# 7 principles of testing.

## Principle 1: testing shows presence of defects

Testing doe not mean a product is bug free.

Testing reduces the probability of unfound bugs.

## Principle 2: Exhaustive testing is impossible

Impossible to test all possible combinations

Testing combinations based on priority

Optimal amount of testing based on risk assessment.

## Principle 3: Early Testing

Testers involved early in software development.

Longer the bug is present the more costly it could be to fix.

## Principle 4: Defect clustering

20% of the modules could cause80% of the problems

With experience you can identify risky modules

Issues remain even after tests pass, repeated functionality that throws error.

## Principle 5: Pesticide Paradox

Don’t use the same tests again and again

Eventually they will no longer find bugs

Adapt existing tests

Add new tests

## Principle 6: Testing is context dependent

Almost every industry is now using software.

Type of testing done is dependent on the industry.

## Principle 7: Fallacy of absence of errors

Product must fulfill the requirements of client!

Finding and fixing defects does not help if the system build is.

Unusable and does not fulfill the user’s needs & requirements.

## Functional testing/unit testing

Tests if code delivers the desired output.

Check interfaces and components.

Focus on finding issues.

Tests only most critical functionalities

If the build passes the smoke testing, then it is considered as a stable build.

## Non-functional/security

The parameter defines how a system is safeguarded against deliberate and sudden attacks from internal and external sources.

Check information leakage by encryption, firewall, etc.

How Easy is the software to use.

How Scalabel is the software.

# Test analyse & Design

Test analyse behandelt WAT

Test design behandelt HOE

## Analyse

* Test basis analyseren
* Test basis evalueren
* Identificeren welke features er getest moeten worden
* Voor elke feature de testcondities afbakenen
* Vastleggen van traceerbaarheid tussen test basis en test condities.

Logisch sequentieel <---> iteratief geimplementeerd.

### Basis

Alle informatie die nodig is om te weten wat testen.

* Vereisten
* Design en implementatie
* Code
* Risk

Testbare variabelen en bijhorende condities bepalen.

System Requirements, functional design, technical design, business requirement.

## Design

* Ontwerpen en prioriteren van test cases
* Nodige data identificeren om testcondities te ondersteunen
* Ontwerpen van de testomgeving en identificeren van benodigde infrastructuur.
* Vastleggen van traceerbaarheid tussen test basis, test condities en test cases.

Mogelijkheden uit test analyse specifieker maken.

* Lower level testing kunnen analyse en design gelijktijdig gebeuren.
* Higher level testing meestal sequentieel werken.

Een set aan acties die een verwacht resultaat moeten opleveren.

* Zeer belangrijk om uitkomst vooraf te bepalen.
* Na uitvoering van acties de gekregen uitkomst vergelijken met de verwachte uitkomst.

### Manueel test Design

Manueel testen brengt risico’s met zich mee

* Dingen over het hoofd zien
* Veel testen nodig
* Tests lopen sneller fout

Manueel testen is zeer arbeidsintensief

* Test design tools generen vaak zelf test cases en bijhorende test data
* Geautomatiseerde uitvoering van test cases
* Geautomatiseerde vergelijking tussen effectieve- en verwachte uitkomst.

# Blackbox testing vs Whitebox Testing

## Whitebox testing

* Transparantie
* Interne werking testen
* I/O tests
* Bespreek de belangrijkste initiatieven van het afgelopen jaar.

Wat word getest?

* Veiligheid risico’s
* Foute of slechte paden
* Verwachte output
* Verwerking van specifieke input
* Conditionele lussen
* Object/Statement individueel testen

Steps:

* Code analyseren
* Code zonder test cases identificeren
* Testen aanmaken

Type testen

* Statement Coverage
* Branch Coverage
* Condition Coverage
* Multiple Condition Coverage
* Path Coverage
* Function Coverage

Testers zijn programmeurs

Complexiteit hangt van project af

Garandeert kwaliteitsvolle en efficiente code

Groot project heeft veel test cases

## Blackbox testing

Niet transparant

End user experience

I/O testing

### Technieken

* Functional testing
* Non-functional testing
* Regression testing

Steps:

* Doelstellingen & specificaties analyseren
* Mogelijke input bepalen
* Verwachte output bepalen
* Test cases maken
* Testen uitvoeren
* Output vergelijken met verwachtingen
* Fouten verteren & opnieuw testen

# Statisch vs dynamisch testen

## Statisch testen

Wordt ook verification testing genoemd

Verificatie van minimum requirements.

Testen waarbij software/code niet uitgevoerd.

Gebeurt voor de compilatie van de code

Zorgt dat de code of het product aan standaarden voldoet

Voorkomen van fouten

Kortere ontwikkelingstijden

Minder kosten en tijd voor testen.

Vooruitgang in productiviteit.

Minder defecten in een later stadium

Is beperkt in mogelijkheden.

## Dynamisch testen

Het testen van het dynamisch gedrag van de code.

Wordt uitgevoerd na de compilatie

Reageert op de input van een gebruiker.

Voor het vinden en het oplossen van fouten.

Testen van de code in een run-time environment

Kwaliteit van de code verhogen

Vinden van bugs in de software

## Dynamisch testen

Ontdekken van defecten die niet ontdekt kunnen worden met statisch analyses

Verhogen van de kwaliteit van het product

Detecteren an security threats

Kan de statisch testresultaten verifieren.

Kost veel tijd, geld, niet gemakkelijk, moeilijk om te bepalen welke kwetsbaarheden te testen.