6th UCAAT User Conference on Advanced Automated Testing

ETSI

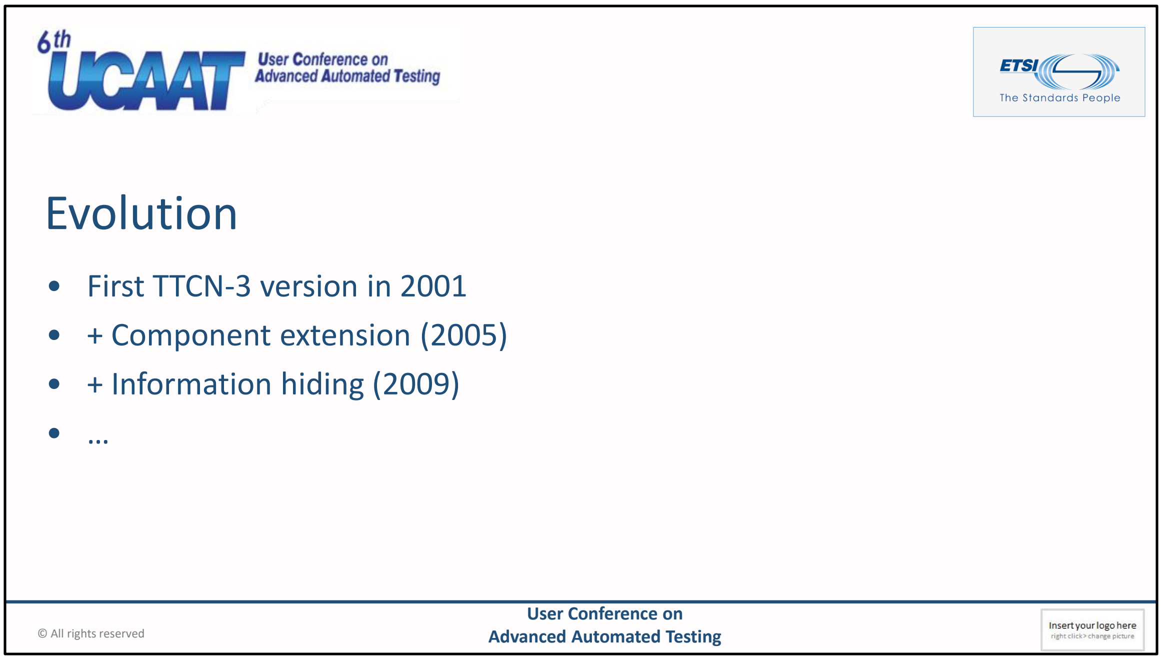
Paris, 16-18 October 2018
Organizer: TESTING SOLUTIONS & SERVICES

Object-oriented Features of TTCN-3

Presented by Kristóf Szabados (on behalf of STF 550)

Insert your logo here
right click> change picture

© All rights reserved


6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People

Evolution

- First TTCN-3 version in 2001
- + Component extension (2005)
- + Information hiding (2009)
- …

© All rights reserved

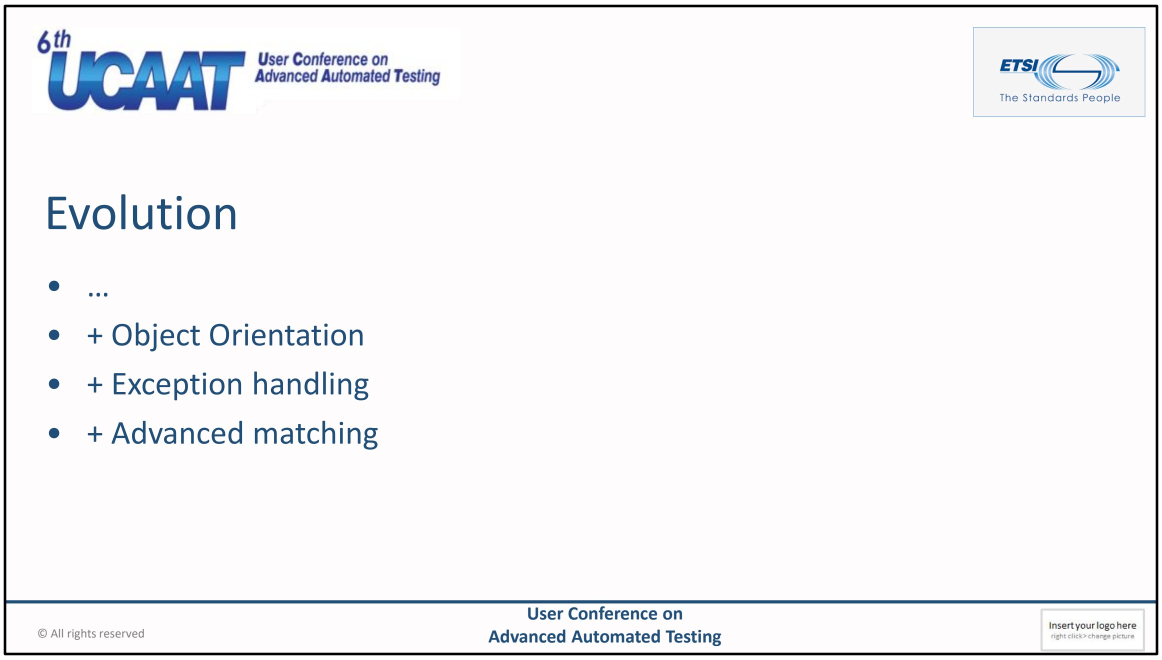
User Conference on Advanced Automated Testing

Insert your logo here right click> change picture

Support for more and more complex test systems is not a new challenge for TTCN-3.

It is quite interesting to look at TTCN-3 from a specific point of view, that of handling complexity.
The very first version of TTCN-3 was create to handle the new complexities of testing modern systems, for example when several test components need to run parallelly and in a concerted way to properly test.
Than came component extension …. Information hiding … etc

First version
http://www.etsi.org/deliver/etsi_es/201800_201899/20187301/01.00.10_50/es_20187301v010010m.pdf
3.1.1: https://www.etsi.org/deliver/etsi_es/201800_201899/20187301/03.01.01_60/es_20187301v030101p.pdf
4.5.1: http://www.etsi.org/deliver/etsi_es/201800_201899/20187301/04.05.01_60/es_20187301v040501p.pdf


6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People

Evolution

- …
- + Object Orientation
- + Exception handling
- + Advanced matching

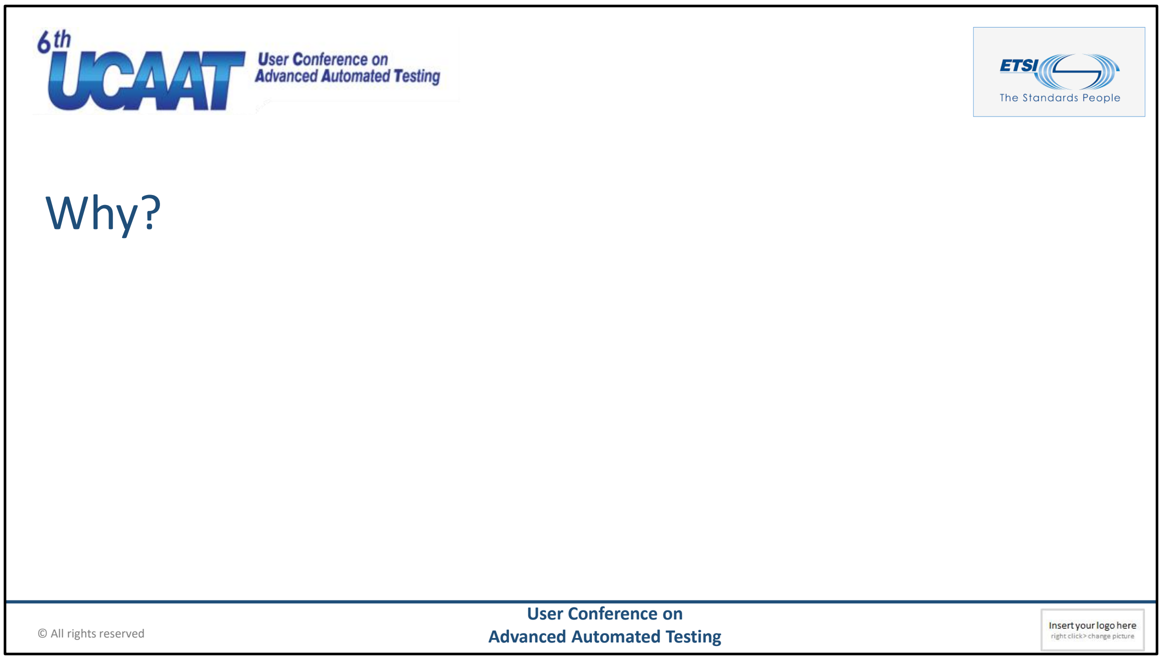
© All rights reserved

User Conference on
Advanced Automated Testing

Insert your logo here
right click > change picture

And some of the most important features of the new additions to and extensions of the language were also meant to further help with addressing complexity.

First version
http://www.etsi.org/deliver/etsi_es/201800_201899/20187301/01.00.10_50/es_20187301v010010m.pdf
3.1.1: https://www.etsi.org/deliver/etsi_es/201800_201899/20187301/03.01.01_60/es_20187301v030101p.pdf
4.5.1: http://www.etsi.org/deliver/etsi_es/201800_201899/20187301/04.05.01_60/es_20187301v040501p.pdf



6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People

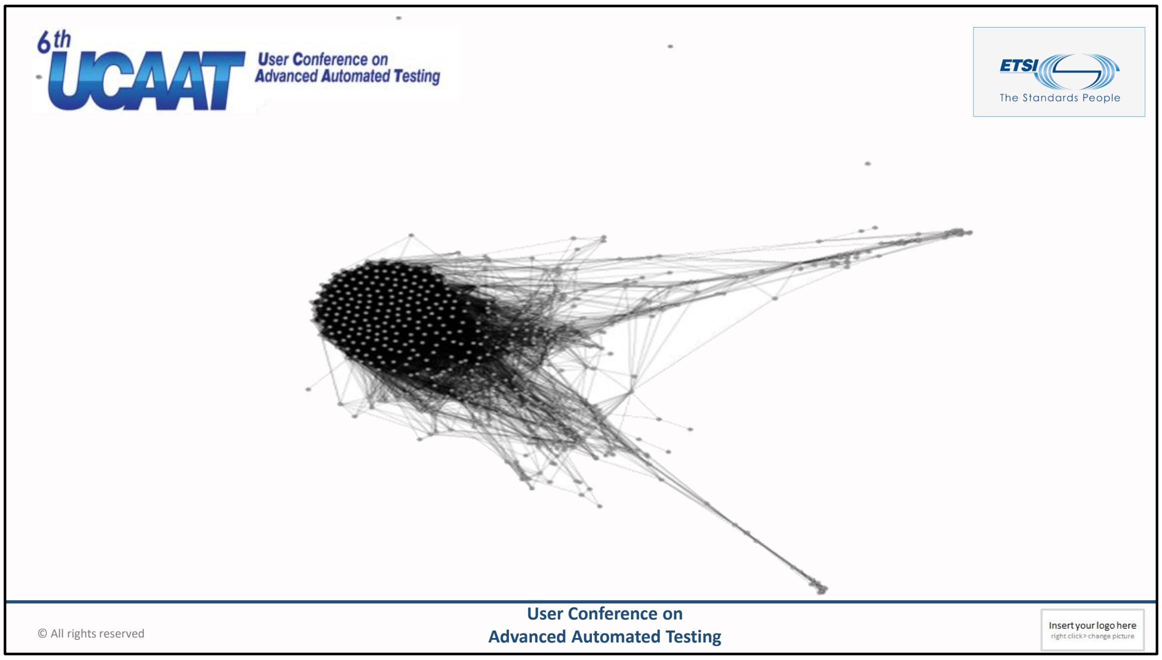
Why?

© All rights reserved

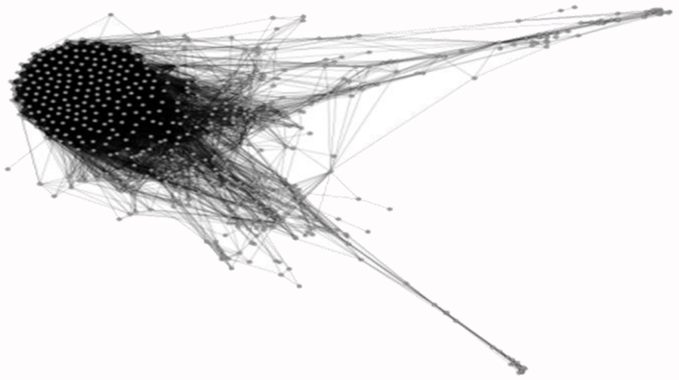
User Conference on Advanced Automated Testing

Insert your logo here
right click> change picture

Why did we need to extend TTCN-3 with Object-oriented features?


6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People



© All rights reserved

User Conference on
Advanced Automated Testing

Insert your logo here
right click> change picture

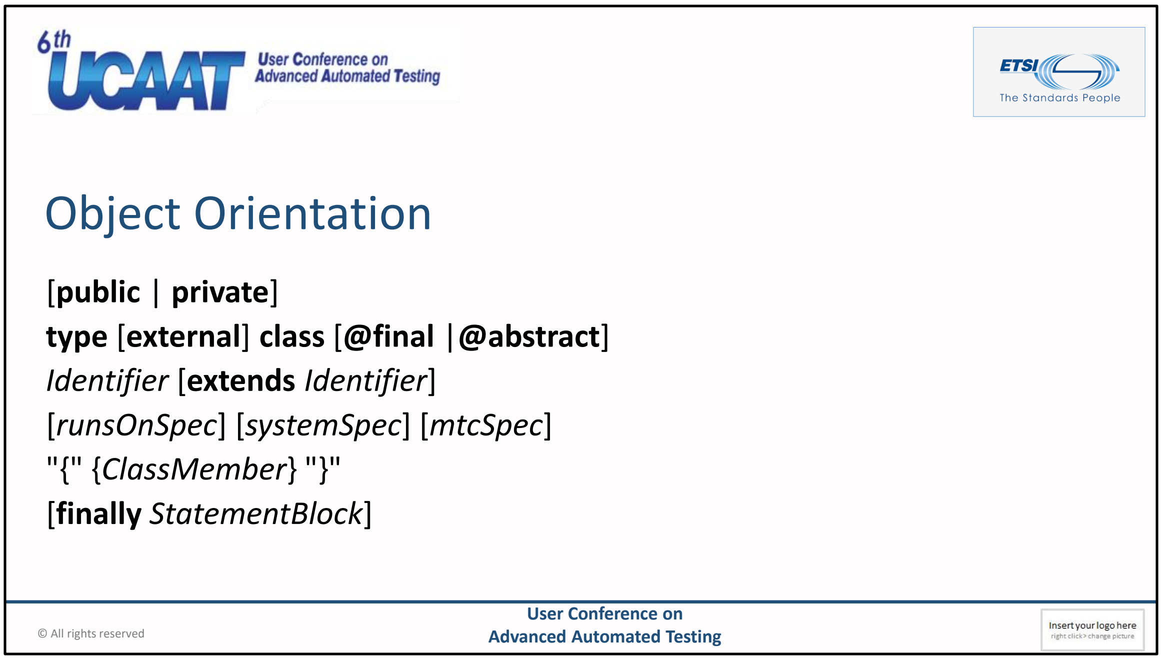
Test are getting bigger + more and more complex !

This is the architecture of a test system.
The points are TTCN-3 and ASN.1 modules.
The lines are import statements connecting the modules.

This network contains small world and scale-free properties.
If we select any 2 nodes, the shortest path between them will contain at most 5 other nodes.
The distribution of the connection follows the power law. Just like the Internet, the human brain and software systems.

At the scale of tests we have nowadays, maintainability and software quality become important questions.
(otherwise development will not be able to keep up)

Related publication:
K. Szabados, Structural Analysis of Large TTCN-3 Projects in proceedings of: Testing of Software and Communication Systems, 21st IFIP WG 6.1 International Conference, TESTCOM 2009 and 9th International Workshop, FATES 2009, Eindhoven, The Netherlands, November 2-4,
2009.

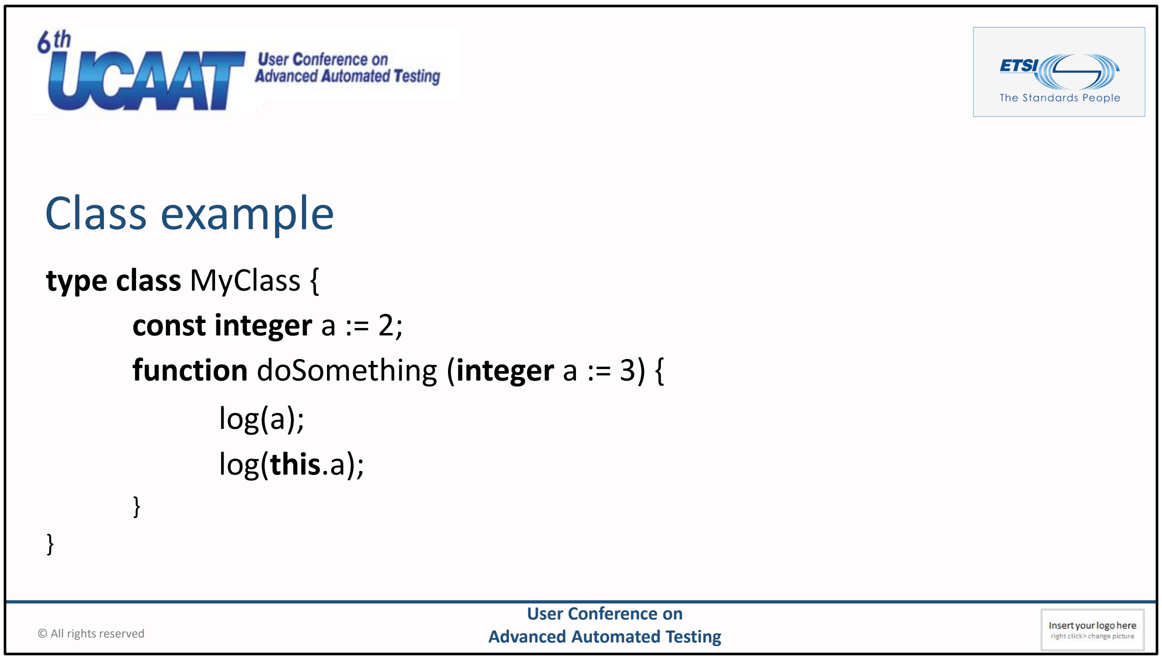


Object Orientation

[**public** | **private**]
type [**external**] **class** [**@final** |**@abstract**]
Identifier [**extends** *Identifier*]
[*runsOnSpec*] [*systemSpec*] [*mtcSpec*]
"{" {*ClassMember*} "}"
[**finally** *StatementBlock*]

© All rights reserved

A new language extension … with quite a lot to unpack here.
I will go through some of the features using examples.


6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People

Class example

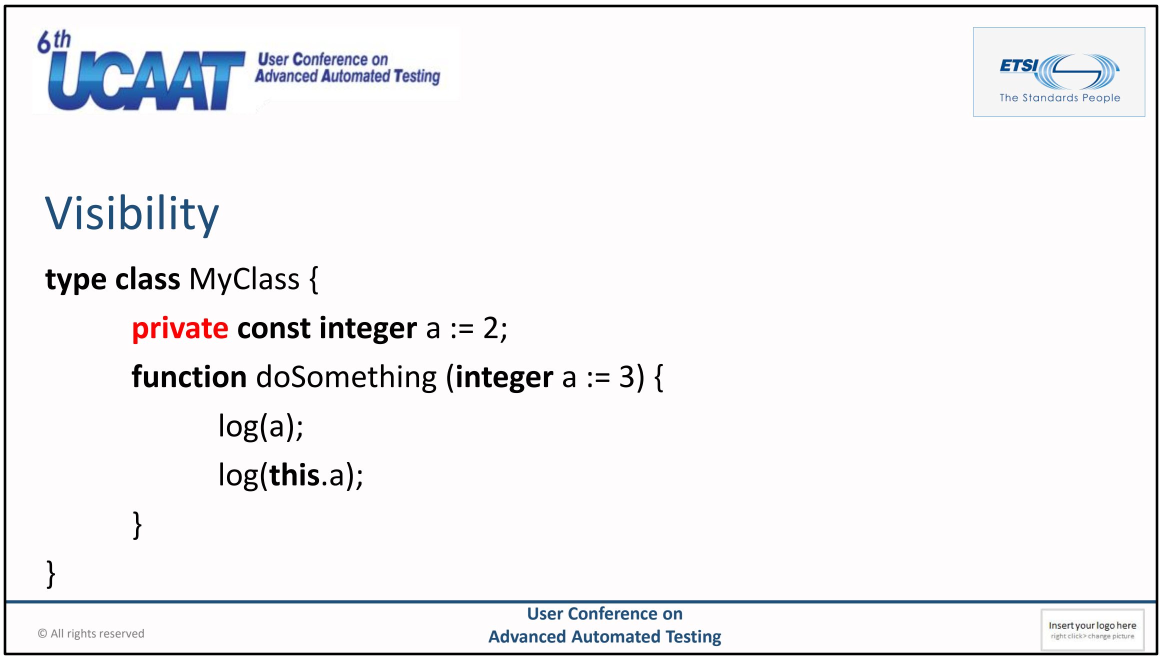
```
type class MyClass {
        const integer a := 2;
        function doSomething (integer a := 3) {
                log(a);
                log(this.a);
        }
}
```

Classes appear as type and as expected from other OO programming languages can contain both members and functions.
This way it will be possible to organize the source code in a way where data and the functions operating on it are kept together.

In the example:
First log logs 3, the second 2.

As present in the example this.a can be used to access a member that has the same name as a formal parameter.

Example code from the standard extension for a simple class.



Visibility

```
type class MyClass {
        private const integer a := 2;
        function doSomething (integer a := 3) {
                log(a);
                log(this.a);
        }
}
```

© All rights reserved

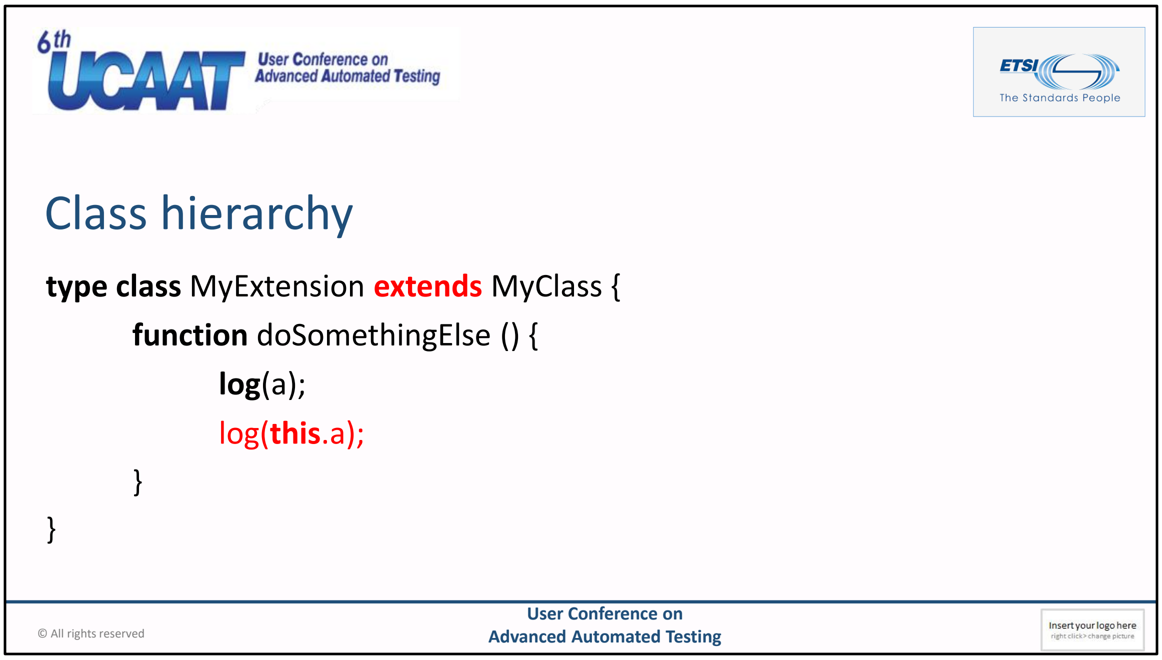
Insert your logo here
right click> change picture

Members can have visibility.
Private members can only be accessed directly by members of the class itself.

Providing support for fine-grained information hiding.

In the example:
First log logs 3, the second 2.

Example code from the standard extension for a simple class.


6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People

Class hierarchy

```
type class MyExtension extends MyClass {
        function doSomethingElse () {
                log(a);
                log(this.a);
        }
}
```

© All rights reserved

User Conference on Advanced Automated Testing

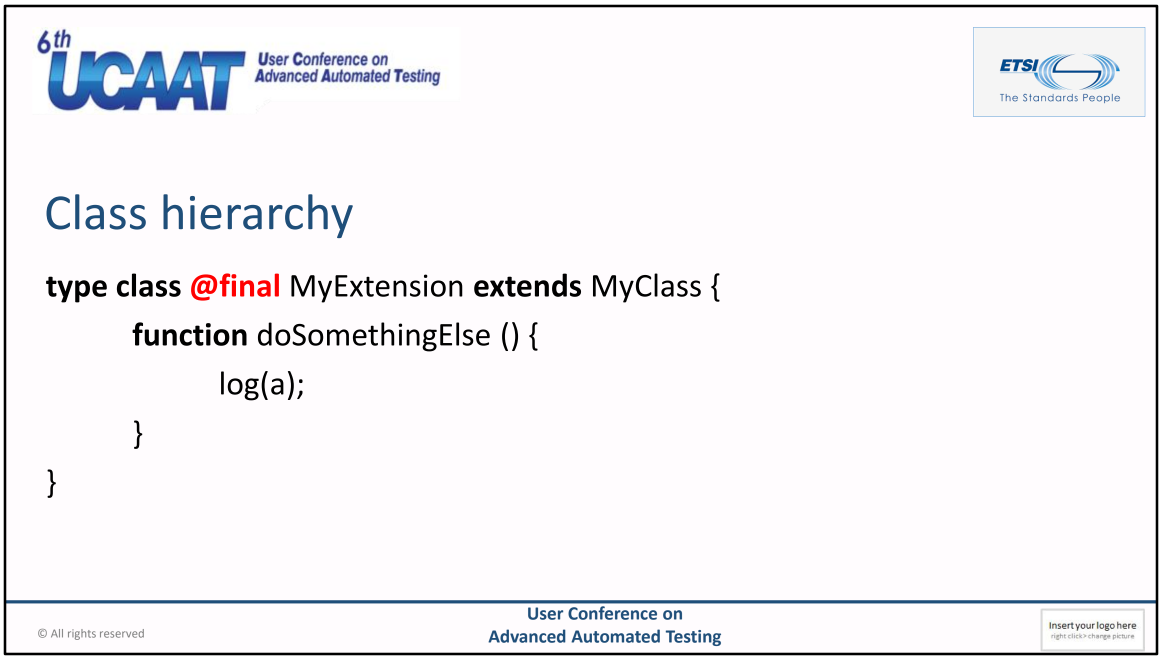
Insert your logo here
right click> change picture

Classes can form a single inheritance hierarchy. That is each class can have zero or one other class it extends. (zero extensions means extending the root class type object)
The extended class is called the superclass, the extending class is the subclass.

In the example: First log logs 3, the second log causes an error as it can not use the private member of it superclass.

Methods can be overridden, but not overloaded !

Example code from the standard extension for a simple class.


6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People

Class hierarchy

```
type class @final MyExtension extends MyClass {
        function doSomethingElse () {
                log(a);
        }
}
```

© All rights reserved

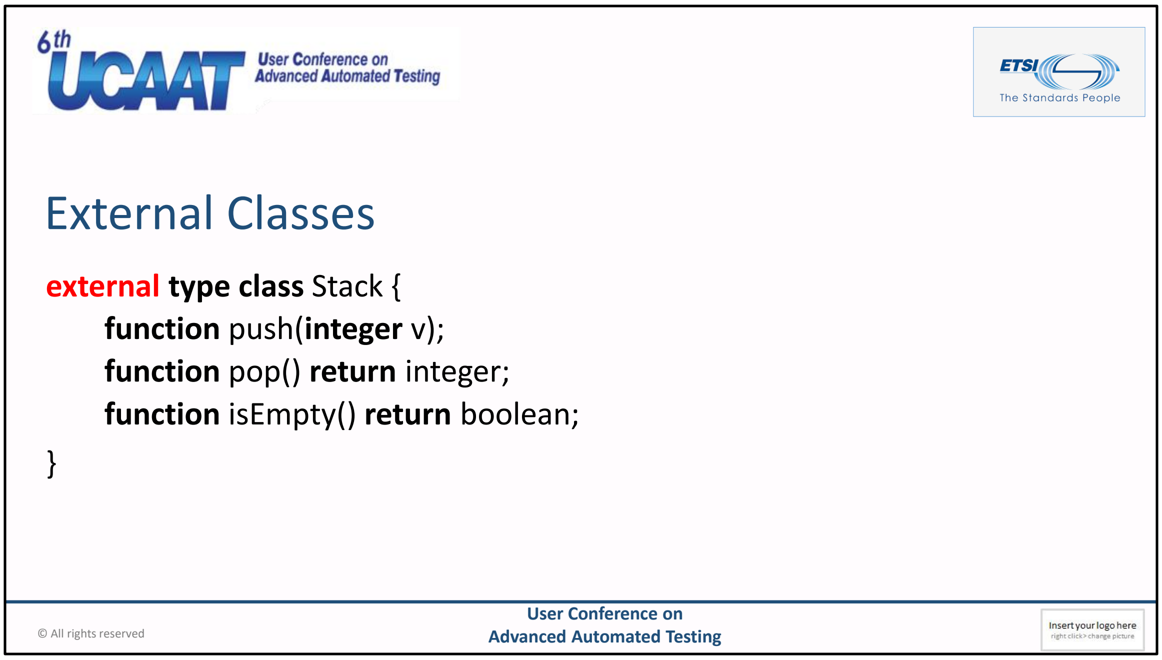
User Conference on
Advanced Automated Testing

Insert your logo here
right click> change picture

Final classes can not be subclassed.
This is good way, to protect one’s library from being extended.

In the example: The log logs 3 (for the default value of the parameter).

Example code from the standard extension for a simple class.



External Classes

```
external type class Stack {
    function push(integer v);
    function pop() return integer;
    function isEmpty() return boolean;
}
```

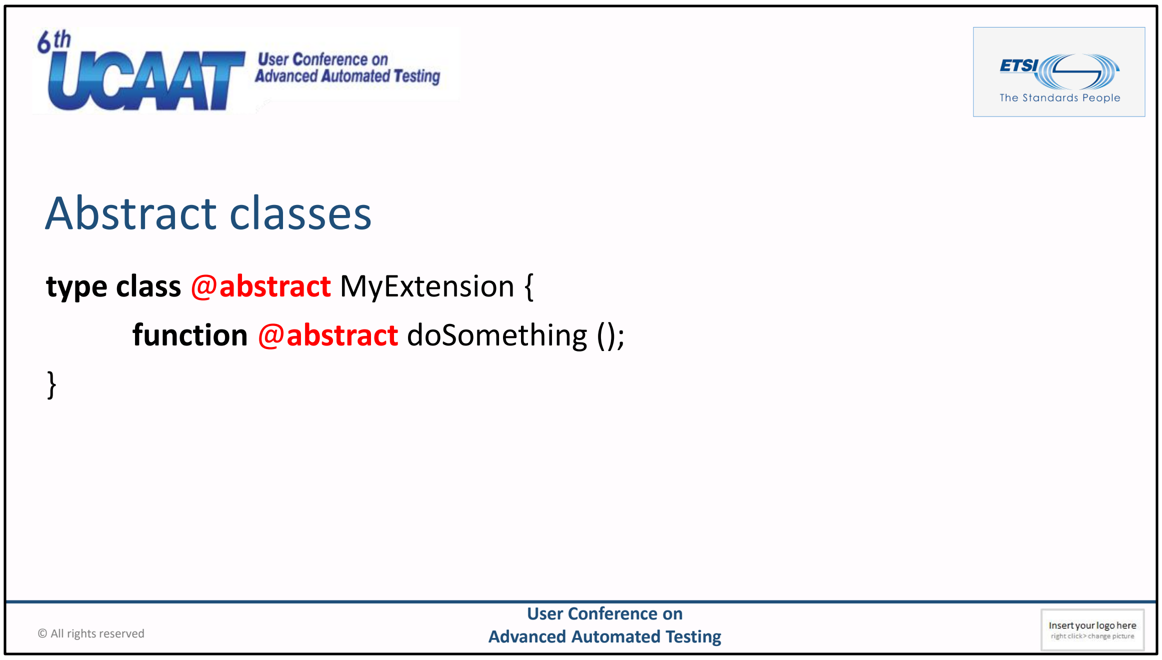
© All rights reserved

User Conference on
Advanced Automated Testing

External classes are a way to connect efficiently implemented external libraries, to TTCN-3 internal usage.
Method calls to objects of external classes are delegated via the platform adapter to the corresponding method of the external class.
And represented inside TTCN-3 with an object of this class, allowing the checking for the proper usage of it's member functions in TTCN-3 code.

This feature can enable the usage of efficient external collection within TTCN-3.
But please note, when using external feature, the usual testing related benefits of TTCN-3 are not available (for example safe termination is not guaranteed if the external library runs into a stack overflow error)

Example code from the standard extension for a simple class.


6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People

Abstract classes

```
type class @abstract MyExtension {
        function @abstract doSomething ();
}
```

© All rights reserved

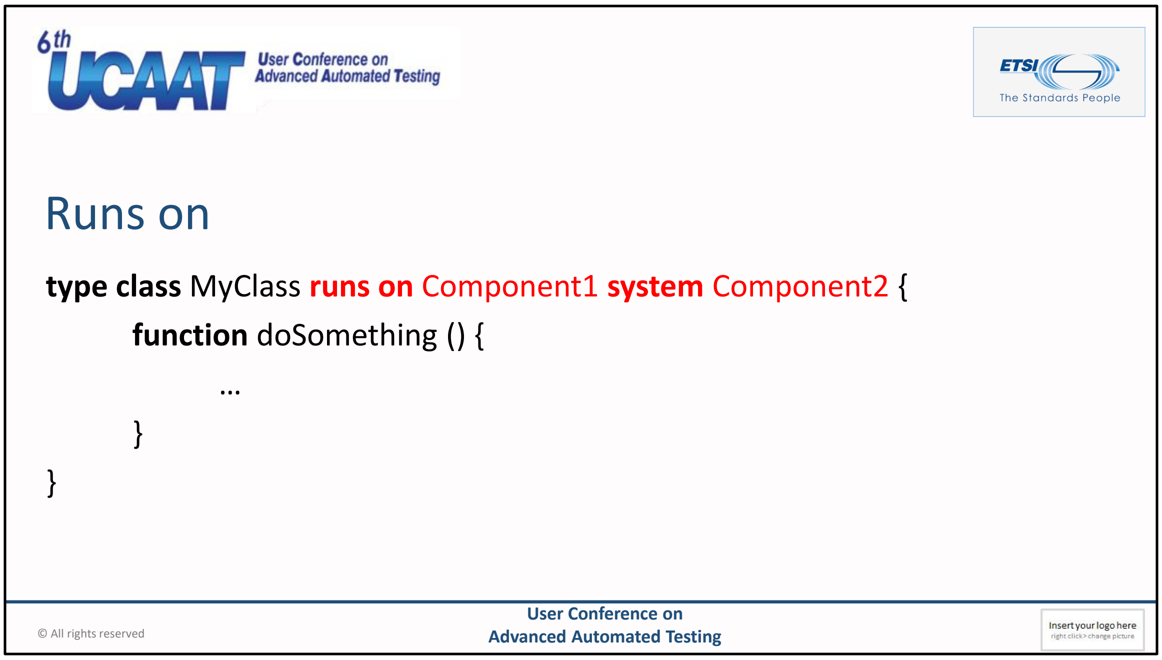
User Conference on
Advanced Automated Testing

Insert your logo here
right click> change picture

Abstract classes can contains abstract member functions.
Non-abstract classes are not allowed to contain abstract functions.
All abstract functions need to be defined by all non-abstract subclasses.

This allows library writers to write their functionalities on an abstract level, and the user to fill in the actual type in her own test code.
For example: the library only needs to know that the objects of this class can log themselves using a “logme” function … but it is up to the user of the library to provide the actual implementation of how her objects should log themselves.

Example code from the standard extension for a simple class.


6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People

Runs on

```
type class MyClass runs on Component1 system Component2 {
        function doSomething () {
                …
        }
}
```

© All rights reserved

User Conference on
Advanced Automated Testing

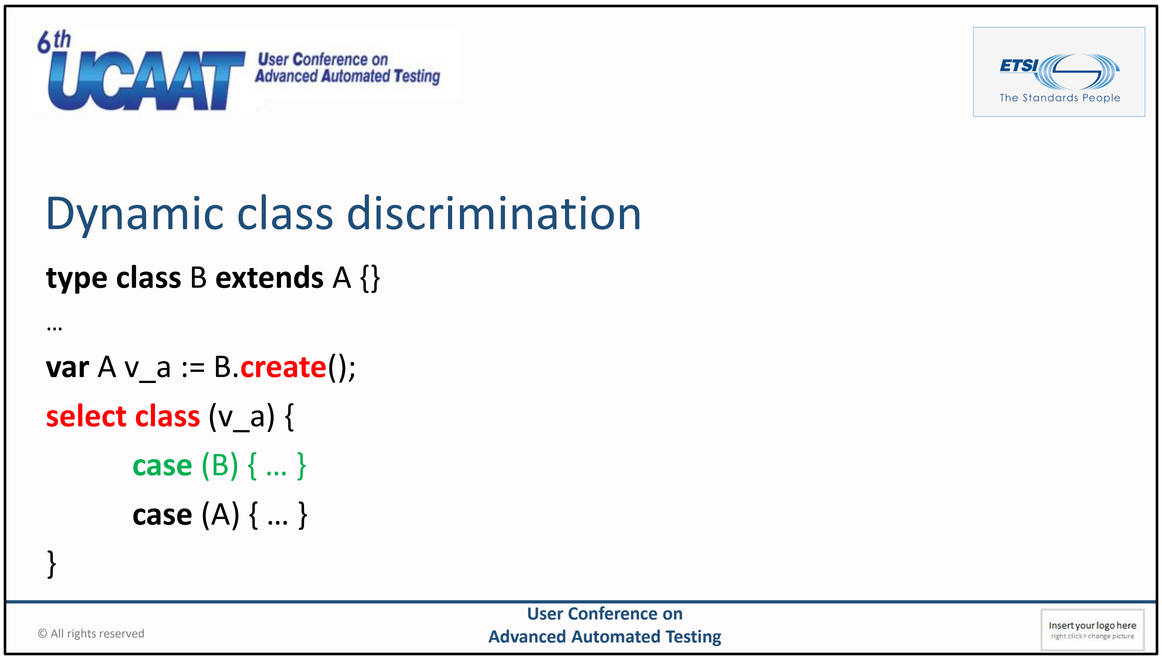
Insert your logo here
right click > change picture

Classes can have runs on, mtc and system clauses, which restrict the component context that is allowed to create objects of that class.

Member functions of a class with runs on, mtc or system clauses shall not have their own clauses, but inherit them from their class.
And objects of such classes can only be created in functions that run on the described components.

In the example: The member function doSomething inherits the runs on, mtc and system clauses of the class.

Example code from the standard extension for a simple class.



Dynamic class discrimination

```
type class B extends A {}
…
var A v_a := B.create();
select class (v_a) {
        case (B) { … }
        case (A) { … }
}
```

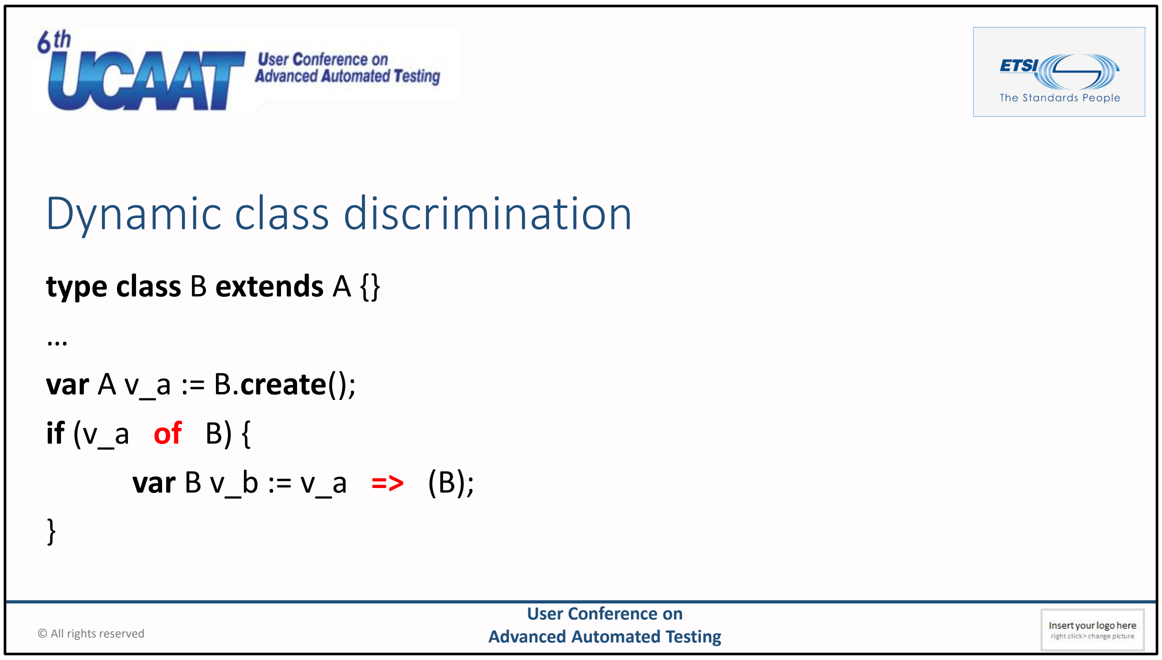
© All rights reserved

Insert your logo here
right click > change picture

An object of a class can be created with the .create() function, which will invoke the constructor of the class (having the name "create")
It is possible to branch the execution of the code based on the dynamic class of the object.

In the example: case (B) will be selected as v_a has B as dynamic type (and A for static type)

Example code from the standard extension for a simple class.



6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People

Dynamic class discrimination

```
type class B extends A {}
…
var A v_a := B.create();
if (v_a   of   B) {
        var B v_b := v_a   =>   (B);
}
```

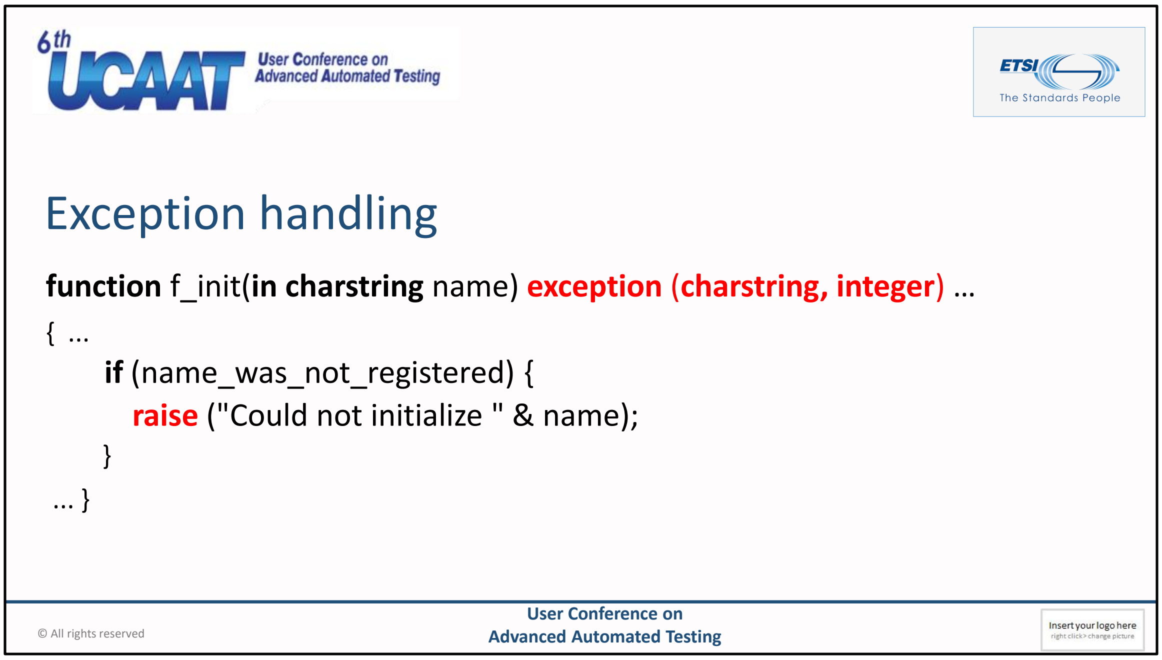
© All rights reserved

Insert your logo here
right click > change picture

Dynamic class discrimination
+ casting

In the example: v_a will be cast to type B before assigned to v_b

Example code from the standard extension for a simple class.



6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People

Exception handling

```
function f_init(in charstring name) exception (charstring, integer) …
{ …
      if (name_was_not_registered) {
         raise ("Could not initialize " & name);
      }
 … }
```

© All rights reserved

User Conference on Advanced Automated Testing

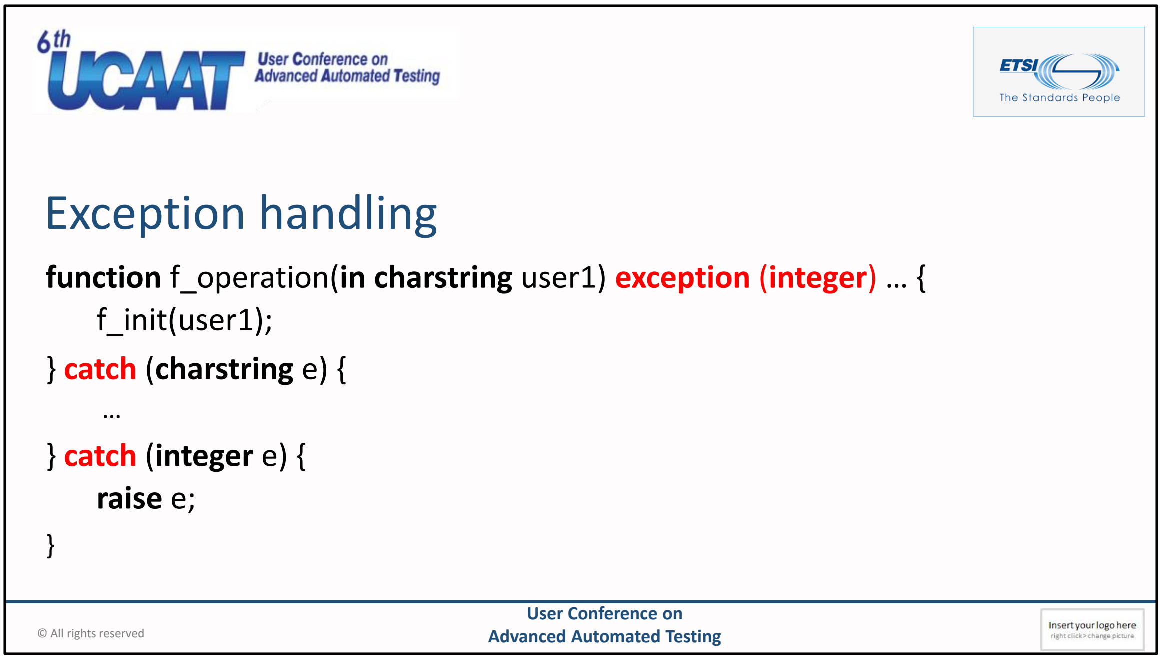
Insert your logo here right click> change picture

An other hardship when creating large libraries is the handling of exceptional execution path and release –ing resources in case of errors.

Exceptions can be thrown to handle exceptional behaviors.
It is also possible (although optional) to list the exceptions that can be thrown inside a function in the function's signature.
Previously functions could indicate such special execution step, by returning error indication values (for example an integer), that on the receiving side is easily forgotten to be handled.
When the exception list is described for the function, tools will have a chance to provide stronger checks (is there an exception that is expected, but not handled here?)

In the example: When the exception is raised f_init terminates.

Example code from the standard extension for throwing an exception.



6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People

Exception handling

```
function f_operation(in charstring user1) exception (integer) … {
      f_init(user1);
} catch (charstring e) {
       …
} catch (integer e) {
      raise e;
}
```

© All rights reserved

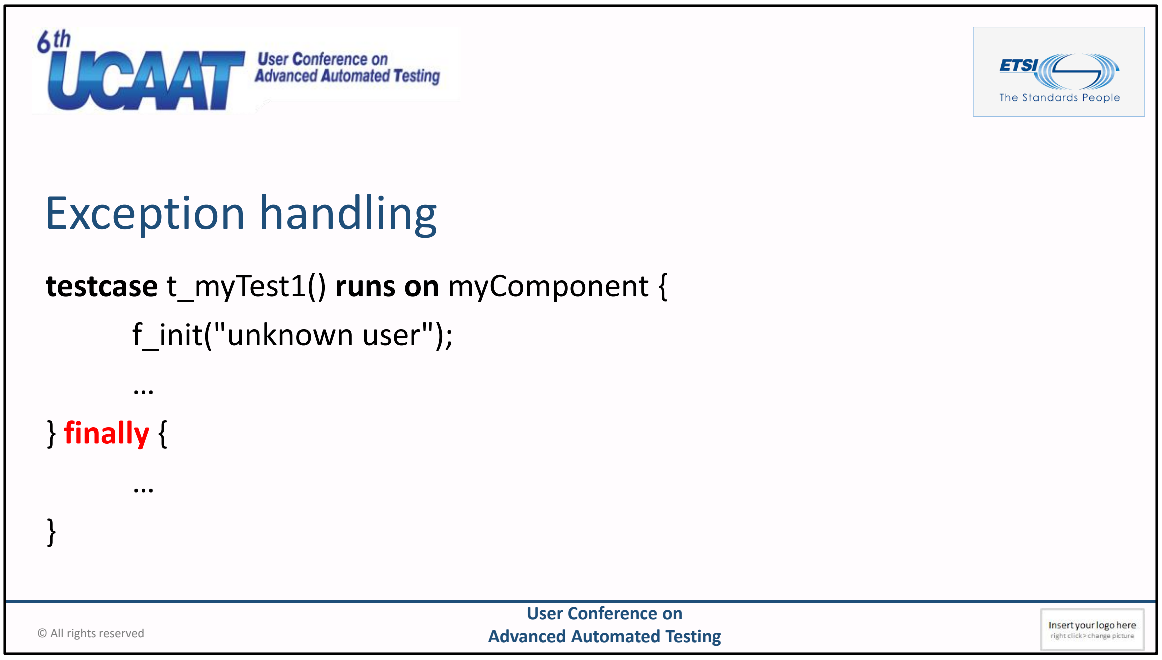
Insert your logo here
right click> change picture

Exceptions do not define new types, already existing types can be used.
For example throwing an exception with a charstring description, or an integer value.

Please note, that there is no try clause in the above code.
Exception catching is not related to a specific try-catch kind of construct like in C++/Java style, but can be added to statementblock (also functions, altsteps, testcases).

In the example:
In both catch cases it is possible to release resources.
The first catch will also terminate the function, returning.
The second catch clause leaves the function raising the exception again so that it can be handled in the calling function.

Example code from the standard extension for catching an exception.



Exception handling

```
testcase t_myTest1() runs on myComponent {
        f_init("unknown user");
        …
} finally {
        …
}
```

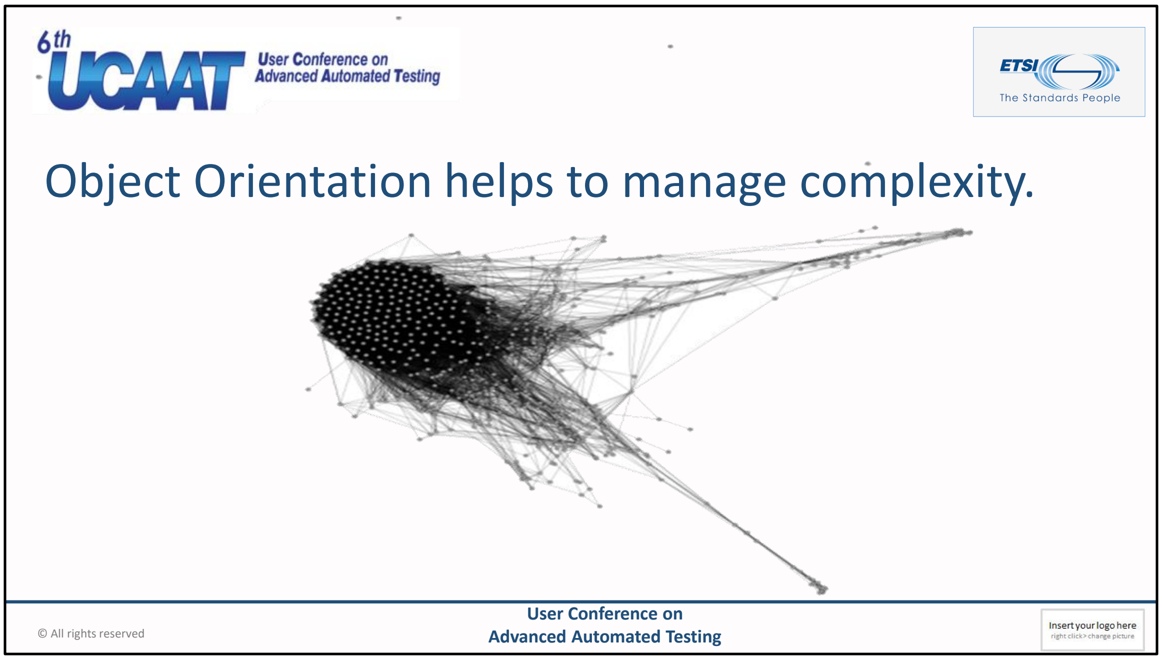
© All rights reserved

Insert your logo here
right click> change picture

In TTCN-3 it will also be possible to use a finally clause.
The contents of this clause get executed when the scope is left, independent of there being an execution or not.
This allows for safe releasing of resources.

In the example:
An exception will be raised in the call to f_init.
Because of the raised exception, execution continues in the finally block.
In the final block component resources can be freed and as the exception is not cought before leaving the testcase, a dynamic error is reported.

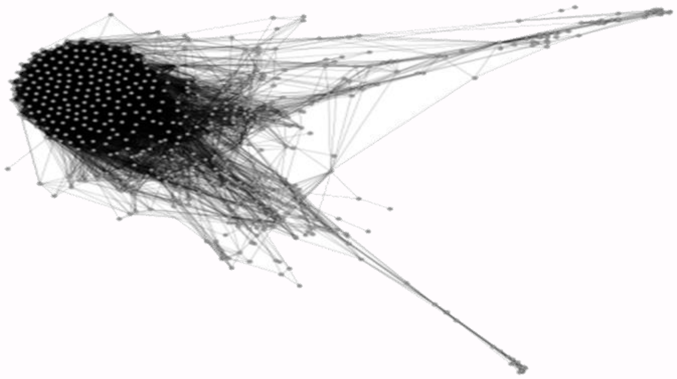
Example code from the standard extension for throwing an exception.



6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People

Object Orientation helps to manage complexity.



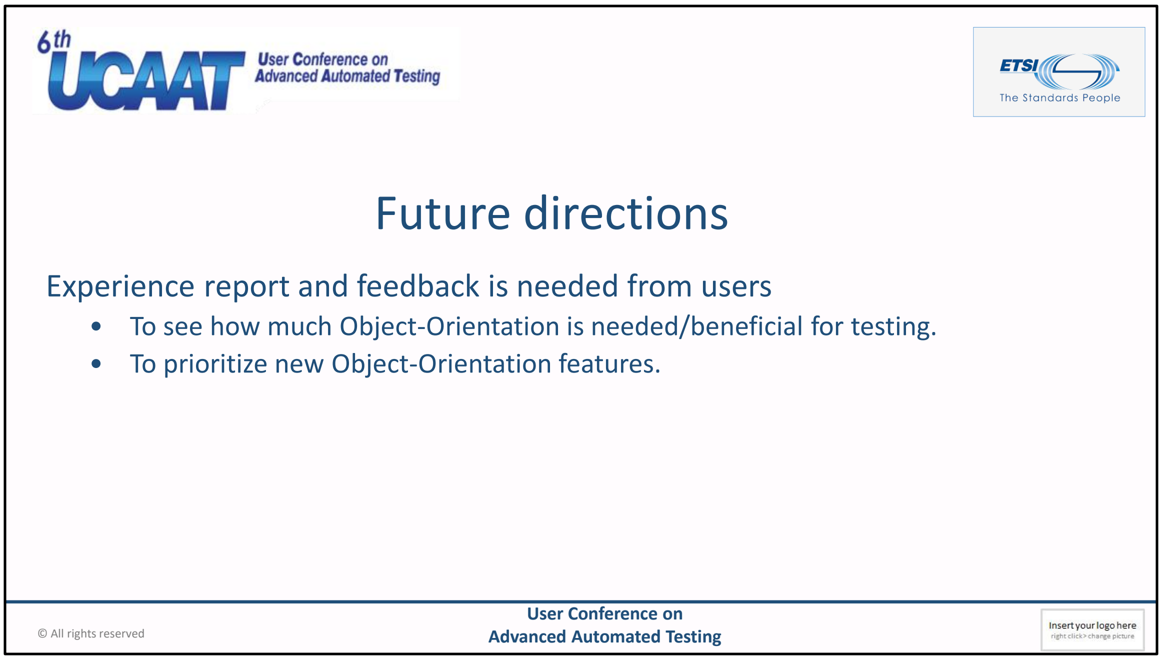
© All rights reserved

User Conference on Advanced Automated Testing

Insert your logo here
right click> change picture

Test are getting bigger + more and more complex !

Object Orientation was seen in many cases to help in creating more maintainable software architectures in software development.



Future directions

Experience report and feedback is needed from users

- To see how much Object-Orientation is needed/beneficial for testing.
- To prioritize new Object-Orientation features.

© All rights reserved

Insert your logo here
right click> change picture

6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People

Thank you for your attention

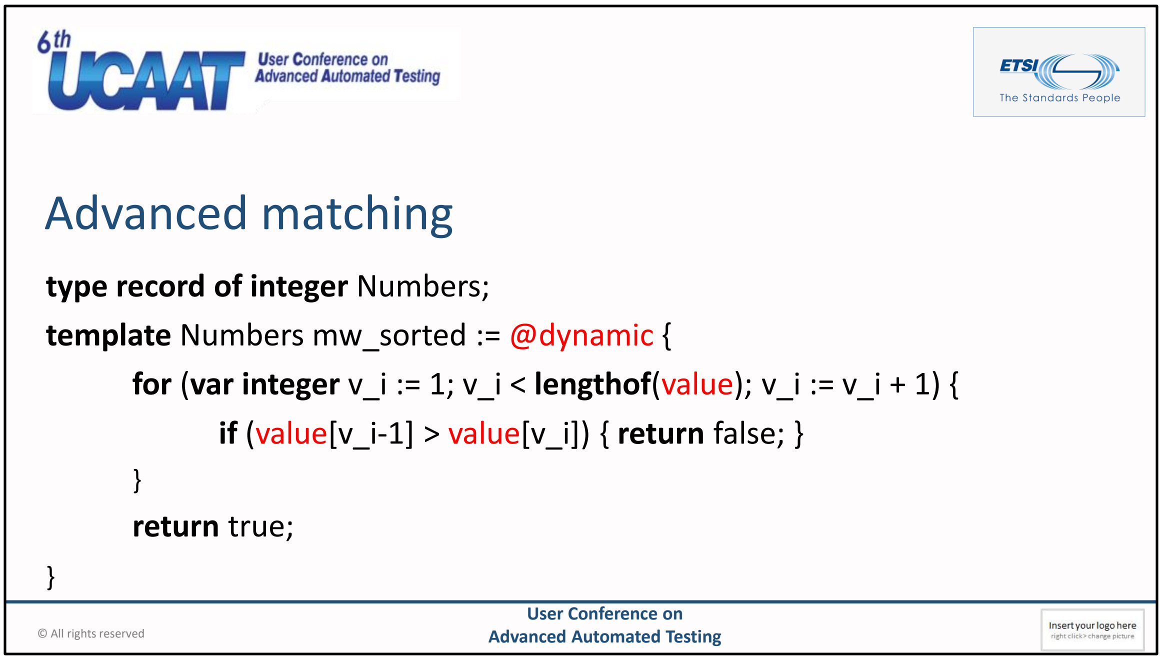
STF550@etsi.org

© All rights reserved

User Conference on
Advanced Automated Testing

Insert your logo here
right click> change picture

Should someone feel like having seen these constructs in C++/Java like languages … we have done a good job, in creating a new language feature that new users can pick up and start using right away.


6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People

Advanced matching

```
type record of integer Numbers;
template Numbers mw_sorted := @dynamic {
        for (var integer v_i := 1; v_i < lengthof(value); v_i := v_i + 1) {
                if (value[v_i-1] > value[v_i]) { return false; }
        }
        return true;
}
```

© All rights reserved

User Conference on
Advanced Automated Testing

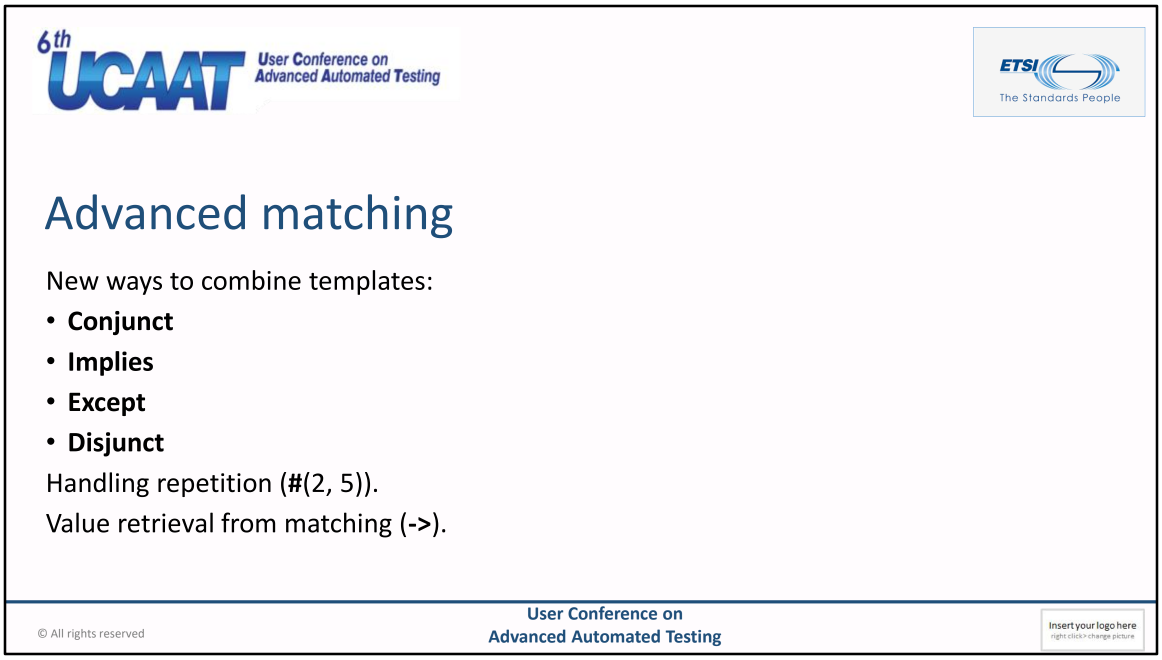
Insert your logo here
right click> change picture

To simplify and increase the power of TTCN-3 template we also introduced advanced matching constructs.

Of which maybe dynamic matching is the most powerful.
Similar to other matching mechanisms, it can be considered as a boolean function that indicates successful matching for the value to be matched by returning the value true and unsuccessful matching by returning the value false.
But in this case the user is able to provide her own algorithm for evaluating the matching criteria.

In the above example value is of type Numbers.
The template uses dynamic matching, and matches if the elements in the record of value are in a descending order.

Example code from the advanced matching standard extension for dynamic matching.


6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People

Advanced matching

New ways to combine templates:

- **Conjunct**
- **Implies**
- **Except**
- **Disjunct**

Handling repetition (**#**(2, 5)).

Value retrieval from matching (**->**).

© All rights reserved

User Conference on Advanced Automated Testing

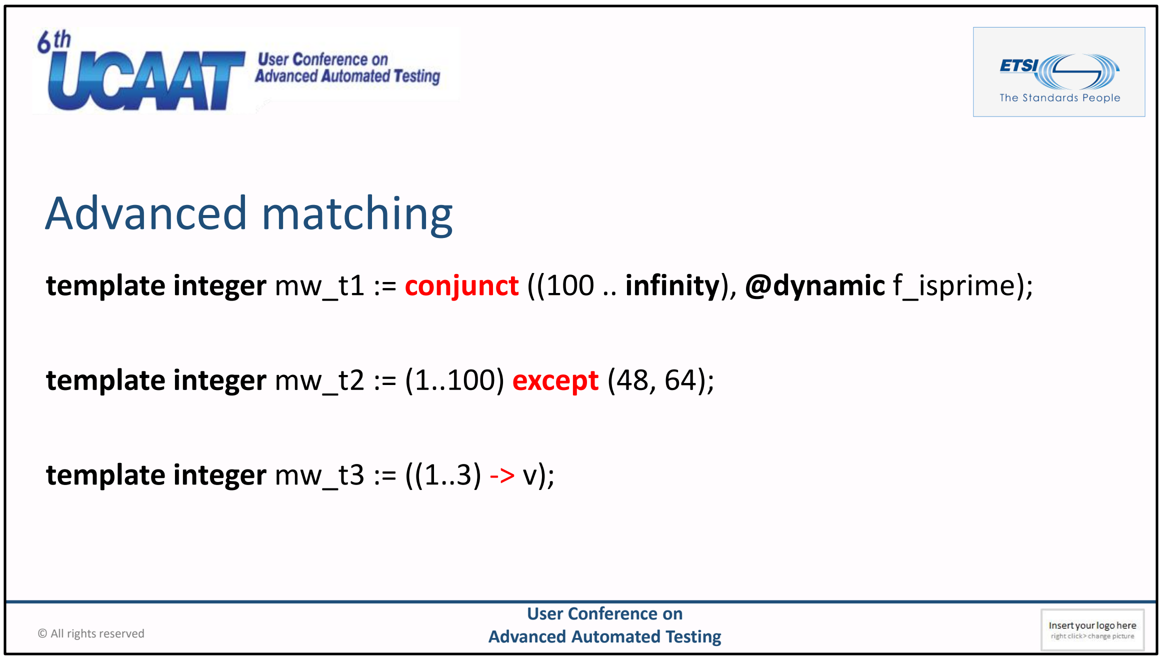
Insert your logo here
right click> change picture

To simplify and increase the power of TTCN-3 template we also introduced advanced matching constructs to combine existing templates.

Conjunct: The conjunction matching mechanism is used when a value shall fulfil several distinct conditions.
Implies: The implication matching mechanism is used when a match is checked only if specified conditions are met.
Except: The exclusion matching mechanism is used when a general matching rule is to be restricted by an exception.
Disjunct: The disjunction matching mechanism specifies one or more alternatives for matching multiple elements of record of, set of or array.

Repetition is a specific symbol used inside record of, array, bitstring, hexstring or octetstring templates to match repeated sequences of items.

In case of a successful template match, the value which matches the template can be assigned to a variable. This can be specified using the “->” symbol.



6th UCAAT User Conference on Advanced Automated Testing

ETSI The Standards People

Advanced matching

```
template integer mw_t1 := conjunct ((100 .. infinity), @dynamic f_isprime);

template integer mw_t2 := (1..100) except (48, 64);

template integer mw_t3 := ((1..3) -> v);
```

© All rights reserved

Insert your logo here
right click > change picture

```
// mw_t1 template matches prime numbers greater than 100
// mw_t2 template matches all integer in the range 1..100 except 48 and 64
// mw_t3 always matches  (1..3) and assigns the value to v.
```

Example code from the advanced matching standard extension.