

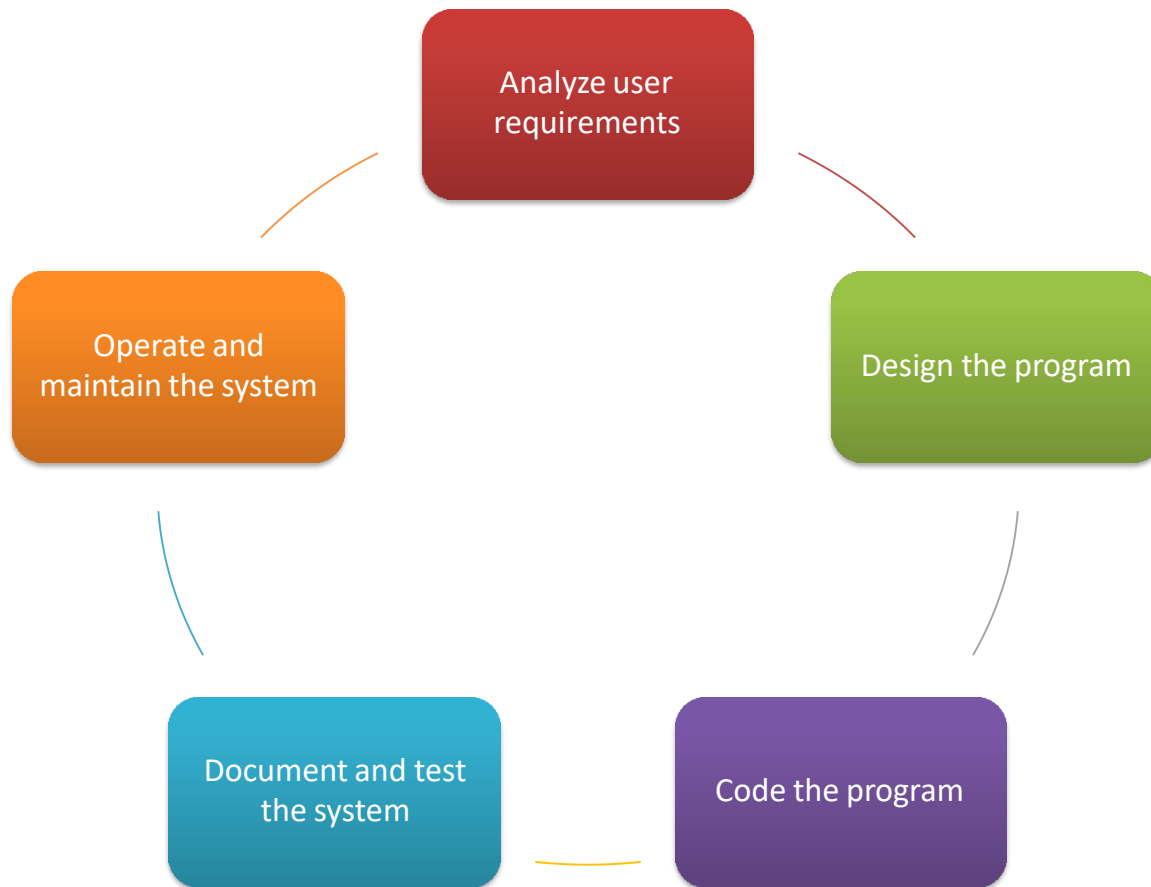


Software Development Approaches. Testing & Logging

Testing

Software Development Life Cycle

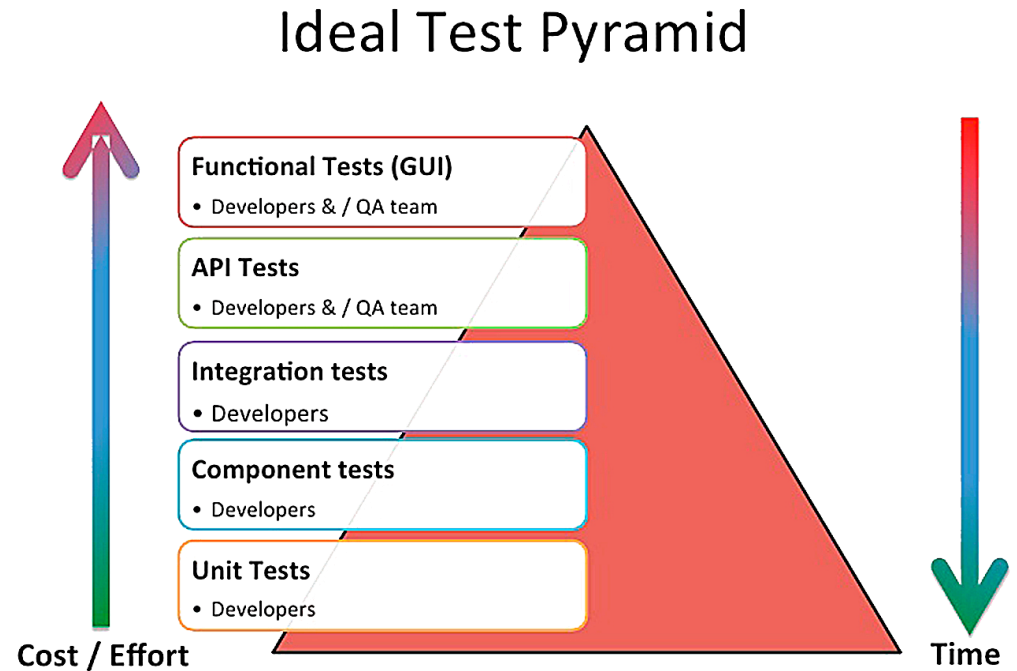
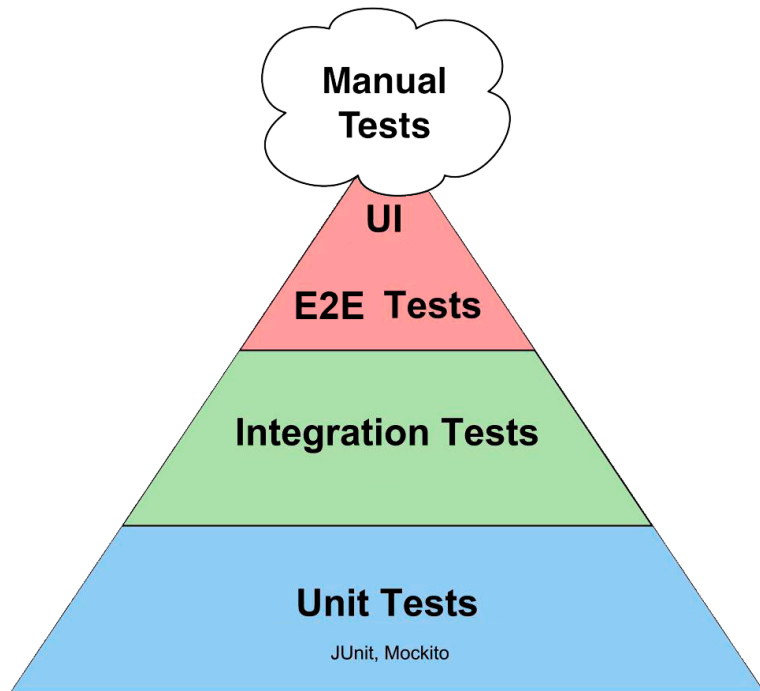
2



- **Software testing** is the process of program execution in order to find bugs.
- **Software testing** is the process used to measure the quality of developed computer software. Usually, quality is constrained to such topics as:
 - correctness, completeness, security;
- but can also include more technical requirements such as:
 - capability, reliability, efficiency, portability, maintainability, compatibility, usability, etc.

Testing Types.

4



Unit Testing is the first level of Software Testing

- **Unit testing** is a procedure used to validate that individual units of source code are working properly.
- The goal of unit testing is to isolate each part of the program and show that the individual parts are correct.
- Any method that can break is a good candidate for having a unit test.

```
<dependency>  
  <groupId>org.junit.jupiter</groupId>  
  <artifactId>junit-jupiter-api</artifactId>  
  <version>5.1.0</version>  
  <scope>test</scope>  
</dependency>
```

FEATURE	JUNIT 4	JUNIT 5
Declare a test method	@Test	@Test
Execute before all test methods in the current class	@BeforeClass	@BeforeAll
Execute after all test methods in the current class	@AfterClass	@AfterAll
Execute before each test method	@Before	@BeforeEach
Execute after each test method	@After	@AfterEach
Disable a test method / class	@Ignore	@Disabled
Test factory for dynamic tests	-	@TestFactory
Nested tests	-	@Nested
Tagging and filtering	@Category	@Tag
Register custom extensions	-	@ExtendWith
Repeated Tests	-	@RepeatedTest

JUnit 4	JUnit 5
fail	fail
assertTrue	assertTrue
assertThat	-
assertSame	assertSame
assertNull	assertNull
assertNotSame	assertNotSame
assertNotEquals	assertNotEquals
assertNotNull	assertNotNull
assertFalse	assertFalse
assertEquals	assertEquals
assertArrayEquals	assertArrayEquals
-	assertAll
-	assertThrows

assertArrayEquals()

The method will test whether two arrays are equal to each other. In other words, if the two arrays contain the same number of elements, and if all the elements in the array are equal to each other.

assertEquals()

The method compares two objects for equality, using their equals() method.

assertTrue() + assertFalse()

The methods tests a single variable to see if its value is either true, or false.

assertNull() + assertNotNull()

The methods test a single variable to see if it is null or not null.

assertSame() and assertNotSame()

The methods tests if two object references point to the same object or not.

JUnit 4	JUnit 5
<code>assumeFalse</code>	<code>assumeFalse</code>
<code>assumeNoException</code>	
<code>assumeNotNull</code>	
<code>assumeThat</code>	<code>assumeThat</code>
<code>assumeTrue</code>	<code>assumeTrue</code>

`assumeTrue` validates the given boolean assumption and abort the test if it's not true.

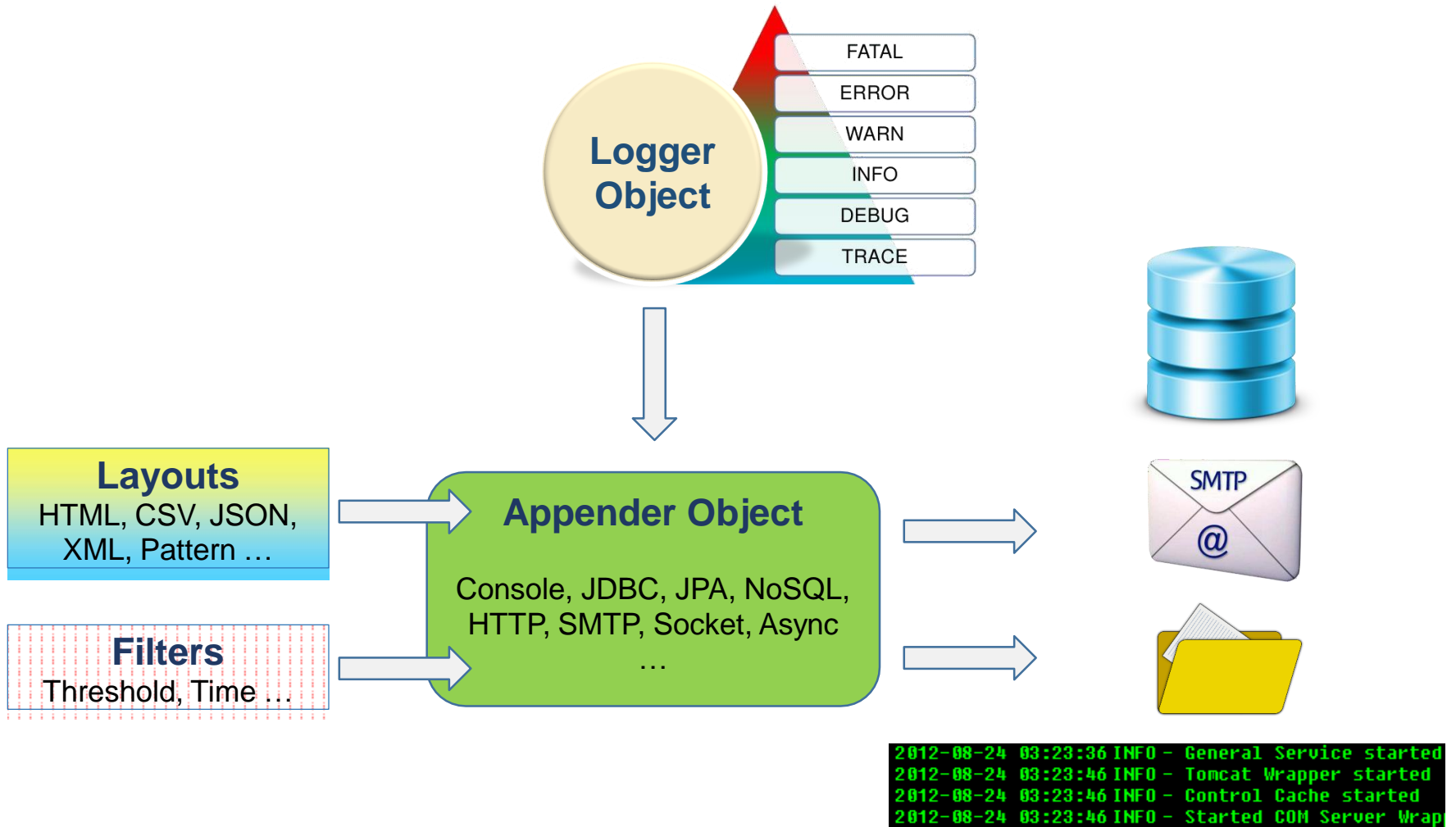
`assumingThat` executes the given Executable only if the assumption is valid, otherwise it does nothing.

Some Logging Frameworks

- ❖ Java Logging Framework (`java.util.logging`, part of the JDK)
- ❖ Apache Log4J
- ❖ LogBack

Some Logging Facades

- ❖ SLF4J (for LogBack, but has adapters for other frameworks)
- ❖ Apache Commons Logging (July 2014)



















































Log4j Levels

14

ALL	Turn on all logging
TRACE	Most detailed information
DEBUG	Detailed information on the flow through the system
INFO	Interesting runtime events
WARN	Potentially harmful situations
ERROR	Other runtime errors or unexpected conditions
FATAL	Severe errors that cause premature termination
OFF	Turn off all logging

Log4j Levels

15

	TRACE	DEBUG	INFO	WARN	ERROR	FATAL
ALL						
TRACE						
DEBUG						
INFO						
WARN						
ERROR						
FATAL						
OFF						

Appender - Console Usage)

20

```
import org.apache.logging.log4j.*;

public class Application {
    private static Logger logger1 = LogManager.getLogger(Application.class);

    public static void main(String[] args) {
        logger1.trace("This is a trace message");
        logger1.debug("This is a debug message");
        logger1.info("This is an info message");
        logger1.warn("This is a warn message");
        logger1.error("This is an error message");
        logger1.fatal("This is a fatal message");
    }
}
```

```
TRACE 2018-03-25 19:08:46.426 Application:25 - This is a trace message
DEBUG 2018-03-25 19:08:46.426 Application:26 - This is a debug message
INFO 2018-03-25 19:08:46.426 Application:27 - This is an info message
WARN 2018-03-25 19:08:46.426 Application:28 - This is a warn message
ERROR 2018-03-25 19:08:46.426 Application:29 - This is an error message
FATAL 2018-03-25 19:08:46.427 Application:30 - This is a fatal message
```


Annotations

since Java 5

Data about program that is not part of the program.

official Oracle java tutorial

Built-in Annotations in Java

```
@SuppressWarnings(value="unchecked")  
public void method() {...}
```

```
@Override  
public void method() {...}
```

```
@Deprecated  
public int oldMethod() {...}
```

```
@SafeVarargs
```

```
@FunctionalInterface
```

```
@Author(name="ObiVanKenobi" date="7/9/3147" )  
public class Droid() {...}
```

Annotations - Hibernate

```
@Entity
@Table(name = "person")
public class Person {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "person_id", nullable = false)
    private Long id;

    @Column(name = "surname", nullable = false, length = 25)
    private String surname;

    @Column(name = "name", nullable = false, length = 25)
    private String name;

    @Column(name = "email", nullable = true, length = 45)
    private String email;

    @ManyToOne
    @JoinColumn(name="city_id", referencedColumnName="city_id")
    private City city;
}
```

Java Annotation Purposes

- ❖ **Information for the compiler** – Annotations can be used by the compiler to detect errors or suppress warnings.
- ❖ **Compile-time and deployment-time processing** – Software tools can process annotation information to generate code, XML files, and so forth.
- ❖ **Runtime processing** – We can define annotations to be available at runtime which we can access using **java reflection** and can be used to give instructions to the program at runtime.

Categories of Annotations

❖ Marker Annotations:

```
@Override  
@TestAnnotation()
```

❖ Single value Annotations:

```
@TestAnnotation("testing")
```

❖ Full Annotations:

```
@TestAnnotation(owner="EPAM", value="Java")
```

Annotations - kinds

Retention policies

source



discarded during compilation

class



discarded during class load

runtime



available for reflection

Annotations - places

Class

Field

Method

Constructor

Parameter
Annotation

Local variable
Package

Annotations - places

Java SE 8

- ❖ Class instance creation expression

```
new @Interned MyObject();
```

- ❖ Type cast

```
myString = (@NonNull String) str;
```

- ❖ implements clause

```
class UnmodifiableList<T> implements  
    @ReadOnly List<@ReadOnly T> { ... }
```

- ❖ Thrown exception declaration

```
void monitorTemperature() throws  
    @Critical TemperatureException { ... }
```


Meta-Annotations

```
@Retention(RetentionPolicy.RUNTIME)
public @interface CustomAnnotation{
}
```

```
@Target(ElementType.METHOD)
public @interface CustomAnnotation{
}
```

```
@Inherited
public @interface CustomAnnotation{
}
```

```
@Documented
public @interface CustomAnnotation{
}
```

Meta-Annotations

@Retention

❖ **RetentionPolicy.SOURCE**

The marked annotation is retained only in the source level and is ignored by the compiler.

❖ **RetentionPolicy.CLASS**

The marked annotation is retained by the compiler at compile time, but is ignored by the Java Virtual Machine (JVM).

❖ **RetentionPolicy.RUNTIME**

The marked annotation is retained by the JVM so it can be used by the runtime environment.

Meta-Annotations

@Target

- ❖ **ElementType.ANNOTATION_TYPE**

can be applied to an annotation type.

- ❖ **ElementType.CONSTRUCTOR**

- ❖ **ElementType.FIELD**

- ❖ **ElementType.LOCAL_VARIABLE**

- ❖ **ElementType.METHOD**

- ❖ **ElementType.PACKAGE**

- ❖ **ElementType.PARAMETER**

can be applied to the parameters of a method.

- ❖ **ElementType.TYPE**

can be applied to any element of a class.

Annotations - custom

```
@Retention(RetentionPolicy.RUNTIME)
@Target(value={ElementType.METHOD})
public @interface InitializingMethod{

    int timesToCall() default 1;

}
```

declaration

```
@InitializingMethod(timesToCall = 7)
public void method myMethod{...}
```

usage

Annotations - custom

```
@Documented
@Target(ElementType.METHOD)
@Inherited
@Retention(RetentionPolicy.RUNTIME)
public @interface MyCustomAnnotation {
    int studentAge() default 18;
    String studentName();
    String stuAddress();
    String stuStream() default "CSE";
}
```

```
@MyCustomAnnotation(studentName="Chaitanya",
                     stuAddress="Agra, India")
public class MyClass { ... }
```

Annotations - custom

```
@Target({ElementType.FIELD, ElementType.METHOD,  
                                                ElementType.PACKAGE})  
@Retention(RetentionPolicy.RUNTIME)  
public @interface MyCustomAnnotation {  
    int studentAge() default 18;  
    String studentName();  
    String stuAddress();  
    String stuStream() default "CSE";  
}
```

Limit the elements in annotation

```
@Documented
@Target(ElementType.METHOD)
@Inherited
@Retention(RetentionPolicy.RUNTIME)
public @interface MyCustomAnnotation {
    int studentAge() default 18;
    String studentName();
    String stuAddress();
    String stuStream() default "CSE";
}
```

```
@MyCustomAnnotation(studentName="Chaitanya",
                     stuAddress="Agra, India")
public class MyClass { ... }
```

Annotations - custom

Note: We can also have array elements in an annotation.

```
@interface MyCustomAnnotation {  
    int count();  
    String[] books();  
}
```

```
@MyCustomAnnotation( count = 2, books = {"C++", "Java"} )  
public class MyClass { }
```


Annotations - custom

Elements of annotation must have one of types:

- ❖ **boolean, byte, short, int, long** (wrapper classes like Integer cannot be used)
- ❖ **String** type
- ❖ enum type
- ❖ class type
- ❖ another annotation type
- ❖ array of any of the above types

Repeating annotations

- ❖ *First, we need to declare a repeatable annotation:*

```
@Repeatable(Schedules.class)
public @interface Schedule {
    String time() default "morning";
}
```

- ❖ *Then, we define the containing annotation with a mandatory value element:*

```
public @interface Schedules {
    Schedule[] value();
}
```

- ❖ *Now, we can use @Schedule multiple times:*

```
@Schedule
@Schedule(time = "afternoon")
@Schedule(time = "night")
void scheduledMethod() {
    // ...
}
```

Not extend annotations

```
public @interface AnAnnotation extends OtherAnnotation
{
    // Compilation error
}
```

