Verification and Validation

Software Reliability and Testing - Barbara Russo

SwSE - Software and Systems Engineering research group

unibz
Freie Universität Bozen
Libera Università di Bolzano
Università Liedia de Bulsan

Verification and Validation

- Software products is imperfect as it is created by human beings
- Verification and Validation techniques are methods to ensure the final product quality

Verification

- Check of consistency of an implementation with a specification
- It is about “How” i.e., the process of building
  - **Are we building the product right?" (B. Boehm)**
- Example: A music player plays (it does play) the music when I press Play

Verification

- Check consistency between two descriptions (roles) of the system at different stages of the development process,
  - e.g., UML class diagram and its code implementation
  - Specification document and UML class diagram
- Chain of Two Roles:
  - Specification ⇒ Implementation (Specification) ⇒ Implementation

Validation

- Degrees at which a software system fulfils the user requirements
- It is about “What” - the product itself
  - **Are we building the right product ? (B. Boehm)**
- Example: A music player plays a music (it does not show a video) when I press Play

Usefulness vs. dependability

- Requirements are goals of a software system
- Specifications are solution to achieve requirements
  - Software that matches requirements ⇒ **useful software**
  - Software that matches specifications ⇒ **dependable software**

Dependability

- Degree at which a software system complies with its specifications (focus on verification)
  - Specifications are solutions to a problem described in requirements ⇒
  - They are prone to defects as they have been written by human beings

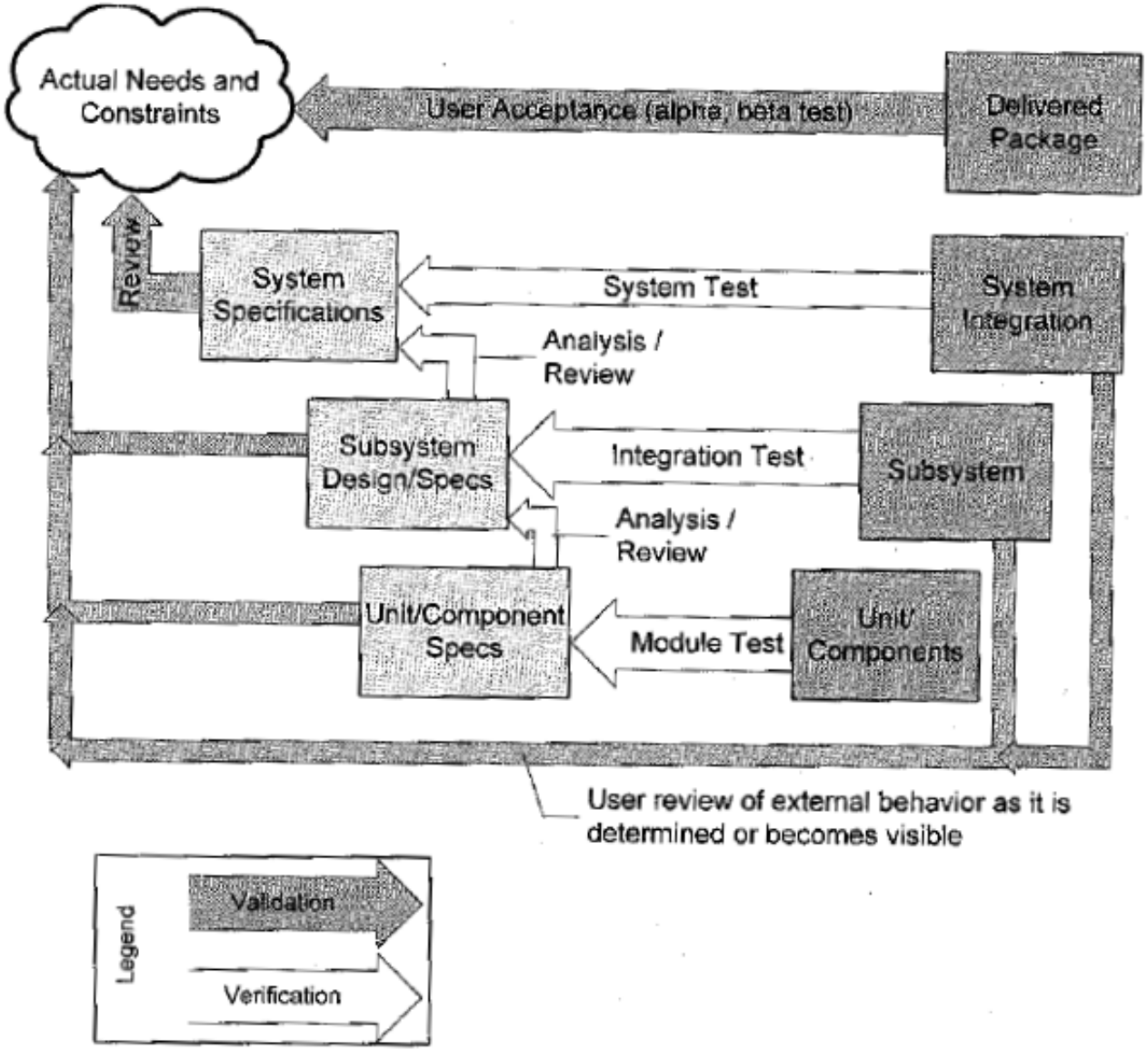

Figure : Verification vs. validation. Source: Pezze and Young, Software Testing and Analysis, Wiley, 2008

Verification vs. Validation

- Validation involves stakeholders' judgment
- Exercise: Discuss a validation technique

Verification vs. Validation

- Verification manly focuses on dependability and concerns four software properties:
  - **Correctness**: consistency with specification
  - **Reliability**: statistical approximation to correctness; probability that a system deviates from the expected behaviour

Verification vs. Validation

- **Robustness:** being able to maintain operations under exceptional circumstances of not full-functionality
- **Safety:** robustness in case of hazardous behaviour (attacks)

Specification Self-consistency

- **Consistency**: Specification vs specification, no conflicts
- **No ambiguity**: open to interpretations, uncertainty
- **Adherence to standards**: consistency with benchmarks

Checking dependability

- How can we check whether our software satisfies any of the dependability properties?
- For example, correctness: given a set of specifications and a program we want to find some logical procedure **(e.g., a proof)** to say that the program satisfies the specifications

Undecidability of problems

Some problems cannot be solved by any computer program (Alan Turing)

The halting problem

Given a program P and an input I, it is not decidable whether P will eventually halt when it runs with that input or it runs forever

Checking a property with algorithms

- Undecidability implies that given a program P and a verification technique T we do not know whether the technique can verify the program in finite time
- ... and even when checking is feasible it might be very expensive

Techniques for verification

- Thus, techniques for verification are inaccurate
- **Optimistic and pessimistic inaccuracy** of a testing technique

Optimistic accuracy

- Optimistic inaccuracy: technique that verifies a property **S can return TRUE on programs that does not have the property (FALSE POSITIVE)**

Example

- Testing is an optimistic technique
- It returns that a program is correct even if no finite number of tests can guarantee correctness

Pessimistic Inaccuracy

- Pessimistic inaccuracy: technique that verifies a property **S can return FALSE on programs that have the property (FALSE NEGATIVE)**
- Conservative technique

Example

- Automatic verification
- It is conservative as it typically uses rules (not heuristics!)

Analysis of a property: confusion matrix

	Pred. TRUE	***Pred. FALSE***
TRUE	*TP*	*FN*
FALSE	*FP*	*TN*

Analysis of a dependability property

- Testing techniques are introduced to analyse the dependability properties of a system

Analysis of programs for dependability

- Optimistic analysis might also return TRUE for non correct programs (it might be FP>0)
- Pessimistic analysis might also return FALSE for correct programs (it might be FN>0)

Example

- Dependability property = **correctness**
- Assume we know how many methods/classes are correct/incorrect in our program (i.e., they are aligned with specifications)
- Use **test coverage technique** to analyse the correctness of methods/classes…
- Determine the accuracy of the technique with the confusion matrix

Example

```
[1] int foo (int a, int b, int c, int d, float e) {
[2]     e;
[3]     if (a == 0) {
[4]         return 0;
[5]     }
[6]     int x = 0;
[7]     if ( (a==b) II ( (c == d) && bug(a) ) ) {
[8]         x=1;
[9]     }
[10]     e = 1/x;
[11]     return e;
[12] }
```

bug(a) = 1 is !a==0 else 0

Property: method correctness. Is this method correct? 100% statement coverage => correct. T(0,0,0,0,0) and T(1,1,0,0,0) cover method statements. Is it FP, FN, TP, TN?

Substituting principle

- In complex system, verifying properties can be infeasible
- Often this happens when properties are related to specific human judgements, but not only

Substituting principle

- Substituting a property with one that can be easier verified
- Constraining the class of programs to verify
- Separate human judgment from objective verification

Example - substitutability

- “Race condition": interference between writing data in one process and reading or writing related data in another process (e.g., an array accessed different threads)
- Testing the **integrity** of shared data is difficult as it is checked at run time
- Typical solution is to adhere to a protocol of serialisation

Serialisation

- When group of objects or states can be transmitted as one entity and then at arrival reconstructed into the original distinct objects

Java object serialisation

- An object can be represented as a **sequence of bytes** that includes the object's data as well as information about the object's type and its types of data
- After a serialised object has been written into a file, it can be read from the file and deserialised: the type information and bytes that represent the object and its data can be used to recreate the object in memory