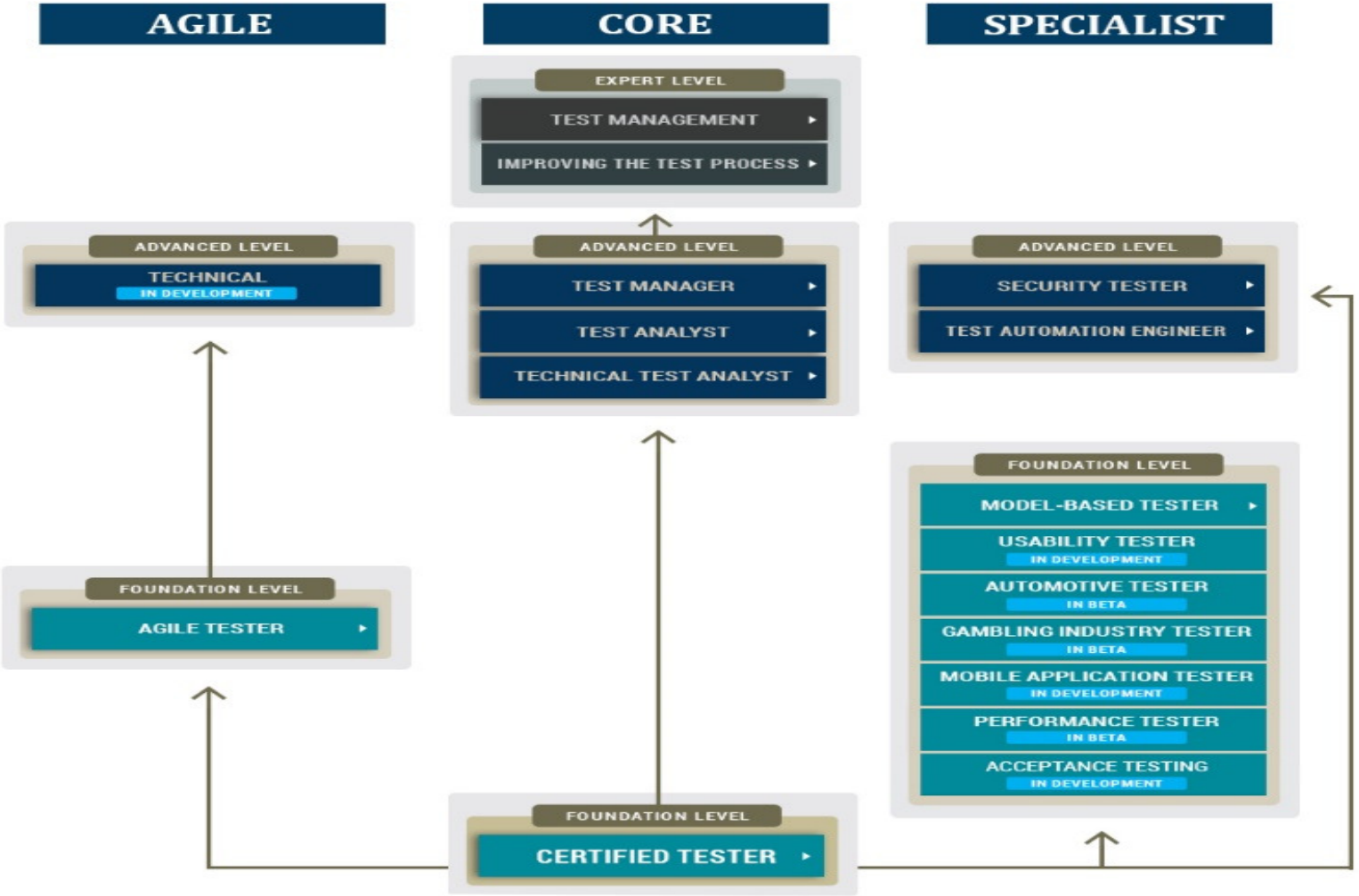
International Software Testing Quality Board (ISTQB)

International Software Testing Qualifications Board (ISTQB) (www.istqb.org)

- ISTQB was set up in 2001 to offer a similar certification scheme to as many countries
- The UK was a founding member of ISTQB
- In 2005, adopted the ISTQB Foundation Certificate syllabus as the basis of examinations for the Foundation Certificate in the UK.
- The Foundation Certificate is now an entry qualification for the ISTQB Advanced Certificate.
- The Certified Tester Foundation Level (CTFL) syllabus has been updated and released in 2018 version

ISTQB
International Software
Testing Qualifications Board



Turkish Testing Board (TTB)

Board Name: Turkish Testing Board
Acronym: TTB

Country/Region: Turkey
Founding Year: 2006

Board's Official website: www.turkishtestingboard.org (http://www.turkishtestingboard.org/)

Languages Supported: Turkish and English

List of Available ISTQB Certifications:

Foundation Level

Agile Tester Foundation

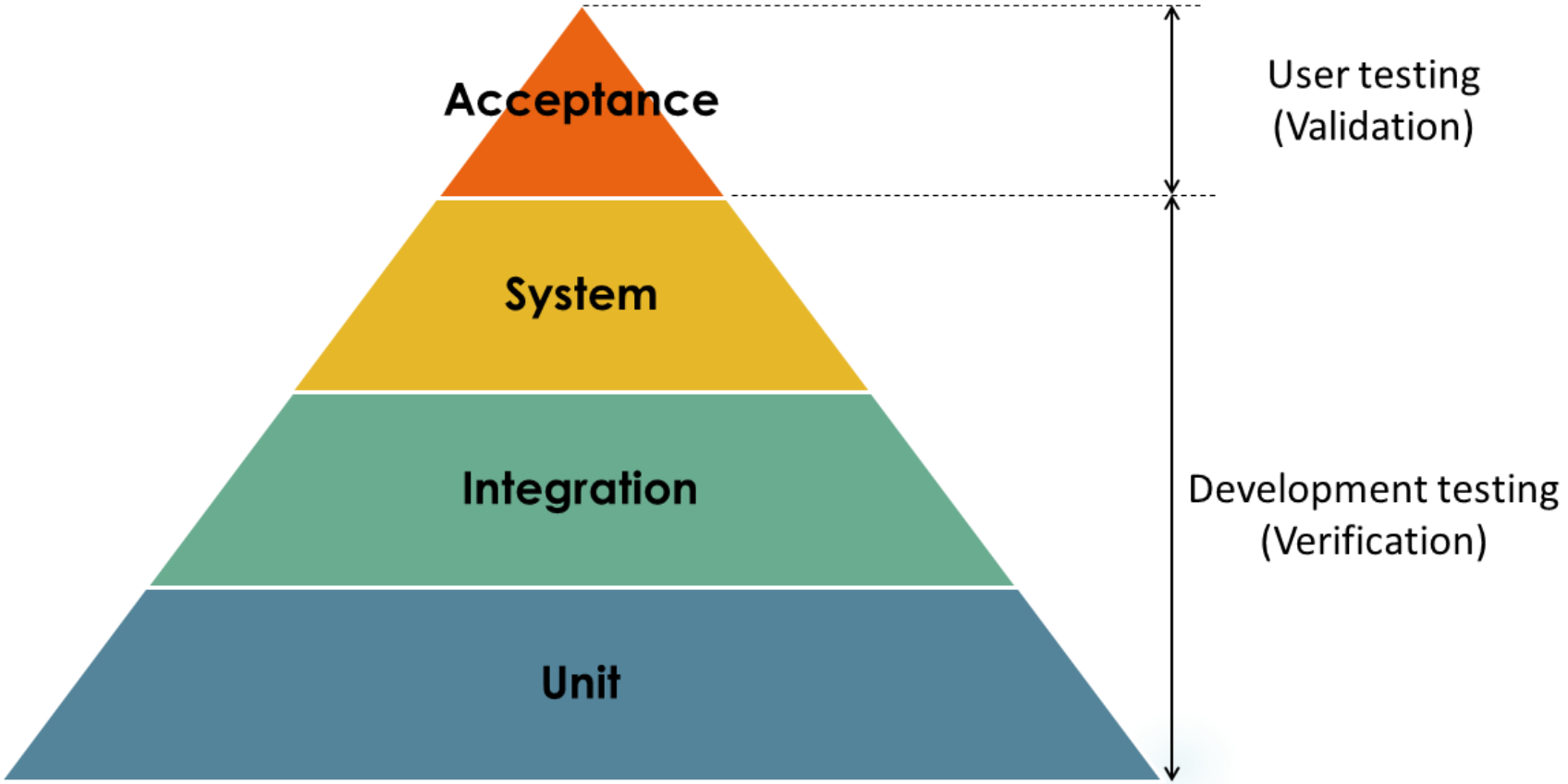
Advanced Level

- Advanced Test Analyst
- Advanced Test Manager
- Advanced Technical Test Analyst

ISTQB–BCS CTFL Examination

- Level of understanding K1 is associated with recall, so that a topic labelled K1 contains information that a candidate should be able to remember but not necessarily use or explain.
- Level of understanding K2 is associated with the ability to explain a topic or to classify information or make comparisons.
- Level of understanding K3 is associated with the ability to apply a topic in a practical setting.
- Level of understanding K4 is associated with the ability to analyze a situation or a set of information to determine what action to take

Verification &Validation For Quality



Testing levels and its relationship with V&V

Program Example

```
class numZero {
public static int   numZero (int[] x)
 {
//  if x == null throw NullPointerException
// else return the number of occurrences of 0 in x
      int   count = 0;
       for (int i = 1;   i < x.length;   i++)
            if (x[i] == 0)
              count++;
return        count;    }   }
```

The **Fault** in the Example

- The fault in this program is that it starts looking for zeroes at index 1 instead of index 0, as is necessary for arrays

First Input: *numZero ([2, 7, 0])* **correctly** evaluates to 1,

Second Input: *numZero ([0, 7, 2])* **incorrectly** evaluates to 0.

- In both of these cases the *fault* is executed.
- Both of these cases result in an error,
- Only the second case results in failure.

Error States in the Example

To understand the **error states,** we need to identify the state for the program.

- **The state** for *numZero* consists of values for *the variables x, count , i* and the *program counter (PC).*
- **The state** at the if statement on the first iteration of the loop is

(x = [2, 7, 0], count = 0, i = 1, PC = if)

- This **state** is **in error,** because the value of **i** should be **0** on the first iteration.

Error States in the Example (for the first input)

However

- The value of count is correct
- The error state does not affect the output,
- The software does not fail.

Finally:

- A state is in error if it is not the expected state, even if all of the values in the state are acceptable.

Error States in the Example (for the first input)

However

- The value of count is correct
- The error state does not affect the output,
- The software does not fail.

Finally:

- A state is in error if it is not the expected state, even if all of the values in the state are acceptable.

Error States in the Example (for the second input)

For the second input:

- **The state** for *numZero* consists of values for *the variables x, count , i* and the *program counter (PC).*
- The **state** is

(x = [0, 7, 2], count = 0, i = 1, PC = if)

- The error affects the variable count
- The error is present in the return value of the method.
- The **failure** results.

Cause-and-effect relationship

- Faults can result in errors.
- Errors can lead to system failures
- Errors are the effect of faults.
- Failures are the effect of errors
- Possible crush of the operating system is a failure

Faulty and Failure

- If a document with an error in it is used to specify a component, the component will be faulty and will probably exhibit incorrect behavior.
- If this faulty component is built into a system the system may fail.
- Failure is not always guaranteed

Distinguish Testing from Debugging

- The definitions of fault and failure allow us to distinguish testing from debugging.

- Big difference is that debugging is conducted by a programmer and the programmer fix the errors during debugging phase.

- Tester never fixes the errors, but rather find them and return to programmer.

Testing and Debugging

Testing and debugging are different kinds of activity

- Debugging is the process that developers go through to identify the cause of bugs or defects in code and undertake corrections.
- Testing is a systematic exploration of a component or system with the main aim of finding and reporting defects.

Testing versus Debugging

- Testing activity is carried down by a team of testers, in order to find the defect in the software.
- Test engineers run their tests on the piece of software and if they encounter any defect
  - This means that actual results don't match expected results
  - Testers report them to the development team
  - Testers also have to report at what point the defect occurred and what happened due the occurrence of that defect.
- All this information will be used by development team to DEBUG the defect.

Testing versus Debugging

TESTING	DEBUGGING
a) Finding and locating of a defect	a) Fixing that defect
b) Done by Testing Team	b) Done by Development team
c) Intention behind is to find as many defect as possible	c) Intention is to remove those defects

Exhaustive Testing of Complex Systems

- Exhaustive testing of complex systems is not possible
- The launch of the smartphone app occurred at the same time as a new phone hardware platform
  - The new app was only available on the new hardware for what many would recognize as the market leader at that time

Exhaustive Testing of Complex Systems

A Case Study:

- The product launch received extensive world-wide coverage with the mapping app.
- In a matter of hours, there were many people wanting to *see their location* in the new app and *see how this compared* with previous ones
- Each phone user was an expert in his/her location and '*test cases*' were generated showing that problems existed.

Exhaustive Testing of Complex Systems

- *If every possible test had been run*, problems would have been detected and rectified prior to the product launch.
- *If every test had been run*, the testing would still be running now, and the product launch would never have taken place

This illustrates one of the general principles of software testing

- With large and complex systems it will **never** be **possible** to **test** everything ***exhaustively***

Exhaustive Testing of Complex Systems

- For the mapping app example, it would be unhelpful to say that not enough testing was done
- The problem was that the right sort of testing was not done because the problems had not been detected

Testing and Risk

- Risk is inherent in all software development.
- The system may not work or the project may not be completed on time
- These uncertainties become more significant as the *system complexity* increase.

Testing and Risk

- We would expect to test an automatic flight control system more than we would test a video game system.
  - Why?

    Because
  - The risk is greater.
- There is greater probability of failure in the more complex system
  - What we test and how much we test ?

    These are related in some way to the risk.
- Greater risk implies more and better testing

A380 Aircraft Problem

- The A380 aircraft has approximately 530 km of cables, 100,000 wires and 40,300 connectors.
- Software is used both to design the aircraft and in the manufacture.
- The large subparts were made in two different countries, with different versions of the software in each manufacturing base.

A380 Aircraft Problem

- When Airbus was bringing together two halves of the aircraft, the different software meant that the wiring on one did not match the wiring in the other.
  - The cables could not meet up without being changed.
- Testing did not test something as straightforward as the versions of the design software
- Each subpart was built according to its own version of the CATIA (Computer Aided Three-Dimensional Interactive Application) software.

The result did not give an aircraft that could fly

Debugging:
Needed to Achieve a Quality Result

- Debugging is essential before testing begins to raise the level of quality of the component or system
- Debugging does not give confidence that the component or system meets its requirements completely.

Testing:
Needed to Achieve a Quality Result

- Testing examines of the behavior of a component or system
- Testing also reports all defects found for the development team
- Testing repeats enough tests to ensure that defect corrections have been effective.

Observations about Testing

Bertrand Myer from Eiffel Software and ETH Zürih

«*Testing is the process of executing a program with the intention of finding errors*»

In other words

«*Testing is about producing failures*»

Edsger Dijkstra

«*Testing can show the presence of bugs but never their absence*»

Testing Goals Based on Test Process Maturity
Bezier Testing Levels

Level 0 There is no difference between testing and debugging

Level 1 The purpose of testing is to show the correctness (software works).

Level 2 The purpose of testing is to show that the software doesn't work.

Level 3 The purpose of testing is not to prove anything specific, but to reduce the risk of using the software.

Level 4 Testing is a mental discipline that helps all IT professionals develop **higher quality** software.

Bezier Testing Levels

- Level 0 is the view that testing is the same as debugging.
- In Level 1 testing, the purpose is to **show correctness**
- In Level 2 testing, the purpose is to **show failures**
- Level 3 testing shows the **presence** , not absence or failure.

If we use software, we expose some risk.

The risk may be **small** and **unimportant,** or the **risk** may be **great** and the consequences **catastrophic,** but **risk is always there.**

In level 3 testing, both testers and developers work ***together*** to reduce risk.

Bezier Testing Levels

- Once the testers and developers are **on the same "team,"** an organization can progress to real Level 4 testing.

Level 4 testing means that the purpose of testing is to improve the ability of the developers to produce high quality software.