

Verification and Validation

The keys to reliable quality software.



Definitions

- Verification is ...
 - the activity of evaluating whether the system is correctly built, i.e. that it fulfills its stated requirements.
 - the process of proving that the developed code can be derived formally (i.e. mathematically) from the specification.
 - the process of determining that an implementation accurately represents the developer's conceptual description and specifications.



Definitions

- Validation is ...
 - the activity of evaluating whether the right system has been built, i.e. that it fulfills the user's needs.
 - the process of determining the degree to which a model is an accurate representation of the real-world from the perspective of the intended uses of the model.
 - the process of demonstrating to the client that the software performs the process required by the client.



Verification is building the product right, and Validation is building the right product.

Reference:
Boehm, B. W., "*Verifying and Validating Software Requirements and Design Specifications*" reprinted in Boehm, B. W. (ed.), *Software Risk Management*, pp. 205 - 218, IEEE Computer Society Press, 1989



Techniques and Methodologies for V&V

Inspection and Testing



Approaches



- Software Verification and Validation both use static and dynamic techniques of system checking to ensure that the resulting program satisfies its specification and that the program as implemented meets the expectations of the stakeholders.
 - Static techniques are concerned with the analysis and checking of system representations throughout all stages of the software life cycle while dynamic techniques only involve the implemented system.

Types of Techniques



Informal	Static	Dynamic	Formal
audit	consistency checking	black-box testing	predicate calculus
desk checking	data flow analysis	execution monitoring, profiling, tracing	lambda calculus
reviews	syntax analysis	regression testing	induction
walkthroughs	semantic analysis	sensitivity analysis	logical deduction
	structural analysis	statistical techniques	inference
	graph-based analysis	stress testing	proof of correctness
	path analysis	white-box testing	
	boundary analysis		

Verification



- Software inspections are concerned with the analysis of the static system representation to discover problems (static verification).
- Inspection is a manual, static technique that can be applied early in the development cycle.
- Inspection is based on reading techniques.

Formal Verification



- Verification may be done in a formal way, using the propositional calculus to prove that code implements the specifications correctly.
 - There are formal specification languages such as VDM and Z to support this.
 - It may also be done less formally by reasoning carefully about the code as it is being developed.
 - The objective is to write correct, readable code in the first place. If the reasoning gets difficult, this may be a sign that the code can be improved.

Validation



- Validation cannot be completed without operational software.
- Prior to implementation, the developer can communicate their understanding of the problem to the client and users.
 - This can be done by providing incomplete models through Rapid Prototyping, or incomplete implementations through Incremental Development.

Software Testing



- Software testing is concerned with exercising and observing product behaviour.
- The system is executed with test data and its operational behaviour is observed
- Testing concerns dynamic verification of system qualities, for example, functionality, reliability and performance.

Software Validation using PMD

Static Analysis for Java Code.
<http://pmd.sourceforge.net/>



What is PMD?



- PMD is an utility for finding problems in Java code using static analysis.
- It was initially written for a DARPA project but it has been open sourced and released on SourceForge.

Rules



- PMD comes with rules to find potential problems such as:
 - Unused local variables
 - Empty catch blocks
 - Unused parameters
 - Empty 'if' statements
 - Duplicate import statements
 - Unused private methods
 - Classes which could be Singletons
 - Short/long variable and method names
- You can also add new rules to PMD.

Best Practices



- Running every ruleset will result in a huge number of rule violations, most of which will be unimportant. Then you have to sort through a thousand line report to find the few you're really interested.
- Instead, start with some of the obvious rulesets - just run unusedcode and fix any unused locals and fields. Then, run basic and fix all the empty if statements and such-like. Then peruse the design and controversial rulesets and use the ones you like via a custom ruleset.

JJGuidelines



- JJGuidelines is a project sponsored by the Belgian government to codify coding practices.
- It uses PMD as the compliance checking utility.
 - The technical paper : Java and J2EE Conventions, Guidelines and Best Practices (v0.0.2 - rev. 4 - November 4, 2003) that describes JJGuidelines can be found at <https://jjguidelines.dev.java.net/>