1303−4173], and Xiaofeng Wang[2][0000−0001−8424−419X]

[1] Institute for Technological Research, Brazil
luiz.viviani@gmail.com
[2] Free University of Bozen-Bolzano, Italy
eduardo.guerra@unibz.it, jorge.melegati@unibz.it, xiaofeng.wang@unibz.it

Abstract. Managing non-functional requirements (NFRs) has been a challenge in software development for many years. These requirements are typically used to make important architectural decisions early in the project, which can be problematic if they are uncertain or unstable. When this uncertainty is not considered when designing the software architecture, changes are often costly and sometimes even unfeasible. Some empirical studies on the subject have already been carried out, but few have focused on the perspective of professionals with extensive experience on the changes and uncertainties of NFRs. This work aims to expand the understanding about the management, clarity and validation of NFRs to fill this gap in the literature. To achieve this goal, a survey was carried out with professionals to find out how NFRs were managed and validated. For the research design, instead of generic questions, the questionnaire focused on some specific types of NFRs to induce participants to recall and report concrete situations. As a result, 40 valid responses were obtained, most from professionals with more than 10 years of experience. The results reveal that a significant number of NFRs were defined after the delivery of software increments (more than 30%) and that revision and change occurred in about a third of the NFRs. Hence, this study presents evidence that NFRs, as the functional ones, can also be uncertain and change frequently, requiring agile approaches and techniques to evolve the software architecture to consider this uncertainty.

Keywords: Non-functional requirements · quality attributes · software maintenance · software evolution · requirements engineering · empirical study

d an important factor in soft-
hitectural decisions
[28], when

Anti-patterns in managing uncertain Non-Functional Requirements

Luiz Viviani
luiz.viviani@gmail.com
Institute for Technological Research
São Paulo, Brazil

Jorge Melegati
jorge.melegati@unibz.it
Free University of Bozen-Bolzano
Bolzano, Italy

Eduardo Guerra
eduardo.guerra@unibz.it
Free University of Bozen-Bolzano
Bolzano, Italy

João Daniel
joao.daniel@alumni.usp.br
Free University of Bozen-Bolzano
Bolzano, Italy

ABSTRACT

Managing non-functional requirements (NFRs) is complex and has been challenging over the years. These requirements are typically used to make important architectural decisions early in the project, which can be a problem if they are uncertain or volatile. Identifying and demonstrating the existence of anti-patterns associated with the negligence and volatility of NFRs is a topic that deserves attention. This study identified five anti-patterns associated with managing NFRs and the volatility of these requirements. Based on a survey applied to 40 professionals with more than 10 years of experience, data were obtained from 144 NFRs of real projects, and dysfunctions were identified. The mapped anti-patterns functions associated with: incorrect perception of needs (UNCERTAINTY); delayed elicitation of NFRs (LATE IDENTIFICATION); immutable NFRs (FROZEN NFR); projects that include validation of feasibility and impacts (FULL PACKAGE); management for all NFRs (INVARIANT MANAGEMENT).

anization → **Embedded systems**; *Re-*
orks → Network reliability.

nti-pattern, volatile, negligence

i, and João Daniel. 2018. Anti-
al Requirements. In *Proceed-*
of Programs (EuroPloP'23).
e/XXXXXXX.XXXXXXX

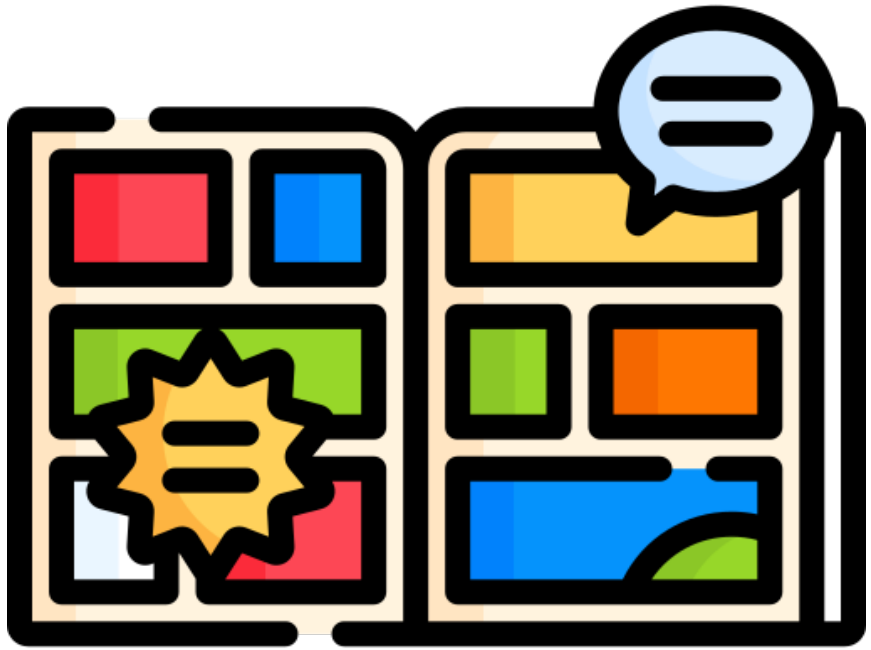
1 INTRODUCTION

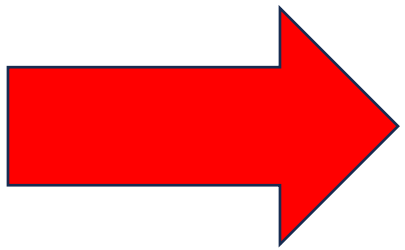
Non-Functional Requirements (NFRs) are considered an important source for software architecture decisions. However, the most important architectural decisions generally are made early in the project, even in agile projects, when some NFRs are uncertain [45]. As changes in software architecture are often expensive when your structure is not prepared to deal with them, late knowledge of new NFRs can represent a great risk for the project [25]. As many requirements (functional and non-functional) are reassessed during the execution of a project, it is vital to be prepared to deal with these changes, reducing possible impacts to the project [7, 40].

Most NFRs are often handled ad-hoc during system testing, and engineers focus their efforts on ensuring that the software functional needs meet the business needs [34]. Functional Requirements (FRs) and NFRs have similar levels of importance to the success of a system [27]. Inadequate ways of dealing with instability, uncertainty, and a possible superficial focus on NFRs can lead to high maintenance costs in the long term, demanding the engagement of experts to accommodate changes [7, 40].

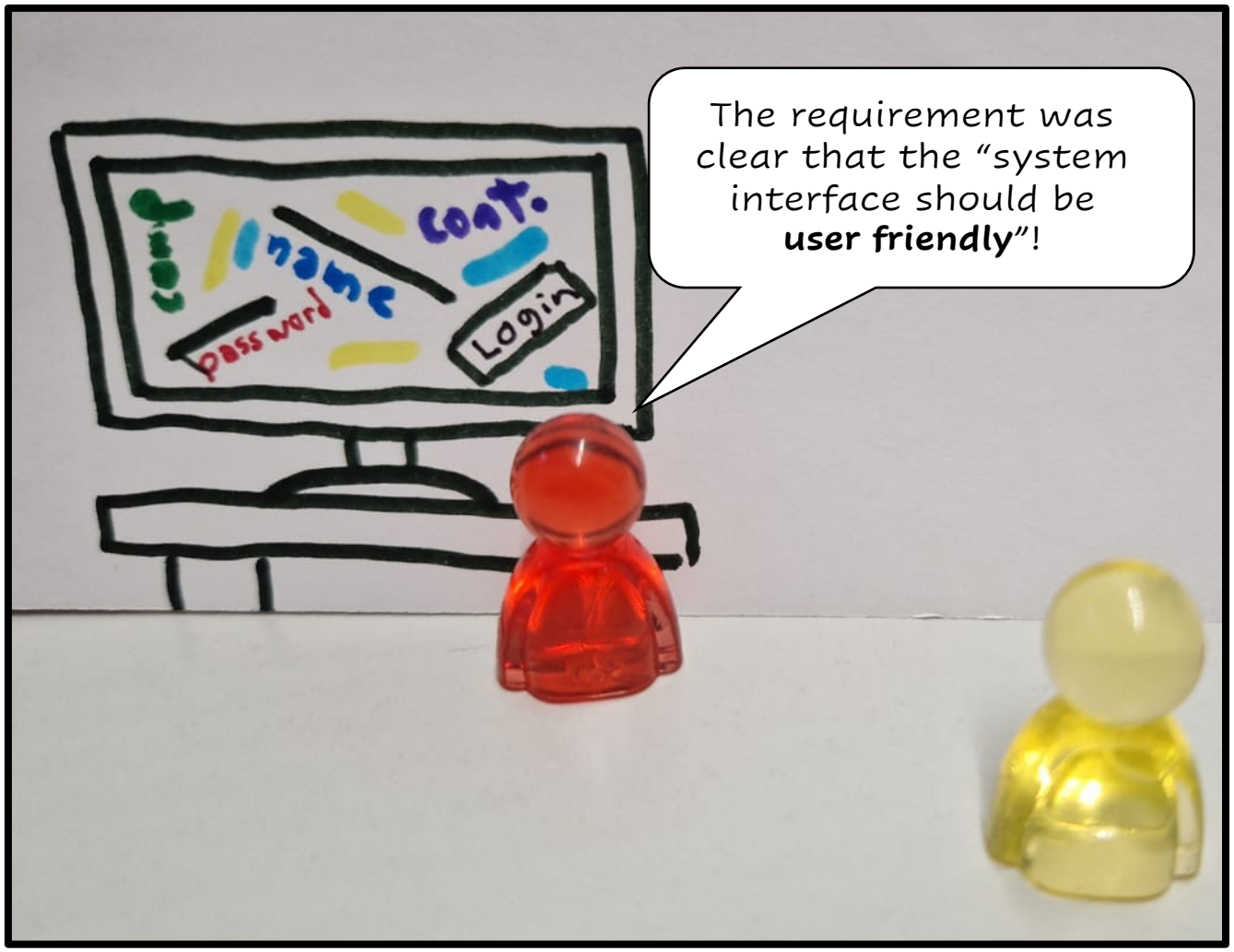
Although the relevance of NFRs is widely accepted, the discourse and approaches on how to deal with them are still oriented and dominated by how to differentiate them from FRs [12, 18]. Currently, more efforts are invested in satisfying NFRs. However, it seems that there is still an uneven emphasis on the importance given to system characteristics and its quality requirements [15, 21].

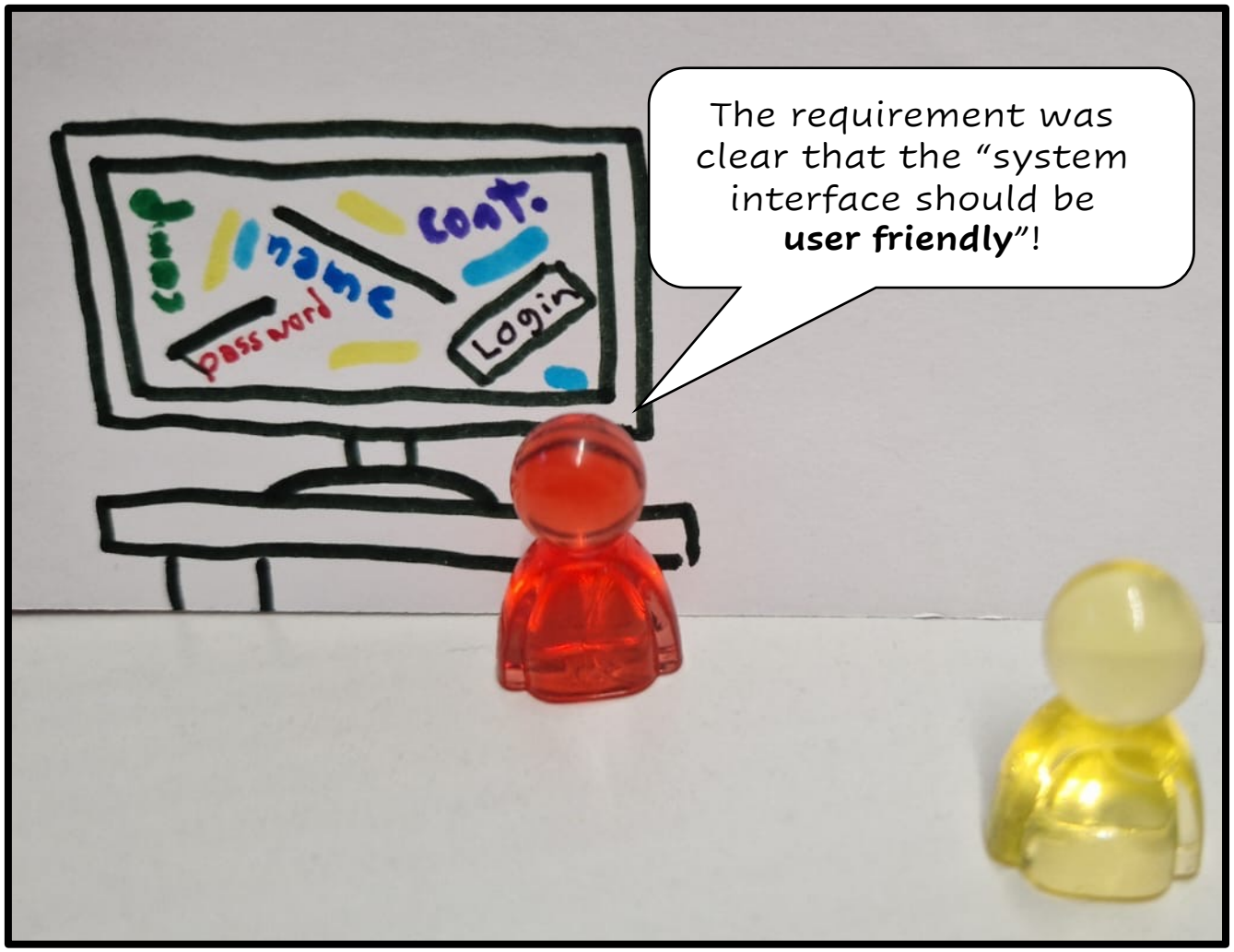
The management of requirements in a software project is complex and meticulous, so in-depth knowledge in the disciplines of requirements engineering and project management is necessary [29, 46]. In this regard, identifying anti-patterns, common behaviors or acti-
in software development
projects [28]





Friendly to whom?

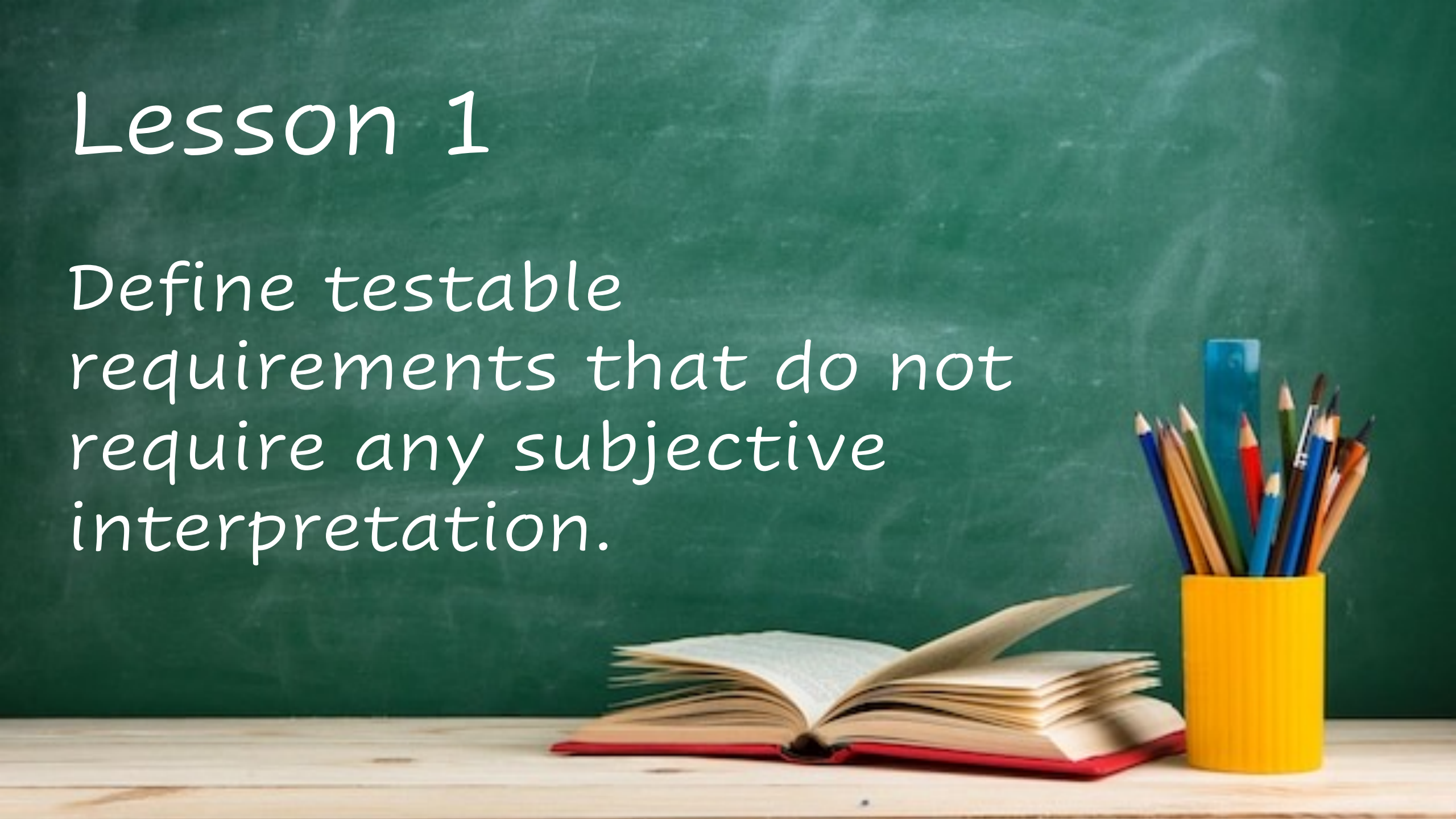

The requirement was clear that the “system interface should be **user friendly**”!
name
password
Login


The requirement was clear that the “system interface should be **user friendly**”!
name
password
Login

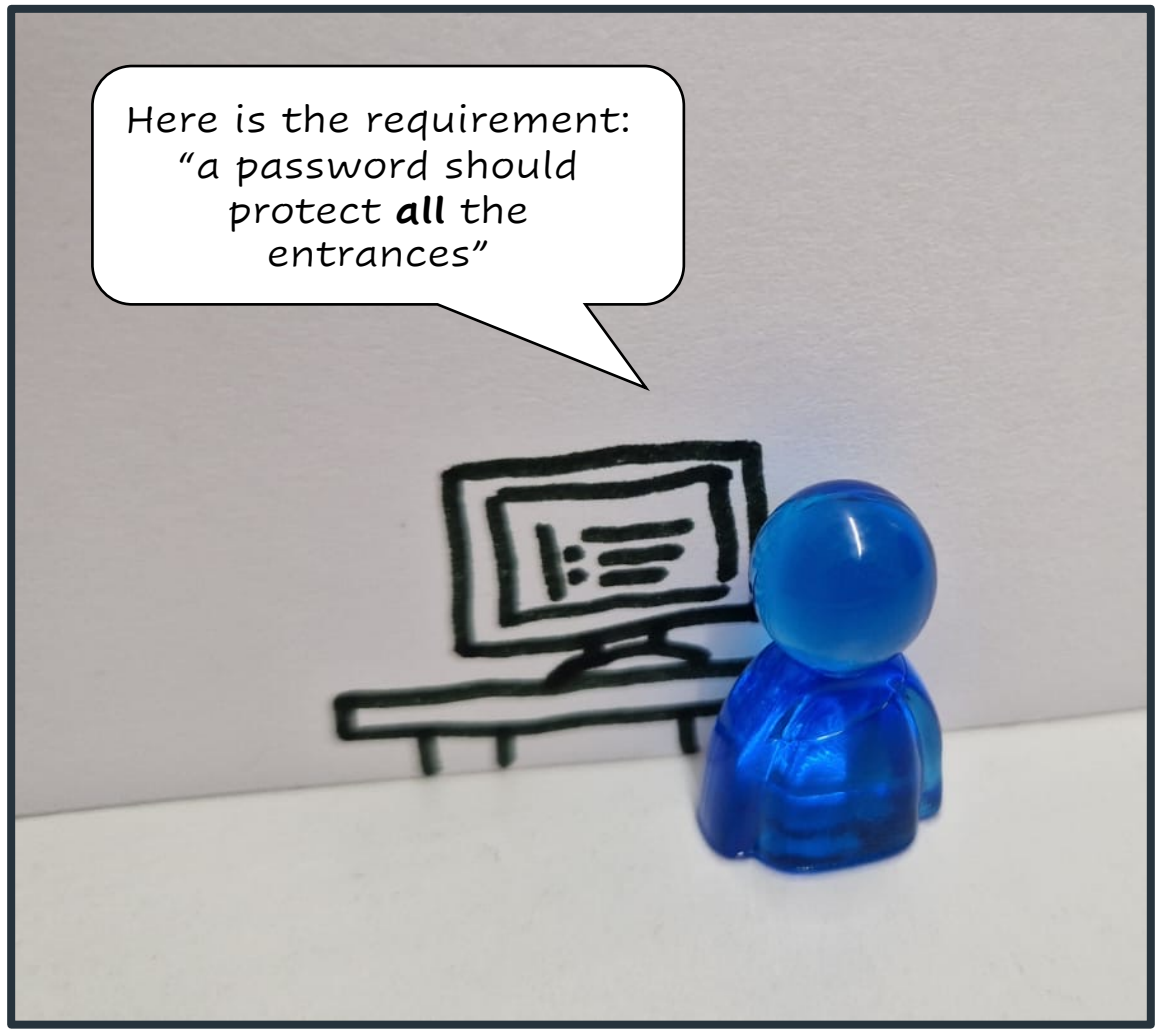

But it looks very friendly to me...
Login

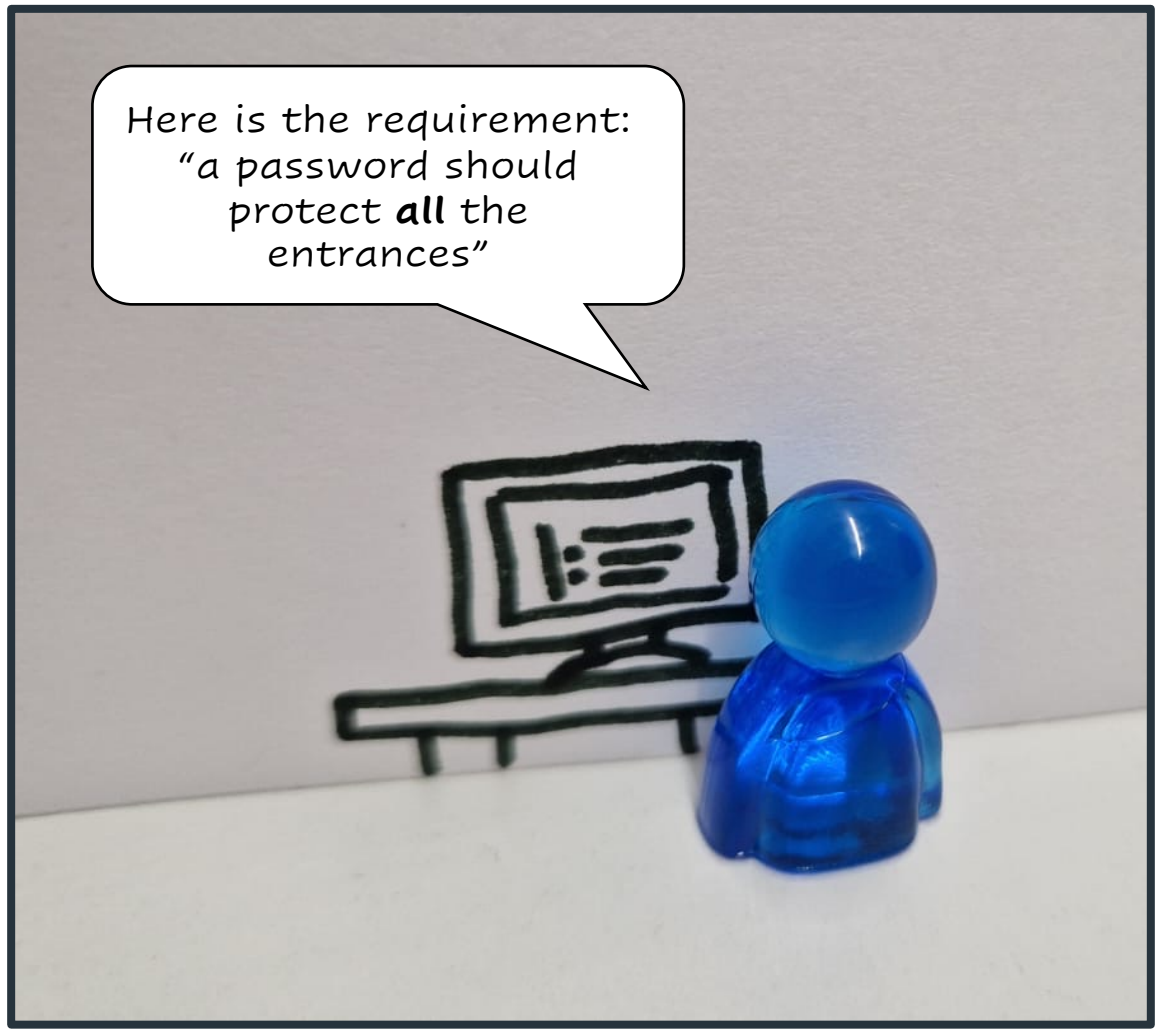
Lesson 1

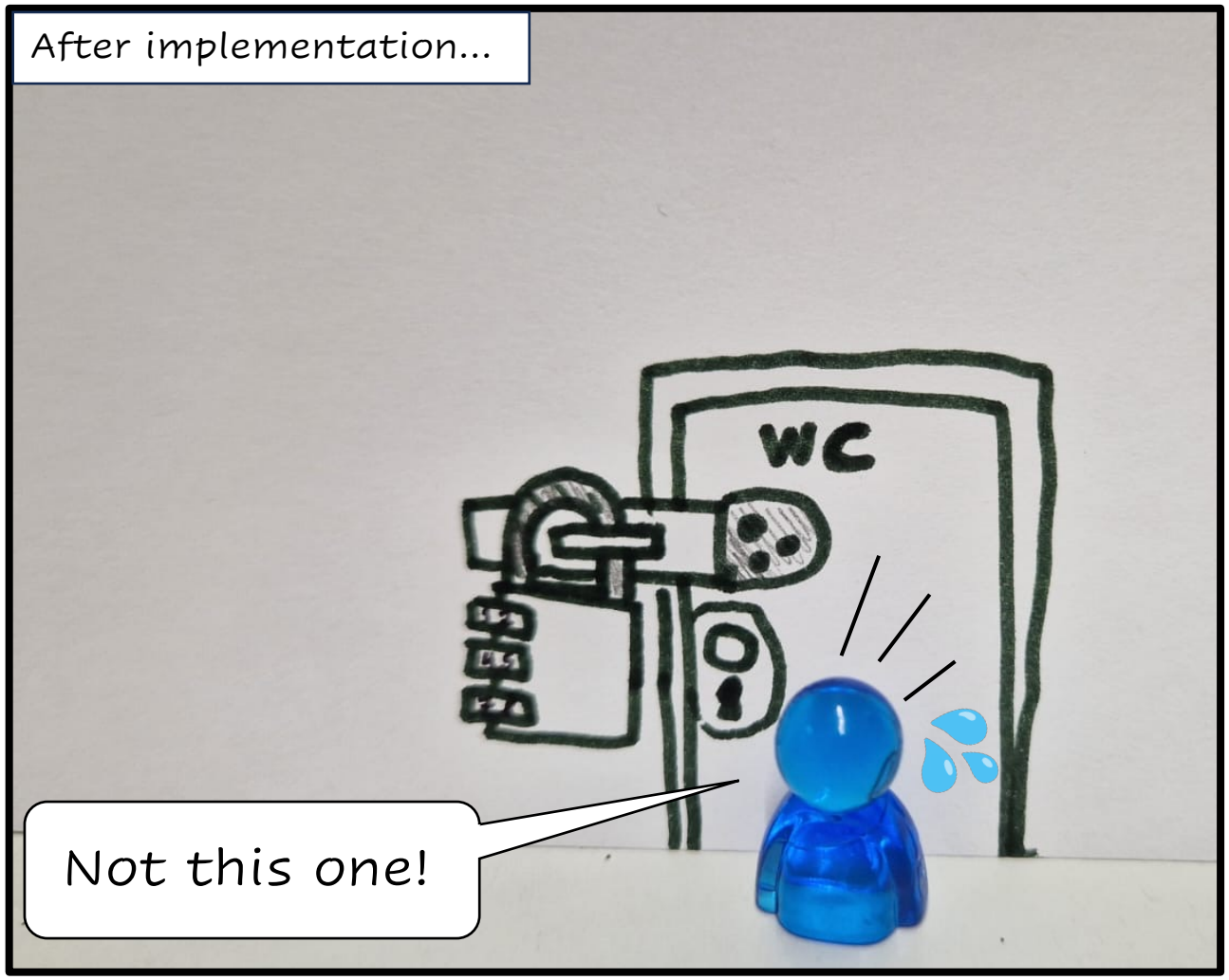
Define testable requirements that do not require any subjective interpretation.



Covering Everything

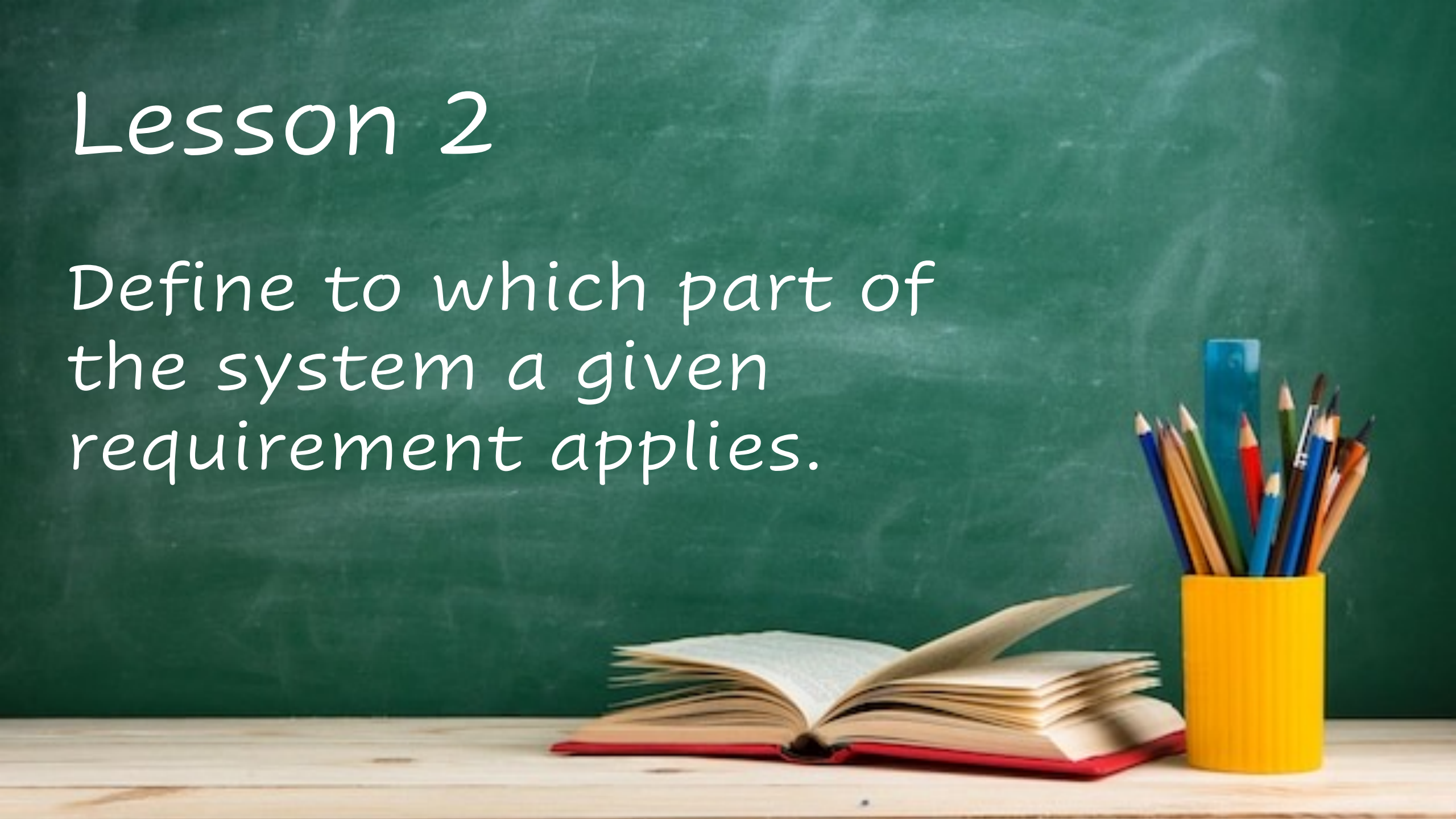

Here is the requirement:
“a password should protect **all** the entrances”


Here is the requirement:
“a password should protect **all** the entrances”


After implementation...
WC
Not this one!

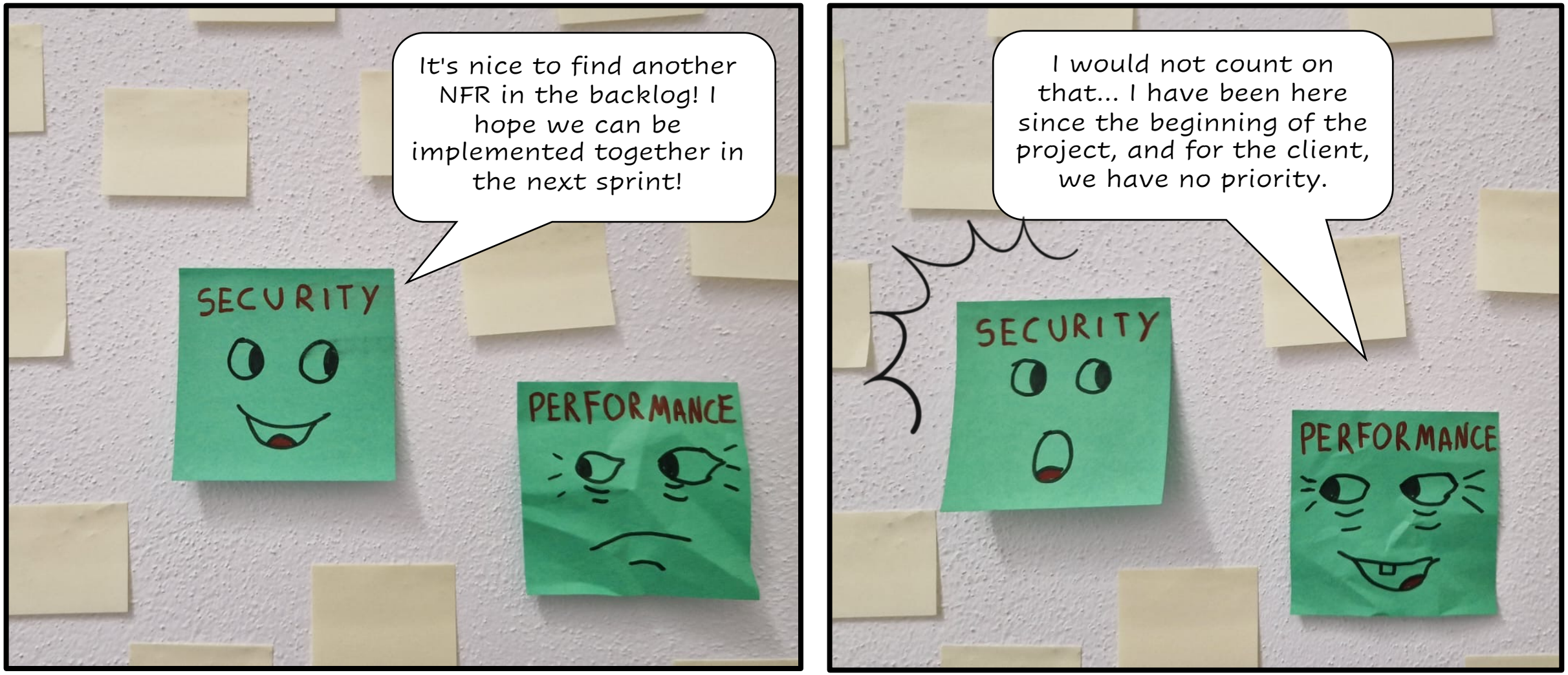
Lesson 2

Define to which part of the system a given requirement applies.



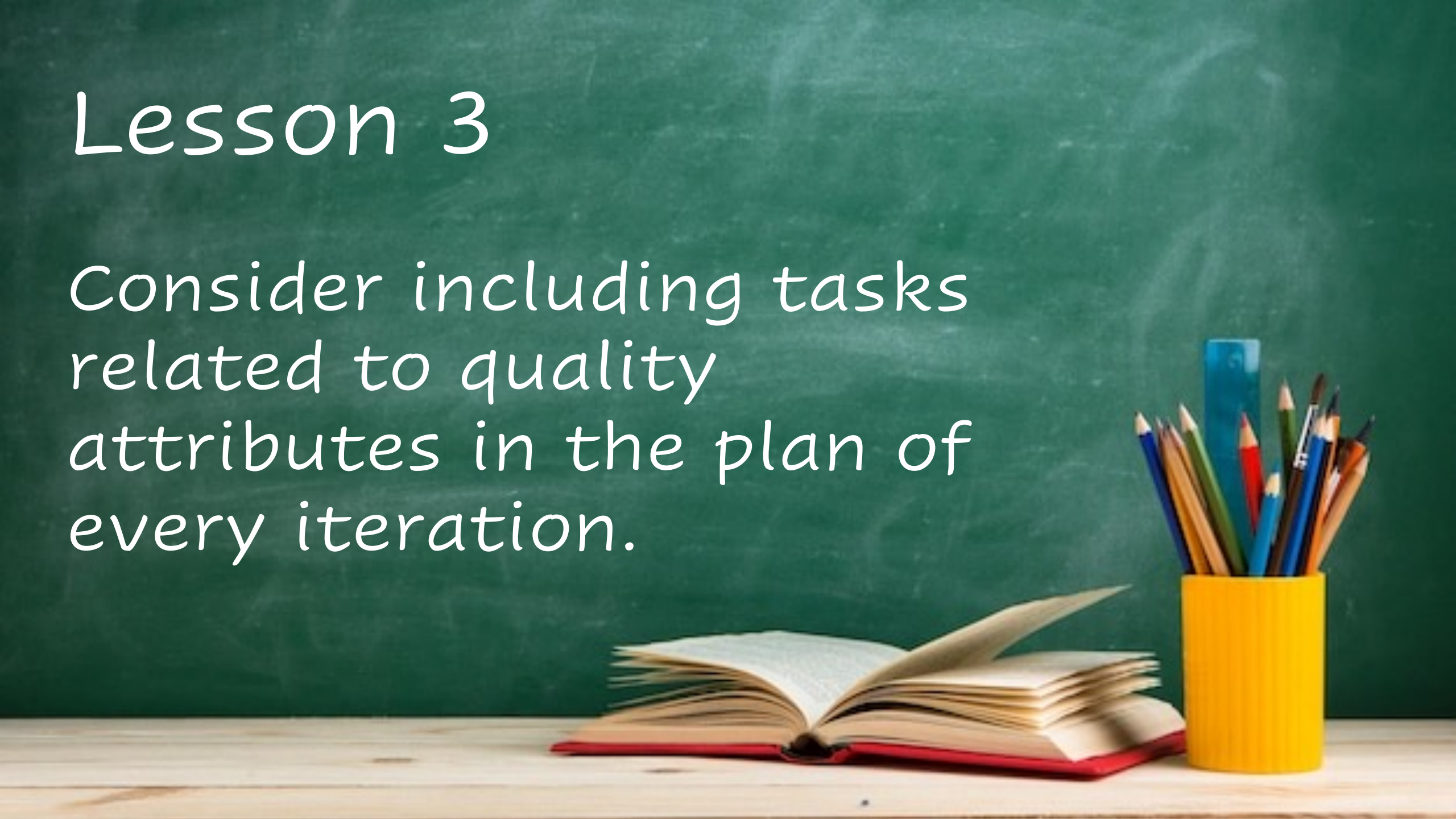
The Old Post-it


It's nice to find another NFR in the backlog! I hope we can be implemented together in the next sprint!
SECURITY
PERFORMANCE

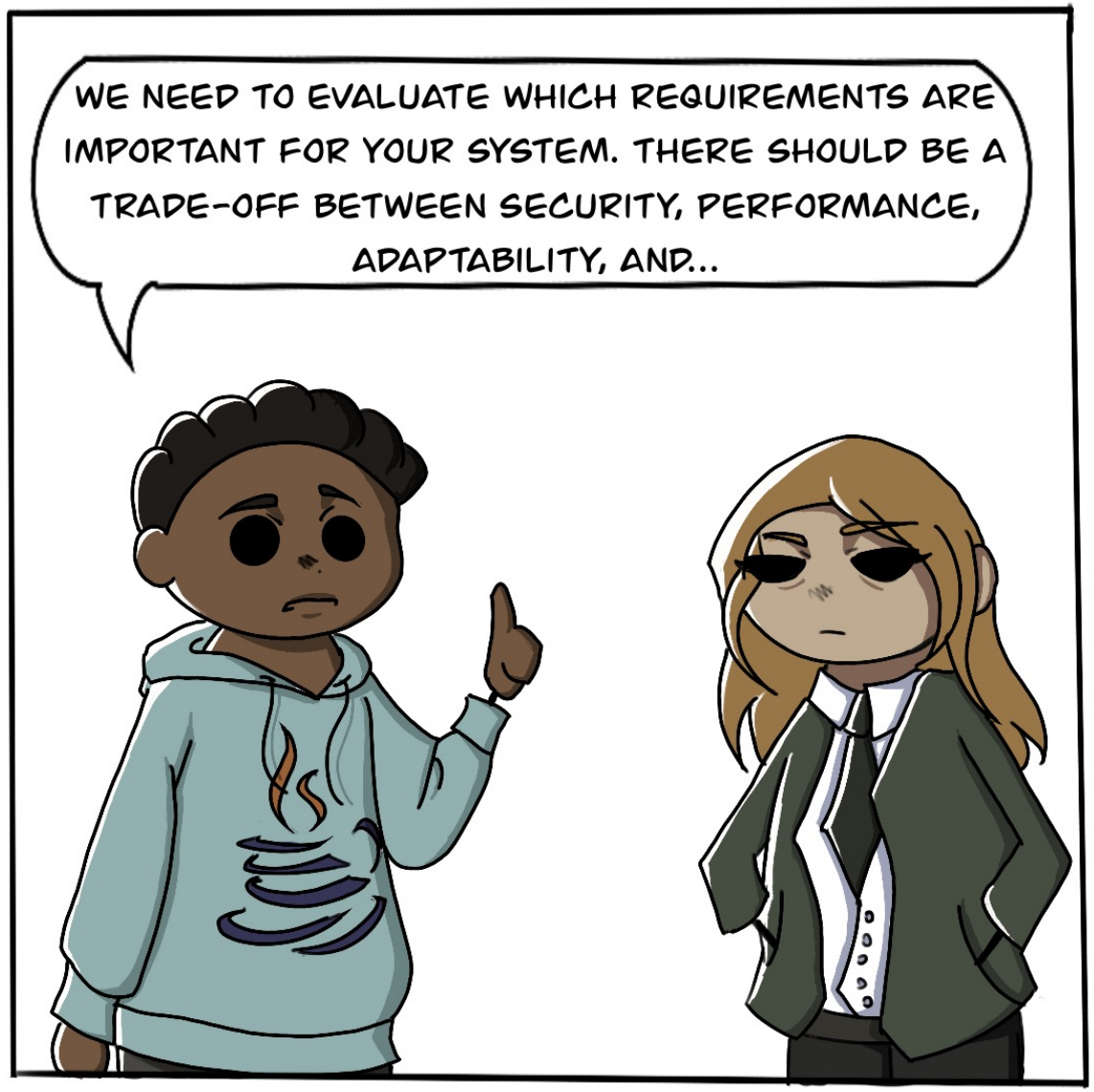

It's nice to find another NFR in the backlog! I hope we can be implemented together in the next sprint!
SECURITY
PERFORMANCE
I would not count on that... I have been here since the beginning of the project, and for the client, we have no priority.
SECURITY
PERFORMANCE

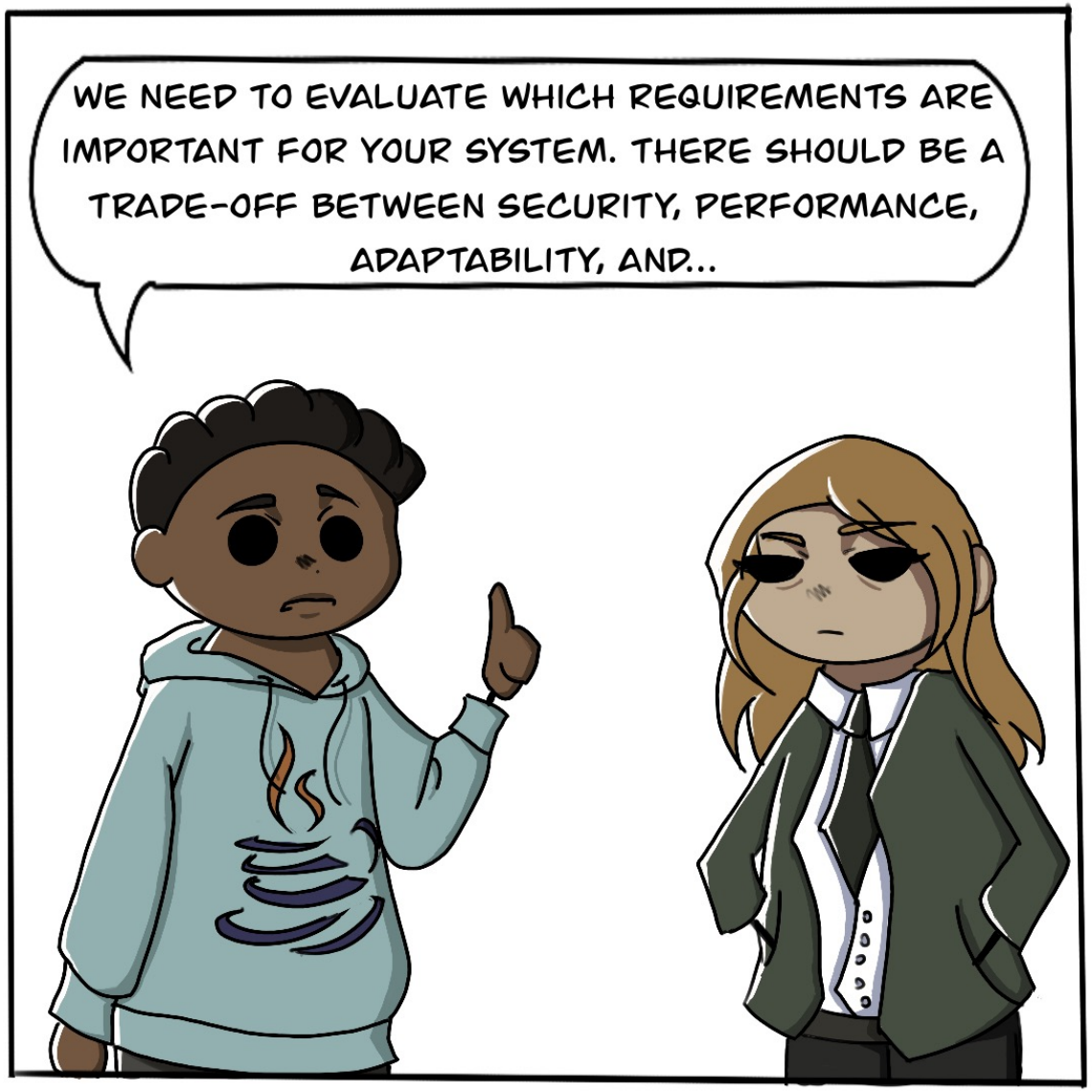
Lesson 3

Consider including tasks related to quality attributes in the plan of every iteration.



Precious Requirements

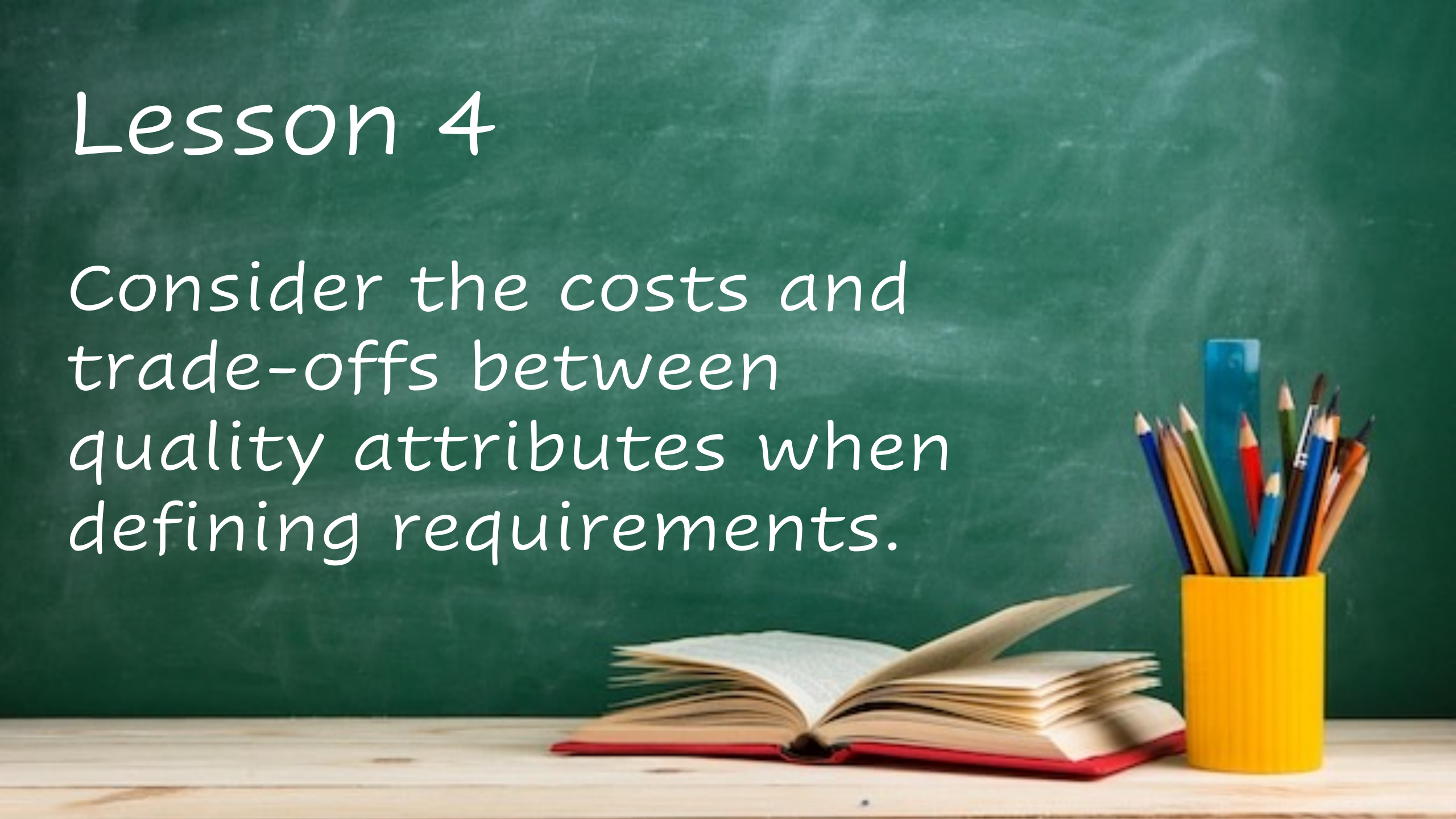

WE NEED TO EVALUATE WHICH REQUIREMENTS ARE IMPORTANT FOR YOUR SYSTEM. THERE SHOULD BE A TRADE-OFF BETWEEN SECURITY, PERFORMANCE, ADAPTABILITY, AND...


WE NEED TO EVALUATE WHICH REQUIREMENTS ARE IMPORTANT FOR YOUR SYSTEM. THERE SHOULD BE A TRADE-OFF BETWEEN SECURITY, PERFORMANCE, ADAPTABILITY, AND...

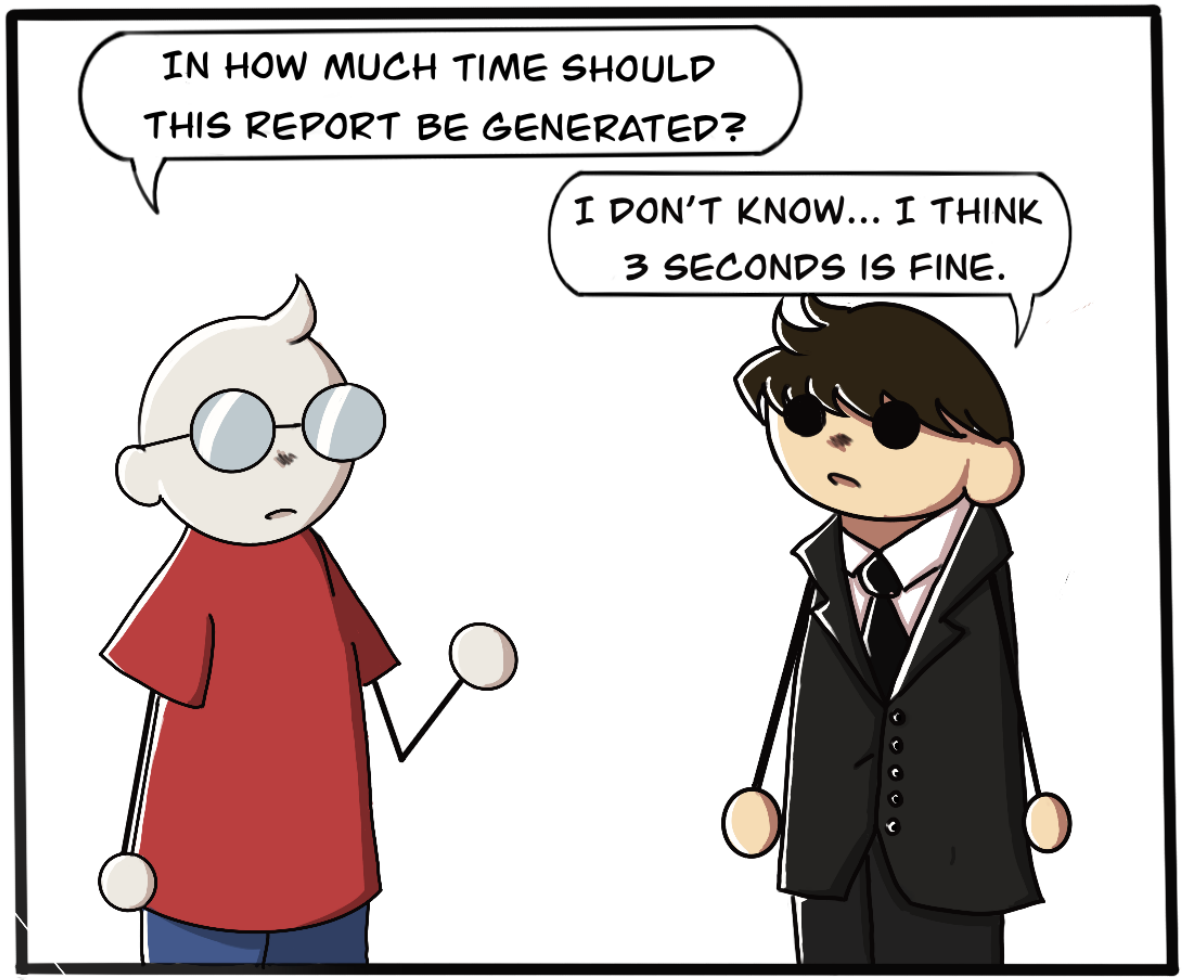

MY PRECIOUS REQUIREMENTS ! WE WANT WANT THEM ALL!

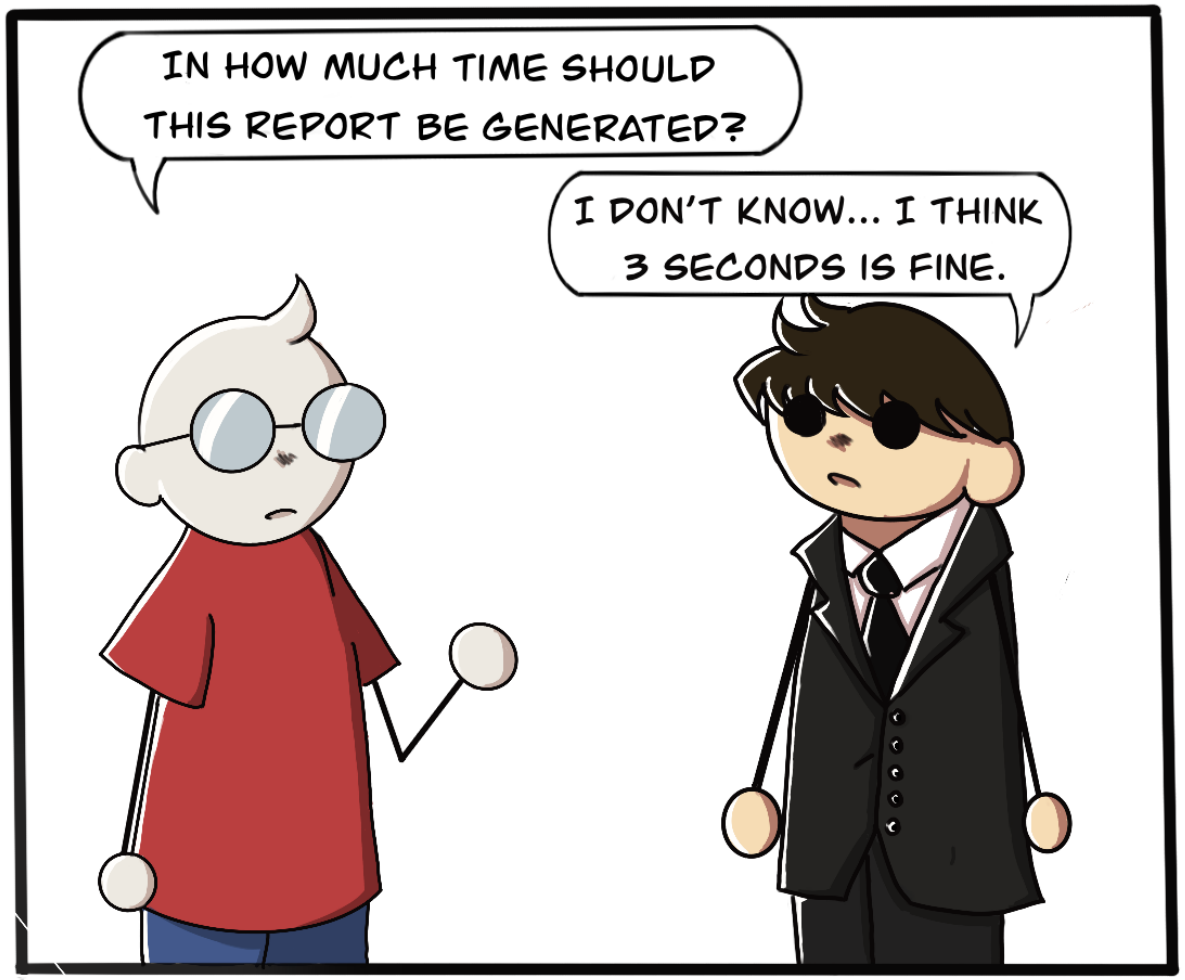
Lesson 4

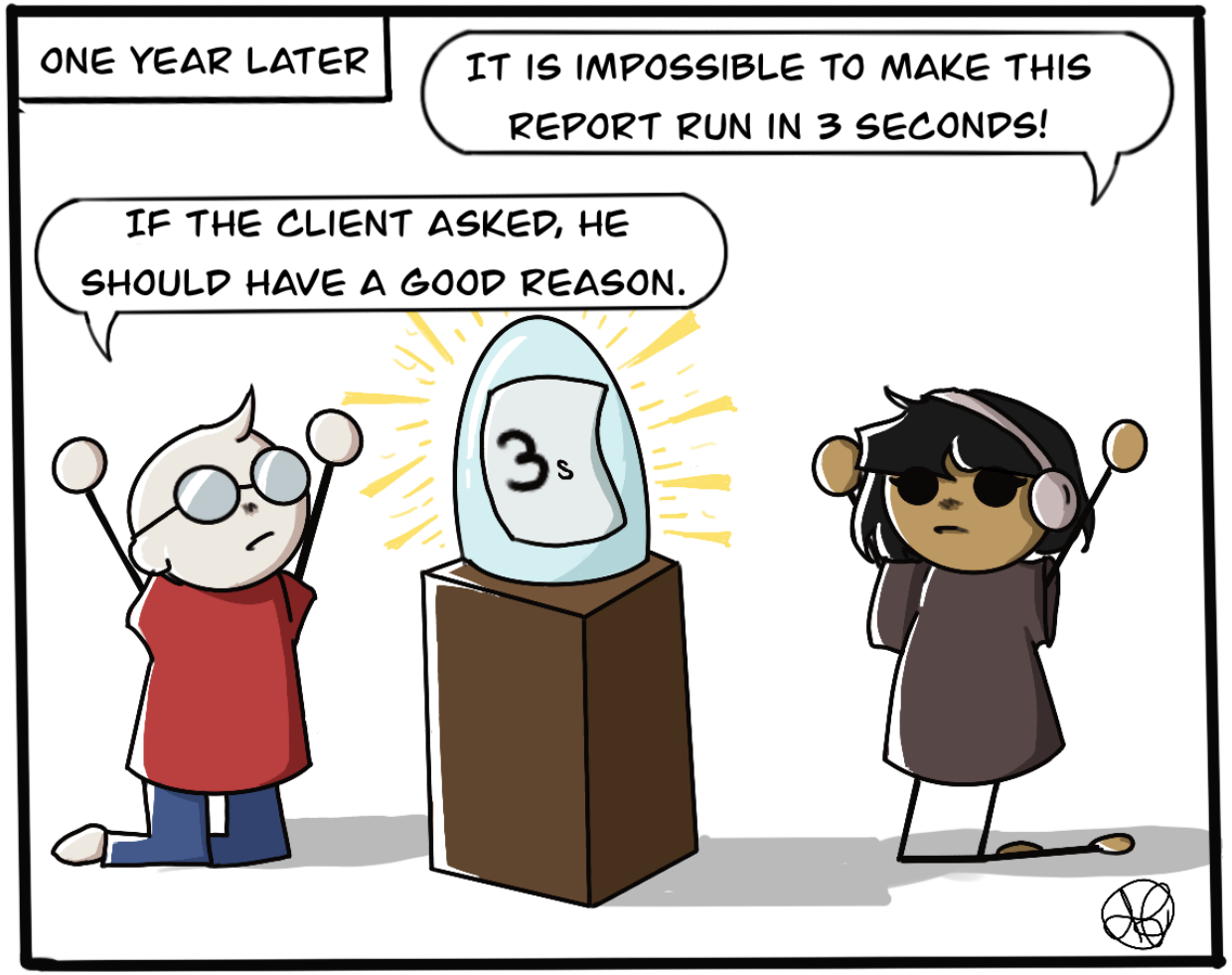
Consider the costs and trade-offs between quality attributes when defining requirements.



Sacred Number

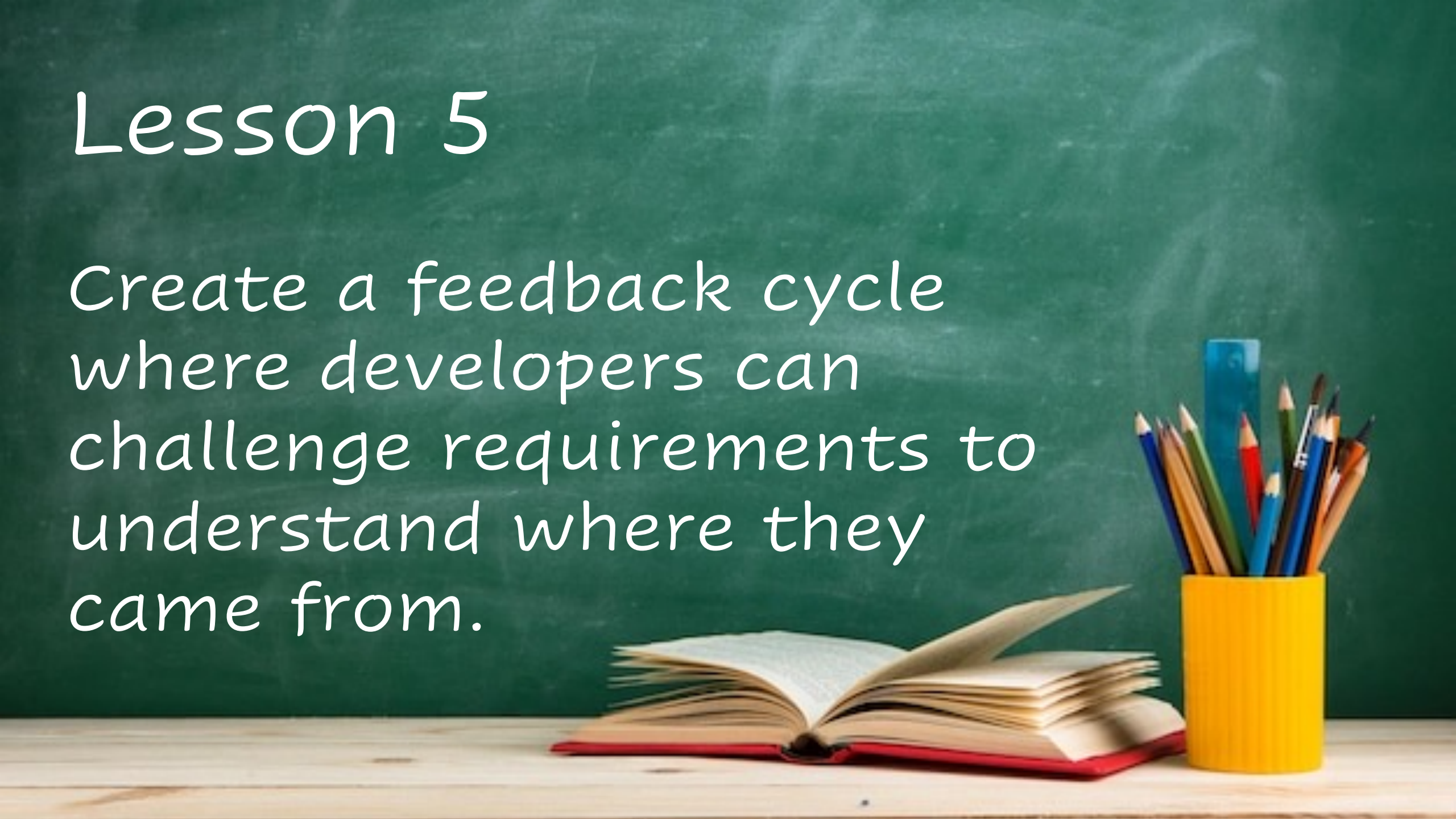

IN HOW MUCH TIME SHOULD THIS REPORT BE GENERATED?
I DON'T KNOW... I THINK 3 SECONDS IS FINE.


IN HOW MUCH TIME SHOULD THIS REPORT BE GENERATED?
I DON'T KNOW... I THINK 3 SECONDS IS FINE.

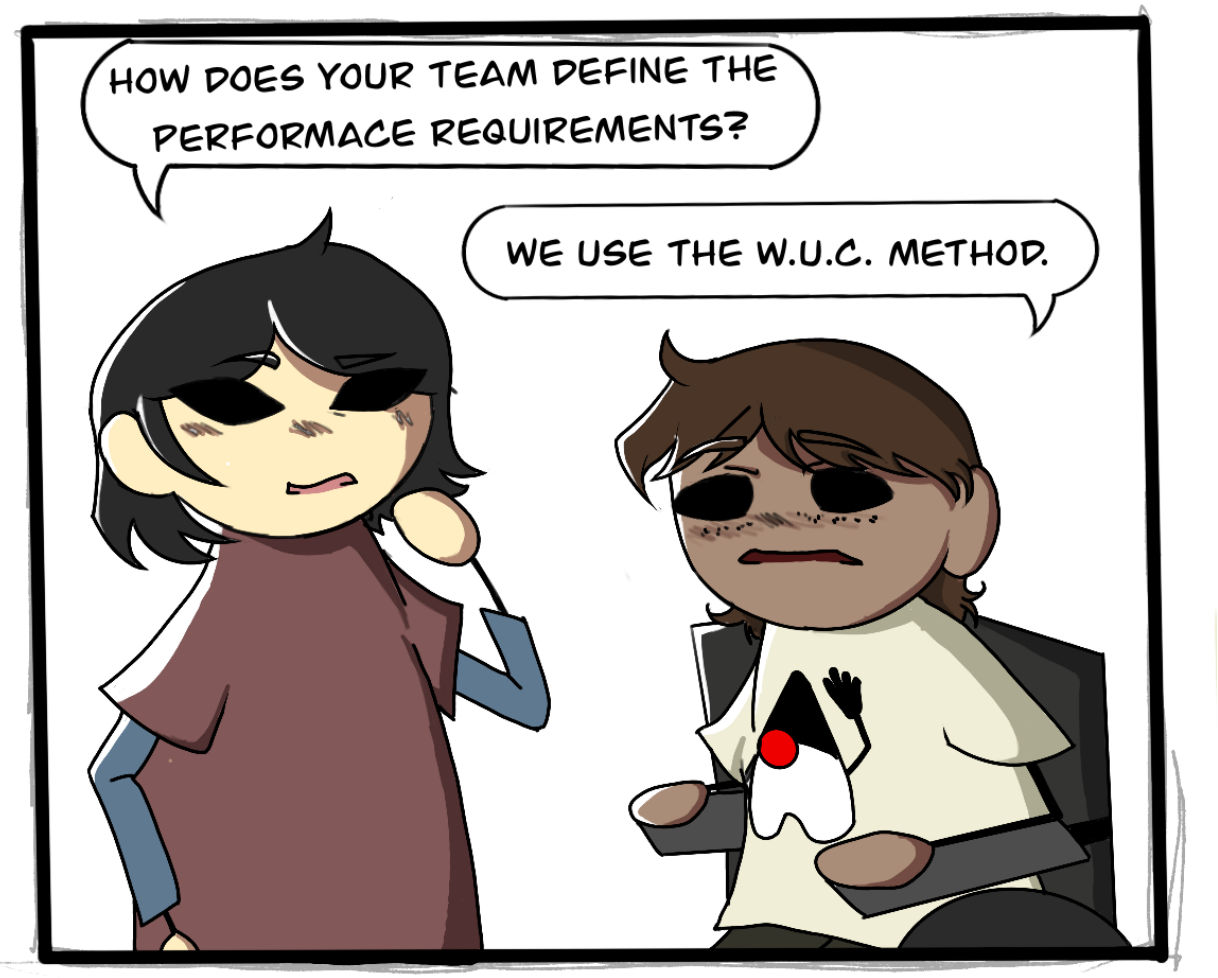

ONE YEAR LATER
IT IS IMPOSSIBLE TO MAKE THIS REPORT RUN IN 3 SECONDS!
IF THE CLIENT ASKED, HE SHOULD HAVE A GOOD REASON.
3s

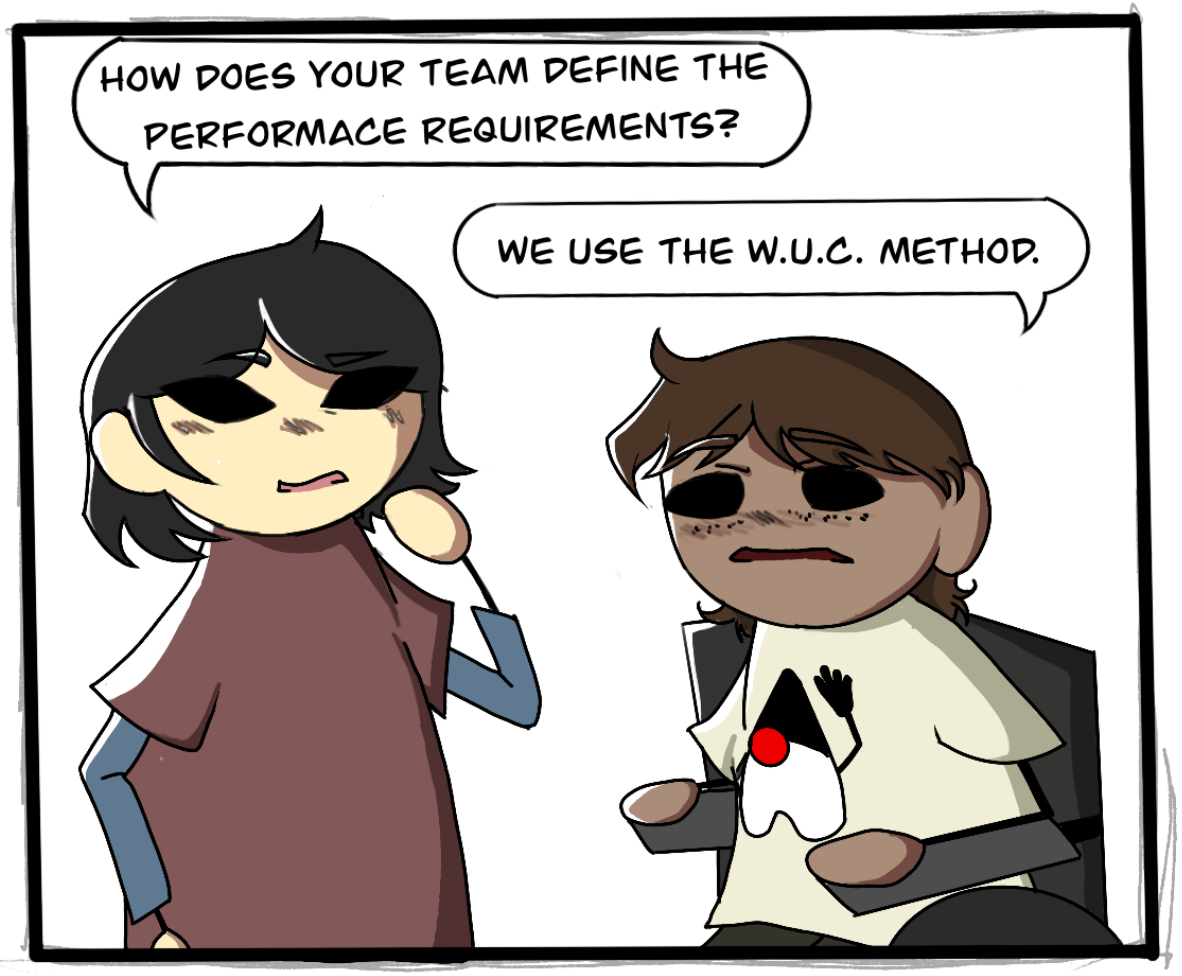
Lesson 5

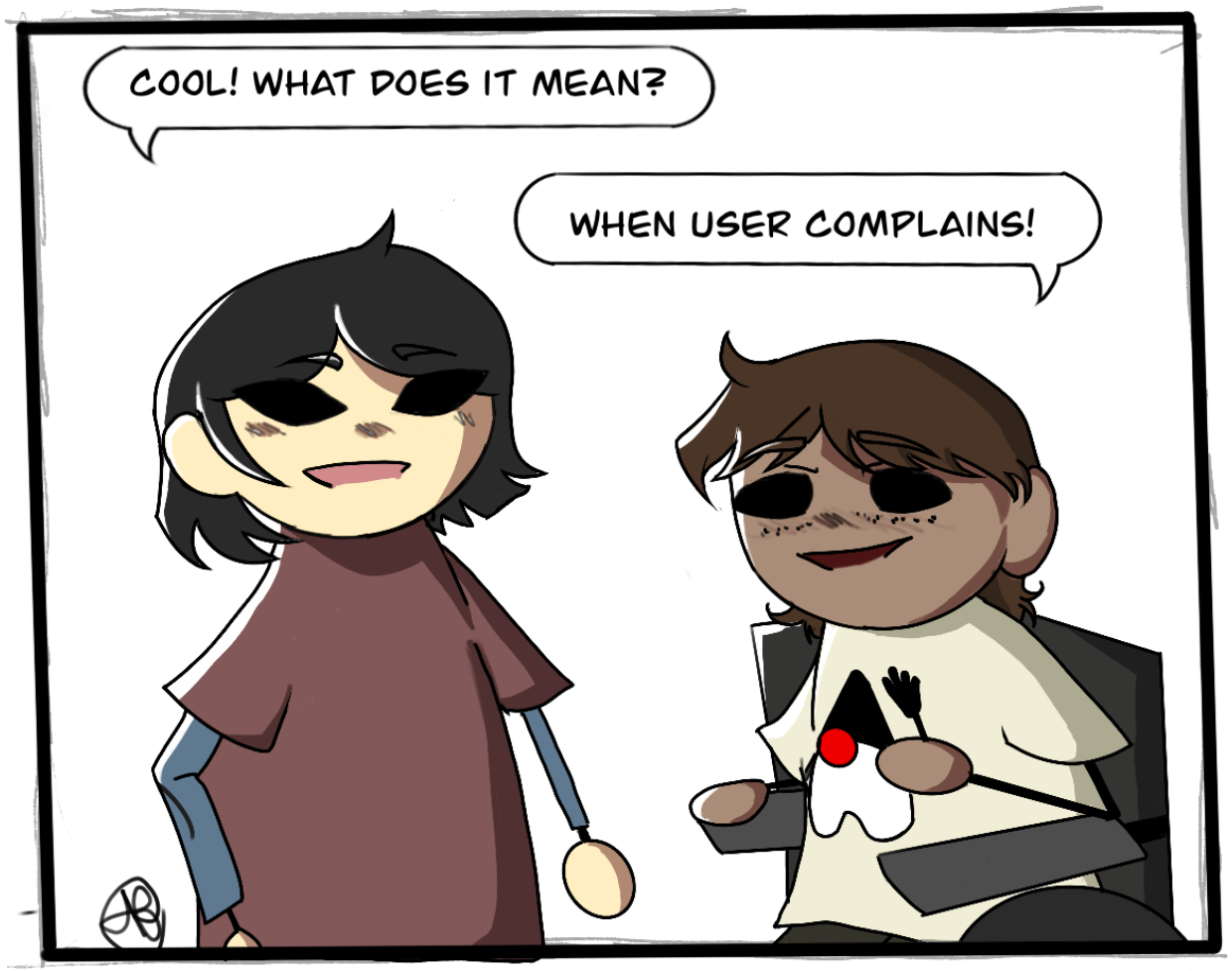
Create a feedback cycle where developers can challenge requirements to understand where they came from.



Most Popular Method


HOW DOES YOUR TEAM DEFINE THE PERFORMACE REQUIREMENTS?
WE USE THE W.U.C. METHOD.

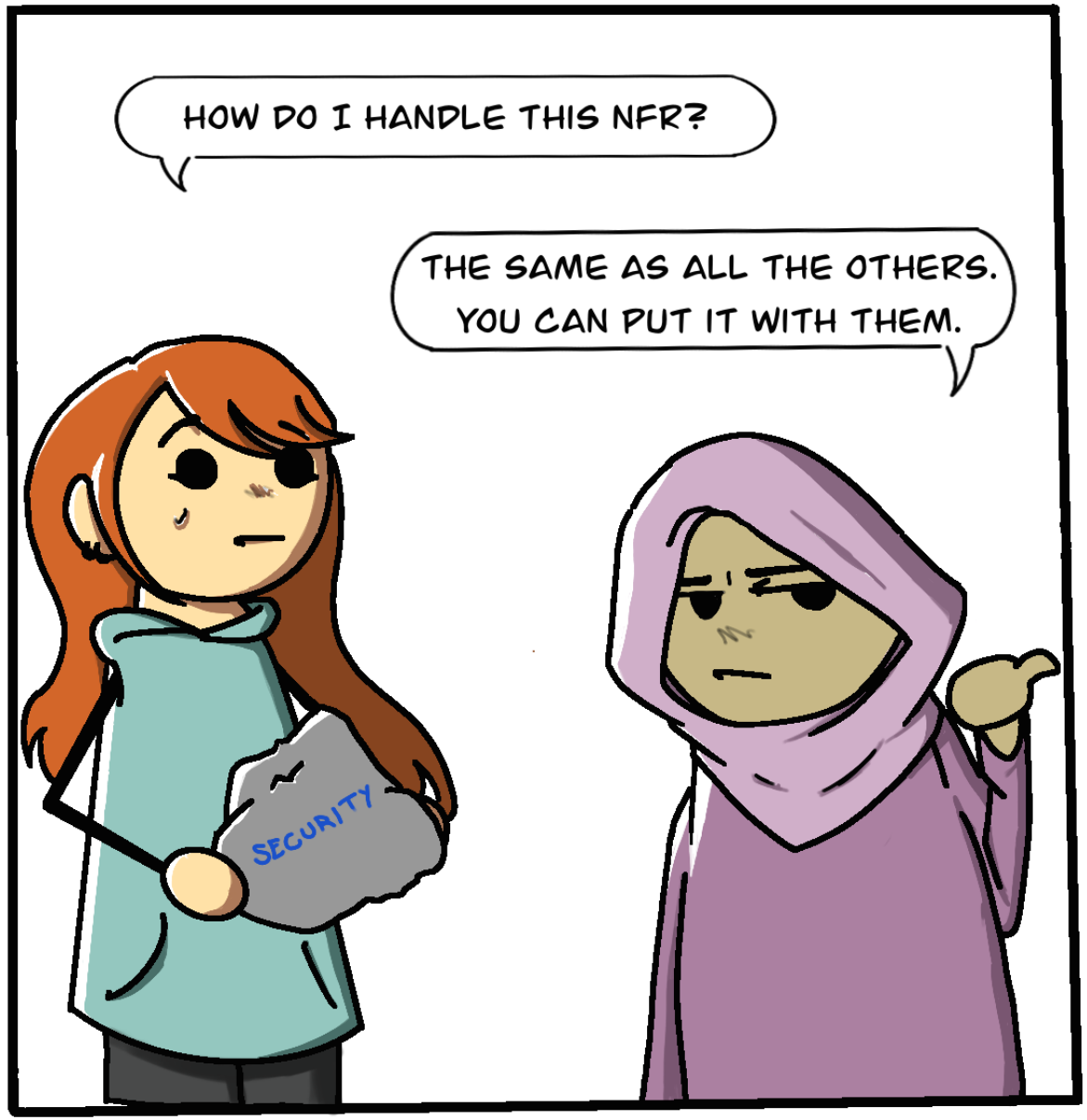

HOW DOES YOUR TEAM DEFINE THE PERFORMACE REQUIREMENTS?
WE USE THE W.U.C. METHOD.

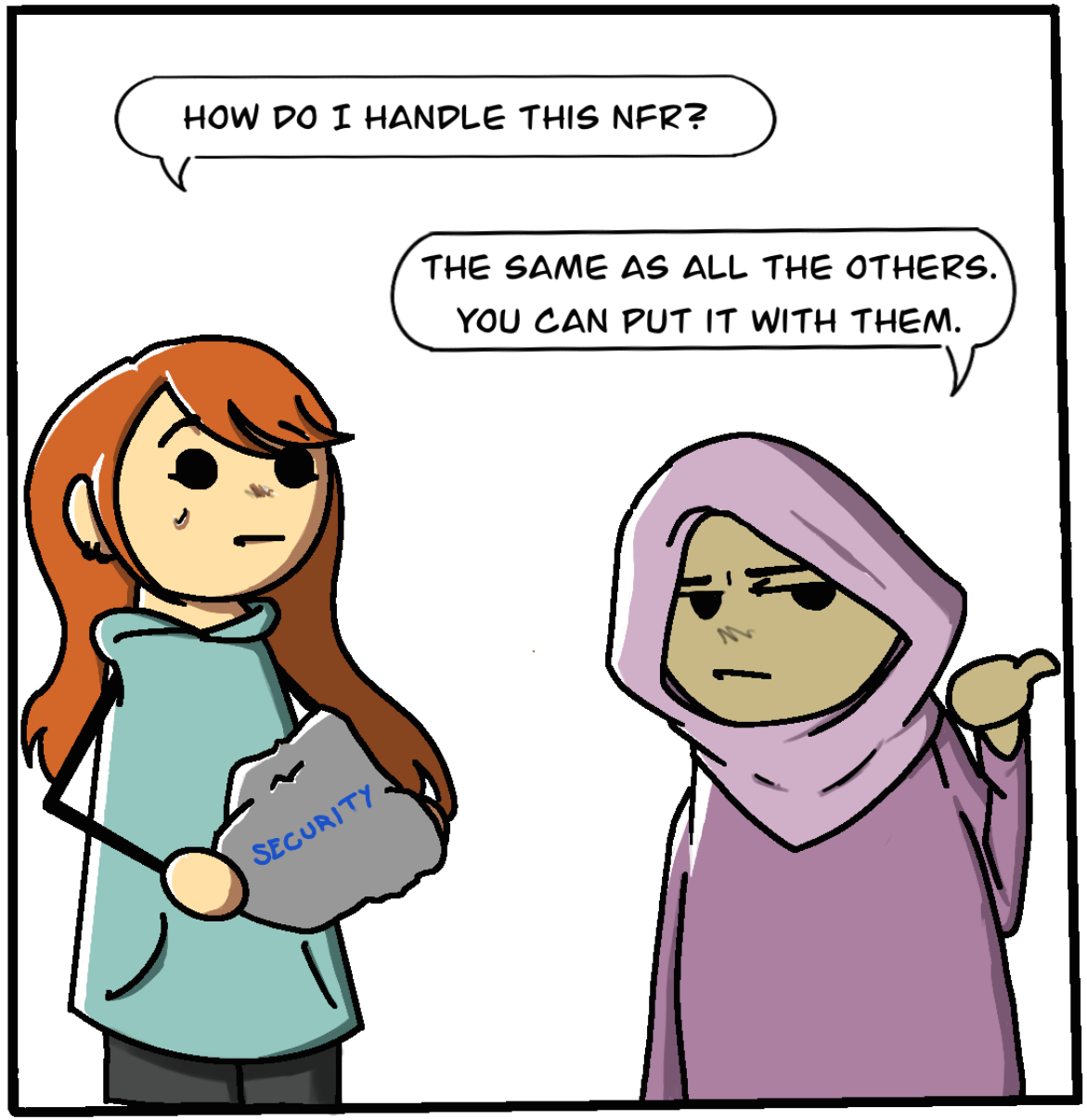

COOL! WHAT DOES IT MEAN?
WHEN USER COMPLAINS!

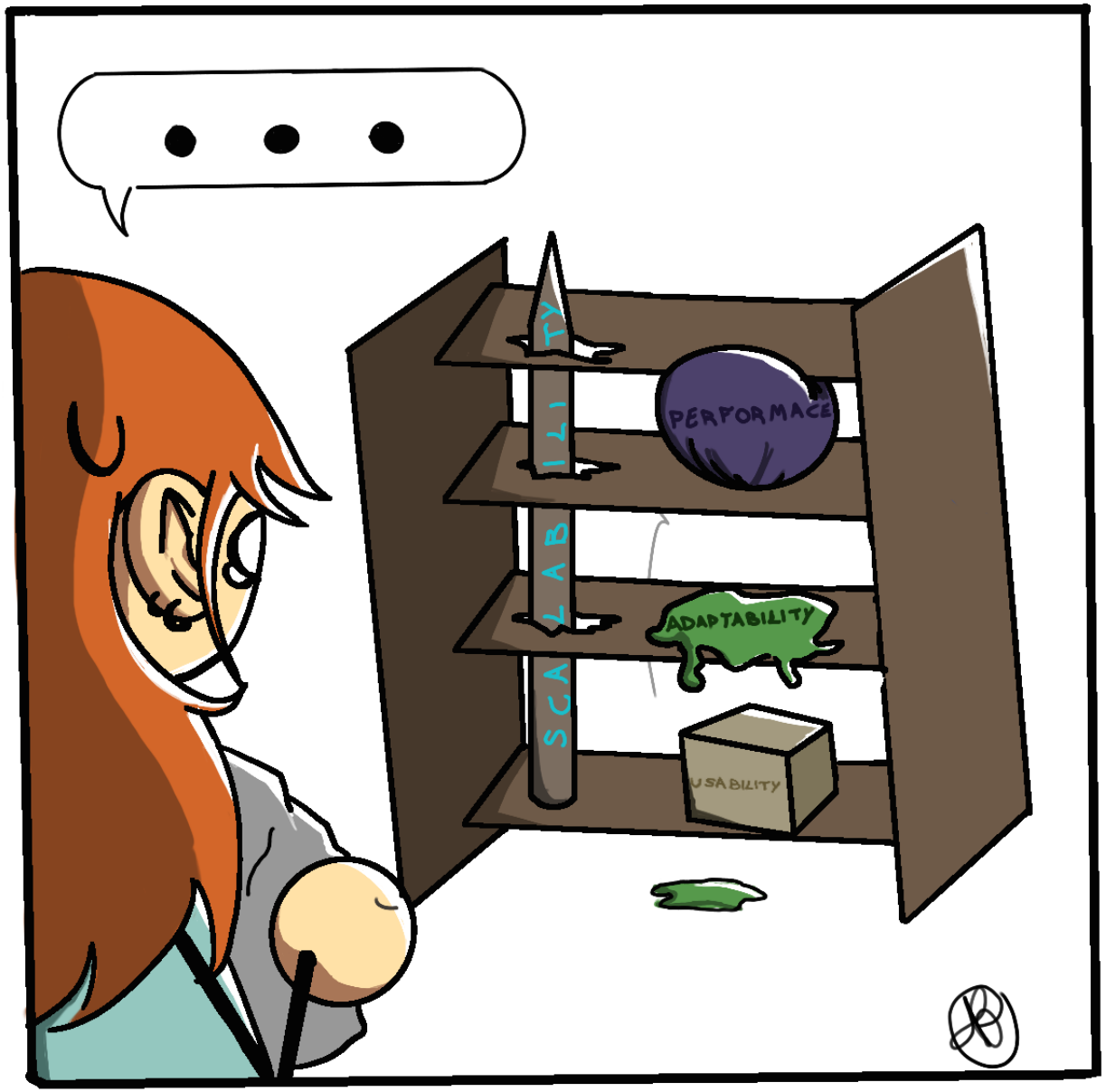
Lesson 6

Do not wait for a user or client manifestation to define and test requirements about quality attributes.

General Storage

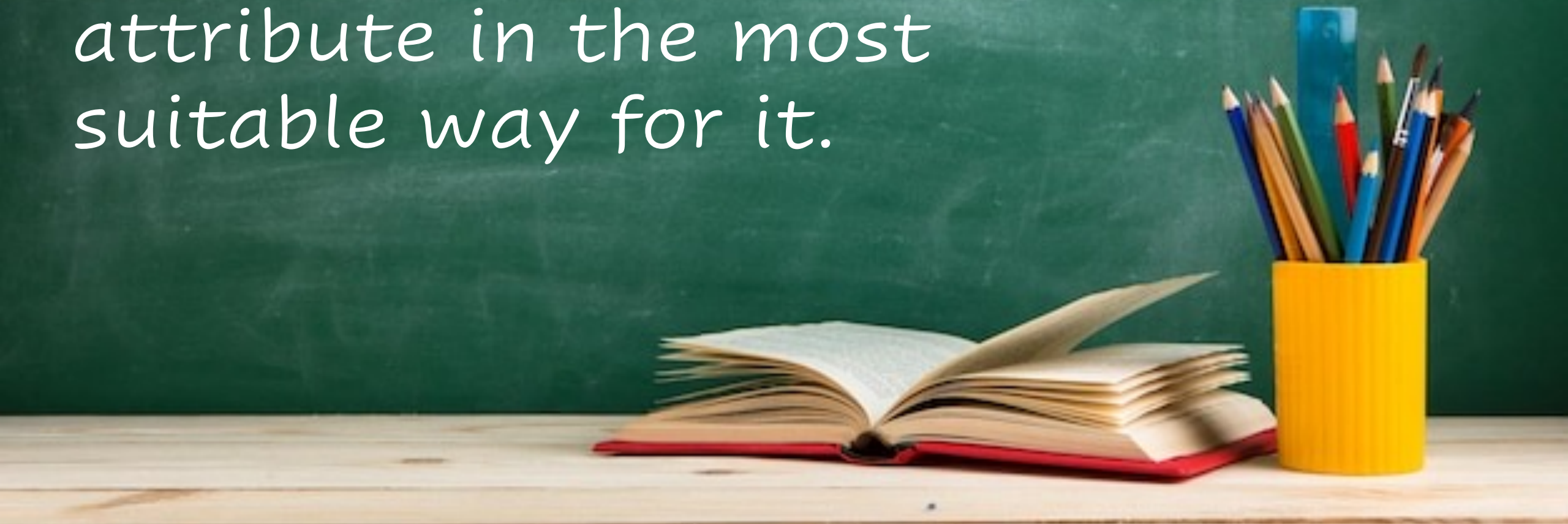

HOW DO I HANDLE THIS NFR?
THE SAME AS ALL THE OTHERS. YOU CAN PUT IT WITH THEM.
SECURITY


HOW DO I HANDLE THIS NFR?
THE SAME AS ALL THE OTHERS. YOU CAN PUT IT WITH THEM.
SECURITY

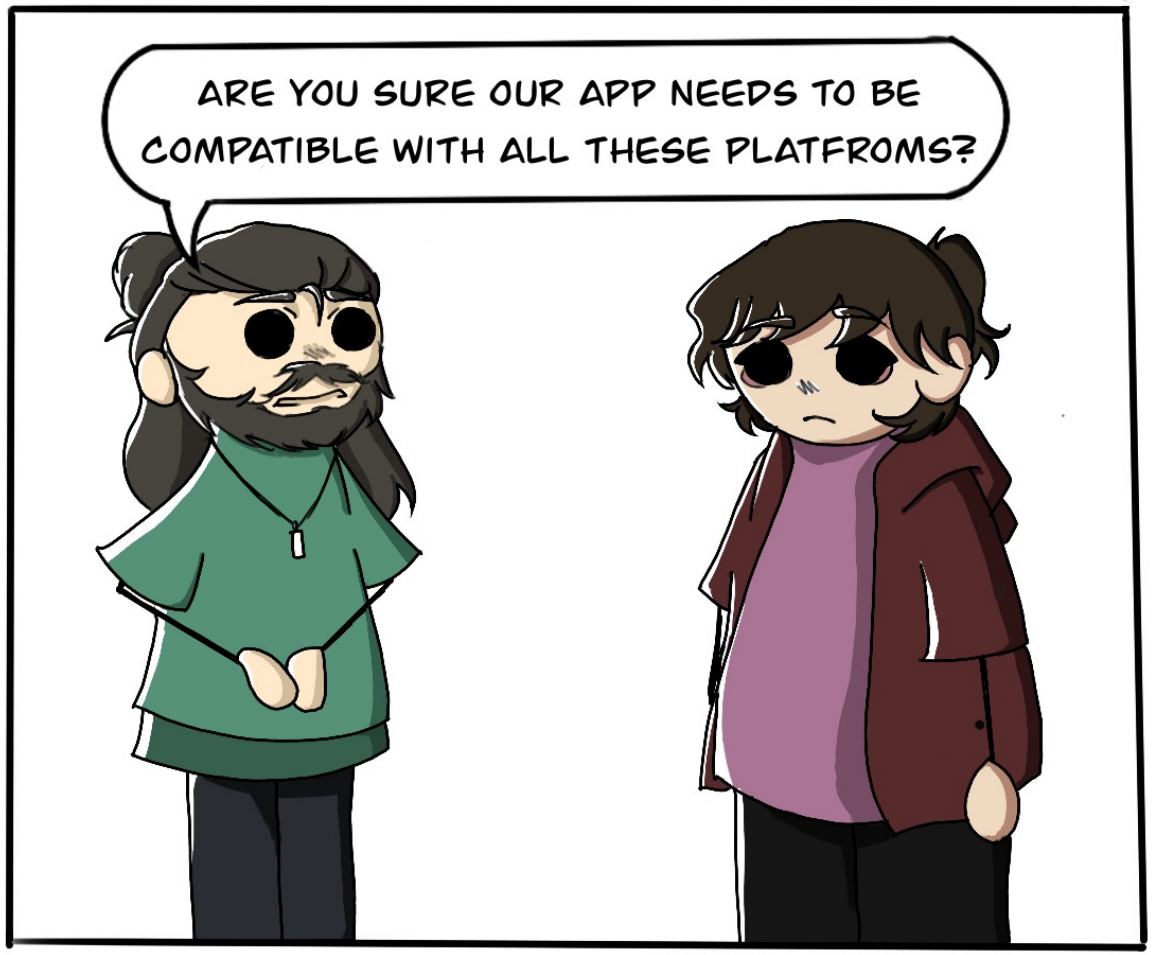

SCALABILITY
PERFORMACE
ADAPTABILITY
USABILITY

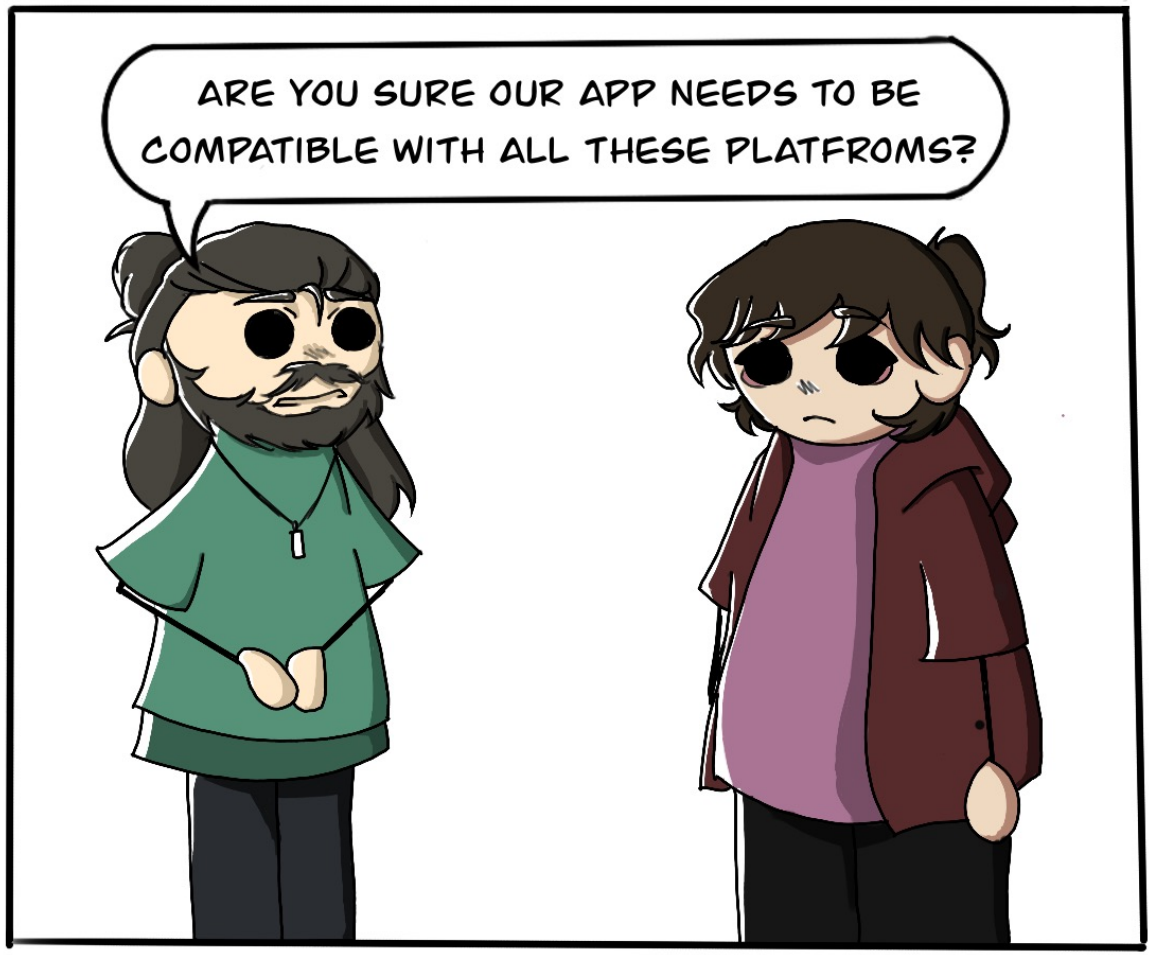
Lesson 7

Threat each type of quality attribute in the most suitable way for it.



Rupestrian Requirement

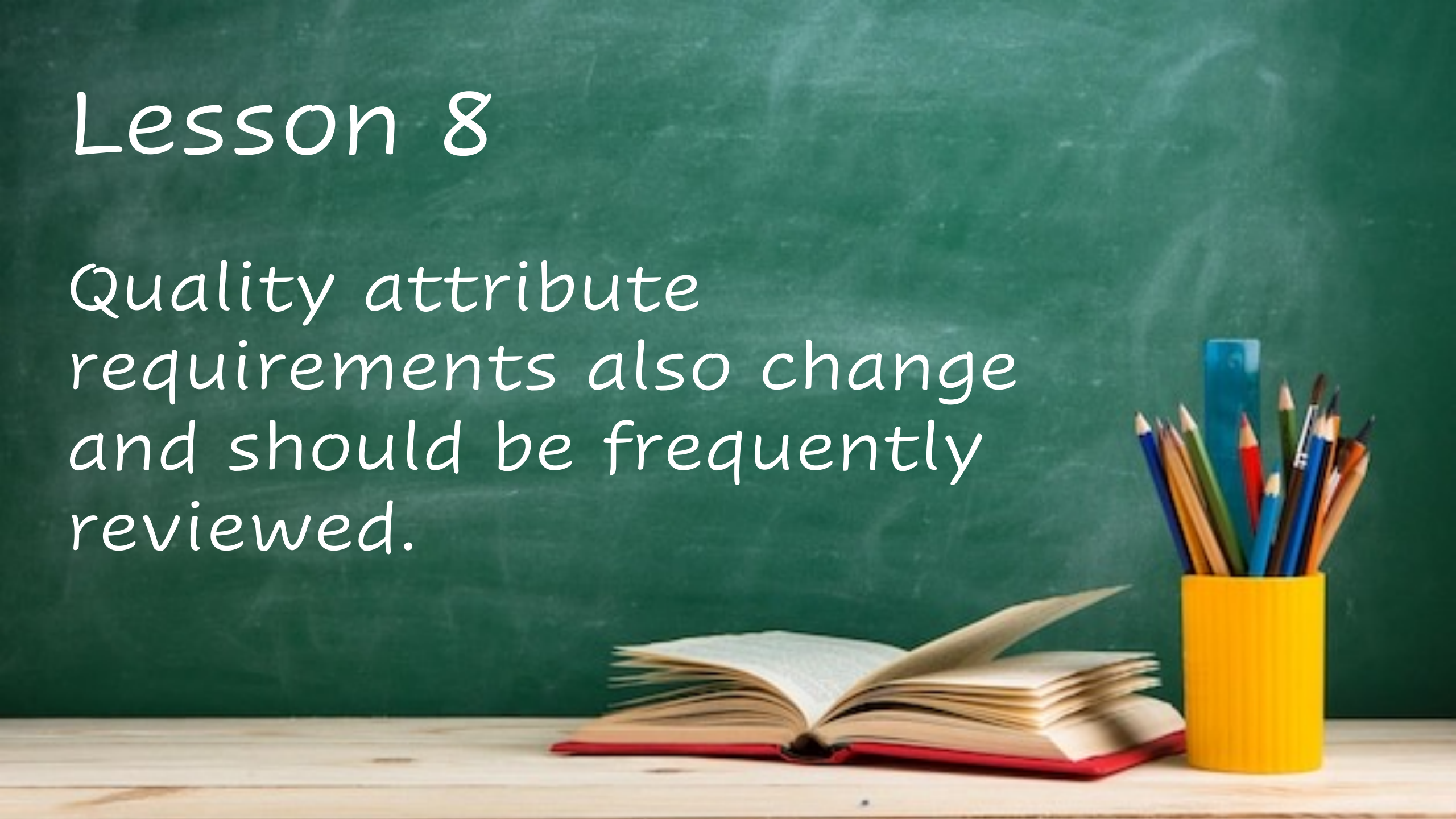

ARE YOU SURE OUR APP NEEDS TO BE COMPATIBLE WITH ALL THESE PLATFROMS?


ARE YOU SURE OUR APP NEEDS TO BE COMPATIBLE WITH ALL THESE PLATFROMS?


BLACKBERRY
THAT'S BEEN HERE SINCE I ARRIVED. NOBODY EVER SAID TO CHANGE IT...

Lesson 8

Quality attribute requirements also change and should be frequently reviewed.



“Change is inevitable. Growth is optional.”

John C. Maxwell

thank you

Eduardo Guerra
eduardo.guerra@unibz.it