## Generelt

Test skulle gerne overholde følgende:

* Troværdig
  + Udviklere accepterer troværdige test
  + Troværdige tests er uden bugs og de tester det rigtige
* Vedligeholdelsesvenlig
  + Udviklere vil stoppe med at vedligeholde og fikse test hvis det tager for lang tid.
  + Dette medfører at man risikerer at miste test.
* Læsevenlig
  + Det er vigtig at kunne læse testen og at man kan finde ud af hvad problemet er hvis den fejler.
  + Uden læsevenlige test kan de andre to punkter ikke bruges, da det bliver svært at vedligeholde og stole på testen.

**Troværdige tests**

En test bør aldrig indeholde:

* Switch, of or else statements
* Foreact, for or while loops

Da det ofte gør at testen tester mere end en ting og gør testen mere kompleks. Det kan medføre:

* At testen er svær at læse og forstå
* Testen er svær at genskabe (fx hvis finde et tilfældigt tal og den pludselig fejler)
* Der er mere sandsyneligt at testen har fejl eller tester der forkerte
* Det er svære at navngive testen, fordi den gør flere ting

## Unit test

Ved brug af unit test kan udvikler forbedre kode kvaliteten men man får en bedre forståelse for funktionen.

Blev introduceret af Kent Beck in Smalltalk.

DEFINITION:

*A unit test is a piece of a code (usually a method) that invokes another piece of code and checks the correctness of some assumptions afterward. If the assumptions turn out to be wrong, the unit test has failed. A “unit” is a method or function.*

DEFINITION:

*SUT stands for system under test, and some people like to use CUT (class under test or code under test). When we test something, we refer to the thing we’re testing as the SUT.*

Der er ingen ide i at skrive en dårligt unit test, med mindre du lærer at skrive en god en af det. Det er spild af tid.

Egenskaber (properties) som en god unit test burde have:

* Den skal være automatiseret og skal kunne gentages
* Den skal være nem at implementere
* Når den er skrevet skal den forblive til fremtidig brug
* Alle skal være i stand til at kører den.
* Den skal kunne køres ved et tryk på en knap.
* Den skal kunne køres hurtigt.

DEFINITION:

*A unit test is an automated piece of code that invokes the method or class being tested and then checks some assumptions about the logical behavior of that method or class. A unit test is almost always written using a unit-testing framework. It can be written easily and runs quickly. It’s fully automated, trustworthy, readable, and maintainable.*

Unit test omfatter 3 hoved handlinger:

* *Arrange* objects, creating and setting them up as necessary
* *Act* on an object
* *Assert* the something is as expected

## Integrations test

DEFINITION – Good unit test:

*Integration testing means testing two or more dependent software modules as a group.*

## Stubs

DEFINITION

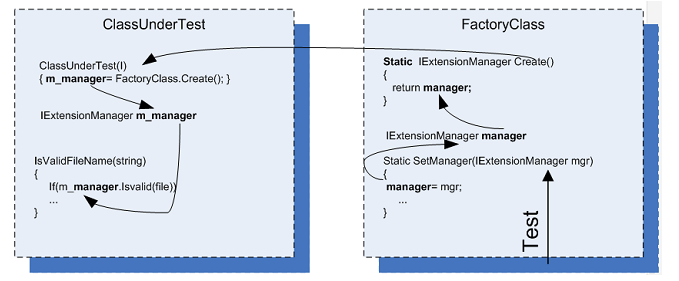
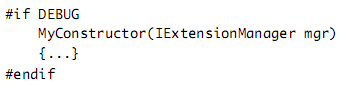
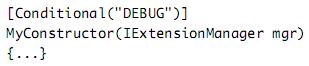
*A stub is a controllable replacement for an existing dependency (or collaborator) in the system. By using a stub, you can test your code without dealing with the dependency directly.*

Tekniker til at bryde afhængigheden til koden der skal testes:

* Extract an interface to allow replacing underlying implementation.
* Inject stub implementation into a class under test.
* Receive an interface at the constructor level.
* Receive an interface as a property get or set.
* Get a stub just before a method call.

Når der bruges stube laves der gerne en klasse om nedarver fra den afhængighed som skal brydes. Derved kan man overskrive de funktioner som der findes i den oprindelige klasse/interface og få det til at gøre som man ønsker det.

Der er 3 måder at indsætte vores stub I den oprindelige funktion:

1. Modtage interfacet i constructor’en og gemme det til senere brug (Constructor injection)
   1. Problemer kan forekomme hvis koden under test skal bruge flere stube for at virke uden afhængigheder. Ved at tilføje flere parametre til contructoren kan den blive sværere at læse og at holde.
   2. Der kan bruges container til at hjælpe med at contructoren bliver overskuelig. (HJÆLP)
2. Modtage interfacet som en property get eller set og gemme det til senere brug (dependency injection)
   1. Benyttes når:
      1. Use this technique when you want to signify that a dependency of the class under test is optional, or if the dependency has a default instance created that doesn’t create any problems during the test. (BETYDER HVAD)
      2. Ændre kode strukturen, og bør derfor prøves undladt, men mindre man har en godt grund *(This method would change the semantics of using the class under test, which is best avoided unless you have a good reason)*
3. Modtage interfacet lige før kaldet i metoden skal bruge det.
   1. Use a factory class 
      1. Det er dog ikke alle klasser som tilader at nogen uden for factory klasen ændre instancen som skal returneres, da den prøver at beskytte indkapslingen.
      2. Løsningen er at indsætte en static set metode, denne skal dog nulstilles efter hver test.
   2. For at skjule sprækkerne man bruger vil at indsætte stubs i releasen kan man skrive. 
   3. Eller 
      1. Denne gør dog at den stadig kan ses af andre programmøre og kan ikke bruges til contructors.
   4. Den kræver tid at sætte sig ind i men ændre ikke på kodestrukturen
   5. En anden måde at gemme sine funktioner på er ved at lave dem internal (HJÆLP)

## Mocks

DEFINITION:

*A mock object is a fake object in the system that decides whether the unit test has passed or failed. It does so by verifying whether the object under test interacted as expected with the fake object. There’s usually no more than one mock per test.*

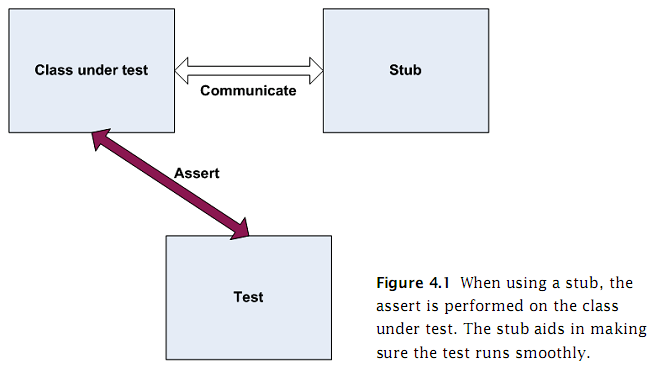
Mock er en smarterer udgave af stubs. Mock minder om stubs, på nær at den vil gøre lidt mere end stubs. Den vil gemme kommunikationshistorien som vil blive kontrolleret senere.

Der benyttes kun én mock per test! De andre falske objekter vil være stubs. Hvis der benyttes mere end en mock per test betyde det normalt at der bliver testet for mere end en ting i testen.

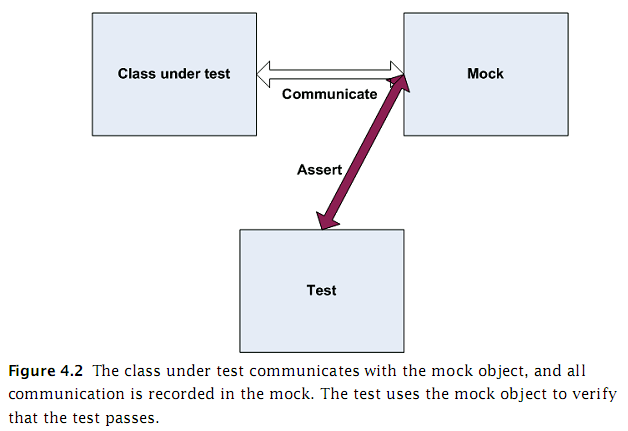
## Forskellen mellem mock og stub

Den største forskel er at stubs ikke kan fejle men det kan mocks.

Stubs erstatter et objekt således at vi kan teste et andet objekt uden problemer. Læg mærke til at Assert bliver brugt mellem Test og kode under test.



Mock objektet benyttes til at se om testen fejler eller ej. Læg mærke til at Assert nu bliver brugt mellem test og mock objektet.



DEFINITION:

*A fake is a generic term that can be used to describe either a stub or a mock object (handwritten or otherwise), because they both look like the real object. Whether a fake is a stub or a mock depends on how it’s used in the current test. If it’s used to check an interaction (asserted against), it’s a mock object. Otherwise, it’s a stub.*

Problemer med håndskrevende mocks og stubs:

* Det tager tid at skrive mocks og stubs
* Det er svært at skrive stubs og mocks til klasser og interfaces som har mange metoder, properties og events
* To save state for multiple calls of a mock method, you need to write a lot of boilerplate code to save the data.
* Hvis man ønsker at tjekke alle parameter på en kaldt metode, bliver man nødt til at skrive mange asserts. Hvis den første assert fejler, vil de andre aldrig bliver kørt, fordi den fejlede assert smider en exception
* Det er svært at genbruge mok og stub kode til andre test

## Dynamisk oprettelse af et falsk objekt

DEFINITION

*A dynamic fake object is any stub or mock that’s created at runtime without needing to use a handwritten implementation of an interface or subclass.*

**Strict mocks**

Strick mock objekter kan kun blive kaldt af metoder som er blevet sat via excpectations. Bliver metoderne ikke kaldt med de rigtigt parametre eller det rigtig navn vil der blive smidt en exception. Testen vil fejle på den første metode som ikke er forventet, hvorved den aldrig komme ned til verify.

Dette betyder at strict mock kan fejle på to måde, enten ved en ikke forventet metode kald, eller hvis en forventet metode ikke bliver kaldt.

**None strict mock – dynamisk**

Den dynamisk mock vil tillade at der bliver kaldt til en metode som ikke er forventet, så længe den ikke returnere en værdi. Hvis metoden skal returnere en værdi, vil den normalt returnere en default værdig (normalt 0 eller null). Nogen frameworks smider exceptions.

Non strict mocks kan kun fejle hvis en forventet metode ikke bliver kaldt.

**Stubs**

Verify vil aldrig kontrollere noget ved stubs.

Gode råd om hvordan man undgår at falde i det åbne huller når der benyttes dynamiske falske objekter kan ses på side 135/161.

## Automated build run automatied tests

Automatiseret bygning til at kører test med skal kunne:

## Get the latest version of all projects in question HJÆLP

* Kompilere alle projekterne I deres seneste version
* Give besked til relevante folk via e-mail hvis noget fejler

Måden at bygge på kan deles op i forskellige byg:

* Nightly build
  + Kører alle de langsomme test
  + Kører alle system test
* Release build
  + Kører nightly build
  + Deploys to server and archives HJÆLP
* CI(countinuous integration)build
  + Kører alle de hurtige test
  + Er færdig efter mindre en 10 min

Gennerelt er det en god ide at dele testene op i hurtige og langsomme test. Efter som interations test ofte tager længere tid en enhedstest vil disse ofte være de langsomme og enhedstest de hurtige.

Det er vigtigt at når der opdateres til den seneste version og koden at testene så følger med da koden har ændres sig, så vil testene også have ændret sig.

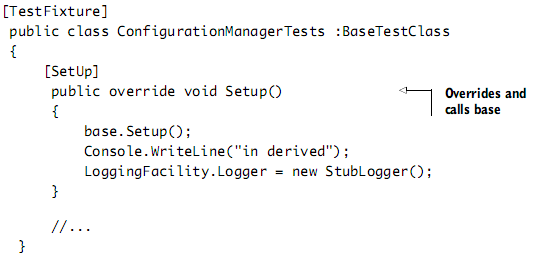
Det er vigtig at når man:

* Hurtigt kan finde alle de test som hører til projektet når man åbner det.
* Kan finde alle de test som hører til en klasse
* Kan finde alle de test som hører til en metode.

Bliver en test klasse for stor og uoverskuelig kan man med fordel trække en funktion ud og ligge den i sin egen klasse med et sigende navn. Resten af funktioner bliver i den oprindelige test klasse.

**Abstract test infrastucture class pattern**

Det er muligt at oprette en base test klasse om indeholder Setup og Teardown for flere klasser som skal bruge det samme. Test klasserne skal derved bare nedarve fra basis klassen og SetUp og TearDown vil blive kørt før hver test. Dette gør at mere kode bliver genbrugt, men desværre gør det testene mindre læselig. Basis test klassens Setup kan også laves virtuel således at den kan overskives af test klasserne.



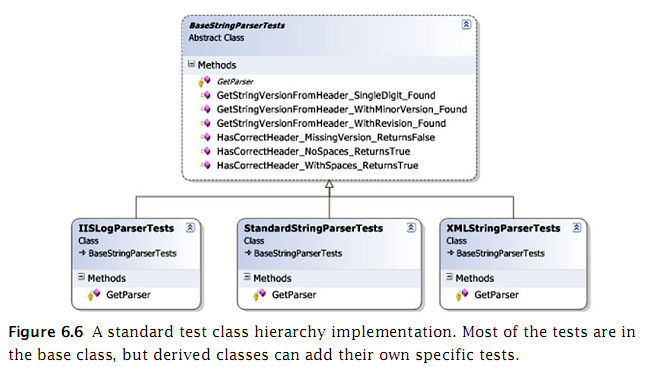
Dette hjælper til at gøre testen mere læselig, da man nu kan se at der er en SetUp. Dvs. at selvom man ikke er interesseret i at lave om i SetUp, kan det stadig være en god ide at overskrive den for at gøre det mere læseligt.

**Template test class pattern**

Her oprettes der en abstrakt klasse som indeholder det nødvendige for at kunne teste op imod et specifikt interface. Klassen indeholder alle de testmetoder som man skal igennem for at hele interfacet er testen. Alle klasse som skal teste mod dette interface kan så arve fra denne klasse og være sikre på at få testet det hele.

**Abstract test driver class pattern**

Her oprettes der en test klasse som tester alle de klasse om normalt nedarves fra den klasse som er undertest. Dvs at man laver testen således at de mapper over med kode under test, så både test og kode under test har en super klasse som der bliver ned arvet fra. Test klassen kan så sætte en attribut som indeholder lige netop den klasse oms der skal testes. (kap 6 side 164/190)



Man kan få vise værktøjer til at refakturer sin kode til at indeholde en af de 3 før nævnte class patterns.

## State-based testing

DEFINITION

*State-based testing (also called state verification) determines whether the exercised method worked correctly by examining the state of the system under test and its collaborators (dependencies) after the method is exercised.*

Kan også kaldes “result-driven testing”. Dette betyder at der teste at slut resultatet er sandt, fx at en property har ændret sig.

Benytter sig gerne af stubs.

## Interacting testing

DEFINITION:

*Interaction testing is testing how an object sends input to or receives input from other objects—how that object interacts with other objects.*

Kan også kaldes “action-driven testing”. Dette betyder at der testes for en specifik handling et objekt udfører, fx. At sende en besked til et andet objekt.

## Test-driven development

## Test navne

* Project
  + [ProjectUnderTest].Tests
* Class
  + [ClassNemw]Test
* Method
  + [MethodName]\_[StateUnderTest]\_[ExpectedBehavior]
    - MethodeName
      * The name of the method you’re testing
    - StateUnderTest
      * The conditions used to produce the expected behavior
    - ExpectedBehavior
      * What you expect the tested method to do under the specified conditions
  + Eksempel
    - IsValidFileName\_validFile\_ReturnTrue

## .Net Framwork Unit-testing

## Unit-testing frameworks are code libraries and modules that help developers unit-test their code, as outlined in table 2.1.

Over test klassen skrives der [TestFixture] således at NUnit ved hvilken klasse den skal teste.

Over hver metode skrive der [Test] således at NUnit ved hvilke metoder der skal testes. Metoderne skal være void og tager ingen parametre.

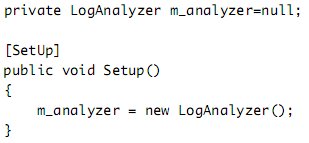
### Setup and Teardown

Funktion med [SetUp] over sig vil blive kørt hver gang en test bliver kørt.

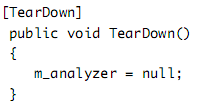
Funktion med [Teardown] over sig vil blive kørt hver gang en test er blevet kørt.

Skal kun bruges til at initiere/nedlægge objekter som er delt mellem metoderne.

SetUp kunne gøre følgende:



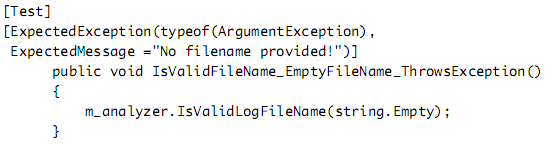
Teardowm kunne gøre følgende:



Findes også SetUp og Teardown til klasser de hedder [TestFixtureSetUp] og [TestFixtureTeraDown].

### Exception

For at teste for exception skal der skrives som følgende:

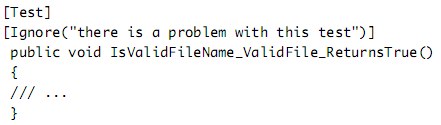


ArgumentException er typen af excepton og ExpectedMessage er den meddelelse man forventer kommer med exceptionen.

Der benyttes ingen assert i test metoden, da [ExpectedException] indeholder denne.

### Ignorere bestemte test

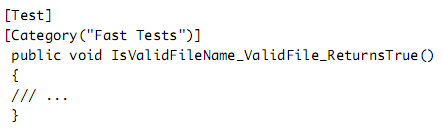
Hvis en test fejler om man er nødt til at comitte den kan man bede NUtint du ignorere således:



### Kategorisere test

**Virker ikke i ReShaper**

Test kan kategoriseres således at det bliver kørt samlet.



## Ord liste

* Accidental bugging
  + Når man kommer til at ødelægge andre dele af koden ved at tilføje eller rette andet kode og ikke køre en test bagefter.

## Definition

DEFINITION:

*Logical code is any piece of code that has some sort of logic in it, small as it may be. It’s logical code if it has one or more of the following: an IF statement, a loop, switch or case statements, calculations, or any other type of decision-making code.*

DEFINITION:

*An external dependency is an object in your system that your code under test interacts with, and over which you have no control. (Common examples are filesystems, threads, memory, time, and so on.)*

DEFINITION

*An isolation framework is a set of programmable APIs that make creating mock and stub objects uch easier. Isolation frameworks save the developer from the need to write repetitive code to test or simulateobject interactions.*

## Gode kode stupmer

* string testName = MethodBase.GetCurrentMethod().Name;
  + Benyttes til at finde navnet på den metode man er inden i