

# Outline

## 1 Verification Techniques

- software testing: search bugs
- black-box and white-box testing
- static and dynamic testing

## 2 Programming by Contract

- assert statements in Python
- preconditions, postconditions, and loop invariants
- type annotations

## 3 Automatic Theorem Proving

- the four-colour theorem

MCS 260 Lecture 29  
Introduction to Computer Science  
Jan Verschelde, 17 July 2023

# Verification Techniques

## objectives and requirements of software verification

Software is a product, subject to quality control.

A product has (industrial) quality if it performs as specified, as expected by the user.

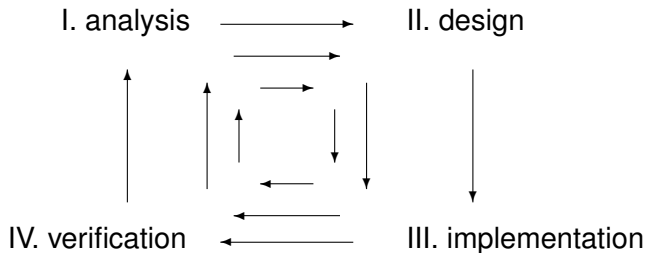
Software testing is part of the software development process (be it waterfall or spiral):

- 1 companies often employ as many software testers as they have developers;
- 2 preliminary versions must pass through beta testing.

The profession of a software tester is just as essential as that of a software developer to achieve quality.

Software testing is that part of software engineering concerned with the systematic search for bugs.

# four stages in the spiral model



# software testing verification techniques

## 1 Verification Techniques

- **software testing: search bugs**
- black-box and white-box testing
- static and dynamic testing

## 2 Programming by Contract

- assert statements in Python
- preconditions, postconditions, and loop invariants
- type annotations

## 3 Automatic Theorem Proving

- the four-colour theorem

# Intel Pentium Floating-Point Division Bug

an infamous software error case study

On 30 October 1994, Dr. Thomas R. Nicely of Lynchberg College traced an unexpected result from his calculations to an incorrect division done by his Pentium PC.

The bug occurs only rarely, but what is really notable is the way Intel handled the situation:

- Although software test engineers had found the bug, management decided not to fix or even announce it.
- Once the bug came out, Intel attempted to diminish its perceived severity.
- Replacement of faulty chips would require proof that the user was affected.

After public outcry, replacements costed \$400 million.  
Now Intel reports known bugs and monitors feedback.

# Software Testing: formal definition of a software bug

The *specification* defines the software:  
how it acts, what it does, and what it does not do.

A *software bug* occurs when one or more of  
the five following rules are true:

- 1 The software doesn't do something that the specification says it should do.
- 2 The software does something that the specification says it shouldn't do.
- 3 The software does something that the specification doesn't mention.
- 4 The software doesn't do something that the specification doesn't mention, but should.
- 5 The software is difficult to understand, hard to use, slow, or viewed by the user as just plain not right.

# software testing verification techniques

## 1 Verification Techniques

- software testing: search bugs
- **black-box and white-box testing**
- static and dynamic testing

## 2 Programming by Contract

- assert statements in Python
- preconditions, postconditions, and loop invariants
- type annotations

## 3 Automatic Theorem Proving

- the four-colour theorem

# Categories of Software Testing

## black-box and white-box

A fundamental requirement is experimental repeatability:  
the same circumstances produce the same results.

We distinguish between

**white**-box testing: choice of input data is based on the internal structure of the program;  
for example: test the functionality offered by one module.

**black**-box testing: choice of input data is based on the functional specification;  
beta testing mostly falls in this category.

White-box (also called glass-box) testing views software from a developer's perspective, whereas black-box testing assumes the user's position.



# software testing verification techniques

## 1 Verification Techniques

- software testing: search bugs
- black-box and white-box testing
- **static and dynamic testing**

## 2 Programming by Contract

- assert statements in Python
- preconditions, postconditions, and loop invariants
- type annotations

## 3 Automatic Theorem Proving

- the four-colour theorem

# Static and Dynamic Testing

combined with white-box and black-box

An additional dimension of testing:

**Static:** *read* specification or source code;

**Dynamic:** *execute* software or test programs.

to the black-box and white-box testing.

We distinguish four types of testing:

- 1 Static black-box testing: test the specification.
- 2 Static white-box testing: inspect the code.
- 3 Dynamic black-box testing: beta testing.
- 4 Dynamic white-box testing: towards debugging.

# Verification and Validation

about static black-box testing

We distinguish between

**Verification:** does software meet its specification?

**Validation:** does software meet user requirements?

The difference often looks subtle:

**Verification:** have we solved the equations right?

**Validation:** have we solved the right equations?

Computing with high working precision (i.e.: many decimal places) may not lead to accurate results.

The Hubble telescope met its specifications on the ground, but not in space.

Never assume that the specification is correct!

Review of the specification requires an understanding of customer expectations, and familiarity of similar products.

# Examining the Code

about static white-box testing

A formal review is often *peer review*:  
done by another programmer (buddy review).

Adhering to coding standards and guidelines  
facilitate code inspections.

A checklist with some types of errors:

- 1 data reference, e.g.: initialization done?
- 2 data declaration, e.g.: of correct type?
- 3 computation, e.g.: overflow or underflow?
- 4 comparison, e.g.: when are two floats zero?
- 5 control flow, e.g.: all cases covered?
- 6 subroutine parameters, e.g.: correct order of input?
- 7 I/O, e.g.: are file formats consistent?

# Dynamic black-box Testing

## beta testing

Two fundamental approaches to testing:

**test to pass:** as an ideal customer would; or

**test to fail:** force errors, provoke exceptions.

Equivalence partitioning reduces the huge set of all possible test cases to a smaller but equally effective set.

Example: test calculator,  $1+2$  is in same class as  $1+5$ ,  
but  $1 + 99999999999999999999999999999999$  is not!

**data** testing: check for random and extreme values, what if enter is pressed without input, test for incorrect input.

**state** testing: verify logic of program through its states. A state is a condition of the software.

# Dynamic white-box Testing

towards debugging

The tests consist of data and procedures.

Isolating the bugs is a common goal with debugging, although dynamic white-box testing is **not** debugging.

We distinguish between

**unit** testing: lowest level, within a module.

**integration** testing: after low level bugs are fixed.

Two approaches:

**Bottom up:** *test drivers* exercise the modules.

Modules may be delivered with test programs that cover its complete functionality.

**Top down:** using *stubs* in place of real modules.

Example: replace thermometer with sensors by a file with test temperature values.

# Applying your Testing Skills

Some specific areas to test:

- 1 configuration testing: suitable hardware?
- 2 compatibility testing: relation to other software?
- 3 foreign language testing: does it makes sense?
- 4 usability testing: easy to use?
- 5 testing the documentation: a good manual?
- 6 testing software security: safe to use?
- 7 website testing: gray-box with html.

For more on the job of software tester:

Ron Patton: *Software Testing*. Second Edition.  
Sams Publishing, 2006.

# software testing verification techniques

## 1 Verification Techniques

- software testing: search bugs
- black-box and white-box testing
- static and dynamic testing

## 2 Programming by Contract

- **assert statements in Python**
- preconditions, postconditions, and loop invariants
- type annotations

## 3 Automatic Theorem Proving

- the four-colour theorem



# Programming by Contract

## the assert statement

To build reliable object-oriented software, the language Eiffel implements the concept of design by contract.

Similar to contracts between humans, we include preconditions and postconditions to any routine.

A **precondition** is a condition on an input parameter.

A **postcondition** is a condition on an output parameter.

Taking invariants into account, one can provide mathematical proofs of correctness.

The most recent version of the Ada standard, Ada 2012, supports contract-based programming.

# the `assert` statement

To implement preconditions and postconditions in Python we use the `assert` statement:

```
>>> x = 4
>>> assert x > 0
>>> assert x < 0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AssertionError
```

Exceptions will be raised if `__debug__` is `True`.

Not meeting the assertions leads to a crash.

Running the program in debug mode means:

test ***and enforce*** all assertions.

Turning on basic optimizations in `python -O` turns off `__debug__`.

# software testing verification techniques

## 1 Verification Techniques

- software testing: search bugs
- black-box and white-box testing
- static and dynamic testing

## 2 Programming by Contract

- assert statements in Python
- **preconditions, postconditions, and loop invariants**
- type annotations

## 3 Automatic Theorem Proving

- the four-colour theorem

# Searching a List: preconditions and postconditions

```
def searchlist(items, item):  
    """  
    Searches a list items for the integer item,  
    returns -1 if the item does not belong to the list,  
    or else returns the position of the item in the list.  
  
    Preconditions:  
    isinstance(items, list) and isinstance(item, int)  
  
    Postconditions:  
    searchlist(items, item) == -1 or  
    searchlist(items, item) == pos and items[pos] == item  
    """
```

- We enforce conditions with `assert`.
- A *loop invariant* is a property of a loop that remains true during the entire execution of the loop.

# software testing

## verification techniques

### 1 Verification Techniques

- software testing: search bugs
- black-box and white-box testing
- static and dynamic testing

### 2 Programming by Contract

- assert statements in Python
- preconditions, postconditions, and loop invariants
- **type annotations**

### 3 Automatic Theorem Proving

- the four-colour theorem

# gradual typing with annotations

- Python is *dynamically typed* as the interpreter determines the type of each variable during execution of a script.
- In a *statically typed* language, the type of each variable is fixed in the code, fixed before execution.
- Python supports *gradual typing* via annotations which allow static type checking.

```
>>> name: str = "hello annotation"
>>> __annotations__
{'name': <class 'str'>}
>>> def search(L: list, i: int) -> int:
...     return L.index(i)
...
>>> help(search)
```

# software testing

## verification techniques

### 1 Verification Techniques

- software testing: search bugs
- black-box and white-box testing
- static and dynamic testing

### 2 Programming by Contract

- assert statements in Python
- preconditions, postconditions, and loop invariants
- type annotations

### 3 Automatic Theorem Proving

- the four-colour theorem

# Automatic Theorem Proving

## Theorem (The Four-Colour Theorem)

*Four colours suffice to color a map so no two adjacent regions receive the same color.*

Conjectured first in 1852 by Francis Guthrie.

Proof in 1976 by Kenneth Appel and Wolfgang Haken at the University of Illinois uses a computer to check 1,936 configurations.

(impractical to check by hand)

Worry: how can proof be correct if program can have bugs?

Georges Gonthier (MS Research, Cambridge):

“A computer-checked proof of the Four Color Problem.”

Using the proof checker Coq in 2005.

Certificate-producing mathematical software.



# Exercises

- 1 Who was Grace Hopper? Read the wikipedia entry about the anecdotal origin of the word bug in connection with computers. Write one paragraph with explanations in your own words.
- 2 Do `conda install pylint` and run it on one of your scripts. Explain the output of this run in your own words.
- 3 Take any of your computer projects (the ones without perfect score) and explain the bugs.
- 4 Can you find any bugs in the solutions to the computer projects posted at the course web site?
- 5 Consider a function to sum a list of integer numbers. Write its specification using pre- and postconditions.
- 6 Use the `assert` statement in the Python code to implement the conditions in the function of the previous exercise.