

Who is playing the role of the customer in 301?



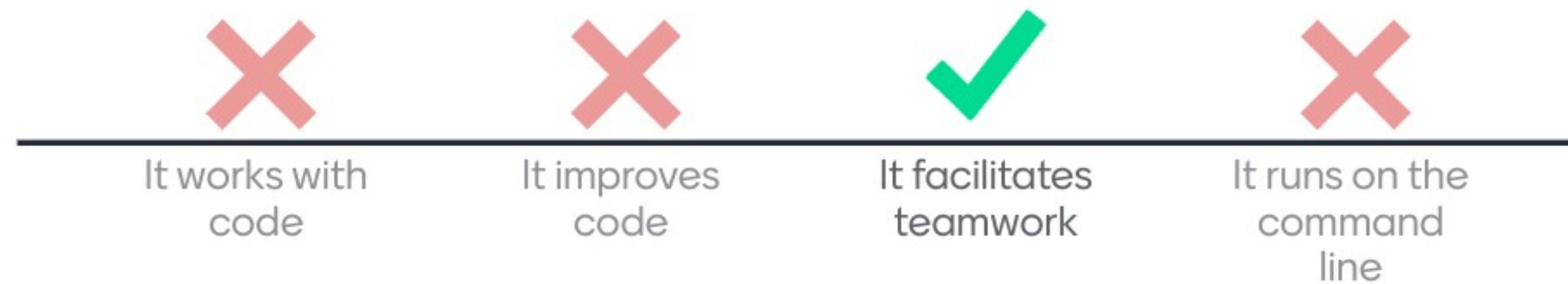
Who is playing the role of the manager in 301?



Who is playing the role of the programmers in 301?



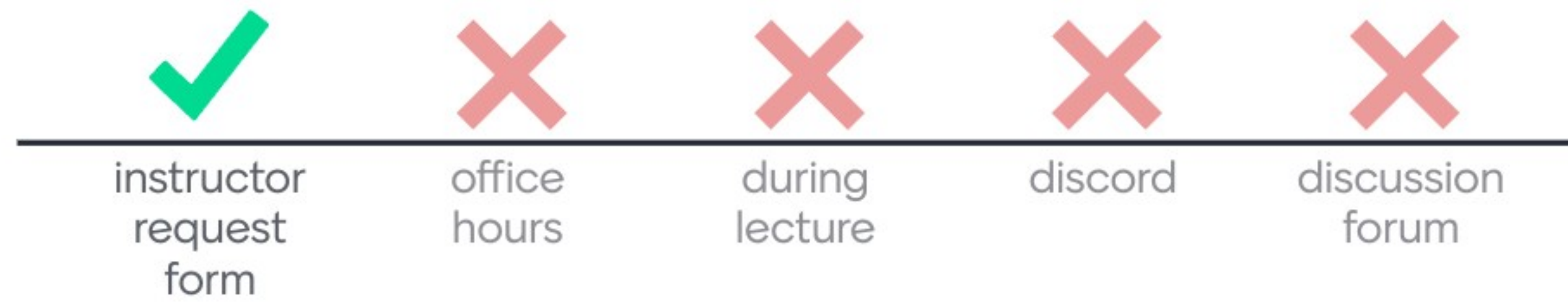
# why is git considered a software engineering tool?



If you have a question about the project requirements, where should you ask it?



If you think your grade is wrong, where should you ask about it?

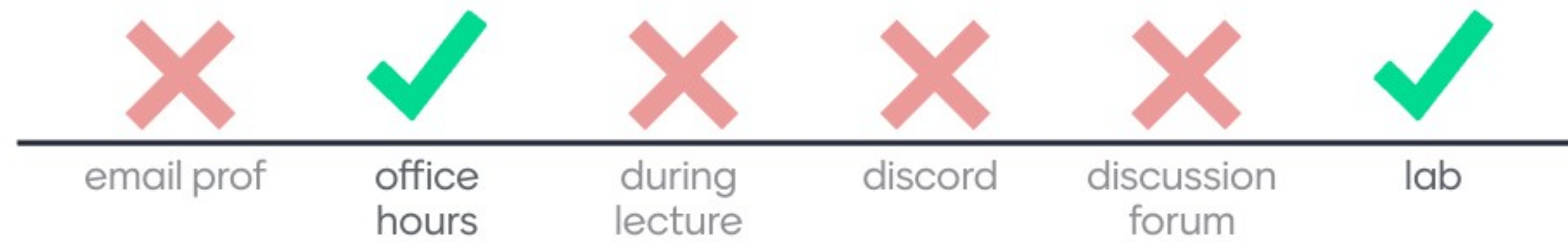




If you have a question about an assignment requirement where should you ask it?



If you need help with your code where should you ask about it?





# Sharing the mentimeter link is...



Discord is a good place to...



Over the history of programming languages we increased...



Making sure the data coming in or out of an object is correct..



Limiting the number of ways the object could be affected or affect other objects...



Breaking big complicated things into a bunch of small simple things...





Making sure we don't repeat ourselves by moving common functions into a shared class/file/method/function...



# UML is useful for... (1/2)



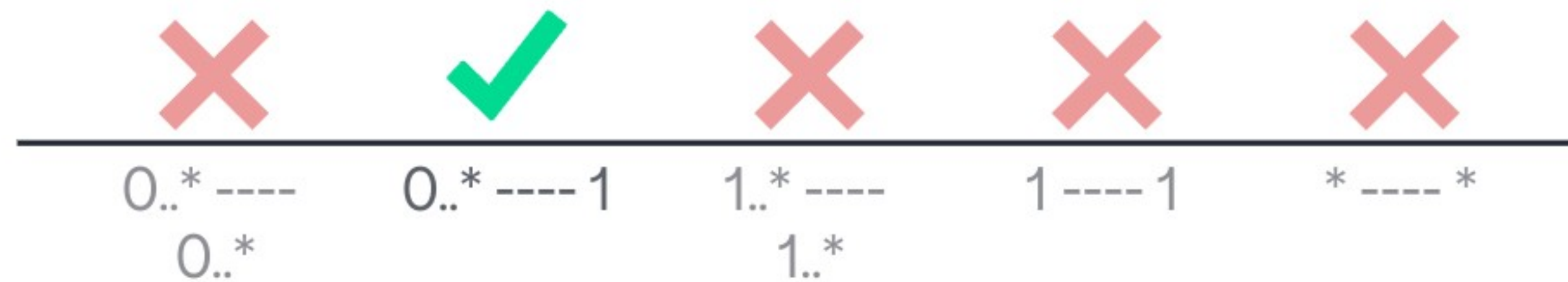
## UML is useful for... (2/2)



every Trainer object has zero or more Pokemon objects, every Pokemon object belongs to a trainer object



every Trainer object references zero or more Pokemon objects, every Pokemon object references one Trainer objects



# No diamond





# Filled-in diamond



# Outlined (empty) diamond



# Triangle arrow



# Skeleton arrow



Which side does the diamond go on



The multiplicity goes on \_\_\_\_\_ the thing that it is counting.



---

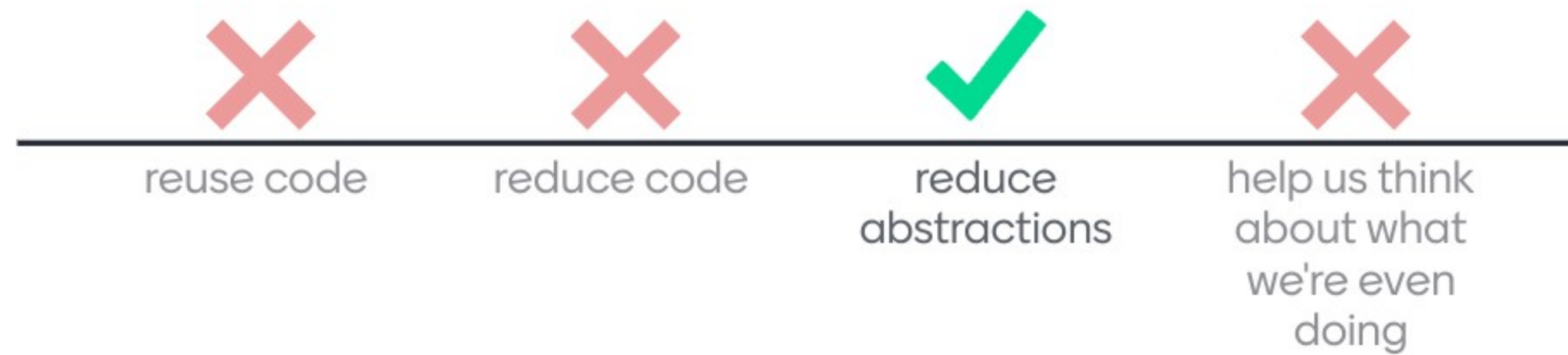
The same side as



The opposite side  
from



# What is NOT a reason to generalize?



Base class is also called a



---

superclass



subclass

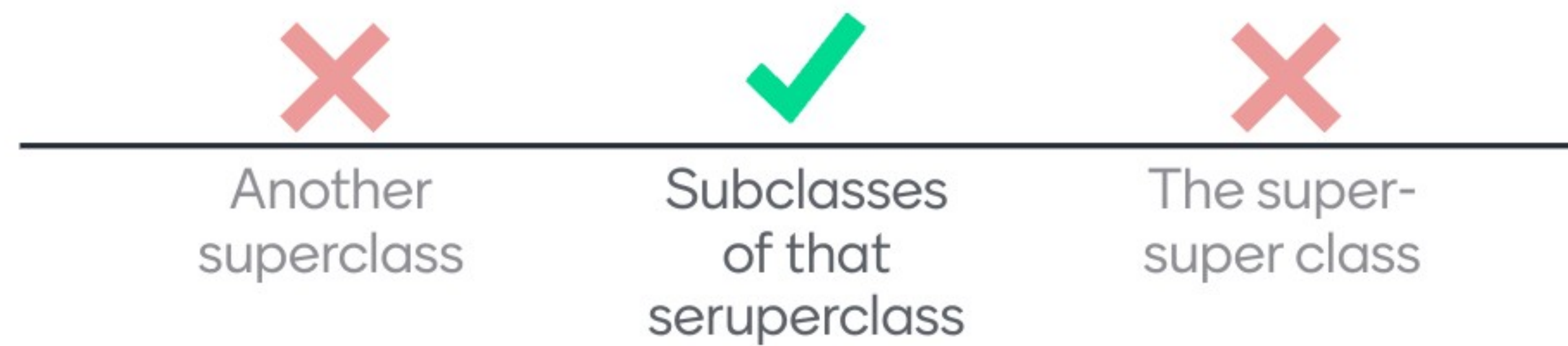


interface

Derived class is also called a







Everywhere you can use a superclass you can also use...



# Which one is a violation of the Liskov substitution principle?

---

			
Adding functionality to the superclass	Adding functionality to the subclass	Missing functionality that was in the superclass in the subclass	Missing functionality in the superclass that is in the subclass

Bulbasaur has growl(). Venusaur has growl() and growth(). Ivysaur has growl(), growth() and solarbeam(). Which should be the superclass?





I have a ClassB extends ClassA that violate the Liskov Substitution Principle... what's one potential way to solve it?



---

Ignore any use  
the missing  
functionality in  
the subclass



Remove  
extends  
ClassA, add  
aggregates  
ClassA



ClassA  
extends  
ClassB

In Java when I call a method, the arguments are passed



Even though its call-by-value, the values passed are usually



# Java constructors are named

---

			
<code>__init__</code>	constructor	:	the same name as the class



# Abstract classes provide



# Concrete (normal) classes provide











# Interfaces provide









# What's the quick rule of thumb to determine if something should be an inheritance?

---

					
Can be used anywhere another class is used without breaking things	"is a"	Has a few method signatures in common with other classes	weak "has a" / whole & parts	"made out of its own" / strong "has a"	"knows about" / "goes well with"









What's the Liskov Substitution principle to determine if something should be an inheritance?

					
Can be used anywhere another class is used without breaking things	"is a"	Has a few method signatures in common with other classes	weak "has a" / whole & parts	"made out of its own" / strong "has a"	"knows about" / "goes well with"

# What's the quick rule of thumb to determine if something should be an association?

---







					
Can be used anywhere another class is used without breaking things	"is a"	Has a few method signatures in common with other classes	weak "has a" / whole & parts	"made out of its own" / strong "has a"	"knows about" / "goes well with"











What's the quick rule of thumb to determine if something should be an aggregation?

---

					
Can be used anywhere another class is used without breaking things	"is a"	Has a few method signatures in common with other classes	weak "has a" / whole & parts	"made out of its own" / strong "has a"	"knows about" / "goes well with"

What's the quick rule of thumb to determine if something should be an composition?

					
Can be used anywhere another class is used without breaking things	"is a"	Has a few method signatures in common with other classes	weak "has a" / whole & parts	"made out of its own" / strong "has a"	"knows about" / "goes well with"



# Is composition relationship enforced in Java?



# For a composition relationship...



---

Part instances are deleted when the Whole instance is



Doesn't have a Whole and Part(s) style relationship



Part instances can be shared (they aren't exclusive)

# For an aggregation relationship...



---

Part instances are deleted when the Whole instance is



Doesn't have a Whole and Part(s) style relationship



Part instances can be shared (they aren't exclusive)

# For an association relationship...



---

Part instances are deleted when the Whole instance is



Doesn't have a Whole and Part(s) style relationship



Part instances can be shared (they aren't exclusive)

The diamond goes on the side of the ...

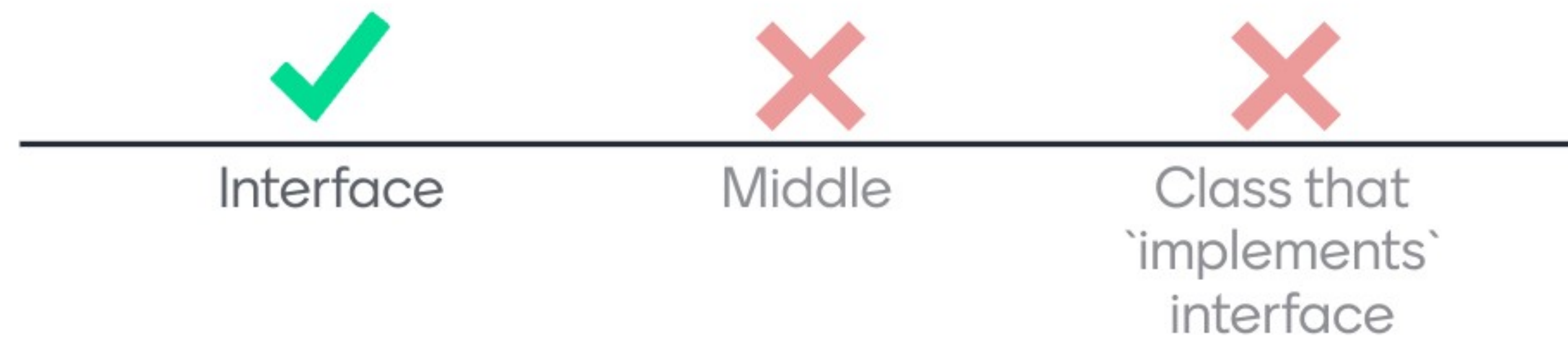


The arrow goes on the side of the ...





The arrow goes on the side of the ...



# What changes the model?



# What updates the views?



What makes things so the user can see them?



What does the user give commands to?



What should be completely INDEPENDENT of the UI toolkit (android/ios/windows...) used?





What should contain the representation of the core concepts (nouns/verbs from OOAD)?



What goes on the top of a CRC card?



What goes on the LEFT of a CRC card?



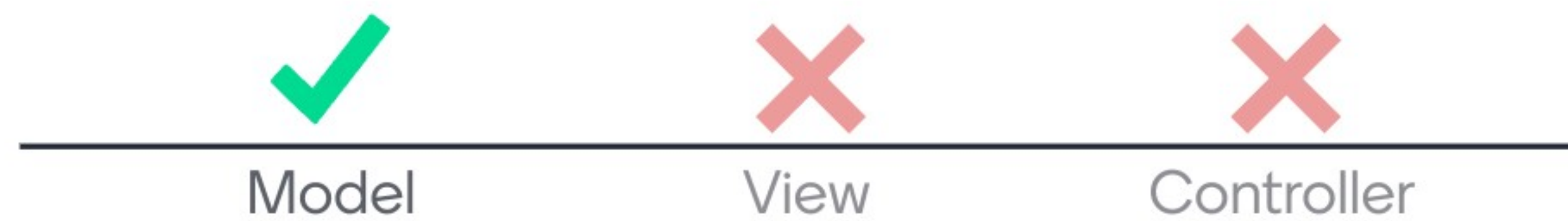
What goes on the RIGHT of a CRC card?



# What goes on the BACK of a CRC card?



In an app about Pokemon, a Pokemon's current level should be kept in the

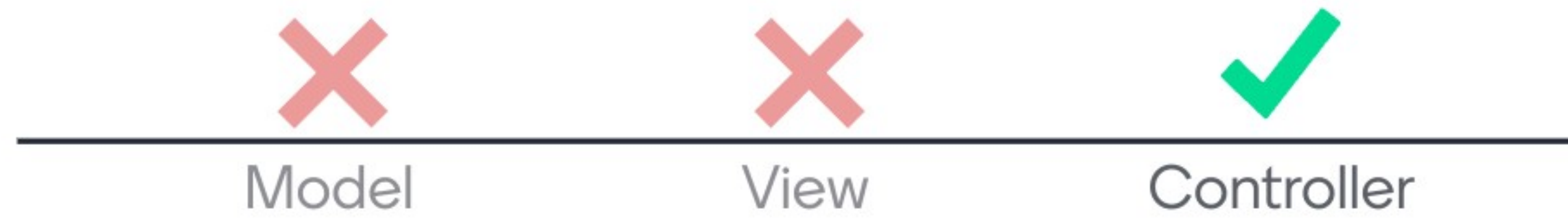




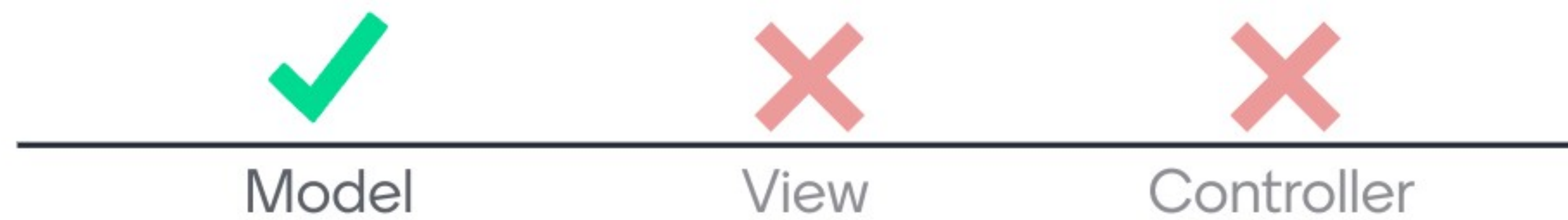
In an app about Pokemon, a Pokemon's picture should be kept in the



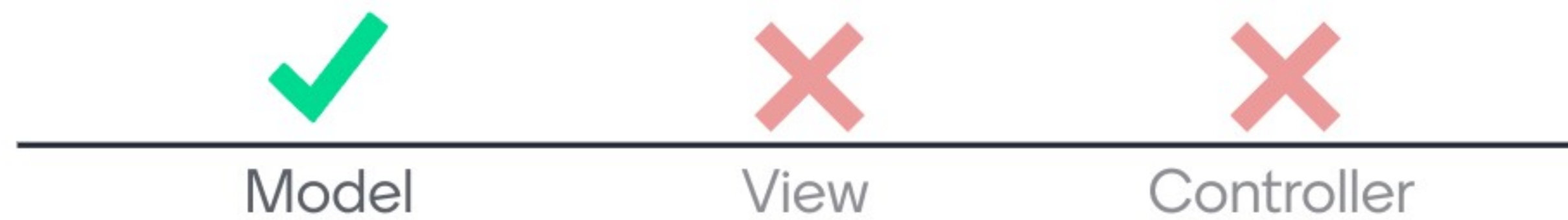
In an app about Pokemon, the player tells what to have their squirtle do a water gun attack?



When I walk my poisoned pokemon, it slowly loses health... what should contain the code for that?



When I save my Pokemon game, what is going to be saved?



What pattern is used in the kind of MVC we do in this class?



In the MVC used for our class, every model usually has how many views?





In the MVC used for our class, every view usually has how many controllers?



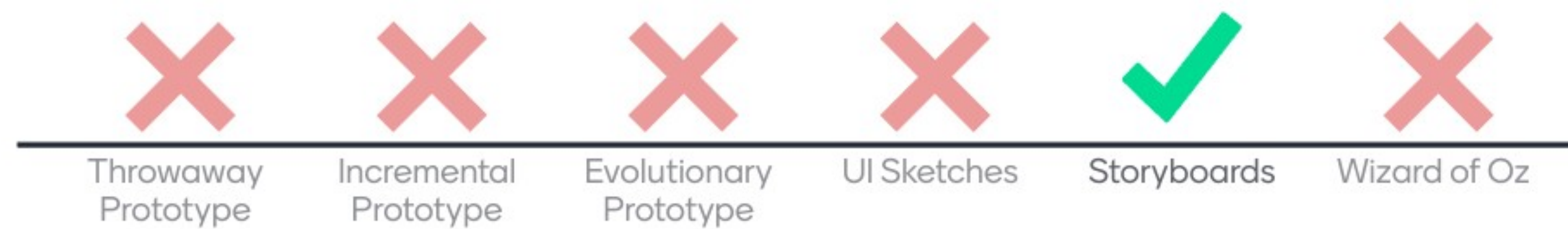
In the MVC used for our class, every controller usually has how many models?



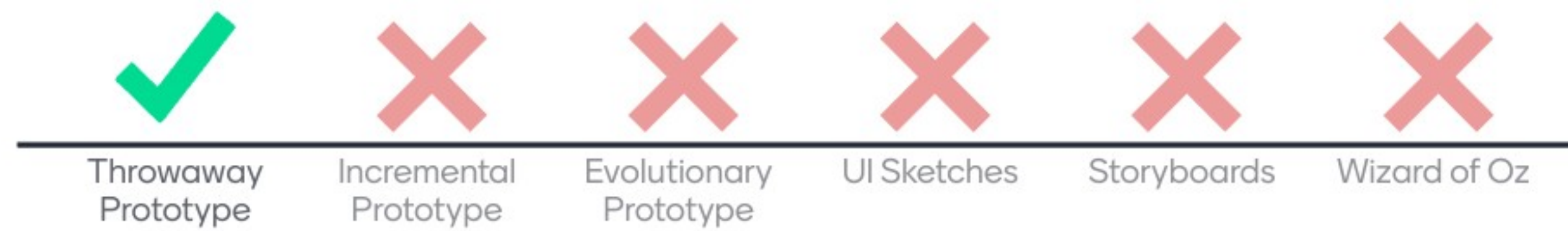
Each iteration of the prototype, every feature gets a little better.



# Visual example of how a user might go through completing a task in the UI



Each iteration we make a new prototype





There's a person running things but it looks like its code doing the work





# Drawings of what the various screens would look like



Customer has to wait a long time



Agile or waterfall: make sure the requirements are correct.



Agile or waterfall: make sure the requirements are correct before starting development.



Agile or waterfall: allow developers to choose what part to work on.



Agile or waterfall: short iterations called sprints.





Agile or waterfall: treats software development like a factory assembly line.



Agile or waterfall: change direction immediately when customer needs change.



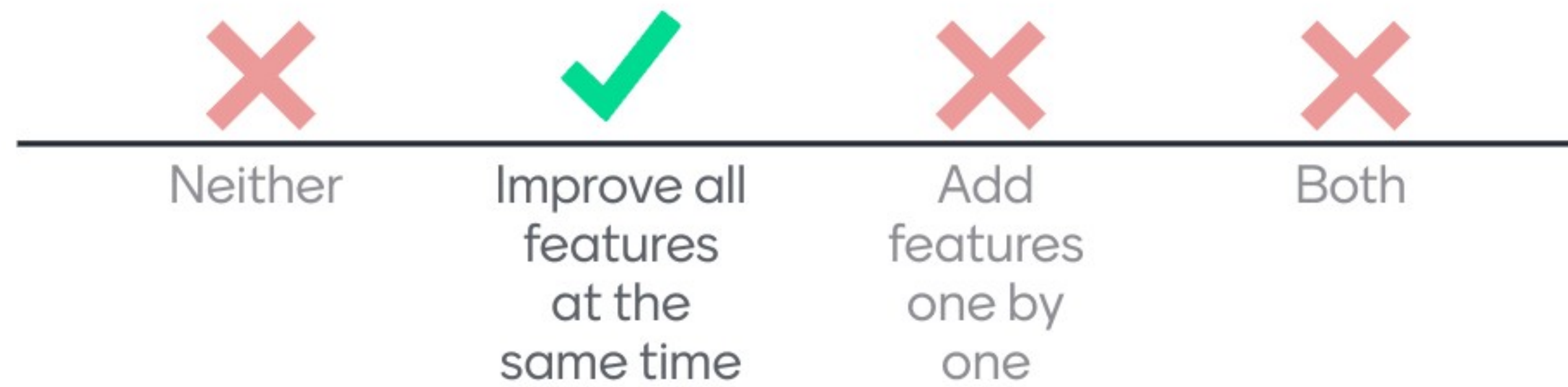
Agile or waterfall: work the same amount this month as last month and next month.



# Agile or waterfall: cares a lot about testing



# Evolutionary prototyping



# Incremental prototyping





