Program correctness

Testing (dynamic)
==========
- test driven development
  - cost of bugfix ~ t^2
- levels
  - unit
  - integration
  - system
  - acceptance
- approaches
  - white-box
  - black-box
  - grey-box
  - fuzzing
- dynamic analyzis
- static analyzis
- code review
- verification
- model checking
- mathematical methods
- practical:
  - contract
  -

Static analysis
==========
- pattern matching (context free)
- AST-based
- flow sensitive
- path sensitive (symbolic execution)
- concolic execution

- true positive
- false positive --> minimize this
- true negative
  - false negative
  -

Static safety
==========
- type system
- Strong type system != Static type system
- e.g. Python has non-static strong type system
- Concept: type system for templates

Dynamic safety
==========
- Contract (C assert, Eiffel contract, C++20 delayed)
- Eiffel (Bertrand Meyer): "Design by Contract" (2000)
  - Eiffel: first order predicate logic
    (including quantifyers: there_exists, for_all)
  across my_list as l
     all l.item.some_property = some_value end

  - precondition "require"
  - postcondition "ensure"
  - invariants   "invariant"
  - assertions   "check"

e.g.
  invariant
    valid_capacity: capacity >0
    valid_item_count: item_count >=0
                  and item_count <= capacity

    definition_of_empty:(empty implies item_count = 0)
                and (item_count = 0 implies empty)
    definition_of_full: (full implies item_count = capacity)
                and (item_count = capacity implies full)

Redefinition - refining of contract
==========
  e.g. inheritance
  Liskov Substitutional Principle

  Der : Base;
  Base b;
  Derived d;

  P1 is stronger than P2 iff P1 => P2

  <Pre',Post'> : Derived is a sub-specification of
  <Pre, Post> : Base
    iff Pre => Pre' and Post' => Post

                (x'->y') <: (x -> y)
                  x' <: x && y <: y'

Covariant return type
==========

contravariant input parameters
(by type theory, identical by practice)

see: https://en.wikipedia.org/wiki/Covariance_and_contravariance_(computer_science)
Eiffel makes itt wrong :
https://www1.icsi.berkeley.edu/~sather/Documentation/EclecticTutorial/node15.html

In C++:

```
Base          { virtual X  f(Y); }
Derived : Base { virtual X' f(Y); }
```

Other methods
==========

- assert
- C++2b Contracts
- C++1x Concept - axioms -- removed from standard
  "operator< is transitive, non-reflexive"
- Java -- perhaps using AspectJ (Gregor Kitzales)

Open issues: WCET (Worst Case Execution Time)

- HUME
- Rust (== Safer C++)
- Agda (Dependent type logic) used among others to prove compilers

Program provers

- Coq
- Idris