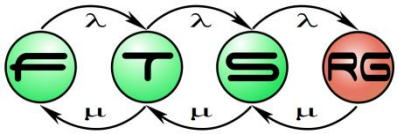
Test oracles

Zoltan Micskei

Budapest University of Technology and Economics
Fault Tolerant Systems Research Group



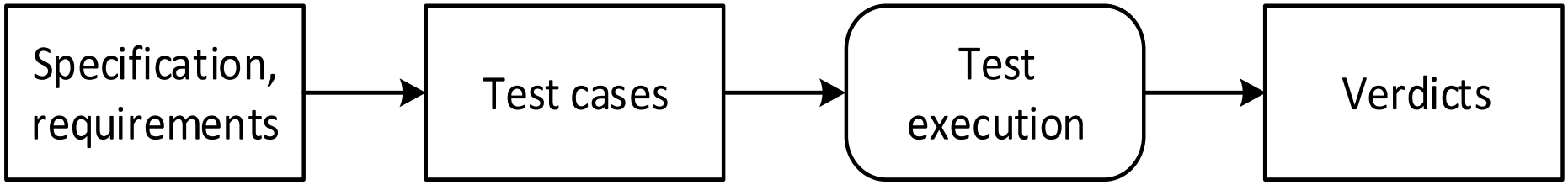
Main topics of the course

- Overview (1.5)
  - Introduction, V&V techniques
- Static techniques (1.5)
  - Specification, Verifying source code
- Dynamic techniques: Testing (7)
  - Testing overview, **Test design techniques**
  - Test generation, Automation
- System-level verification (3)
  - Verifying architecture, Dependability analysis
  - Runtime verification

Learning outcomes

- Recall the definition of test oracle (K1)
- Explain different types of test oracles (K2)

Recap: Basic (simplified) concepts



- Test case
  - a set of test inputs, execution conditions, and expected results developed for a particular objective
- Test oracle
  - A principle or mechanism that helps you decide whether the program passed the test
- Verdict: result (pass / fail / error / inconclusive…)

EXERCISE: Expected result

1. The tax for a vehicle is based on the volume of the engine. Under 1000 cm^3 the tax is 0.1 EUR / cm^3, otherwise 0.2 EUR / cm^3. Hybrid cars qualify for a 10% discount.
   - Selected input #1: volume = 800 cm^3, hybrid = no
   - Expected result: ?
2. The software analyzes data from a particle detector searching for the mass of a new, yet unconfirmed particle.
   - Expected result: ???

Specifying perfectly the expected results is not realistic in many cases

Challenges with test oracles #1

“Non-testable programs”

“Programs which were written in order to determine the answer in the first place. There would be no need to write such programs, if the correct answer were known.”

Source: E. J. Weyuker, “On testing non-testable programs,” Comput. J., vol. 25, no. 4, pp. 465–470, Nov. 1982.

Challenges with test oracles #2

- The software can pass checking all expected results in the tests, but still be incorrect, e.g.
  - Returns the correct value, but way too slow, does not use authentication, uses hard to read UI…
  - The specification used as oracle also misses important special conditions

- The test fails, but the software behaved correctly
  - E.g. some rare external condition was not considered

See also: Test Oracles; Oracle Problem

How to think about test oracles?

Oracles are incomplete, partial, fallible

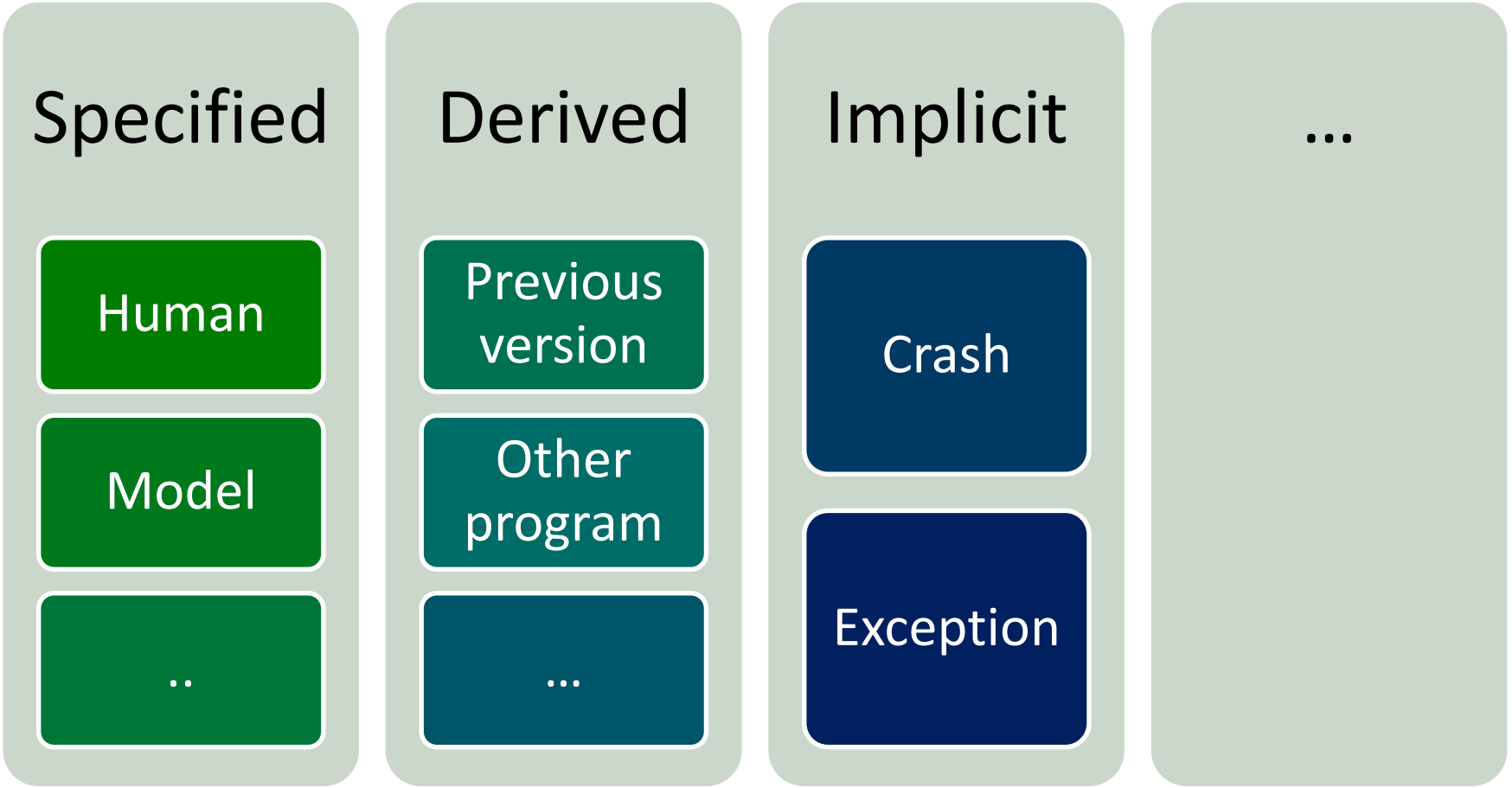
- **Incomplete**: you can never specify fully the expected outcome (~full state of computer)
- **Partial**: it works only for some part of input; it can decide only if an outcome is bad…
- **Fallible**: can cause a wrong verdict

Oracles are heuristics
(decision rule that is useful but not always correct)

TYPES OF TEST ORACLES

Heuristics, Techniques and Examples

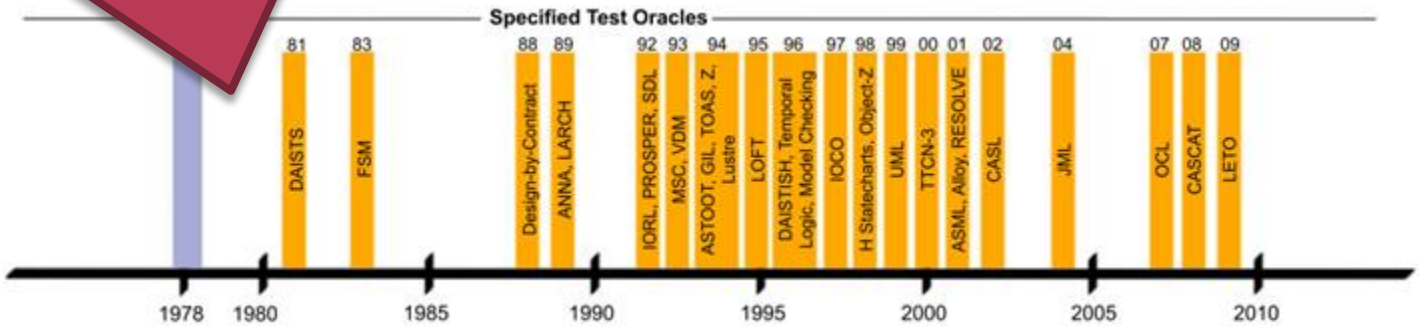
Types of test oracles



Source: The Oracle Problem in Software Testing: A Survey

Specified oracles

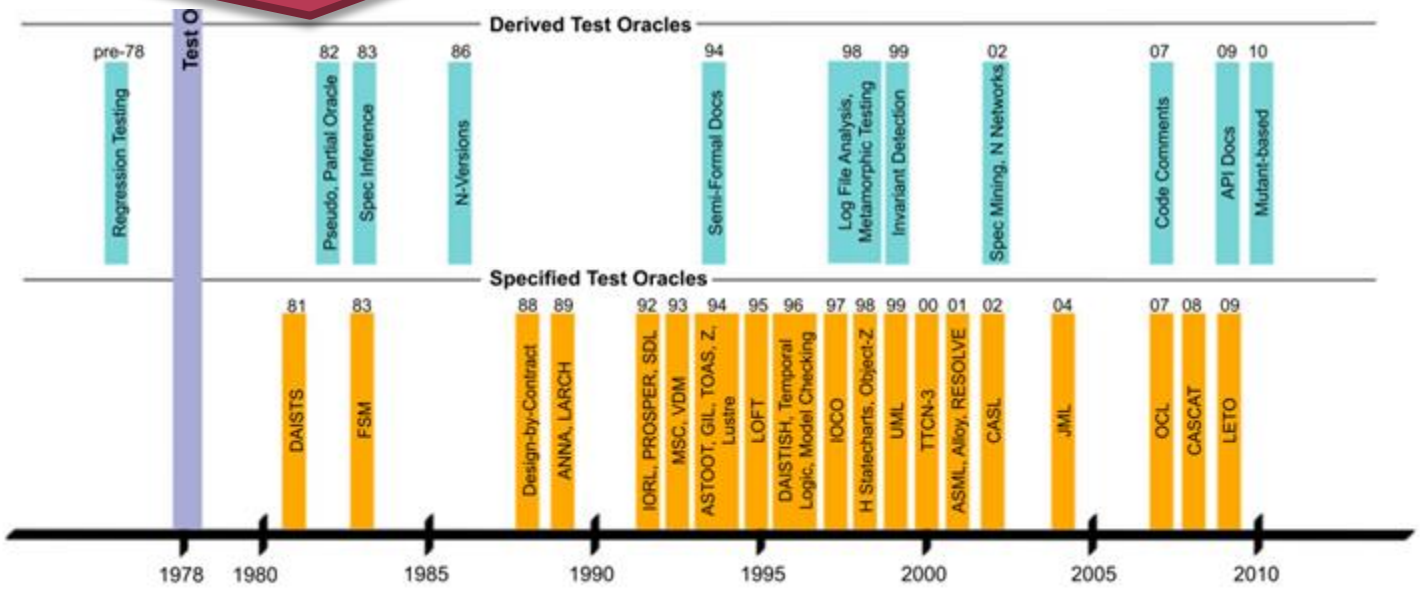
- Humans
- Textual specification
- Models (FSM, UML…)
  - Pre/post-conditions (Design-by-Contract)
  - -> see MBT lecture



Derived oracles

Derive verdict from various other artefacts, e.g.

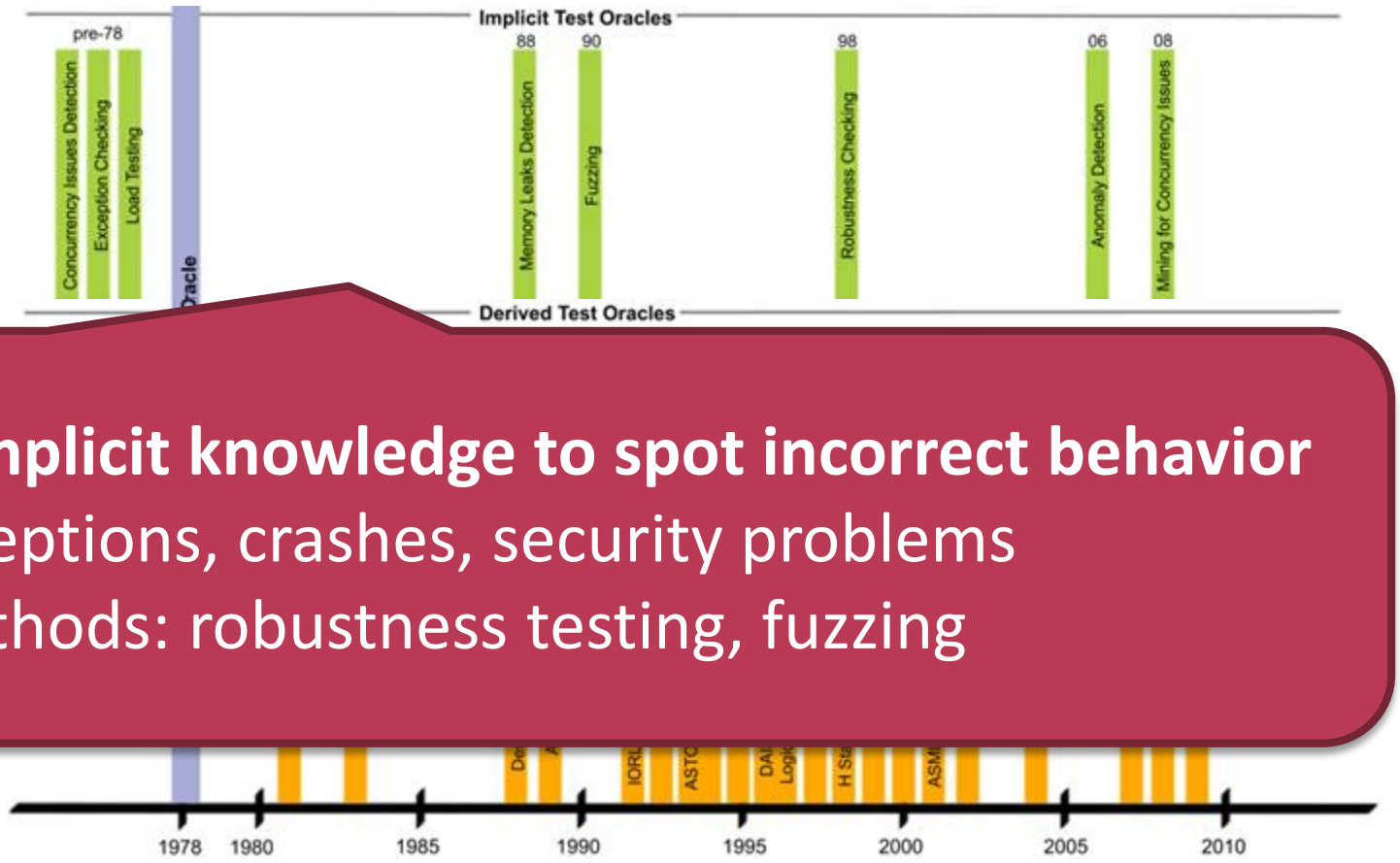
- Compare output with the output of
  - Previous program versions (Regression testing)
  - Different implementations (N-version programming)
  - Only difference can be detected (not pass/fail)!
- Assumptions, validity checks, invertible function



Examples for derived oracles

- Compare the actual output with the output from a previous build/release
- Compare with a program implementing the same specification or API (e.g. POSIX, C/C++)
- Metamorphic relations, e.g. if we add an AND expression to a query, the number of results should be <= number of results of original query
- Integration: calculating is hard, but a candidate result can be checked easily with derivation
- Do not 4.556^4 exactly, but it should be >0

Implicit oracles



Use implicit knowledge to spot incorrect behavior

- Exceptions, crashes, security problems
- Methods: robustness testing, fuzzing

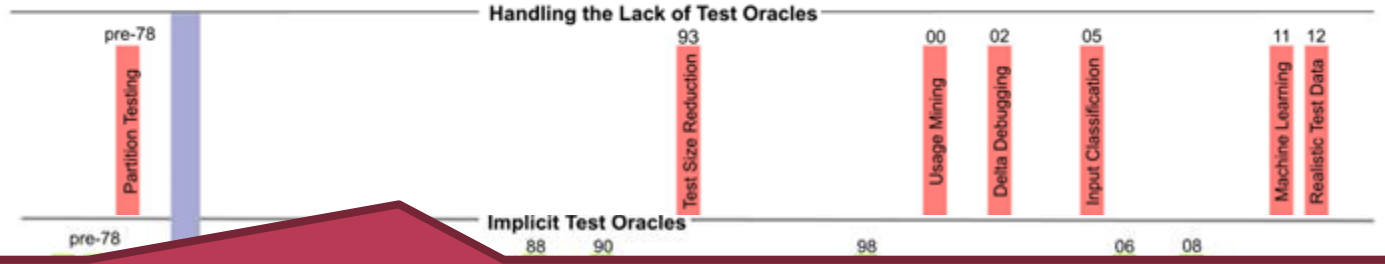
Examples for implicit oracles

- We may not know what the correct result is, but it is definitely bad, if the program
  - throws an unhandled exception to the UI
  - terminates or even crashes the machine
  - makes a buffer overflow possible
  - uses too much HW resources / runs too long

- Typical methods:
  - Fuzzing: generating random or guided random inputs
  - See Code-based test generation lecture

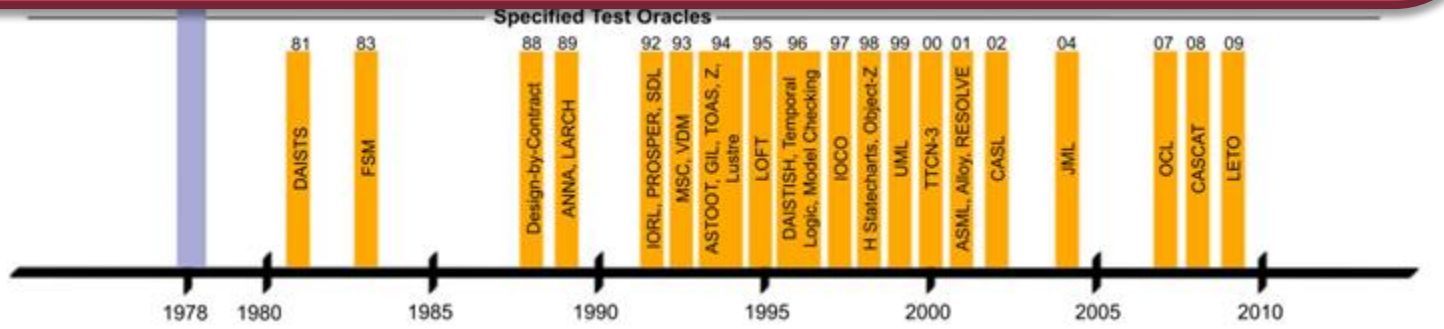
Handling the lack of oracles

BARR ET AL.: THE ORACLE PROBLEM IN SOFTWARE TESTING: A SURVEY



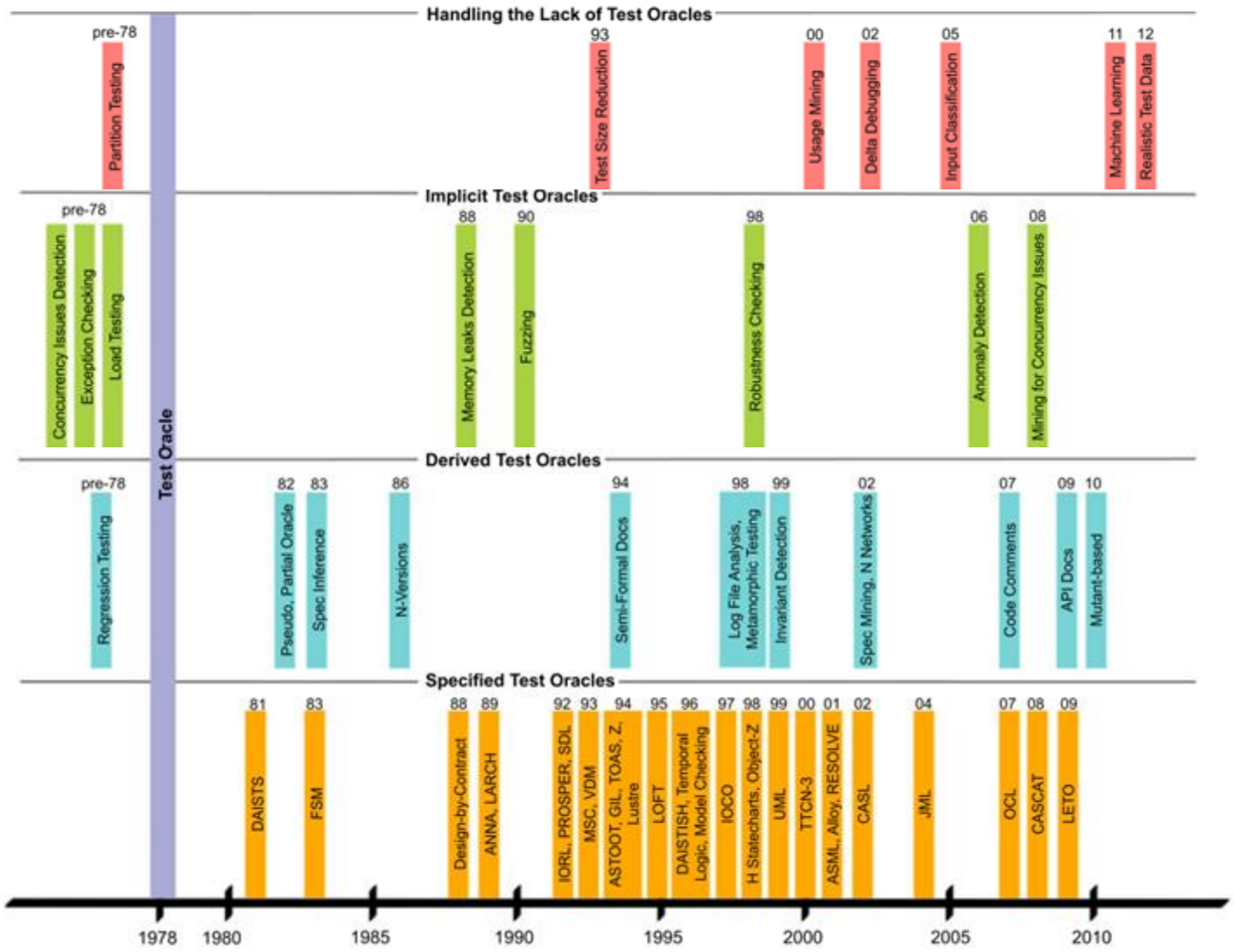
Try to minimize the work of human testers

- Reduce test suite size
- Simplify failing tests (see delta debugging)
- Input classification (group similar outputs)
- Machine learning



Summary of test oracles

BARR ET AL.: THE ORACLE PROBLEM IN SOFTWARE TESTING: A SURVEY



Conclusion

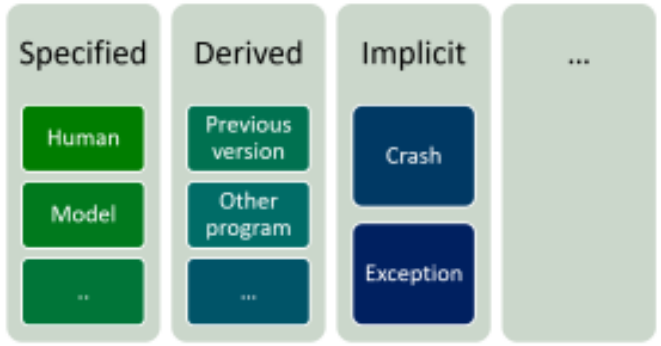
How to think about test oracles?

Oracles are incomplete, partial, fallible

- **Incomplete**: you can never specify fully the expected outcome (~full state of computer)
- **Partial**: it works only for some part of input; it can decide only if an outcome is bad…
- **Fallible**: can cause a wrong verdict

Oracles are heuristics
(decision rule that is useful but not always correct)

Types of test oracles



Source: The Oracle Problem in Software Testing: A Survey

Summary of test oracles