selsoft
build better, deliver faster

Temiz Kod (Clean Code)

6. Bölüm - Test Güdümlü Programlamaya Giriş

Eğitmen:
Akın Kaldıroğlu
Çevik Yazılım Geliştirme ve Java Uzmanı

Konular

- Doğruluk
- TDD (Test-Driven Development)
  - TDD Life-Cycle
- Testing
- Unit Testing
- Integration Testing
- Refactoring
- Resources

Doğruluk

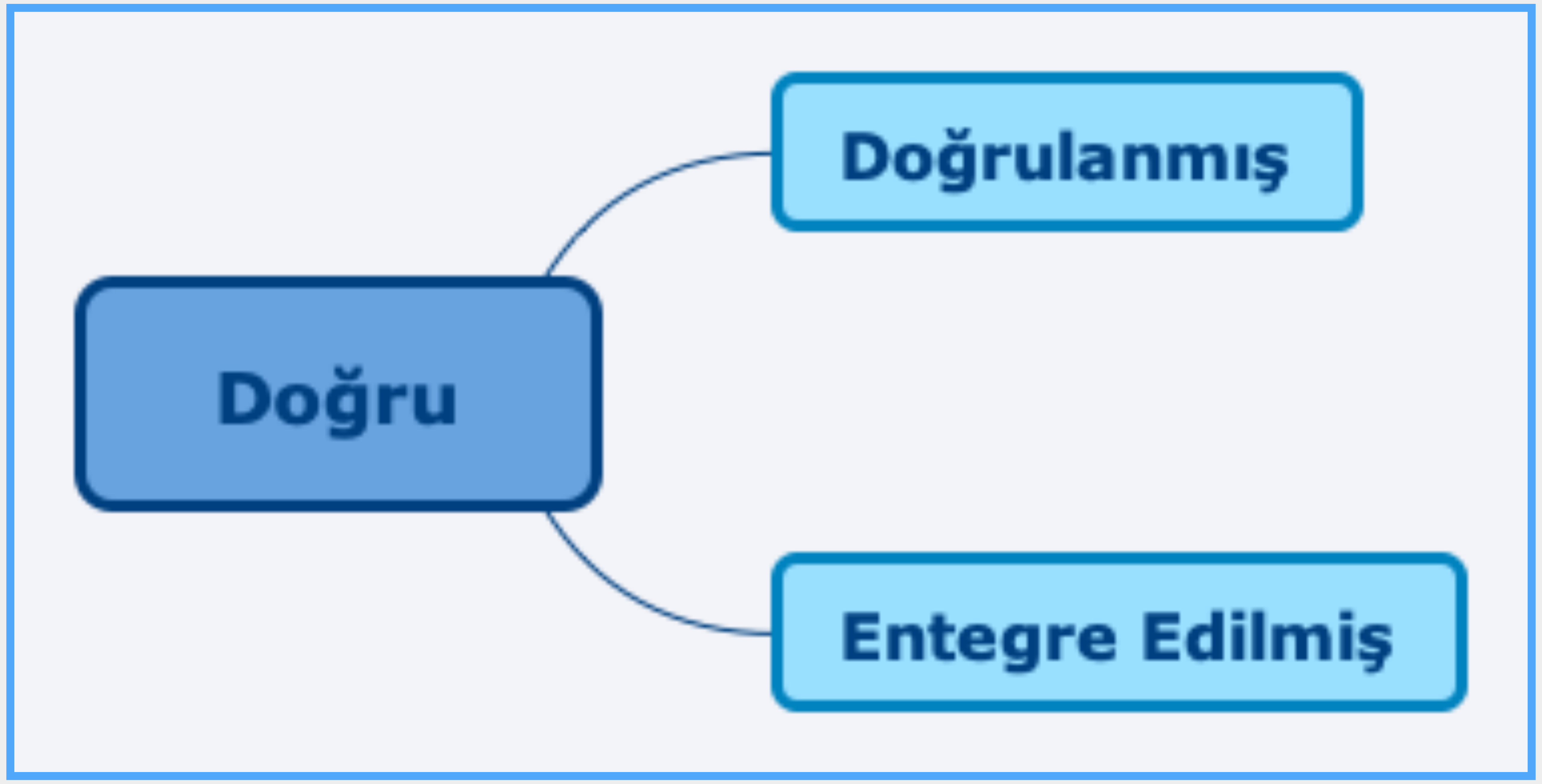
selsoft

build better, deliver faster

Doğru Kod - I

- **Temiz kod doğrudur (correct).**
- Bir kodun kendi başına doğruluğu ancak **birim testi**yle (**unit test**) sınanabilir.
- Bir kodun çevresindeki diğer kod parçalarıyla birlikte doğru çalıştığı ise ancak **entegrasyon testi**yle sağlanabilir.
- Birim ve entegrasyon testleri TDD yaklaşımı ile yapılırsa, doğruluk ve tamlık konusunda daha iyi seviyeler elde edilebilir.

Doğru Kod - II



Test-Driven Development

selsoft

build better, deliver faster

TDD & XP

- TDD is an approach to software development created within the paradigm of the **eXtreme Programming** (**XP**) by **Kent Beck**.
- Values of XP are communication, simplicity, feedback, courage, and respect.
- TDD is a technique beside other ones such as **Pair Programming** in XP.

How TDD Conceived - I

- In his book “**Test-Driven Development By Example**” while thanking to those that he owes for the book Beck says:

  Finally, to the unknown author of the book which I read as a weird 12-year-old that suggested you type in the expected output tape from a real input tape, then code until the actual results matched the expected result, thank you, thank you, thank you.

How TDD Conceived - II

- Beck later says:

  I thought, what a stupid idea. I want tests that pass, not tests that fail. Why would I write a test when I was sure it would fail. Well, I’m in the habit of trying stupid things out just to see what happens, so I tried it and it worked great.

  I was finally able to separate logical from physical design. I’d always been told to do that but no one ever explained how.

Goal of TDD - I

- **Primary goal of TDD is specification not validation.**
- The central notion of TDD is “test-driven” not just “test”!
    - That’s why, “test-driven” qualifies “development”.
- It is about separating what from how at the most concrete level of software development namely coding.

Goal of TDD - II

- According to Ron Jefries **the goal of TDD is clean code that works.**
- So TDD is about the quality of
  - Implementation: Does it work? or Do we build the right product?
  - Design: Is it well structured? or Do we build the product right?

What is TDD?

- TDD is an evolutionary approach to development which combines test-first development where you write a test before you write just enough production code to fulfill that test and refactoring.
- Scott Ambler says **TDD = Refactoring + TFD**

Kent Beck Says - I

- K. Beck says in “**Test-Driven Development by Example**”:
- **In TDD, we**
    - **Write new code only if an automated test has failed,**
    - **Eliminate duplication.**

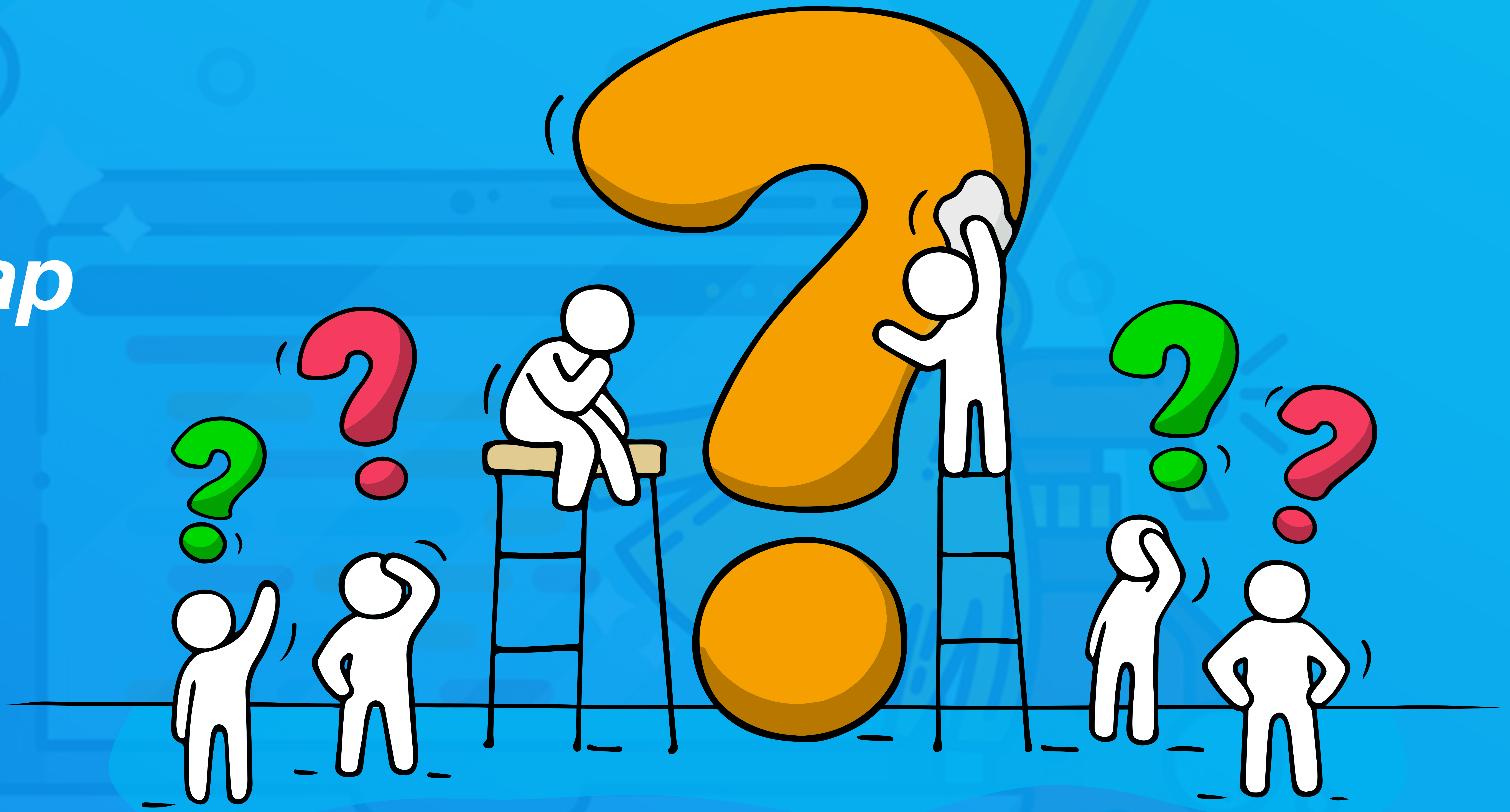
Kent Beck Says - II

- We must design organically, with running code providing feedback between decisions.
- We must write our own tests, because we can't wait 20 times per day for someone else to write a test.
- Our development environment must provide rapid response to small changes.
- Our designs must consist of mainy highly cohesive, loosely coupled components, just to make testing easy.

About TDD

- TDD is mostly considered as a technique to avoid bugs and to make sure that the business logic works as expected.
    - Achieving high percentage in code coverage becomes most important target.
    - Because it serves to having a bug-free code base.
- But TDD is more than that.
    - In fact it becomes a design tool when its test-first and refactoring principles are applied.

Soru ve Cevap Zamanı!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

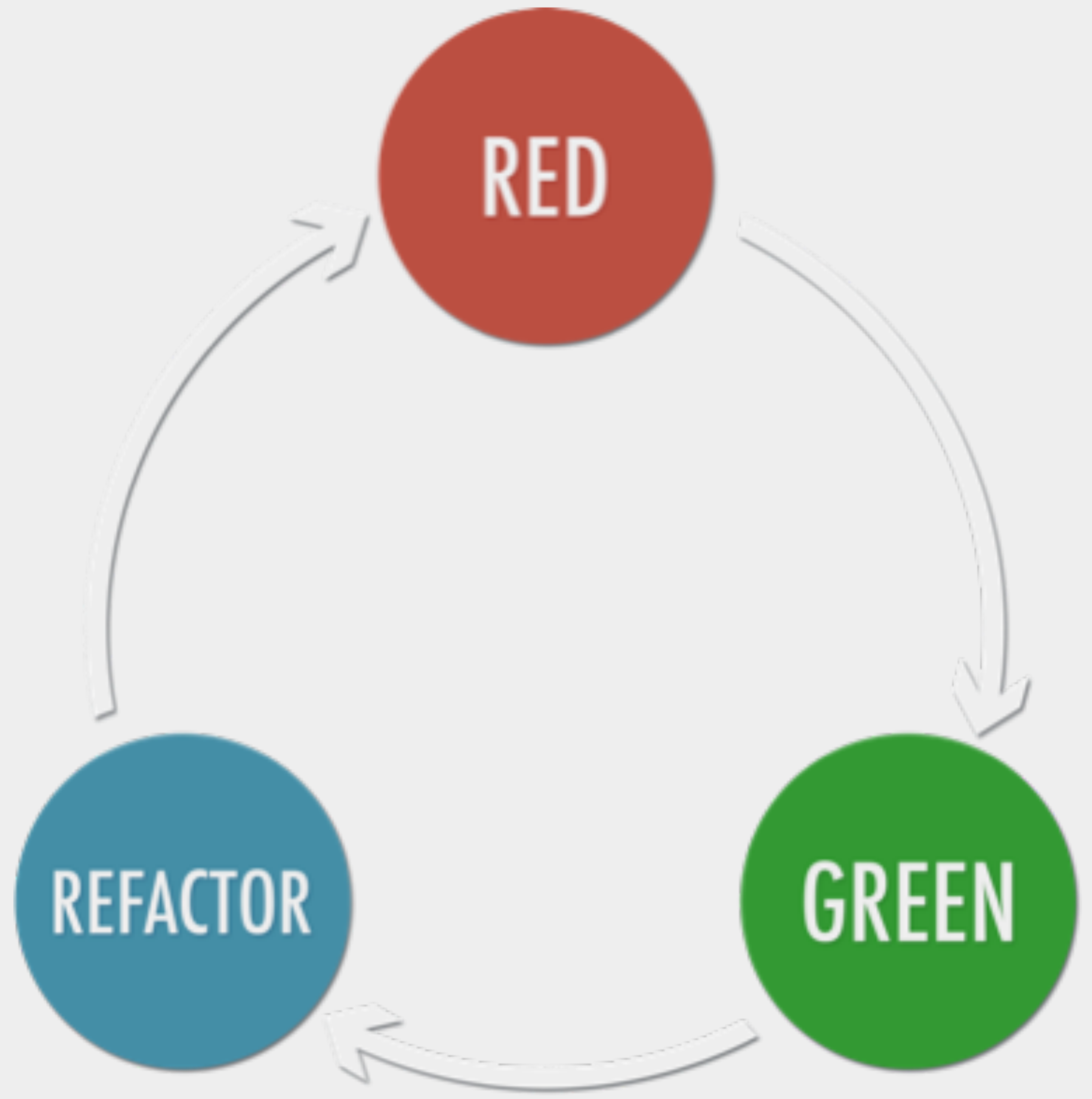
selsoft

build better, deliver faster

TDD Life-Cycle

TDD Life Cycle - I

- TDD life cycle has three steps, **RED-GREEN-REFACTORING**:
  - **RED**: Testing
  - **GREEN**: Coding
  - **Refactoring**
- **RED** => test failed
- **GREEN** => test passed
- **REFACTOR** => Cleaning the code



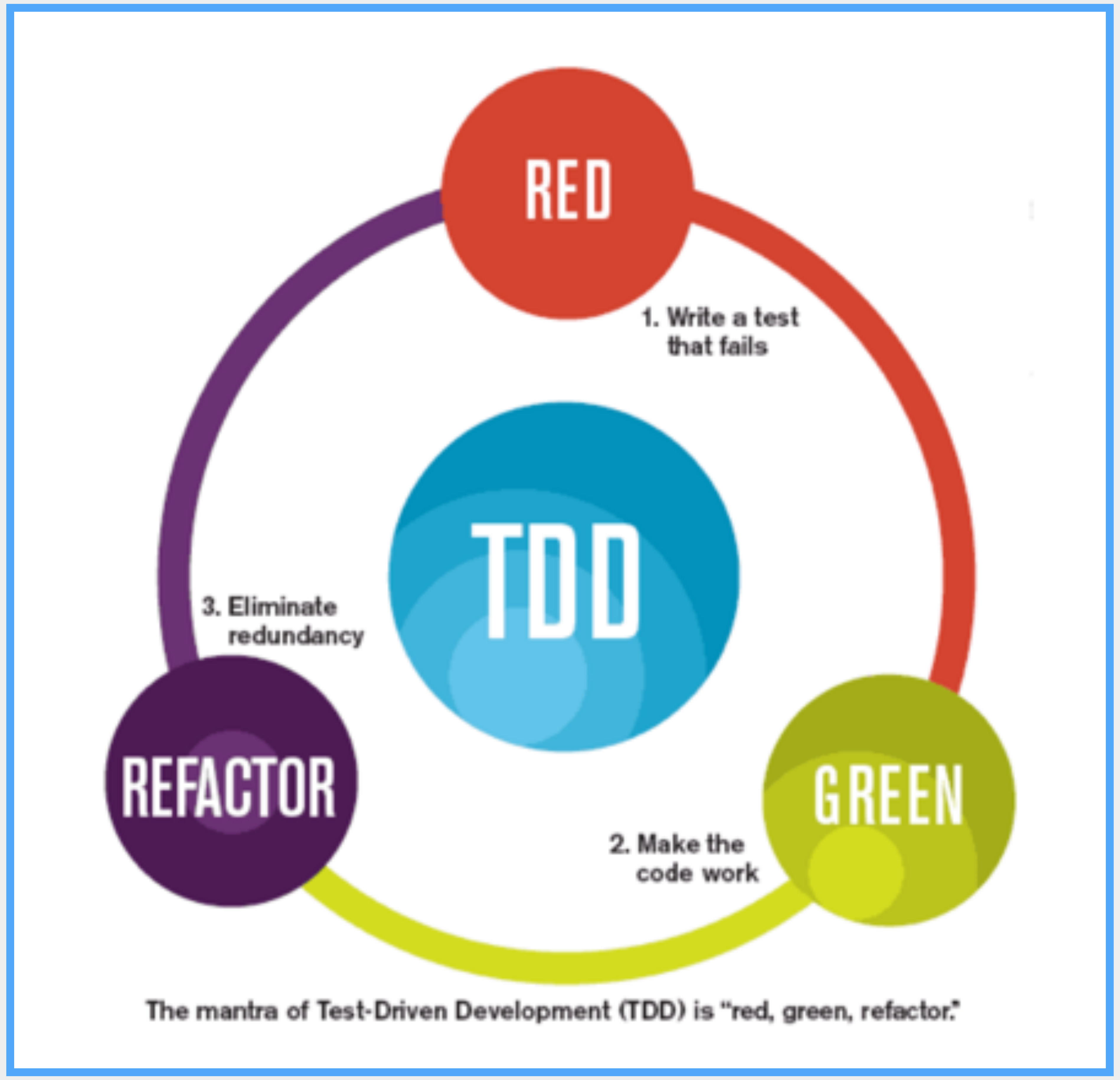
TDD Life Cycle - II

- Martin Fowler says:

Write a test for the next bit of functionality you want to add,

Write the functional code until the test passes,

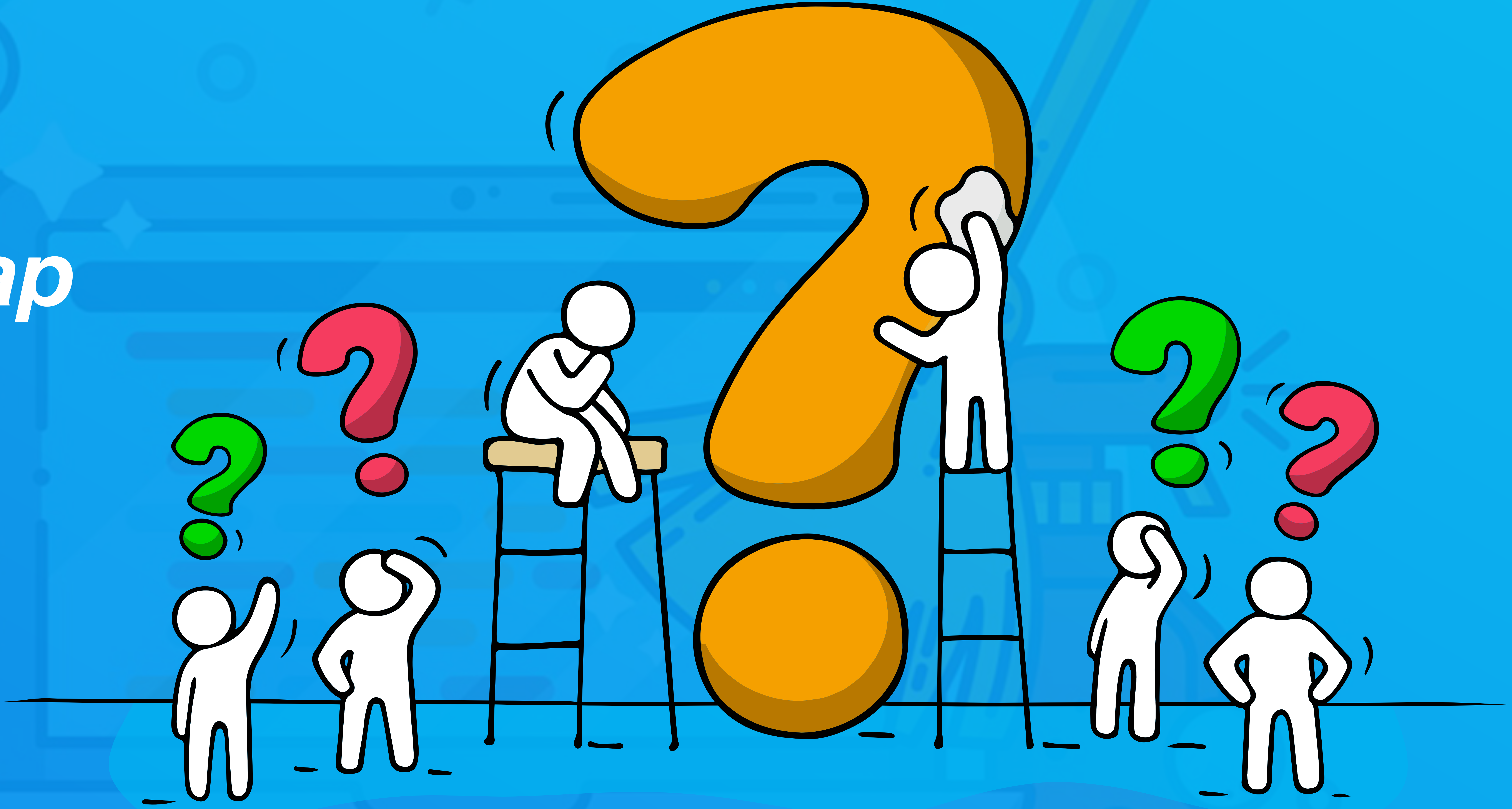
Refactor both new and old code to make it well structured.


RED
1. Write a test that fails
TDD
GREEN
2. Make the code work
REFACTOR
3. Eliminate redundancy
The mantra of Test-Driven Development (TDD) is "red, green, refactor."

TDD Life Cycle - III

- TDD is a style of development where:
  - You maintain an exhaustive suite of programmer tests,
  - No code goes into production unless it has associated tests,
  - You write the tests first,
  - The tests determine what code you need to write.
- There is no code unless there is a test that requires it in order to pass.
- When there are no more tests the implementation finishes.

Soru ve Cevap Zamanı!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

Testing

selsoft

build better, deliver faster

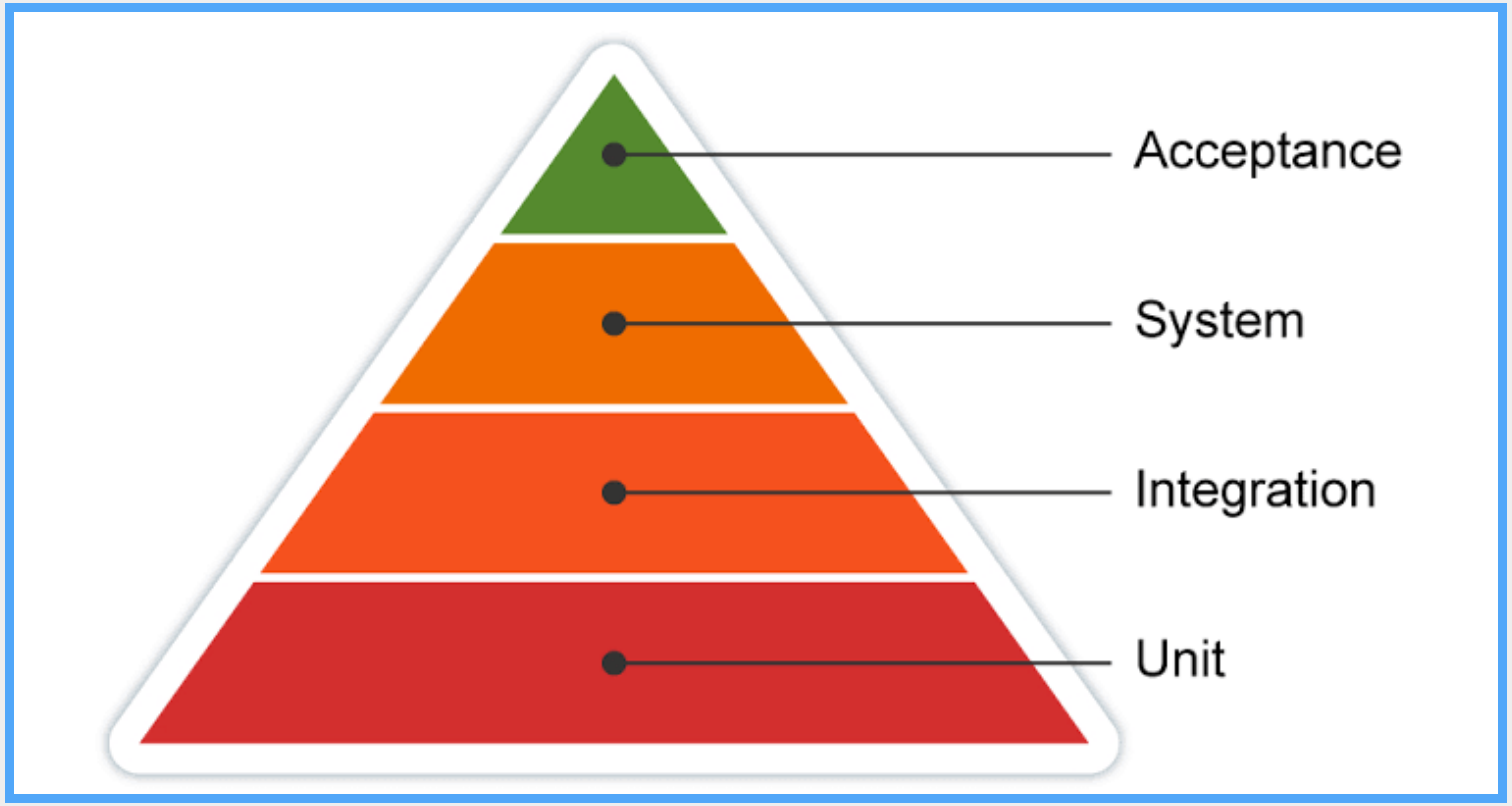
Test Class and Method

- To test is a verb meaning "**to evaluate**"
- For Beck, main wisdom regarding the testing:

No software engineers release even the tiniest change without testing, except the very confident and the very sloppy.

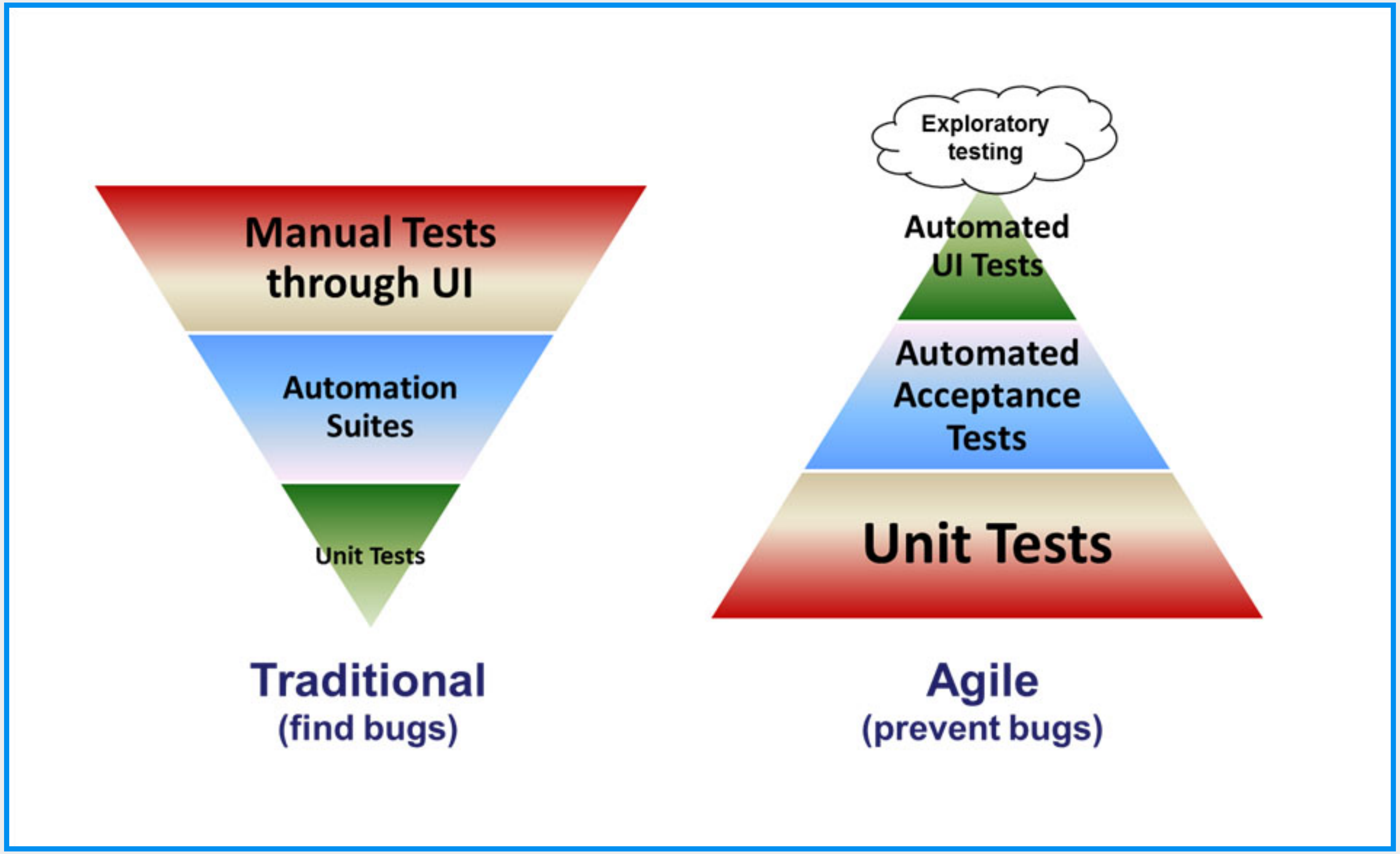
Testing Levels - I

- Test pyramid gives a big picture of tests at different levels.



Testing Levels - II

- There are several issues regarding the test pyramid:
  - Upper levels take longer, requires more infrastructure thus are more expensive than lower ones
  - While lower levels are done by developers, upper levels are done by QAs and users
  - For lower levels preventing bugs are the focus but for upper levels the focus shifts to finding bugs
  - Number of tests are many more in lower levels than upper ones


Manual Tests through UI
Automation Suites
Unit Tests
Traditional
(find bugs)
Exploratory testing
Automated UI Tests
Automated Acceptance Tests
Unit Tests
Agile
(prevent bugs)

Unit Testing

selsoft
build better, deliver faster

Testing in TDD

- Test in TDD is mainly unit testing.
- So TDD is a strategy to prevent bugs not to find them.
- TDD implements main testing philosophy of Beck: **test early, test often, test automated**
- Since the number of unit tests grows very fast they should be managed and run automatically.

What is Unit? - I

- In OOP, the unit is class, in procedural programming it is function.
  - But a class consists of methods in terms of functionality.
- It depends on how you define it or on the context.
- Sometimes it is a single method or sometimes it is a bunch of classes behind a single method being tested.

What is Unit? - II

- **Unit = Birim**
  - Unit is sometimes transated as “bileşen” but that’s not correct because bileşen is component, unit test is not bileşen testi!
- **Unit test = Birim Testi**
- It’s been around longer than TDD and even XP.
- When we talk about ”test” we mean unit test of TDD, i.e. test done before writing the implementation code,
  - Test-first development!

When Unit Testing? - I

- Unit tests can be written before or after writing the functional code they test.
  - The difference between these two are drastic.
- If you write them after writing the code, the code determines the unit tests.
- It is still good but what unit tests do is mostly the confirmation of the code through testing.
  - It is a misunderstanding of T in TDD.

When Unit Testing? - II

- If you first write the unit tests and then write the code, the unit tests becomes a design tool.
  - In which case tests become the specification, what determines the code.
- Unit tests convert requirements into design that is expressed in code.

When Unit Testing? - III

- Robert Martin says:

  The act of writing a unit test is more an act of design than of verification. It is also more an act of documentation than of verification. The act of writing a unit test closes a remarkable number of feedback loops, the least of which is the one pertaining to verification of function.

Unit Tests As Documentation

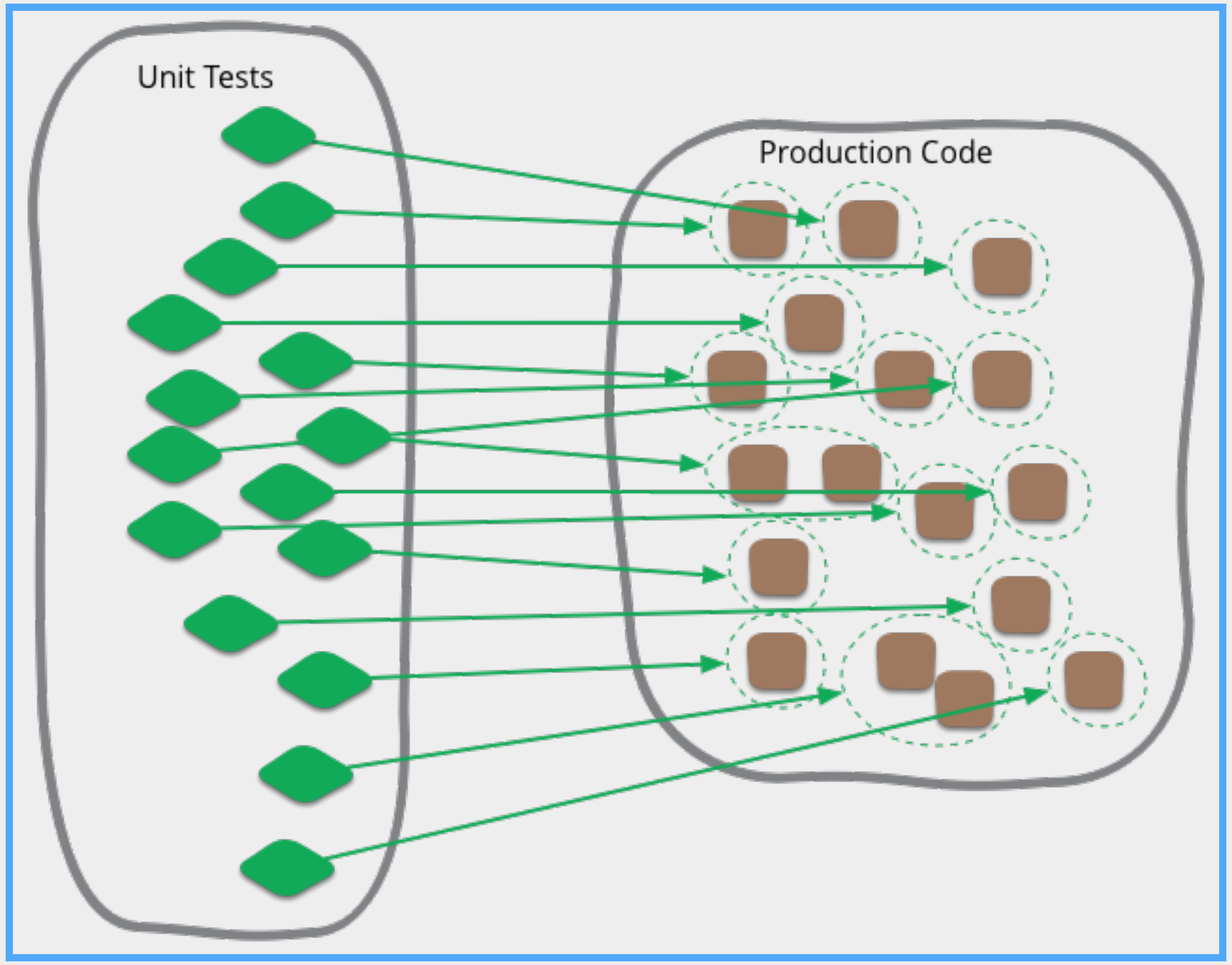
- Tests of TDD provide living documentation for the interfaces of the units.
    - To understand what a class does you can look at the tests associated with that class.
    - It is much easier to read test code than real code in order to understand what the real code does.
- Implementation code provides all needed details while test code acts as the description of the intent behind the production code.

Code Coverage - I

- Code or test coverage is a measure used to describe the degree to which a software system is tested.
- In code coverage it is mostly the degree in terms of percentage to which the code is unit tested.
- Don’t confuse test coverage with test quality.
- Higher test coverage is of course better if the quality of the test is not sacrificed to reach that coverage.

Code Coverage - II

- Main aim of unit testing is there would be no non-trivial code that is not unit-tested!
    - High-coverage with high-quality tests



Programmer/Developer Test

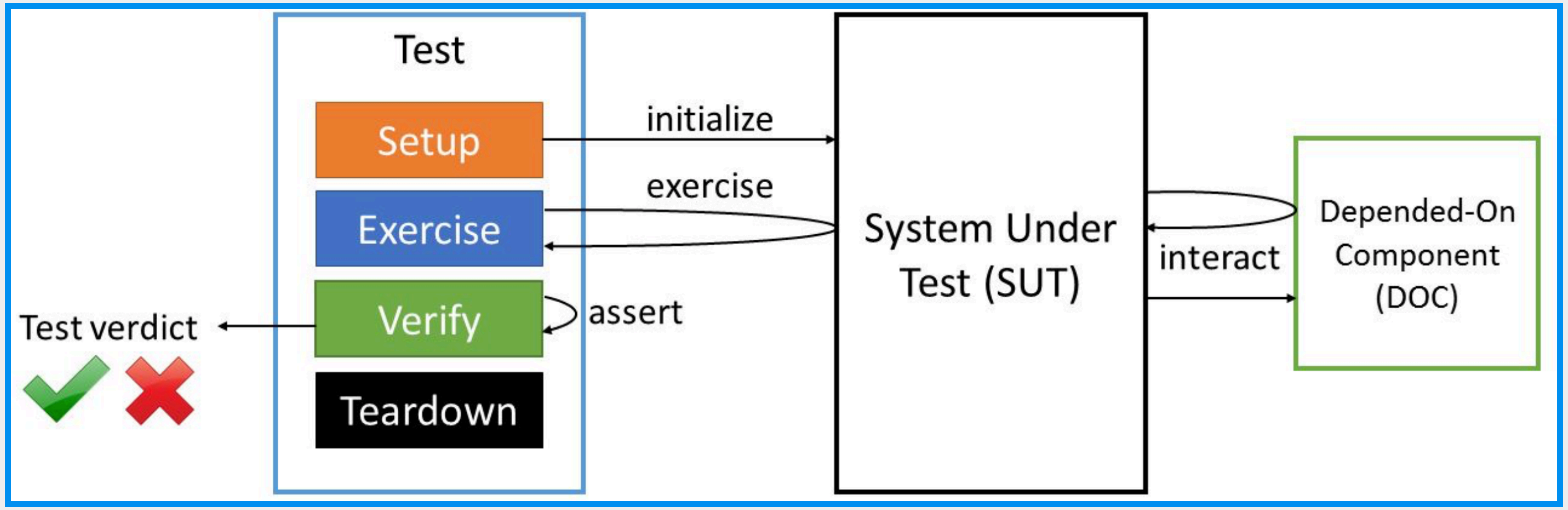
- Unit tests are written by the programmers.
    - So don’t wait your code to be tested!
- That’s why unit test is also called programmer/developer test.
- In TDD unit test are written by the programmer whıo would develop the functional code.

Some Unit Test Terms - I

- Test/Unit Test
- Unit/System under test (U/SUT)
- Dependent on component (DoC)
- Test Double, Mock objects & Mocking
- Test case and Test Suite
- Test class and Test method
- Arrange, Act, Assert/Assemble, Activate, Assert
- Test fixture
- Setup
- Assertion
- Teardown

SUT & DOC

- Arrange or Setup
- Act or Exercise
- Assert or Verify
- Teardown



Exercise: CF Converter - I

- Implement following requirement and unit test it to make sure the implementation works correctly:
  - The system shall convert between Celcius and Fahrenheit degrees using following formula:

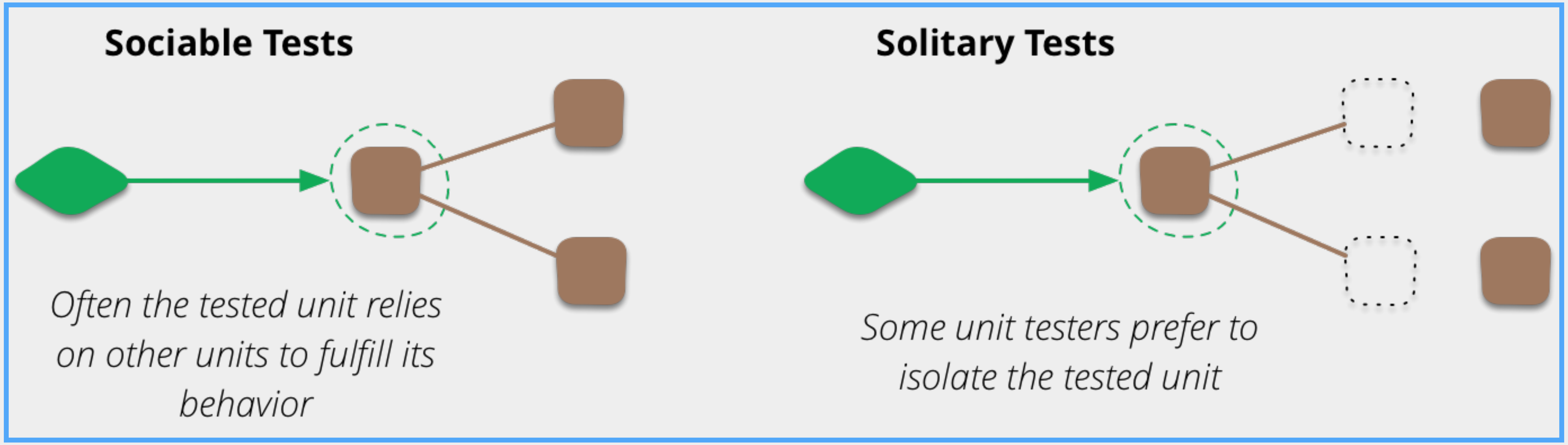
```
T(°F) = T(°C) × 9/5 + 32

T(°C) = (T(°F) - 32) × 5/9
```

Exercise: CF Converter - II

- Converter should notify the user for invalid values.
- Following input values are invalid:
  - For Celcius, the values smaller than -273
  - For Fahrenheit the values smaller than -459,40

Isolation

- One of the distinctive features of unit testing is isolation.


Sociable Tests
Often the tested unit relies on other units to fulfill its behavior
Solitary Tests
Some unit testers prefer to isolate the tested unit

Mocking - I

- To implement solitary tests, it is necessary to isolate units from their dependent objects for testing
- For this purpose mocking is used.
- Mocking means creating mock objects and training them for required and temporary behavior until the real objects are available,
- There are many mock object frameworks that can be used in unit tests.

Mocking - II

- Sometimes the objects that unit depends on are not ready yet,
- Or there are some infrastructural objects such as HttpRequest needed for the unit tests but it is hard or impossible to provide that setup,
  - Deferring the unit tests is not a good solution at all!
- Mocking helps in those situations too.

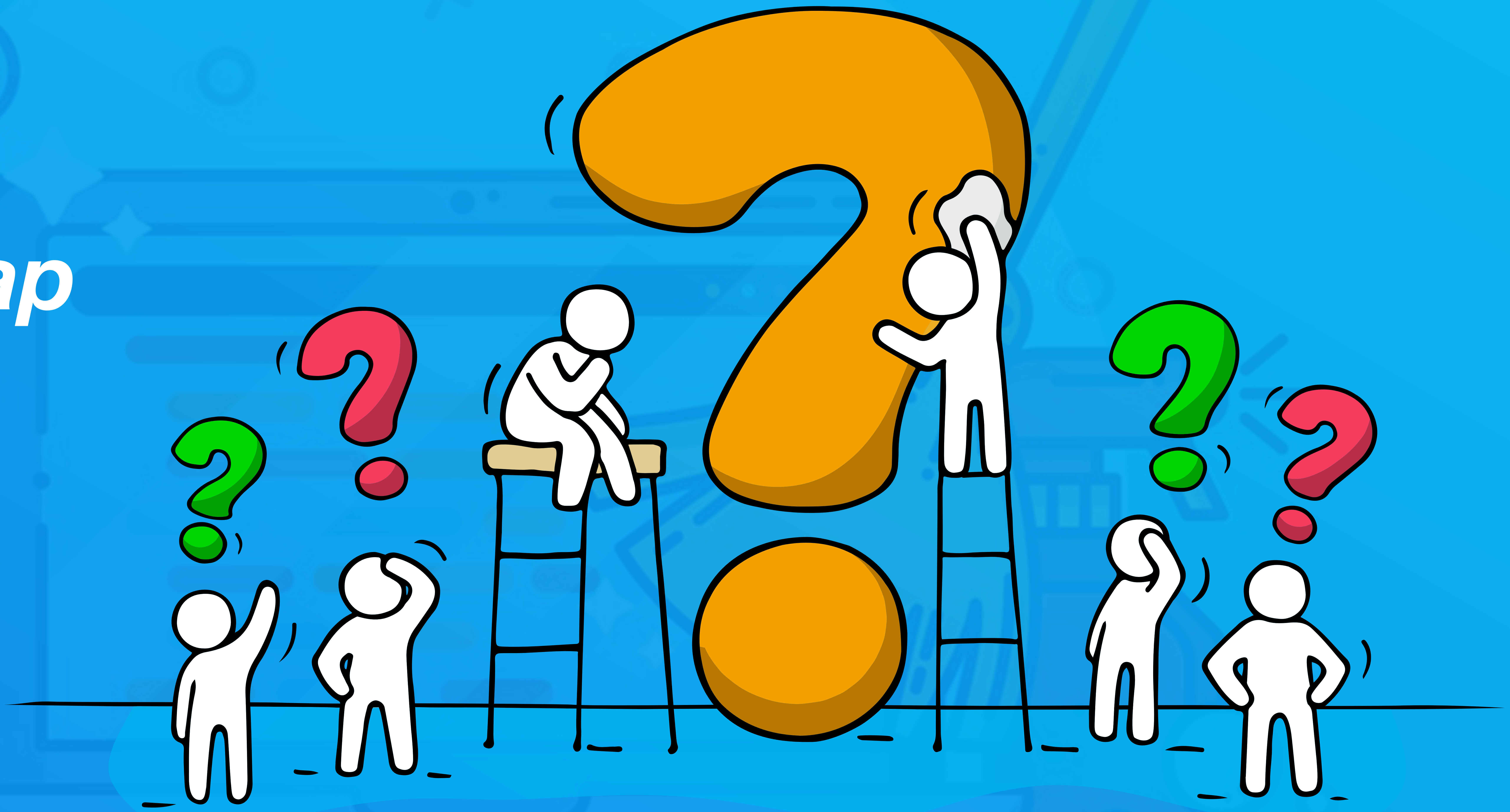
White Box or Black Box

- In terms of the software system, it is a white-box testing:
  - Because it tests the internal structures of the system and
  - It does not always test the functionalites of the system visible from an external perspective.
- In terms of the components of the software system, it is a black-box testing:
  - Because it tests the internal structures of the system through their APIs treating them as black boxes.

Unit Testing As Regression

- Unit tests are used for the purpose of regression testing after any changes are made to the code base.
- When unit tests are run automatically it becomes much easier to run all necessary tests as regression tests.
- New bugs discovered during the regression tests are fixed in refactoring.
- With this the code base are kept without bugs.

Soru ve Cevap Zamanı!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

Integration Testing

selsoft
build better, deliver faster

Integration Testing

- Even though each unit of a system works well in isolation they may have some problems with working together.
- So the scope of the test for unit tests is narrow including only one unit while that of integration tests is wider including several units.

Refactoring

selsoft
build better, deliver faster

What is Refactoring?

- Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure.
- It is a disciplined way to clean up code that minimizes the chances of introducing bugs.
- Refactoring means improving the design of the code after it has been written.

Why Refactoring?

- Refactoring is the third and last step in TDD life cycle.
- Because software corrupts as it grows.
- To be able to keep up a sustainable pace it is a must to improve the structure of the code.

How To Refactor? - I

- Main activities of refactoring may be:
  - Removing unnecessary code
  - Correcting any deviations from architectural and functional design decisions
  - Eleminating the code duplication
  - Restructuring the existing units to make them more cohesive and less coupled such as dividing classes and methods

How To Refactor? - II

- Applying design patterns to improve the design
- Improving non-functional aspects of the code such as performance
- Making the code cleaner for example by renaming things
- And others such as documentation, etc.

Tools and Code Reviews

- Using static code checkers and making code reviews periodically helps refactoring.

Neglecting Refactoring

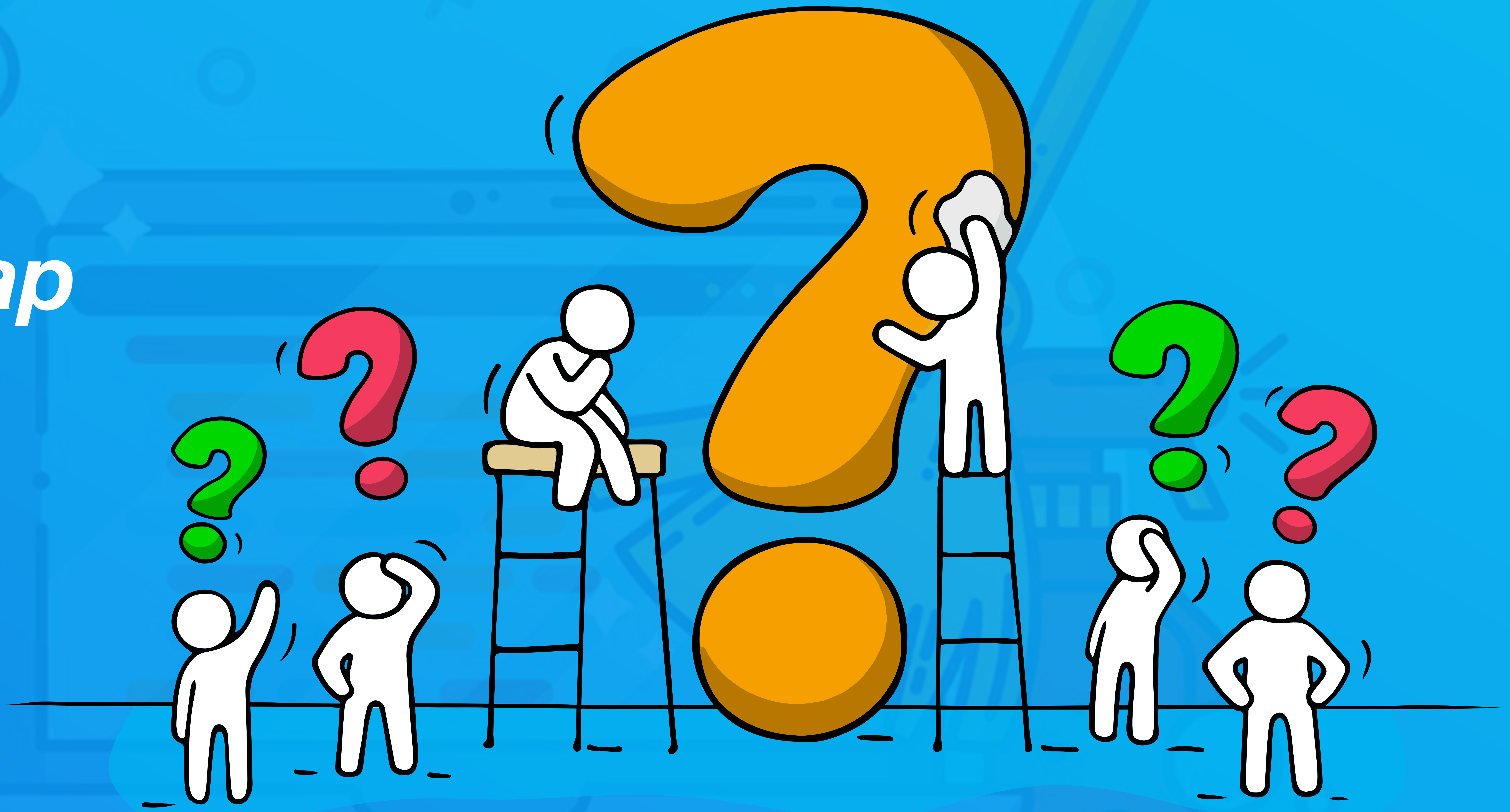
- Refactoring is the most neglected step of TDD.
- If you avoid sparing time for refactoring and don’t make it a natural part of the development, most benefits of TDD get lost.
- If you neglect refactoring even unit tests would get harder due to corruption of software and eventually you give up doing TDD.

First Step in Refactoring

- M. Fowler says in his book **Refactoring**:

  Whenever I do refactoring, the first step is always the same. I need to ensure I have a solid set of tests for that section of code.

Soru ve Cevap Zamanı!



selsoft
build better, deliver faster

info@selsoft.com.tr
selsoft.com.tr

Resources

selsoft

build better, deliver faster

Resources - I

- **Test-Driven Development By Example**
  - Kent Beck, Addison-Wesley, 2002
- **xUnit Test Patterns: Refactoring Test Code**
  - Gerard Meszaros, Addison-Wesley, 2007
- **Test-Driven Development: A Practical Guide**
  - David Astels, Prentice Hall, 2003

Resources - II

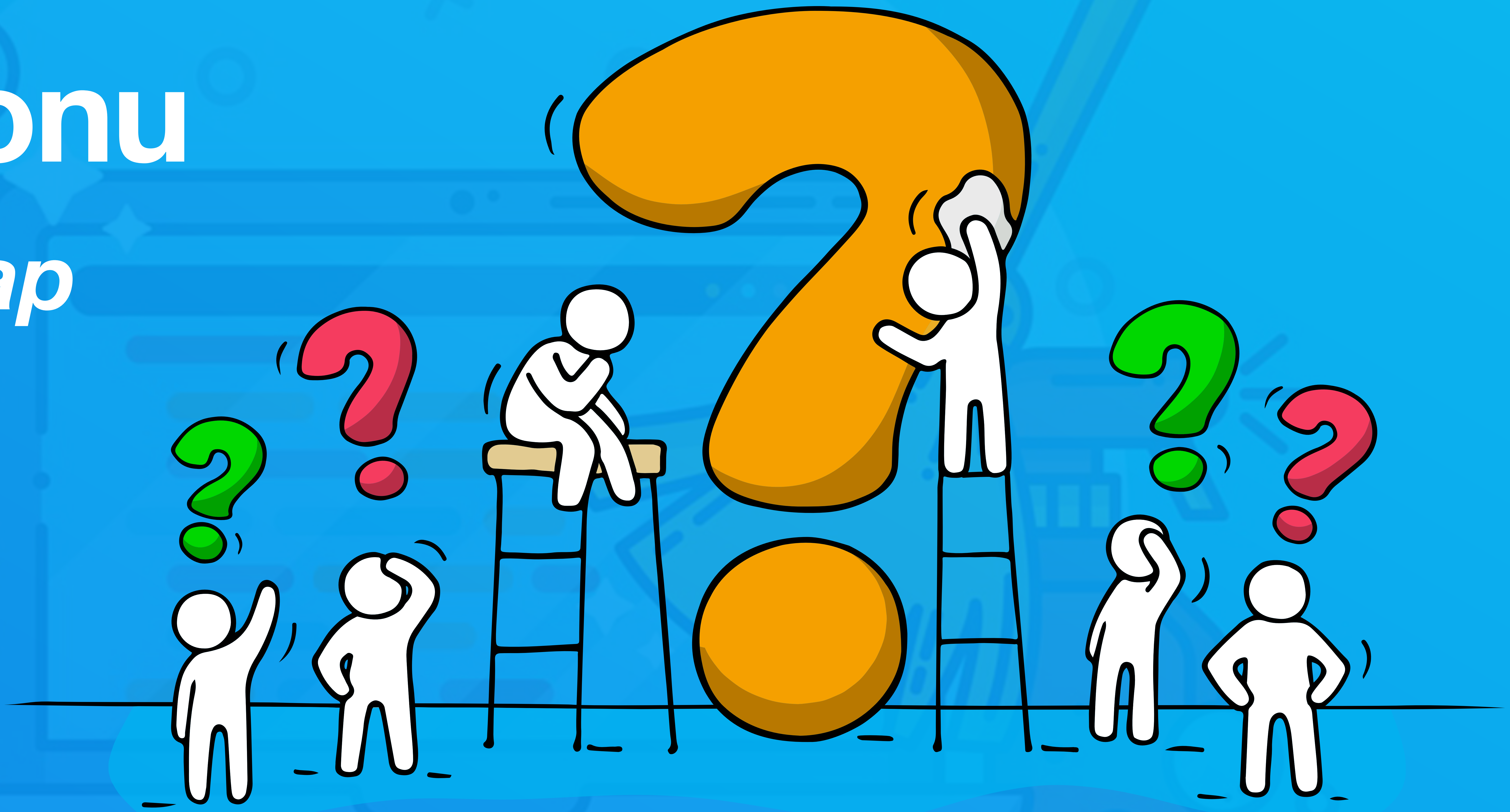
- **Unit Testing Principles, Practices, and Patterns**
  - Vladimir Khorikov, Manning, 2020
- **Test-Driven Java Development**
  - Viktor Farcic and Alex Garcia, Packt, 2015
- **Effective Unit Testing**
  - Lasse Koskela, Manning, 2013

Resources - III

- **The Art of Unit Testing: with examples in C#**
  - Roy Osherove, Manning, 2018
- **Modern C++ Programming with Test-Driven Development: Code Better, Sleep Better**
  - Jeff Langr, Pragmatic Bookshelf, 2013

Bölüm Sonu

Soru ve Cevap Zamanı!



selsoft

build better, deliver faster

info@selsoft.com.tr

selsoft.com.tr