

# Lektionen 5-8



Bildquelle: <https://laptrinhx.com/topic/213/cac-dinh-nghia-va-thuat-ngu-trong-kiem-thu-phan-mem-phan-2>

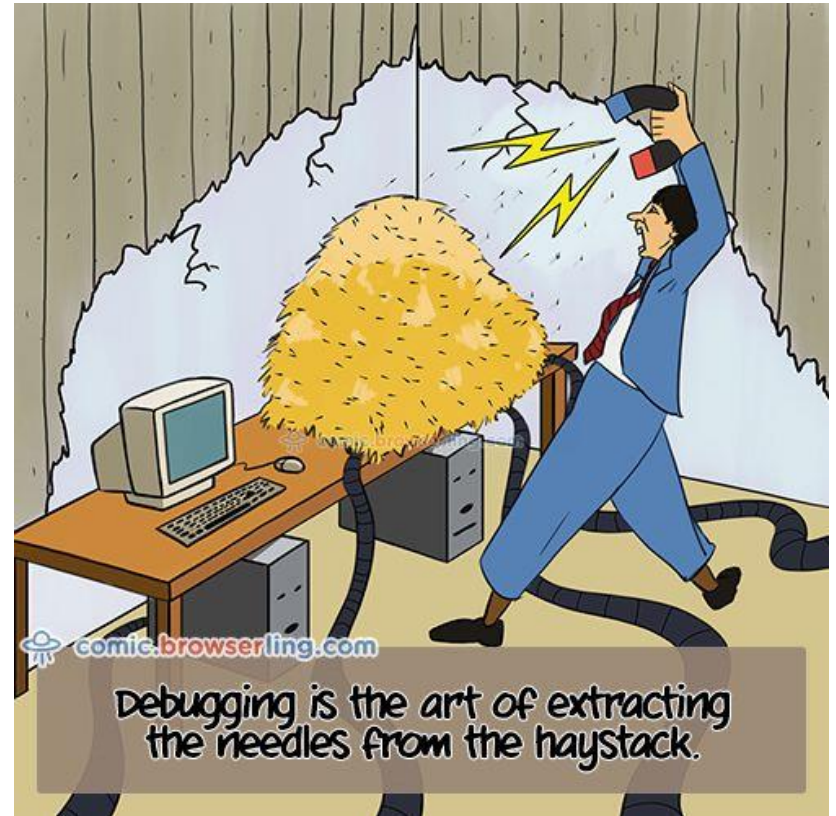
# Agenda

---

- Testbegriff
- Was ist Qualität?
- Qualitätsmerkmale
- Vorgehensmodelle
- Unit Testing Frameworks
- Grundlagen Unit Tests
- My first Unit Test with NUnit

# Testbegriff

- Softwaretesting
  - Lokalisierung von Defekten in der Software
- Debugging
  - Fehlerbereinigung
  - Fehlerkorrektur
- Weitere Ziele des Testens?



# Was ist Qualität?

---



Bildquelle: <http://tip-web.de/de/about/qualitaet>

# Testen und Qualität


---

- Durch das Testen werden Defekte identifiziert, deren Behebung zur Steigerung der Softwarequalität führt.

# Qualitätsmerkmale nach ISO9126 (1)

---

- Funktionalität
- Zuverlässigkeit
- Benutzbarkeit
- Effizienz
- Änderbarkeit
- Übertragbarkeit



Funktionale Q-Merkmale  
Was macht die SW?

Nicht-Funktionale Q-Merkmale  
Wie macht es die SW?

# Aufgabe

---

Lesen Sie den Text zu dem Q-Merkmal, welches Ihnen zugewiesen wurde. Besprechen Sie den Text mit Ihrer Gruppe.

Erstellen Sie eine Kurzpräsentation (max. 3 Min.) und nennen Sie dabei drei Beispiele aus der realen Welt. Jemand aus der Gruppe hält den Vortrag.

**Arbeitsform:** 6 Gruppen

**Zeit:** 20 Minuten

**Besprechung / Feedback:** In der Klasse

# Qualitätsmerkmale nach ISO9126 (2)

---

- Ein Softwaresystem kann nicht alle Qualitätsmerkmale gleich gut erfüllen
- Die Priorisierung der Qualitätsmerkmale dient nicht nur für die Entwicklung, sondern auch für den Test, zur Bestimmung der Intensität der Überprüfung der einzelnen Qualitätsmerkmale

Wurde durch ISO/IEC 25000 abgelöst



# Qualitätsmerkmale nach ISO9126 (3)

---

Apropos «Priorisierung»...

# Übung «Priorisierung der Q-Merkmale»

---

Nach welchen drei Q-Merkmalen wurde die Ihnen zugewiesene Kaffeemaschine entwickelt? Sortieren Sie diese nach Priorität!

Bilden Sie vier Gruppen.

**Arbeitsform:** Gruppenarbeit

**Zeit:** 15 Minuten

**Besprechung / Feedback:** In der Klasse

# Übung «Priorisierung der Q-Merkmale»

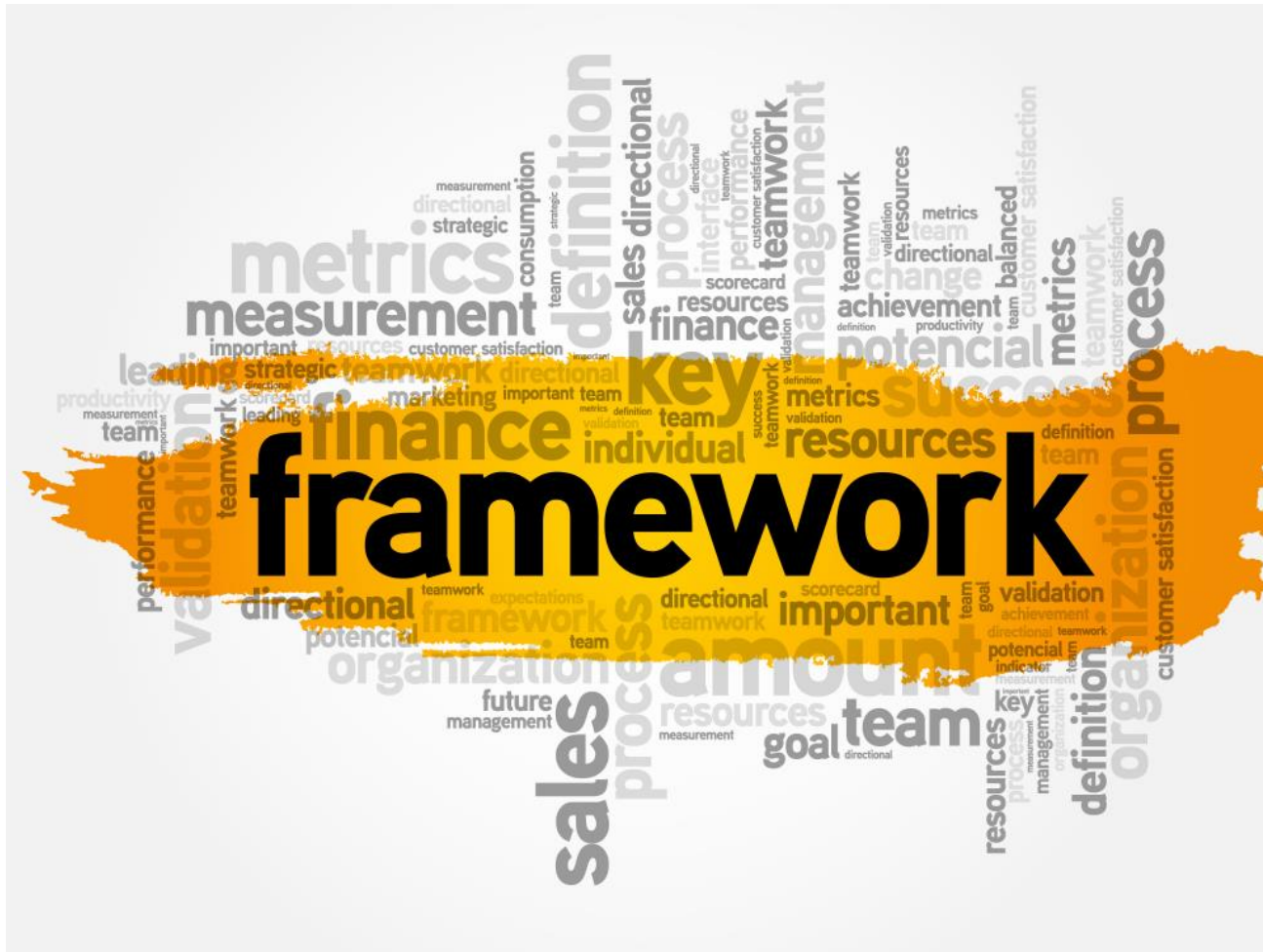


senn  
Kaffee



Bildquelle:  
<https://yellow.local.ch/de/d/Fislibach/5442/Kaffeemaschinen/Senn-Kaffee-AG-lqhJL1j3-PVn-JS2arNnw>  
<https://www.kaffeetech.ch/>  
<https://www.amici.ch/de/x1-mie/516-x1-anniversary-inox.html>  
<http://haus.onlinemarket24.ch/produkt/nespresso-kaffeemaschine-koenig/>

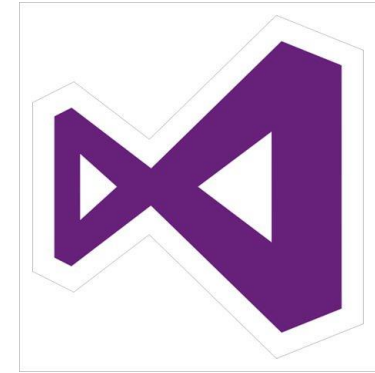
# Unit Testing Frameworks



Bildquelle: <https://steemit.com/howto/@danielwong/all-about-the-frameworks>

# Testing Frameworks für .NET (Auswahl)

---



# Grundlagen – Unit Tests (1)

---



Bildquelle: <https://simpleprogrammer.com/tdd-unit-testing/>

# Grundlagen – Unit Tests (2)

---

- Begriff «Unit Test» erstmals in Kombination mit Smalltalk in den 1970er Jahre genannt
- Code-Qualität verbessern
- Tieferes Verständnis für funktionale Anforderungen von Klassen und Methoden

# Definition – Unit Tests

---

Nach Roy Osherove (Aussprache «Oschrov»)

«Ein Unit Test ist ein automatisiertes Stück Code, das eine zu testende Unit of Work aufruft und dann einige Annahmen über ein einzelnes Endresultat dieser Unit prüft. Ein Unit Test wird fast immer mithilfe eines Unit-Testing-Frameworks erstellt. Er kann einfach geschrieben und schnell ausgeführt werden. Er ist vertrauenswürdig<sup>1</sup>, lesbar und wartbar. Seine Ergebnisse sind konsistent, solange der Produktionscode nicht geändert wird.»

<sup>1</sup> vertrauenswürdig/zuverlässig (aus dem Englischen «trustworthy»)

Quelle: «The Art of Unit Testing, S. 26, 2. Auflage 2015, mitp Verlags GmbH & Co. KG»



# Definition – Unit of Work

---

«Eine Unit of Work ist die Summe aller Aktionen, die zwischen dem Aufruf einer öffentlichen Methode innerhalb eines Systems und einem erkennbaren Endresultat beim Test dieses Systems stattfinden.

---

Quelle: «The Art of Unit Testing, S. 26, 2. Auflage 2015, mitp Verlags GmbH & Co. KG»

# Definition – SUT/CUT

---

«SUT steht für «System Under Test», manche verwenden auch den Begriff CUT («Class Under Test» oder «Code Under Test»). Wenn Sie etwas testen, bezeichnen Sie das, was Sie testen, als das SUT.»

Wir am ZbW verwenden in allen Beispielen SUT

---

Quelle: «The Art of Unit Testing, S. 26, 2. Auflage 2015, mitp Verlags GmbH & Co. KG»

# Grundsatz für das Schreiben von Unit Tests

---

- Es werden nur «gute» Unit Tests geschrieben.  
Schlechte Unit Tests sind Zeitverschwendung!!!

# Eigenschaften eines «guten» Unit Tests

---

Ein Unit Test *sollte* die folgenden Eigenschaften haben:

- Er sollte automatisiert und wiederholbar sein.
- Er sollte einfach zu implementieren sein.
- Einmal geschrieben, sollte er auch morgen noch relevant sein.
- Jeder sollte in der Lage sein, den Test auf einen Knopfdruck hin laufen zu lassen.
- Er sollte schnell ablaufen.
- Die Resultate sollten konsistent sein (die Tests geben immer den gleichen Wert zurück, wenn zwischen den Testläufen nichts geändert wird).
- Er sollte die volle Kontrolle über die getestete Unit haben.
- Er sollte komplett isoliert sein (er läuft völlig unabhängig von anderen Tests).
- Wenn er fehlschlägt, sollte es einfach zu erkennen sein, was erwartet wurde und wie das Problem lokalisiert werden kann.

Quelle: «The Art of Unit Testing, S. 28, 2. Auflage 2015, mitp Verlags GmbH & Co. KG»

# Integrationstests

---

- Sind nicht schnell
- Sind nicht konsistent
- Beinhalten eine oder mehrere Abhängigkeiten der zu testenden Unit.

«Wenn also beispielsweise der Test die reale Systemzeit, das reale Dateisystem oder die reale Datenbank verwendet, dann hat er sich schon auf das Gebiet des Integration Testing begeben.»

«Integrationstests erhöhen das Risiko für ein anderes Problem: das Testen zu vieler Dinge auf einmal.»

Quelle: «The Art of Unit Testing, S. 29, 2. Auflage 2015, mitp Verlags GmbH & Co. KG»

# Was muss nicht getestet werden

---

Gemäss Roy Oshero:

«Properties (Getters/Setters in Java) sind gute Beispiele für Code, der gewöhnlich keine Logik enthält und somit nicht eigens von den Tests adressiert werden muss. Es handelt sich um Code, der wahrscheinlich von einer Unit of Work, die Sie testen, verwendet wird. Es besteht daher keine Notwendigkeit, ihn direkt zu testen. Aber seien Sie auf der Hut: Sobald Sie irgendeine Überprüfung in eine Property einbauen, sollten Sie sicherstellen, dass die Logik getestet wird.»

---

Quelle: «The Art of Unit Testing, S. 34, 2. Auflage 2015, mitp Verlags GmbH & Co. KG»

# Hands on | nUnit (Live-Coding)



Bildquelle: <https://think360studio.com/9-best-online-websites-to-learn-coding-easily/>

# Hands on | NUnit (Nachcoden)

---

1. Leere Solution einrichten  
*(ZbW.Testing.FirstUnitTest)*
2. Zwei Class-Libraries (.NET-Framework) erstellen  
*(ZbW.Testing.FirstUnitTest.Library / ZbW.Testing.FirstUnitTest.Library.UnitTests)*
3. FunctionRepository-Klasse erstellen  
*public bool GreaterThan10(int value)*
4. Nuget Package zu \*.UnitTests-Projekt hinzufügen (NUnit)
5. Testklasse schreiben  
*[TestFixture] / [Test]*  
*Sowie Assert.IsTrue(result)*

**Arbeitsform:** Einzelarbeit

**Zeit:** 20 Minuten

**Besprechung / Feedback:** In der Klasse

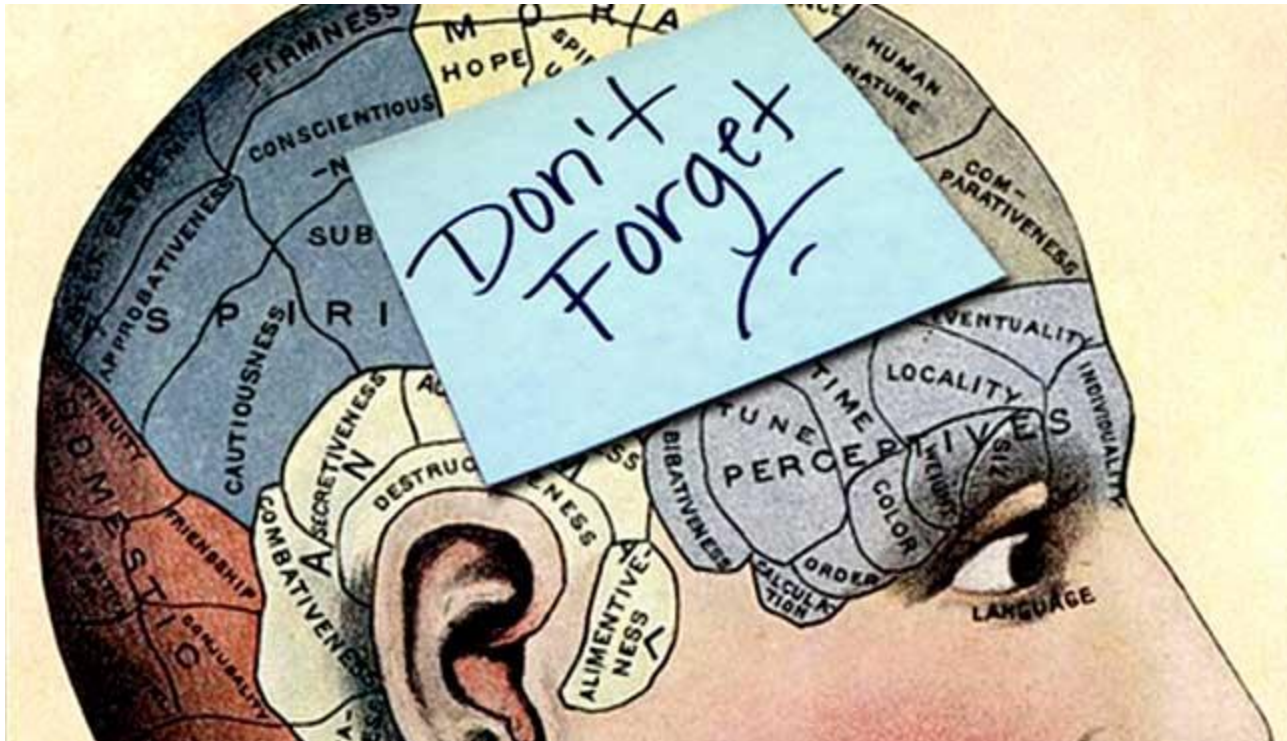


# Übung

---

- Erstellen Sie **zwanzig** Methoden, welche Sie anschliessend testen.
  - IsValidEMailadress(string) : bool
  - IsStringLongerThan10(string) : bool
  - GetStringLength(string) : int
  - AreNumbersInString(string) : bool
  - ToUpperString(string) : string
  - IsIntValid(string) : bool
  - Usw.

# Was nehmen wir vom Unterricht mit?



Bildquelle: <http://floraremedia.com/keep-mind-memory-sharp/>

# Hausaufgaben

---

- Machen Sie die Übung mit den 20 UnitTests fertig
- nächstes Mal ca. 200 Zeilen Code für eine Code Review mitbringen

# Ergänzende Unterlagen

---

Code Beispiele des heutigen Unterrichts unter:

<https://github.com/michikeiser/ZbW.Testing.FirstUnitTest>