UNIT 1 - Fundamentals of Test and Analysis

A FRAMEWORK FOR TEST AND ANALYSIS

Validation and Verification

Degrees of Freedom

Varieties of Software

VALIDATION AND VERIFICATION

Assessing the degree to which a software system actually fulfills its requirements, in the sense of meeting the user’s real needs, is called ***validation.***
Fulfilling requirements is not the same as conforming to a requirements specification.
A specification is a statement about a particular solution to a problem, and that proposed solution may or may not achieve its goals.

Verification is checking the consistency of an implementation with a specification.
Here “*specification*” and *“implementation”* are roles, not particular aritificats.
The purpose of verification is to determine if the system is well-engineered and error free.

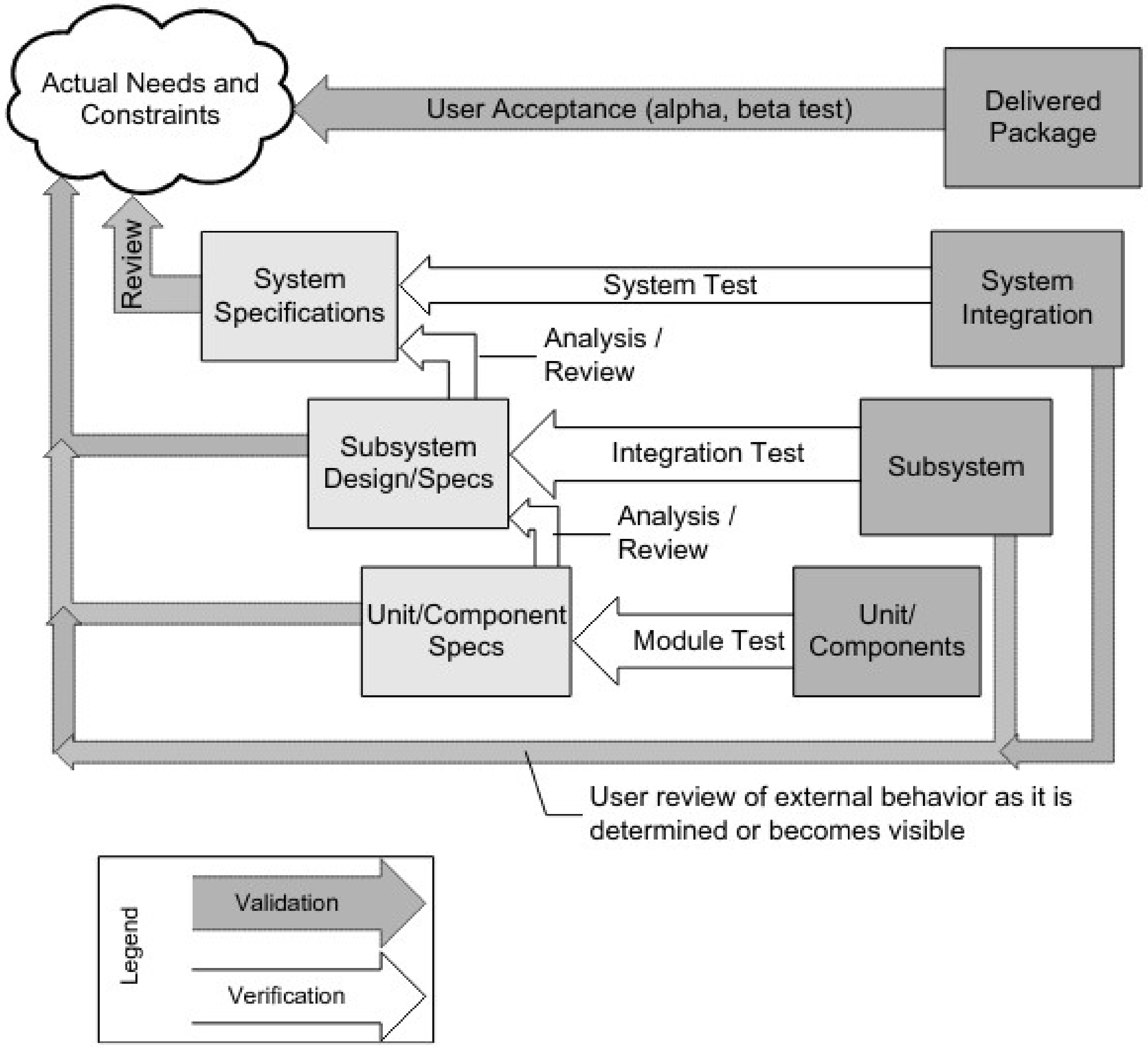
STATIC TESTING

Static testing techniques are a strong tool to increase software development quality and productivity by supporting engineers in identifying and correcting their own errors early in the software development process.

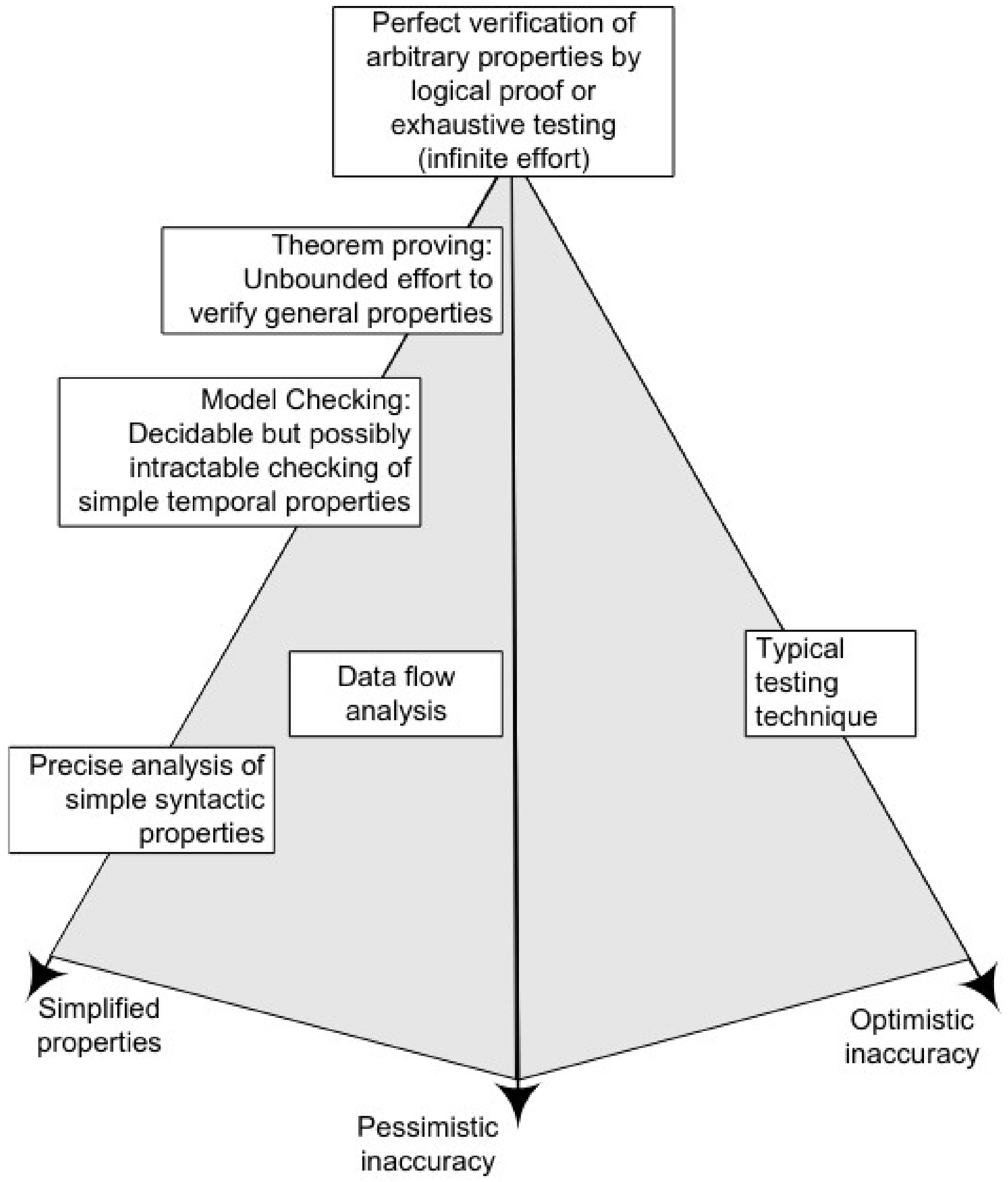
Defects found during static testing include departures from standards, missing requirements, design flaws, non-maintainable code and conflicting interface definitions.

VERIFICATION VS VALIDATION

Verification	**Validation**
Document, design and program *verification* is a static practice	*Validation* is a dynamic process used to validate and test the real product
It does not include running the code	It always entains running the code
It is based on human *verification* of papers and data	It is computer-based program execution
Verification procedures include inspections, reviews, walkthroughs, and desk-checking among others	*Validation* employs techniques such as black box testing, gray box testing and white box testing among others


Actual Needs and Constraints
User Acceptance (alpha, beta test)
Delivered Package
Review
System Specifications
System Test
System Integration
Analysis / Review
Subsystem Design/Specs
Integration Test
Subsystem
Analysis / Review
Unit/Component Specs
Module Test
Unit/ Components
User review of external behavior as it is determined or becomes visible
Legend
Validation
Verification

DEGREES OF FREEDOM


Perfect verification of arbitrary properties by logical proof or exhaustive testing (infinite effort)
Theorem proving: Unbounded effort to verify general properties
Model Checking: Decidable but possibly intractable checking of simple temporal properties
Data flow analysis
Typical testing technique
Precise analysis of simple syntactic properties
Simplified properties
Pessimistic inaccuracy
Optimistic inaccuracy

```
class Trivial
{
   static int sum(int a, int b)
   {
      return a+b;
   }
}
```

int is 32 bits.

2^32 * 2^32 = 2^64 ~10^21

At one nanosecond per test case, this will take approximately 10^21 seconds, or 30000 years.

Exhaustive testing cannot be completed in any finite amount of time.

In theory, undecidability of a property *S* merely implies that for each verification technique for checking *S*, there is at one “pathological” program for which that technique cannot obtain a correct answer in finite time. It does not imply that verification will always fail or even that will usually fail, only that it will fail in at least one case. In practice, failure is not only possible but common, and we are forced to accept a significant degree of inaccuracy.
A technique of verifying a property can be inaccurate in one of two directions are:

1. **Optimistic inaccuracy:**

- We may accept some program that do not possess the property.
- It may not detect all violations
- Example: Testing, because no finite number of tests can guarantee correctness

2. Pessimistic inaccuracy:

- It is not guaranteed to accept a program even if the program does possess the property being analyzed, because of false alarms.
- Examples: Automated program analysis

A software verification technique that errors only in the pessimistic direction is called conservative analysis.

However, a conservative analysis will often produce a very large number of spurious errors reports, in addition to a few accurate reports. A human may, with some effort, distinguish real faults from a few spurious reports, but cannot cope effectively with a long list of purported faults of which most are false alarms. Often only careful choice of complementary optimistic and pessimistic techniques can help in mutually reducing the different problems of the techniques and produce acceptable results.

Spurious: outwardly similar or corresponding to something without having its genuine qualities

3. **Simplified properties:**

- It reduces the degree of free by simplifying the property to check
- Example: Model Checking

It substitutes a property that is more easily checked, or constraining the class of programs that can be checked.

Many examples of substituting simple, checkable properties for actual properties of interest can be found in the design of modern programming languages

```
int i, sum;
int first=1;
for(i=0;i<10;i++)
{
   if (first)
   {
      sum=0,first=0;
   }
   sum+=i;
}
```

- In C language, a compiler cannot provide precise static check.
- In java neatly solves this problem by making code like this illegal: variable must be initialized on *all* program control paths, whether or not that path can ever be executed.

VARIETIES OF SOFTWARE

The *“generic”* software testing techniques are at least partly applicable to most varieties of software, particular application domains (ex: real-time and safety-critical software) and construction methods (ex: concurrency and physical distribution, graphical user interfaces) call for particular properties to be verified, or the relative importance of different properties, as well as imposing constraints on applicable techniques. Typically a software system does not fall neatly into one category but rather number of relevant characteristics that must be considered when planning verification.

CONCLUSION

Validation and verification are critical components of software development. If comprehensive verification and validation are not carried out, a software team may be unable to build a product that meets the expectations of stakeholders. Verification and validation reduce the chance of product failure and raise the reliability of the final product

In summary, Precise answers to verification questions are sometimes difficult or impossible to obtain in theory and in practice. Verification is therefore an act of compromise, accepting some degree of optimistic inaccuracy or pessimistic inaccuracy or choosing to check a property that is only an approximation of what we really wish to check. Building great software products requires mastering the skill of picking the right trade-off. This requires experience and a strong understanding of what lies ahead. The most successful trade-offs are those that minimize risk while setting up the groundwork to address the negative side of the compromise.

THANK YOU