Software-Testing

Lektionen 13-16

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

Agenda

- Teststufen
- Standortbestimmung
- Retrospektive
- Code Review
- Test Coverage mit dotCover

Teststufen - Übersicht

- Der **Komponententest** prüft, ob jeder einzelne Softwarebaustein für sich die Vorgaben seiner Spezifikation erfüllt. (S. 44 – 51)
- Der **Integrationstest** prüft, ob Gruppen von Komponenten wie im technischen Systementwurf vorgesehen zusammenspielen. (S. 52 – 60)
- Der **Systemtest** prüft, ob das System als Ganzes die spezifizierten Anforderungen erfüllt. (S. 60-63)
- Der **Abnahmetest** prüft, ob das System aus Kundensicht die vertraglich vereinbarten Leistungsmerkmale aufweist. (S. 64-67)

Teststufen - Aufgabe

- Jede Gruppe bereitet eine der Teststufen vor und präsentiert diese **ausführlich** in einer Präsentation (nutzt Whiteboard, Flipchart, Slides)

- **Systemtest und Abnahmetest werden von einer Gruppe vorbereitet**

Arbeitsform: Gruppenarbeit (drei Gruppen)
Zeit (Vorbereitung): 45 Minuten
Zeit (Präsentation): max. 5 Minuten
Besprechung / Feedback: In der Klasse

P A U S E

Standortbestimmung

- Zeit: 90min
- Hilfsmittel:
  - Voraussichtlich: Spick - A4 Vorderseite hangeschrieben
  - Keine alten Prüfungen
- Befinden sich in der Datenablage
- Fragerunde beim nächsten Mal

Retrospektive

- Nehmt euch 5 Minuten Zeit und bewertet den Kurs, dessen Inhalt und den Dozent.
- Einteilung in MAD / SAD / GLAD
- Vorstellung in der Klasse

Arbeitsform: Einzelarbeit
Zeit (Vorbereitung): 5 Minuten
Besprechung / Feedback: In der Klasse

Review - Einführung

- Oberbegriff für statische Prüfverfahren die von Personen (nicht Maschinen) durchgeführt werden. → **Strukturierte Gruppenprüfung**
- Review als Mittel zur Qualitätssicherung von Dokumenten

- **Ziel: Unstimmigkeiten sowie Fehler feststellen**

Review – Positive Auswirkungen

- Kostengünstige Fehlerbeseitigung (möglichst früh durchführen)
- Entwicklungszeiträume werden dadurch verkürzt
- Kostenreduzierung, da Fehler früh erkannt werden
- Wissensaustausch unter den Personen
- Gesamtes Team fühlt sich verantwortlich

Review - Probleme

- Persönliche Kritik vs. Kritik am Dokument

Review – Vorgehen IEEE 1028 (1)

- Planung
  - Welche Dokumente / Was / Von wem?
- Einführung
  - Ziele des Reviews sowie benötige Informationen kommunizieren, Testorakel
- Vorbereitung
  - Individuelle Vorbereitung ist der Schlüssel zum Erfolg

Review – Vorgehen IEEE 1028 (2)

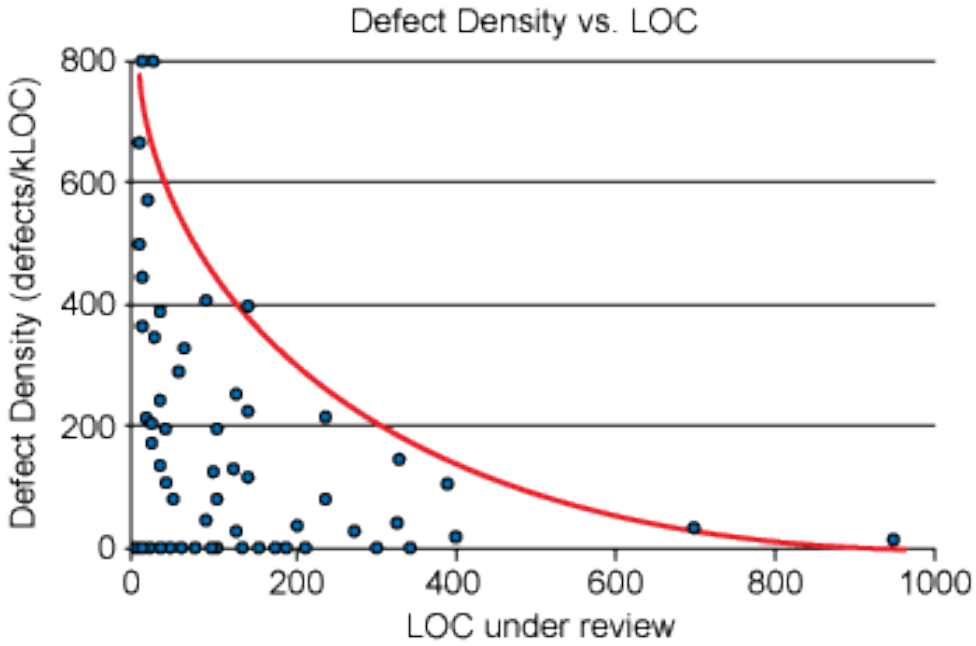
- Reviewsitzung
  - Durch Sitzungsleiter (Moderator) geführt
  - Nicht der Author sondern das Dokument wird geprüft
  - Es werden keine Lösungen entwickelt
  - Ziel: Dokument wird unverändert akzeptiert, verbessert oder muss neu geschrieben werden

Review – Vorgehen IEEE 1028 (3)

- Überarbeitung
  - Author überarbeitet das Dokument auf Basis der Reviewergebnisse
- Nachbereitung
  - Überarbeitung wird durch eine andere Person kontrolliert.
  - Verbesserungen im Entwicklungsprozess umsetzen

Code Review (1)

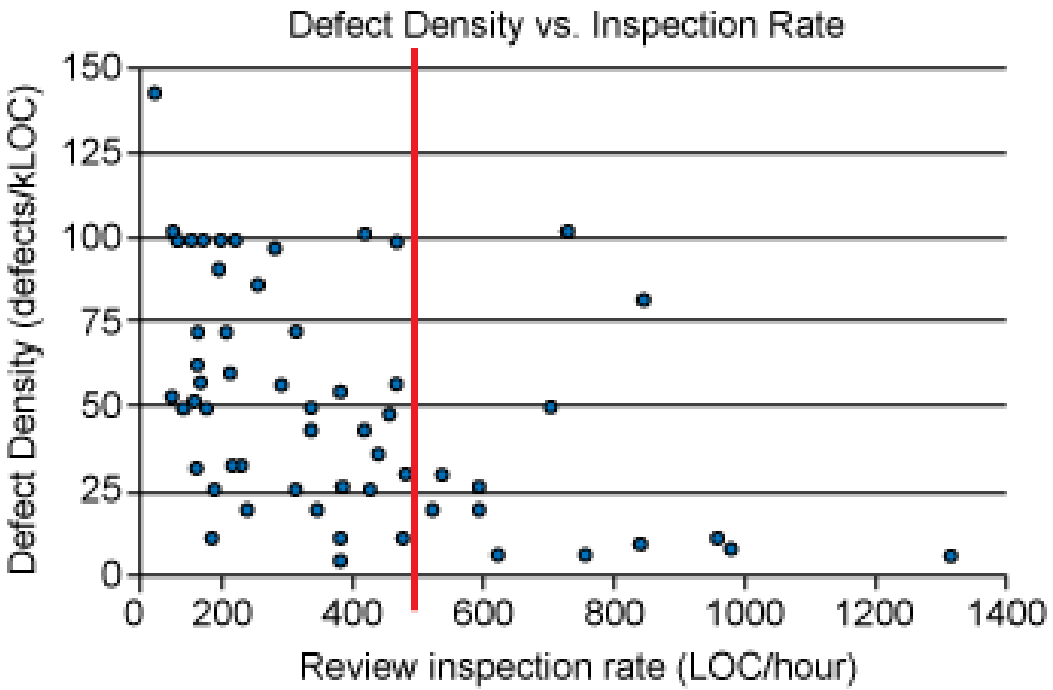
- Maximal 200 – 400 Codezeilen (LOC) pro Review bearbeiten.



Quelle: https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/

Code Review (2)

- Inspection rates should under 500 LOC per hour



Quelle: https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/

Code Review (3)

- Eine Code Review sollte nicht länger als 60 Minuten dauern
- Benutze Checklisten für immer wiederkehrende Fehlerhandlungen.

Quelle: https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/

Code-Review-Arten

- Ad-Hoc-Review
  - Spontane Peer Code Review (Nur der Zeitpunkt ist geplant)
- Code Review in Meetings
  - Geplantes Peer Code Review (Umfang, Zeitpunkt, Testobjekt ist definiert)

Peer Code Review - Ablauf

- Wöchentlich z.B. 30 Minuten
- Maximal 1 – 2 Reviewer
- Zu Beginn der Review wird eine Checkliste erstellt, welche Teile der Software Reviewt werden sollen.
- Werden Probleme während einer Review identifiziert, sollen gemeinsam Lösungen besprochen werden. Deren Umsetzung kann in der nächsten Review bewertet werden.

Review - Kommunikation

«Der Kommunikationsstil eines Reviewers spielt ebenfalls eine große Rolle für die Beurteilung einer Review durch die befragten Mitglieder der Mozilla Developer Community. So wünschen sie sich, dass ein Reviewer grundsätzlich **freundlich, zugewandt und unterstützend** auftritt. Er muss dabei jedoch auch dazu in der Lage sein, **Probleme deutlich zu kommunizieren und sich durchzusetzen**. Der Tonfall ist dabei besonders wichtig: Kommentare sollten konstruktiver Natur sein und so vermittelt werden, dass der Autor des Codes sie nicht persönlich nimmt. Das ultimative Kriterium für eine gute Code-Review ist für die Mozilla-Developer aber natürlich die Code-Qualität. Sie soll durch die Review in einem Maße steigen, das der Entwickler des Codes alleine nicht erreichen könnte.»

Quelle: https://entwickler.de/online/development/code-review-best-practice-295369.html

Übung: Peer Code-Review durchführen

Bilden Sie zweier Gruppen und bestimmen Sie Autor und Reviewer. Der Autor wählt einen Code aus, welcher er in den letzten Tagen/Wochen geschrieben hat. Auf Basis dieses Quelltextes führen Sie ein Peer Code Review durch. Wechseln Sie nach ca. 20 Minuten die Rollen.

Dokumentieren Sie die Ergebnisse und stellen Sie Ihre Erfahrungen in der anschliessenden Feedbackrunde in der Klasse vor.

Arbeitsform: Zweier- / Dreiergruppen
Zeit (Vorbereitung): 40 Minuten
Besprechung / Feedback: In der Klasse

Let’s code!

Bildquelle: http://www.loslegen.net/los-gehts/

Weitere Attribute

- Aufbau und Abbau

```
[OneTimeSetUp]
0 references | 0 changes | 0 authors, 0 changes
public void OneTimeSetUp()
{
}

[OneTimeTearDown]
0 references | 0 changes | 0 authors, 0 changes
public void OneTimeTearDown()
{
}
```

```
[SetUp]
0 references | 0 changes | 0 authors, 0 changes
public void SetUp()
{
}

[TearDown]
0 references | 0 changes | 0 authors, 0 changes
public void TearDown()
{
}
```

Weitere Attribute

- **[OneTimeSetUp]**
  - Methode wird ***einmal vor*** dem Testen der Klasse ausgeführt
- **[OneTimeTearDown]**
  - Methode wird ***einmal nach*** dem Testen der Klasse ausgeführt
- **[SetUp]**
  - Methode wird ***vor jedem*** Test ausgeführt
- **[TearDown]**
  - Methode wird ***nach jedem*** Test ausgeführt

Weitere Attribute

- Kategorisieren

```
[Category("LongRunningUnitTest")]
0 references | 0 changes | 0 authors, 0 changes
public void ThisIsALongRunningUnitTest()
{
}

[Category("ShortRunningUnitTest")]
0 references | 0 changes | 0 authors, 0 changes
public void ThisIsAShortRunningUnitTest()
{
}
```

Weitere Attribute

- Ignorieren

```
[Ignore("Dieser Test ist sinnlos, daher wird er ignoriert")]
0 references | 0 changes | 0 authors, 0 changes
public void IgnoreThisTest()
{
}
```

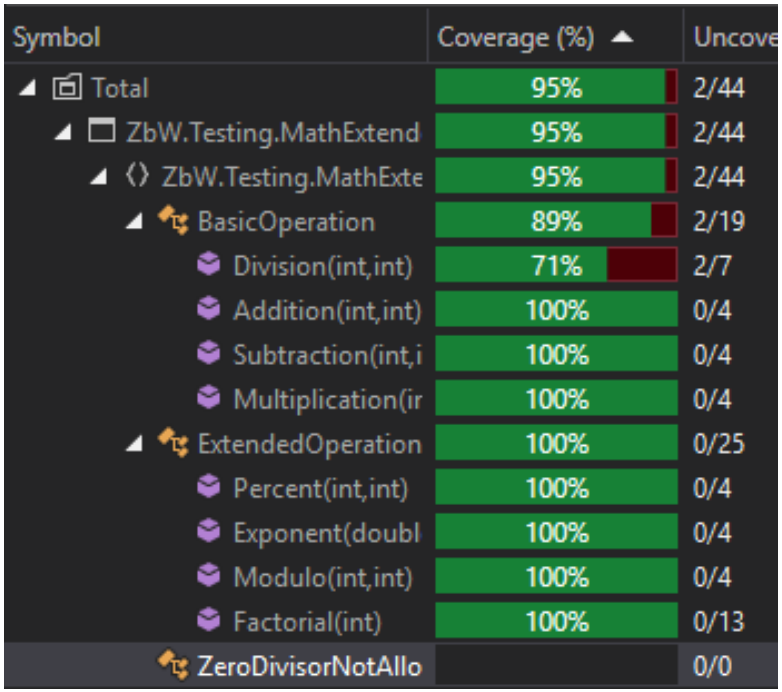
Übung: Weitere Attribute

Wenden Sie sämtliche Attribute die Sie kennen gelernt haben an.

- **[OneTimeSetUp]**
- **[OneTimeTearDown]**
- **[SetUp]**
- **[TearDown]**
- **[Categorie]**
- **[Ignore]**

Arbeitsform: Einzelarbeit
Zeit (Vorbereitung): 15 Minuten
Besprechung / Feedback: In der Klasse

Testabdeckung ermitteln – Live Coding



Symbol	Coverage (%)	Uncove
Total	95%	2/44
ZbW.Testing.MathExtend	95%	2/44
ZbW.Testing.MathExte	95%	2/44
BasicOperation	89%	2/19
Division(int,int)	71%	2/7
Addition(int,int)	100%	0/4
Subtraction(int,i	100%	0/4
Multiplication(ir	100%	0/4
ExtendedOperation	100%	0/25
Percent(int,int)	100%	0/4
Exponent(doubl	100%	0/4
Modulo(int,int)	100%	0/4
Factorial(int)	100%	0/13
ZeroDivisorNotAllo		0/0

```
1 reference | Michael Keiser, 21 days ago | 1 author, 1 change
public double Division(int dividend, int divisor)
{
    if (divisor == 0)
    {
        throw new ZeroDivisorNotAllowedException();
    }

    var quotient = (double)dividend / divisor;
    return quotient;
}
```

Testabdeckung ermitteln – Live Coding

Beispiel-Code unter
https://github.com/michikeiser/ZbW.Testing.MathExtended

Prinzip

- Alle studierenden schauen zuerst zu
- Nach der Vorführung erstellen sie selbstständig eine Test Coverage

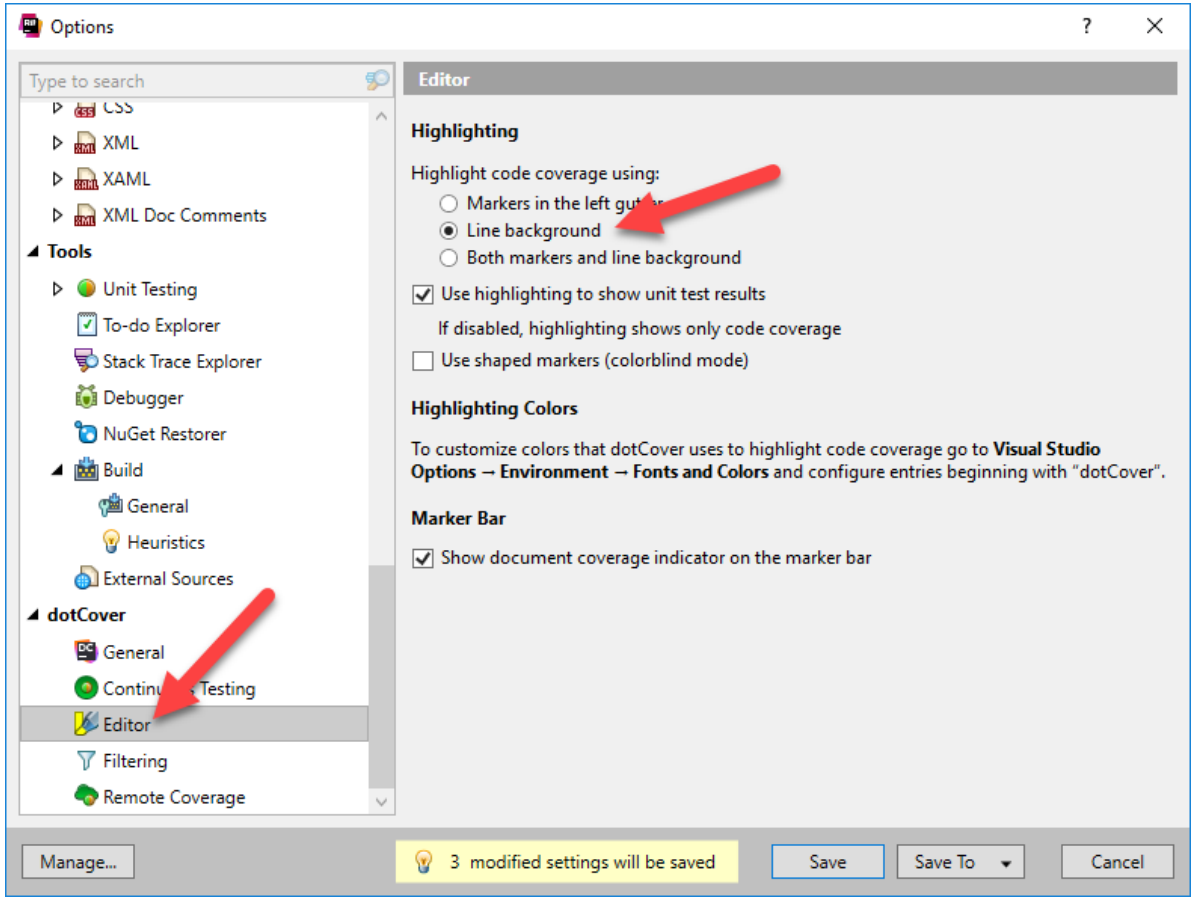
Test Coverage berechnen

- ReSharper > Unit Tests > Cover All Tests from Solution
- Fenster «Unit Test Sessions» und «Unit Test Coverage» erklären
- «DivisionByZero_Calculate_ReturnsException» auskommentieren um zu zeigen, wie es aussieht, wenn eine Linie nicht abgedeckt ist.

Anschliessend die Einstellungen vornehmen und die Test Coverage ermitteln

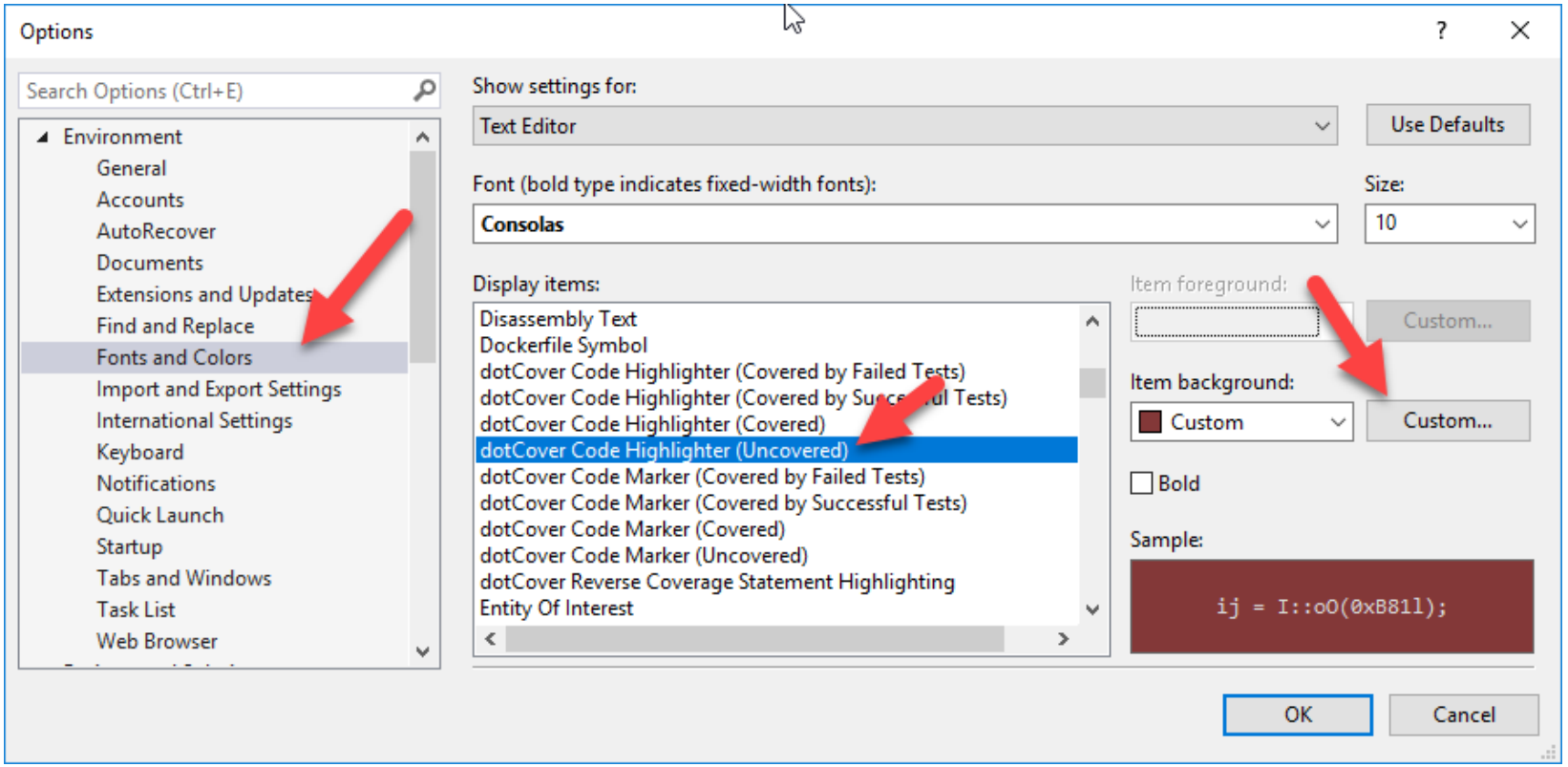
dotCover – Einstellungen (1)

- ReSharper > Options…

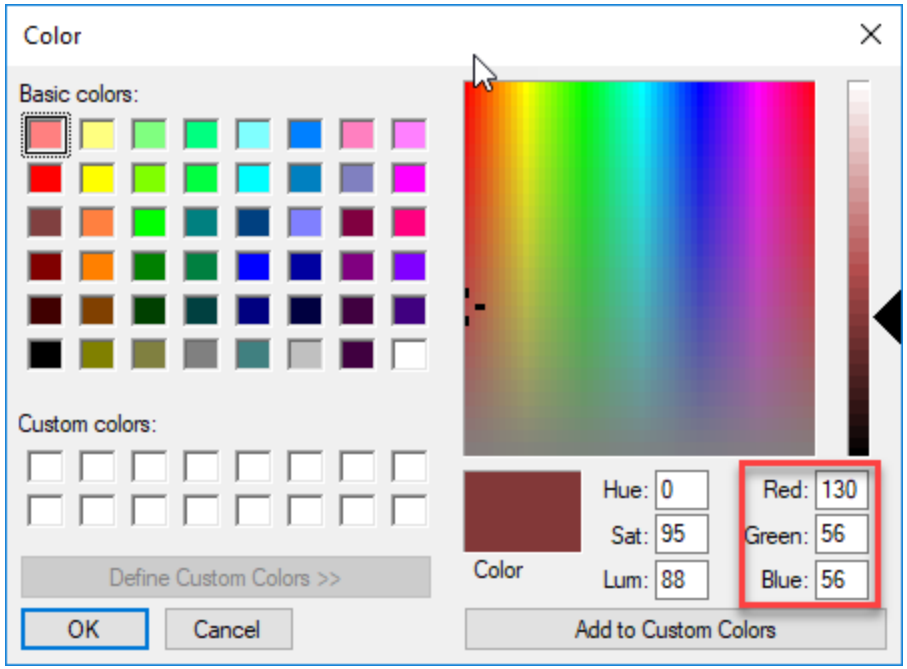


dotCover – Einstellungen (2)

- Tools > Options



dotCover – Einstellungen (2)



Hands on «Test Coverage»

- Beispiel-Code unter
  - https://github.com/michikeiser/ZbW.Testing.MathExtended
- Test Coverage berechnen
  - ReSharper > Unit Tests > Cover All Tests from Solution
  - «DivisionByZero_Calculate_ReturnsException» auskommentieren

Continuous Testing

Idee:

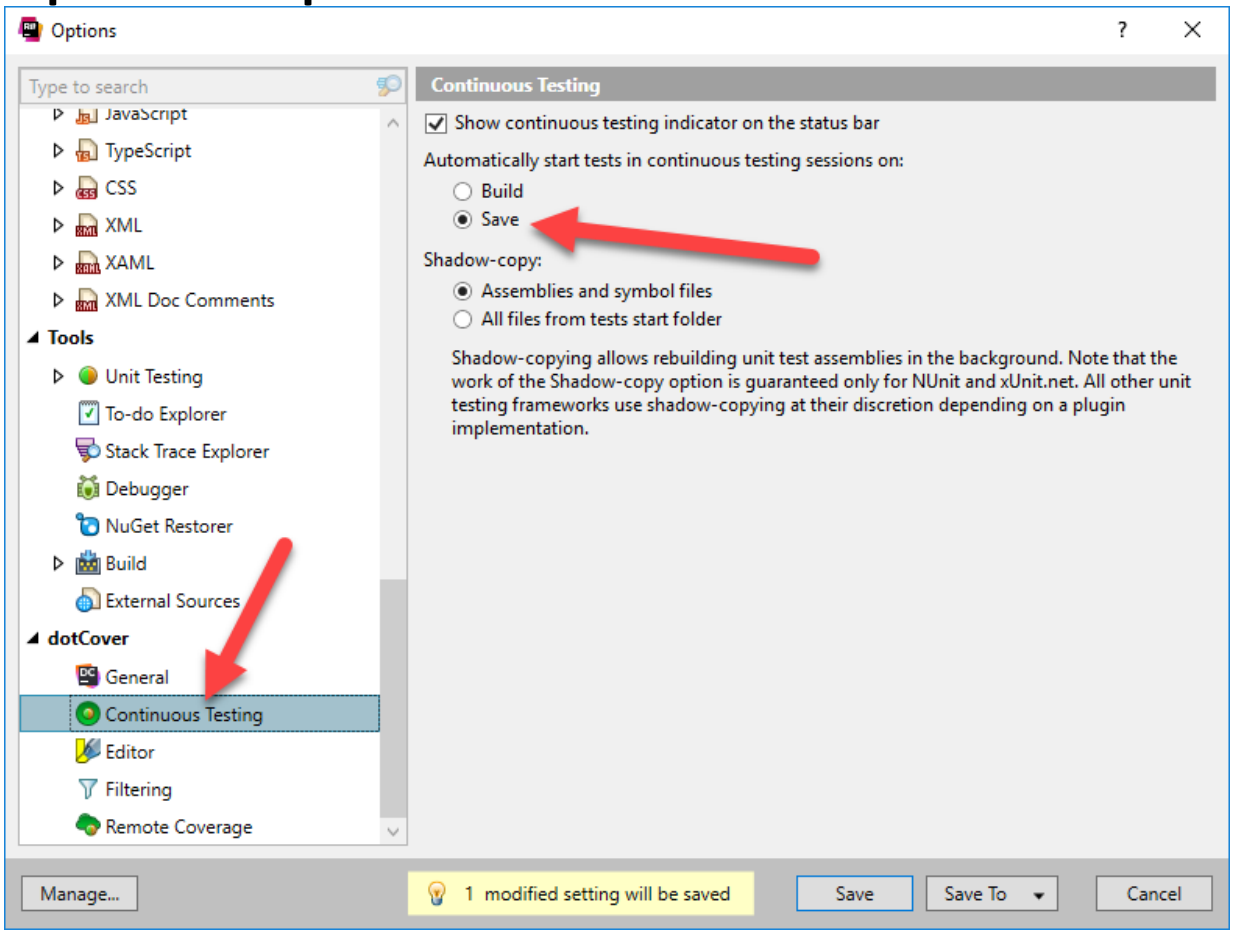
- Sobald gespeichert wird, soll das System alle relevanten Tests ausführen, um mir möglichst schnell ein Feedback über den Zustand der Software zu liefern.

Lösung:

- Continous Testing

dotCover – Einstellungen (2)

- ReSharper > Options…



Weitere Attribute

- Methode für die Berechnung der Test Coverage NICHT berücksichtigen

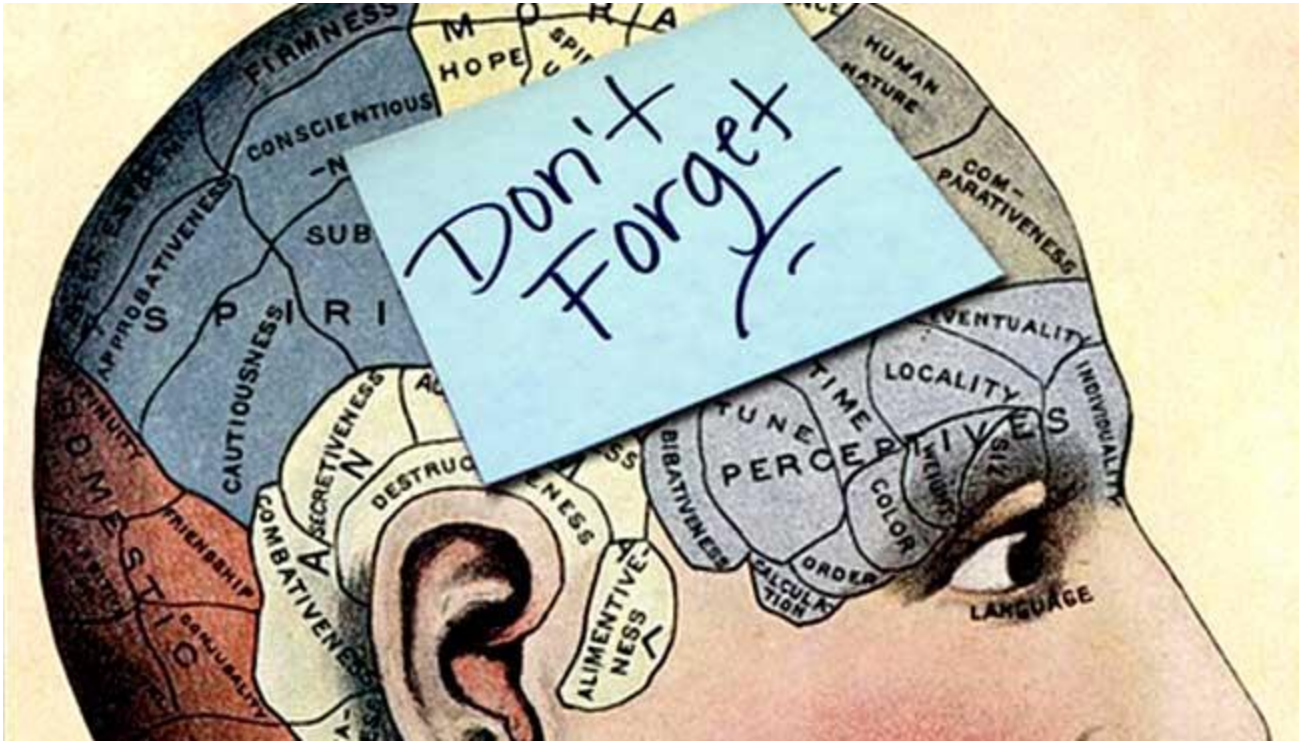
```
[ExcludeFromCodeCoverage]
0 references | 0 changes | 0 authors, 0 changes
public void DoSomeThing()
{
}
```

Hands on «dotCover»

- Probieren Sie die eben gelernten Features von dotCover aus.

Arbeitsform: Einzelarbeit
Zeit (Vorbereitung): 20 Minuten
Besprechung / Feedback: In der Klasse

Was nehmen wir vom Unterricht mit?



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

Hausaufgaben

- Unterreicht nachbearbeiten
- Kapitel 3 und 4 lesen