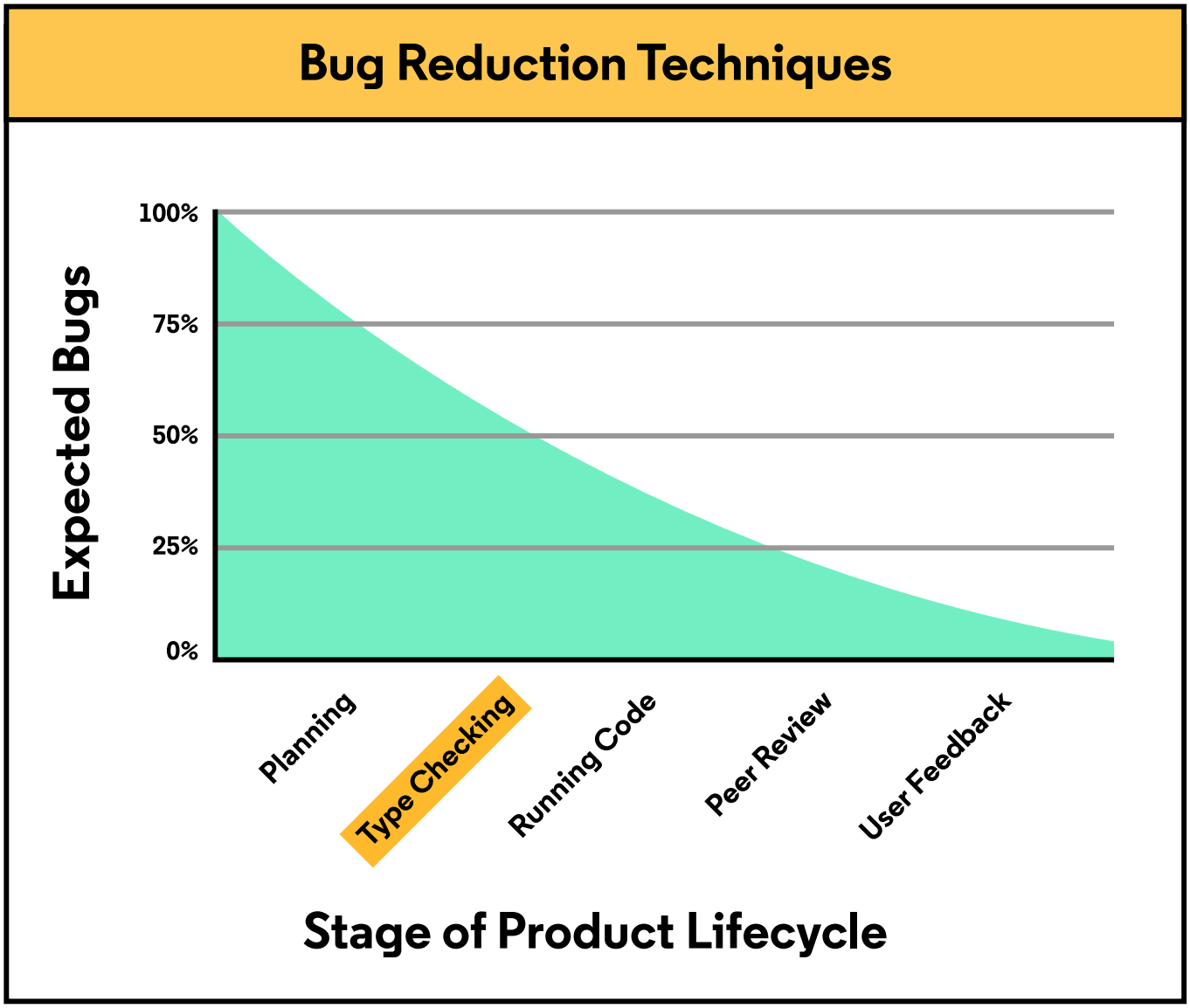
Introduction to Interfaces

Since programming is complex, it's easy to make mistakes. For example, maybe we try to use an `int` like a `string`. This would cause a type error. C# has rules built-in that check for those mistakes before we use that code in the real world.

The below diagram estimates the number of bugs in our code at each stage of development. We discover and fix some of those bugs at each stage, so the number decreases from left to right. Sadly we can't fix every bug. But if we can catch bugs earlier, then we save ourselves work and we save users frustration.

We say that C# is "type-safe" because it helps us catch bugs at one of the earliest stages: the type checking stage.



The difference between an `int` and a `string` is clear, but what about custom-defined classes? Say we made `Fruit` and `Vegetable` types. Can we use `PlantSeed()` on both? Do they both have a `Seed` property? The computer running the program needs more information.

In this lesson we'll dive into *interfaces*, which are sets of actions and values that describe how a class can be used. This lets the computer check that we are using each type legally, thus preventing a whole group of type errors!

In this lesson you'll learn how to:

- build your own interface
- write classes that implement an interface

☑Instructions

Based on the code in **Program.cs**, we can expect the `Sedan` class to have:

- a `Speed` property
- a `LicensePlate` property
- a `Wheels` property
- a `Honk()` method

By the end of this lesson you'll build an interface that guarantees that all Sedans have these behaviors and attributes.