

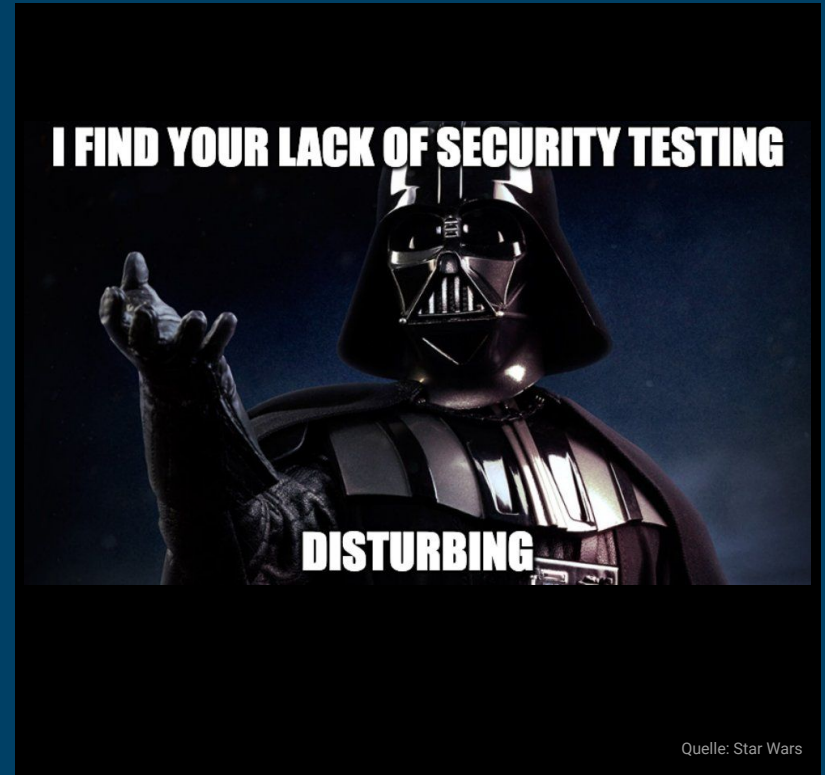
# Security Testing For Coders

Eine Verfolgungsjagd  
auf die CIA

[jasie.de/security-testing](https://jasie.de/security-testing)

[@jasie@machteburch.social](https://twitter.com/jasie@machteburch.social)

[speakerinnen.org/de/profiles/jana-sieber](https://speakerinnen.org/de/profiles/jana-sieber)



# (abstract)

---

In der wilden Welt des Internets, wo Bugs und Sicherheitslücken lauern wie hungrige Raubtiere in der Savanne, braucht es mutige Entwickler, die es wagen, den Kampf aufzunehmen. Dieser Vortrag ist für die Code-Krieger, die bereit sind, sich der Herausforderung zu stellen und ihre Code-Sicherheit auf die nächste Stufe zu heben.

Mein Ziel: euch mit dem Wissen auszustatten, um automatisierte Tests zu schreiben, die die dunklen Ecken eures Codes beleuchten und die fiesen Bugs aufspüren, die dort lauern.

Wenn ihr wollt, dass euer Code nicht nur funktional einwandfrei, sondern auch sicher(er) ist, dann kommt mit mir auf eine Verfolgungsjagd nach der CIA äh den CIA ;-)

—  
Vor  
langer,  
langer  
Zeit...

Quelle:  
[thepeakmagazine.com.sg](http://thepeakmagazine.com.sg)



# Über mich



@jasie@machteburch.social • jasie.de •



# Gliederung

---

- I. Motivation
- II. Hintergründliches
- III. Security Test Cases
- IV. Und außerdem



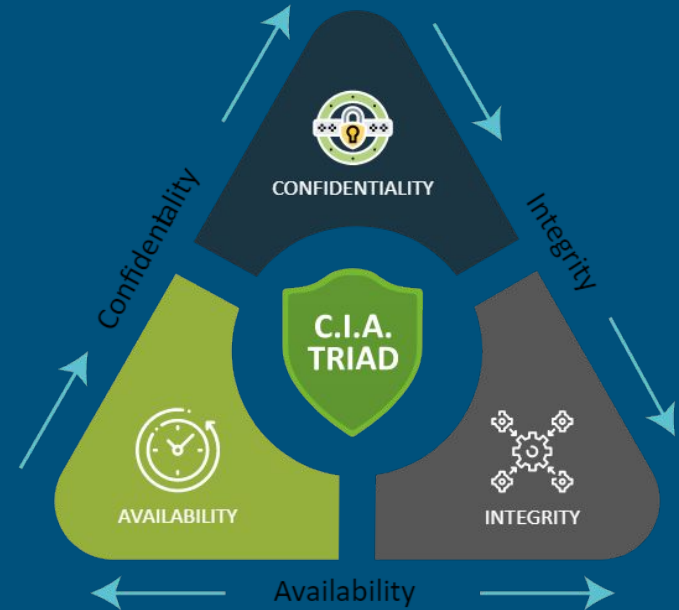
You are a lucky bug. I'm seeing that you'll be shipped with the next five releases.

# I) Motivation

---

# Softwaresicherheit

- SW-Sicherheit = Zustand
- IT Schutzziele: **CIA**  
Vertraulichkeit – Integrität – Verfügbarkeit  
(Datendiebstahl - Defacement - (D)DOS)
- **Böswilligkeit vs Unabsichtlichkeit**  
(Hacker vs User ☐)



# Berühmte Software-Katastrophen

- Vertraulichkeit
  - Facebook Business, 2015
  - Vergabe von Admin-Berechtigungen ohne Access Control
- Integrität
  - Mariner 1 Crash, 1962
  - Formel von Programmierer nicht korrekt übertragen
- Verfügbarkeit
  - Ausweisnotstand in UK, 1999
  - System überlastet





# Wozu die Mühe...

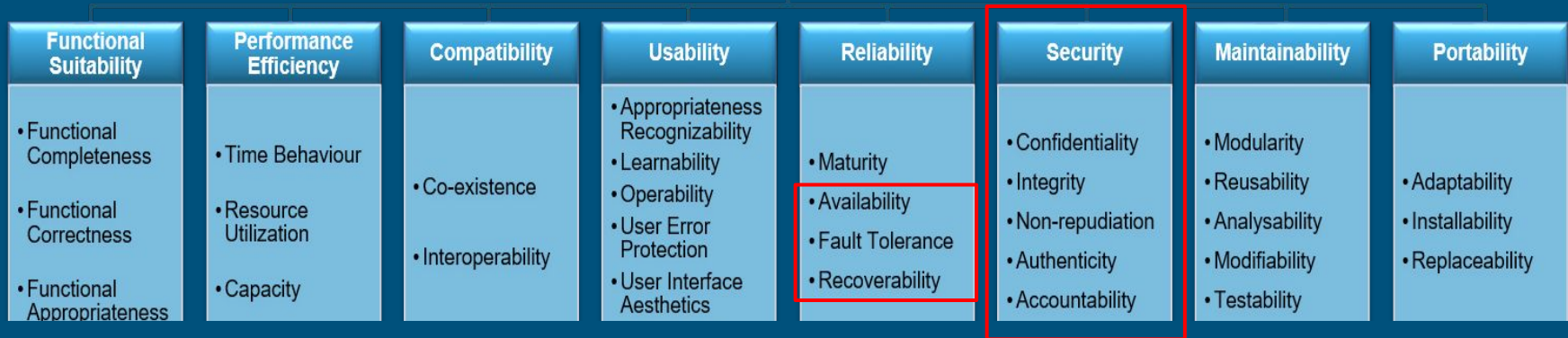
[ISO 25010]

System und Software-Engineering –

Qualitätskriterien und Bewertung von System und Softwareprodukten (SQuaRE)



© iso.org



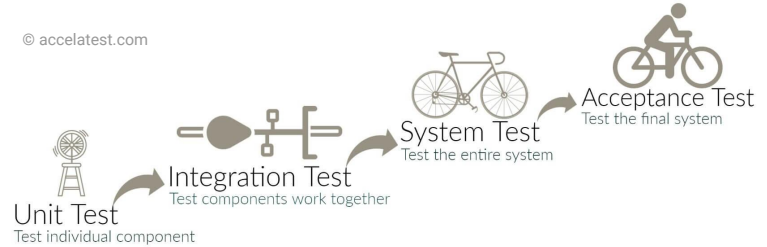
Quelle: [websitesecuritystore.com](https://www.websitesecuritystore.com)

# II) Hintergrundliches

---

# Teststufen

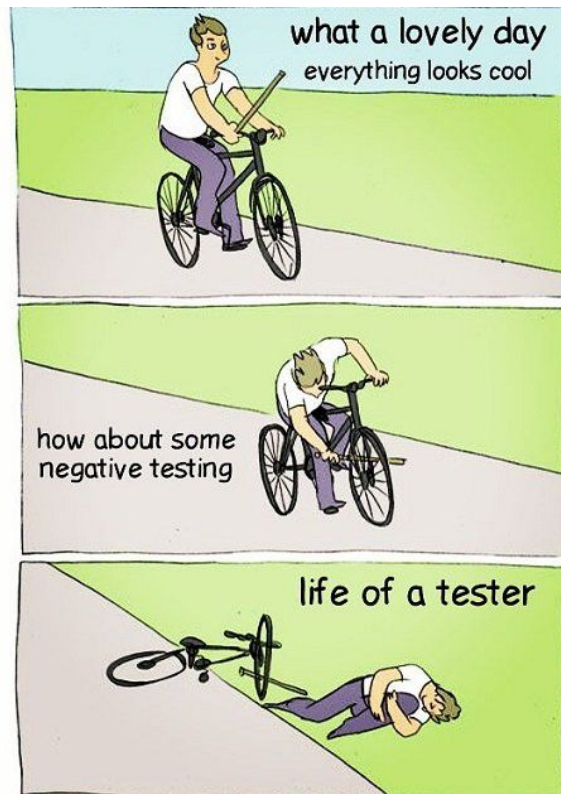
- **Component (Unit) Tests**
  - Inneres der Komponenten, Mocking der Abhängigkeiten
- **Integration Tests**
  - Schnittstellen, Interaktion der Komponenten



Quelle: tsh.io

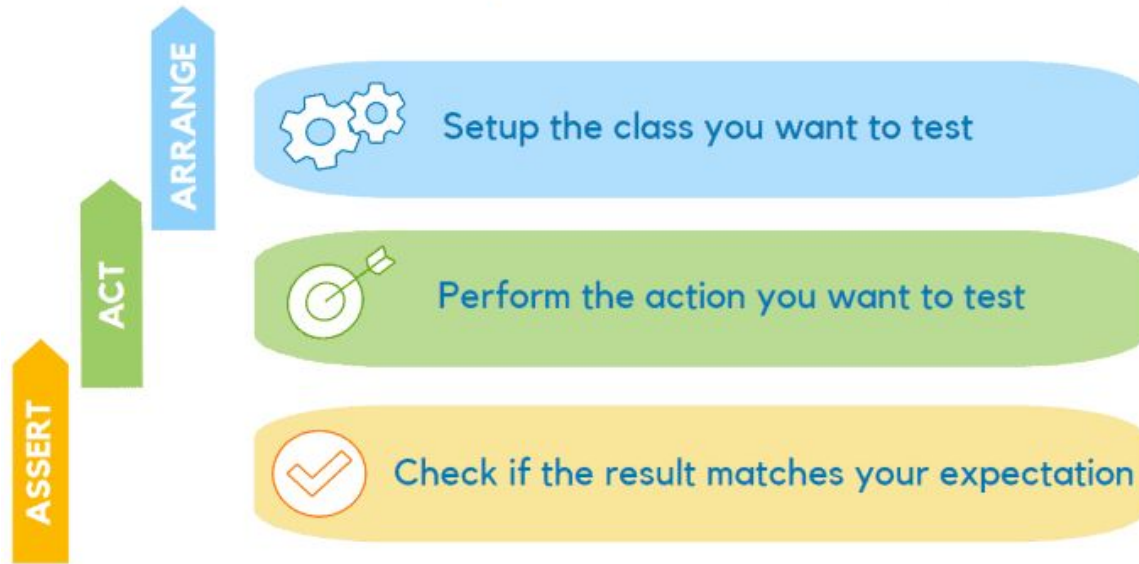
# Positive vs Negative Testing

- **Positiv-Tests** 😇
  - mit validem Input
  - prüft korrekte Funktionalität
  - bestätigt das Erwartete
- **Negativ-Tests** 😈
  - mit invalidem/unerwartetem Input
  - prüft Robustheit, Performanz, Sicherheit
  - findet Schwachstellen



# Arrange Act Assert (AAA)

---



# Parametrisierung

---

- variierender Input bei gleicher Erwartung
- gegen Zufallstreffer (false positive)

```
[TestCase(12, 3, 4)]  
[TestCase(12, 2, 6)]  
[TestCase(12, 4, 3)]  
public void Divide__With_valid_input__Should_succeed(int dd, int dv, int q)  
{  
    // Act  
    var divisionResult = MathHelper.Divide(dd, dv);  
  
    // Assert  
    Assert.That(divisionResult, Is.EqualTo(q), $"because {dd} and {dv} are valid");  
}
```

C#

# III) Security Test Cases

---

# Test Case: Unauthorized

---

- Bsp.: Endpunkt "`{base}/api/jedis`"
  - mit HTTP Methods `GET POST PUT DELETE`
- **Unauthorized** ([401](#)) Test  
(eigentlich 'unauthenticated')
- Test auf *Confidentiality*
  - Input:  
keine authentication credentials
  - Expected:  
401 Error Response  
(statt erfolgreichem Query/Command)





# Test Case: Unauthorized - Beispiel

---

```
[Test]
public void GetById_Without_auth_Should_fail_with_unauthorized()
{
    // Arrange
    Logout();

    // Act
    var response = api.CallGetById(idOfJedi1);

    // Assert
    Assert.That(
        response.StatusCode,
        Is.EqualTo(HttpStatusCode.Unauthorized),
        "because the GET request was made without authentication");
}
```

C#

# Test Case: Forbidden

---

- Bsp.: Endpunkt "`{base}/api/lightsabers`"
  - mit HTTP Methods `GET POST PUT DELETE`
- **Forbidden** ([403](#)) Test (eigentlich 'unauthorized')
- Test auf *Confidentiality*
  - Input: insufficient rights to a resource
  - Expected: 403 Error Response (statt erfolgreichem Query/Command)



# Test Case: Forbidden - Beispiel

---

```
[Test]
public void Create__As_padawan__Should_fail_with_forbidden()
{
    // Arrange
    LoginAsPadawan();

    // Act
    var response = api.CallCreate(LightsaberColours.Blue);

    // Assert
    Assert.That(
        response.StatusCode,
        Is.EqualTo(HttpStatusCode.Forbidden),
        "because the POST request was made by an unauthorized padawan");
}
```

C#

# Test Case: NotFound

---

- Bsp.: Endpunkt "`{base}/api/siths`"
  - mit HTTP Methods `GET PUT DELETE`
- **NotFound** ([404](#)) Test
- Test auf *Integrity*
  - Input:  
a valid resource value
  - Expected:  
404 Error Response  
(statt erfolgreichem Query/Command)



# Test Case: NotFound - Beispiel

---

C#

```
[Test]
public void Delete__Existing_sith__Should_succeed()
{
    // Act
    var response = api.CallDelete(idOfDarthMaul);

    // Assert
    Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.OK));

    var response = api.CallGetById(idOfDarthMaul);

    Assert.That(
        response.StatusCode,
        Is.EqualTo(HttpStatusCode.NotFound),
        $"because a sith with the ID {idOfDarthMaul} does not exist");
}
```

# Weitere Response Test Cases

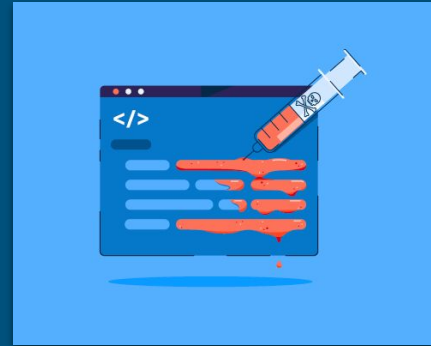
---

- 400 BadRequest,  
z.B. bei malformed request syntax
- 409 Conflict,  
z.B. beim Aktualisieren von Ressource mit  
widersprüchlichem Payload
- 429 Too Many Requests,  
z.B. bei zu vielen Anfragen in bestimmter Zeit




# Test Case: SQL Injection

- Input Validation,  
falls manuelle custom Queries notwendig



Quelle: learn.g2.com

```
txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;
```

Benutzerkennung	Passwort	
<input type="text" value="105 OR 1=1"/>	<input type="password" value="....."/>	
		<input type="button" value="Anmelden"/>

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

# Test Case: SQL Injection - Beispiel

---

```
[TestCase("99 OR 1=1")]
[TestCase("99'; DROP TABLE users; --")]
public void GetById_With_malicious_input__Should_fail(string injectedInput)
{
    // Act
    var response = api.CallGetWithCustomSql(injectedInput);

    // Assert
    Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.BadRequest));

    Assert.That(response.Content, Is.Null,
        $"because the ID {input} is malicious");
}
```

C#



# Test Case: multipart/form-data

---

- Input Validation bei File Upload  
(z.B. png, pdf, xlsx, json, ...)
- Test auf *Integrity*
  - Input:  
invalides File oder invalider File Content
  - Expected:  
keine Verarbeitung



Quellen:  
Lord of the Rings  
/ Star Wars  
/ Life of Brian  
/ Fifth Element

# Test Case: multipart/form-data - Beispiel 1

---

```
[Test("invalid_file.txt")]
```

```
[Test("empty_file.json")]
```

C#

```
public void Upload_With_invalid_file__Should_fail_with_bad_request(string fileName)
{
    // Arrange
    var testfile = new FormFile(memoryStream, 0, stream.Length, fileName, fileName);

    // Act
    var response = api.CallUploadJsonFile(testfile);

    // Assert
    Assert.That(response.StatusCode, Is.EqualTo(HttpStatusCode.BadRequest));
}
```

# Test Case: multipart/form-data - Beispiel 2

---

```
[TestCase("{\"name\": \"Jason1\", \"age\": null, }")] // Extra comma (,) in object
[TestCase("{\"name\": \"Jason2\", , \"age\": null}")] // Extra comma (,) in object
[TestCase("{\"name\": \"Jason3\", \"age\": null}")] // Closing bracket wrong
[TestCase("{\"name\": \"Jason4\", \"age\": }")] // Missing value in name value pair in object
[TestCase("{\"name\": \"Jason5\", \"age\" }")] // Missing : after name in object
[TestCase("{}")] // Missing name in object
public void Upload_With_invalid_content_Should_fail(string content)
{
    // Act
    var methodResult = content.DeserializeJson(out object objectResult);

    // Assert
    methodResult.Should().BeFalse();
    objectResult.Should().BeNull();
}
```

C#

# Test Case: Exceptions

---

- Werfen von Exceptions testen (falls ihr sie aus Gründen nicht wegfangt)

```
[Test]
public void TestException()
{
    Assert.Throws<InvalidOperationException>()
        .And.Message.EqualTo("message")
        .And.Property("MyParam").EqualTo(42),
        () => throw new MyException("message", 42));
}

Assert.Throws<NullReferenceException>(MethodThatThrows);
Assert.Throws<ArgumentNullException>(MethodThatThrows);
Assert.Throws<UriFormatException>(MethodThatThrows);
```

C#

# Test Case: Logging

---

- Protokollierung von Ereignissen testen, insbesondere bei Problemen

```
public int? Divide (int dividend, int divisor)
{
    try
    {
        logger.LogInformation("Dividing {Dividend} with {Divisor}", dividend, divisor);
        return dividend / divisor;
    }
    catch (Exception ex)
    {
        logger.LogError(ex, "Error during divide");
        return null;
    }
}
```

C#

# Test Case: Logging - Beispiel

---

```
[Test]
public void Divide__With_invalid_divisor__Should_log()
{
    // Arrange
    var logger = new FakeLogger<Math>();
    var math = new Math(logger);

    // Act
    var result = math.Divide(10, 0);

    // Assert
    Assert.That(result, Is.Null);
    Assert.That(logger.Collector.LatestRecord, Is.Not.Null);
    Assert.That(logger.Collector.LatestRecord.Message, Is.EqualTo("Error during divide"));
}
```

C#

# Test Case: Encryption / Hashing

- testbar bei **Encryption**:
  - Mindestlänge (abhängig von Input)
  - Regex
  - Decrypt mit falschem, aber sehr ähnlichem Key
  - Klartext nicht in Chiffre
- testbar bei **Hashing**:
  - Mindestlänge
  - gleiche Länge (unabhängig vom Input)
  - Regex
  - Salt (nie gleicher Hash bei gleichem Input)



# Assertion Extensions

---

- z.B. 'FluentAssertions', 'AssertJ', ...

```
theObject.Should().NotNull();  
theObject.Should().BeOfType(typeof(string));  
theObject.Should().BeXmlSerializable();  
theObject.Should().BeBinarySerializable();  
  
theInt.Should().BeGreaterThanOrEqualTo(0);  
theInt.Should().BeInRange(1, 10);  
theInt.Should().NotBeNegative();  
  
theDatetime.Should().NotBeBefore(1.February(2010));  
timeSpan.Should().BeCloseTo(new TimeSpan(13, 0, 0), 10.Ticks());
```



# Weitere Test Cases

- Input Sanitization
  - Unsichere Chars vor Processing bereinigt?
- Output Encoding
  - Chars vor Ausgabe im Client enkodiert?
- DB Transactions
  - Funktionieren Rollbacks korrekt?
- Performance
  - Geht der Code durch Überlastung in die Knie?
- Race Conditions
  - Mehrere zeitgleiche Operationen ausgeführt?

*Knock Knock  
Race condition!  
Who's there?*

*Multithreaded programming*



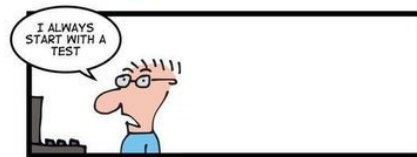
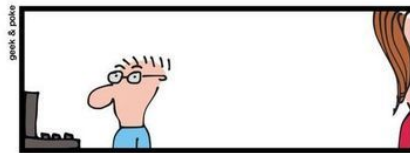
Quelle: devrant.com

# Aufräumen nicht vergessen!

- Connections schließen
- DB leeren
- Log leeren
- Ausloggen
- ...

```
[TearDown]
public void TearDown()
{
    _dbContext.Dispose();
    DatabaseContext.ClearDatabase(TestingDbContext);
}
```

## SIMPLY EXPLAINED



TDD

# IV. Und außerdem

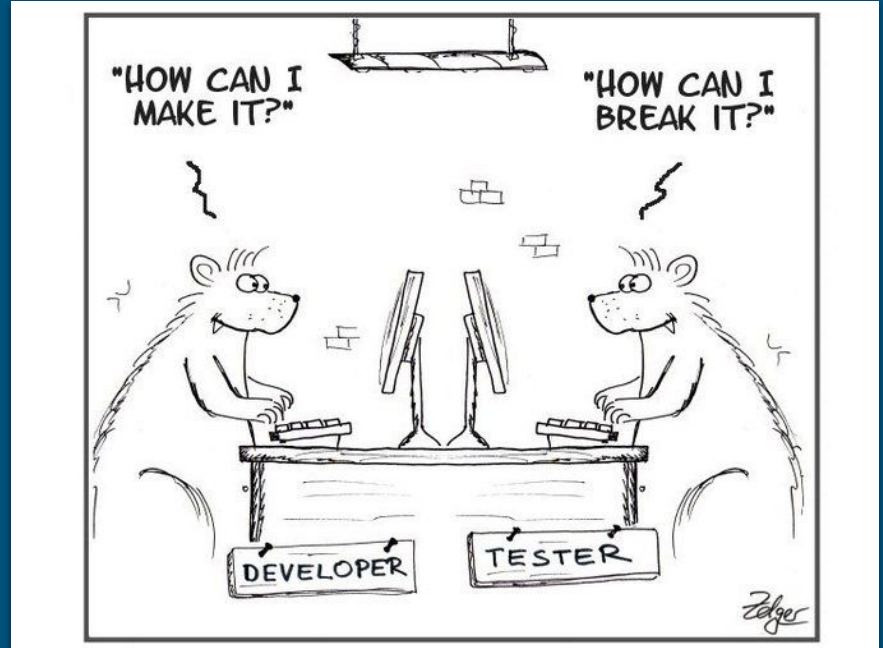
---

# Mindset

---

Denkt immer zuerst an die  
**Negativ-Tests!**

Seid teuflisch kreativ!



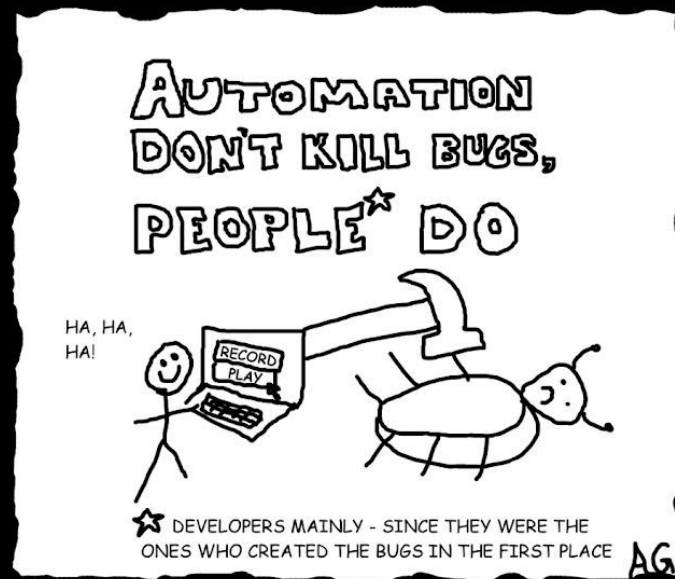
# Statische Analyse

Nutzt die Analyse-Tools Eurer IDE!

Ignoriert die Warnungen nicht!

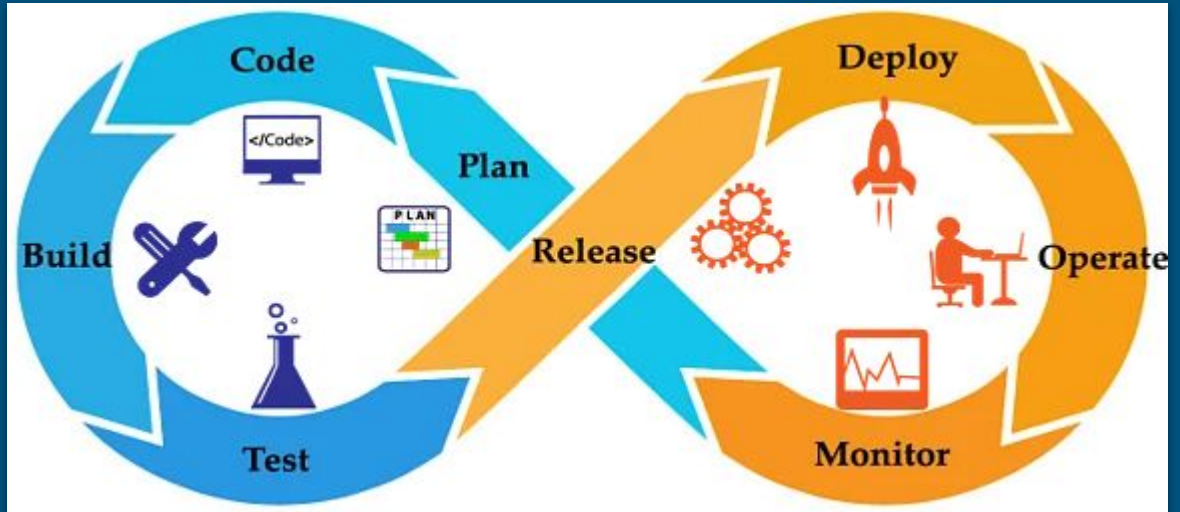
Static Application Security Testing (SAST) Tools:

[owasp.org/www-community/Source\\_Code\\_Analysis\\_Tools](https://owasp.org/www-community/Source_Code_Analysis_Tools)



# CI/CD

Lasst die Tests  
regelmäßig laufen!  
(Regression Testing)



Quelle: medium.com

# Test Driven Development

---

Schreibt die Tests doch mal vor dem  
Produktiv-Code!

(oder wenigstens zeitgleich)

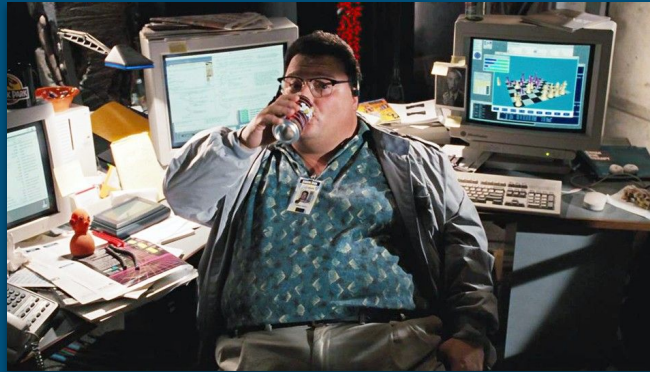


**bug** driven  
development

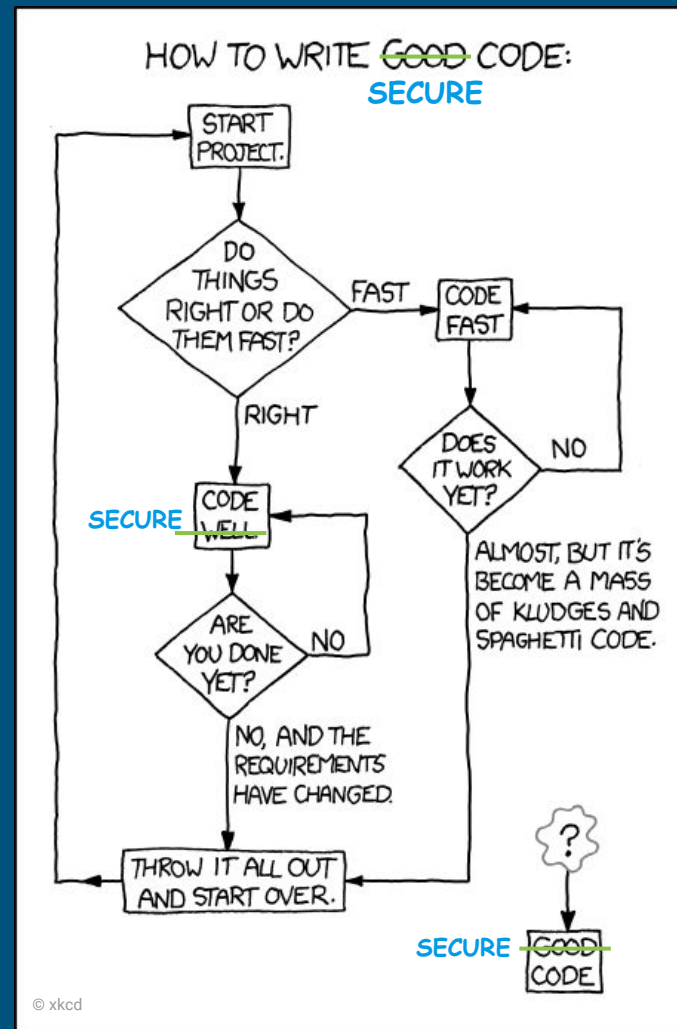


**test** driven  
development

# Fazit



Download & mehr:  
[jasie.de/security-testing](https://jasie.de/security-testing)





# Weiterführend /1

---

- [1] zu Schutzzielen (CIA):  
StickyMinds: **Using the Principles of the CIA Triad to Implement Software Security** (2016)  
[stickyminds.com/article/using-principles-cia-triad-implement-software-security](https://stickyminds.com/article/using-principles-cia-triad-implement-software-security)
  
- [2] zu ISO/IEC 25010:  
Medium: **Understanding ISO/IEC 25010: Unraveling the Complexity of Product Quality Evaluation** (2023)  
[medium.com/@gbrigens/understanding-iso-iec-25010-unraveling-the-complexity-of-product-quality-evaluation-d47f32ba60c1](https://medium.com/@gbrigens/understanding-iso-iec-25010-unraveling-the-complexity-of-product-quality-evaluation-d47f32ba60c1)
  
- [3] zu Mariner I Crash:  
TUM: **Mariner I** (2002)  
[www5.in.tum.de/lehre/seminare/semsoft/unterlagen\\_02/erdfenr/website/mariner.html](http://www5.in.tum.de/lehre/seminare/semsoft/unterlagen_02/erdfenr/website/mariner.html)
  
- [4] zu OWASP & ausgenutzten Schwachstellen:  
Horangi: **Real Life Examples of Web Vulnerabilities (OWASP Top 10)** (2022)  
[horangi.com/blog/real-life-examples-of-web-vulnerabilities](https://horangi.com/blog/real-life-examples-of-web-vulnerabilities)

# Weiterführend /2

---

- [5] zu Teststufen:  
Medium: **ISTQB Foundation [...]: Testing Throughout the Software Development Lifecycle** (2019)  
[medium.com/@HugoSaxTavares/istqb-foundation-level-syllabus-part-2-of-6-e85155a27ccf#94d8](https://medium.com/@HugoSaxTavares/istqb-foundation-level-syllabus-part-2-of-6-e85155a27ccf#94d8)
- [6] zu Positive vs Negative Testing:  
QA Madness: **Negative vs Positive Testing? Always Go for Both! Here's Why** (2023)  
<https://www.qamadness.com/negative-vs-positive-testing-always-go-for-both-heres-why/>
- [7] zu Security Unit Tests:  
Medium: **Shift Left Security, Security Unit Tests [...] Key Practices for Secure Development** (2023)  
[medium.com/@rethinkyourunderstanding/shift-left-security-security-unit-tests-and-owasp-top-10-and-ai-key-practices-for-secure-e41b060a8057](https://medium.com/@rethinkyourunderstanding/shift-left-security-security-unit-tests-and-owasp-top-10-and-ai-key-practices-for-secure-e41b060a8057)