

WEB-UI TESTS

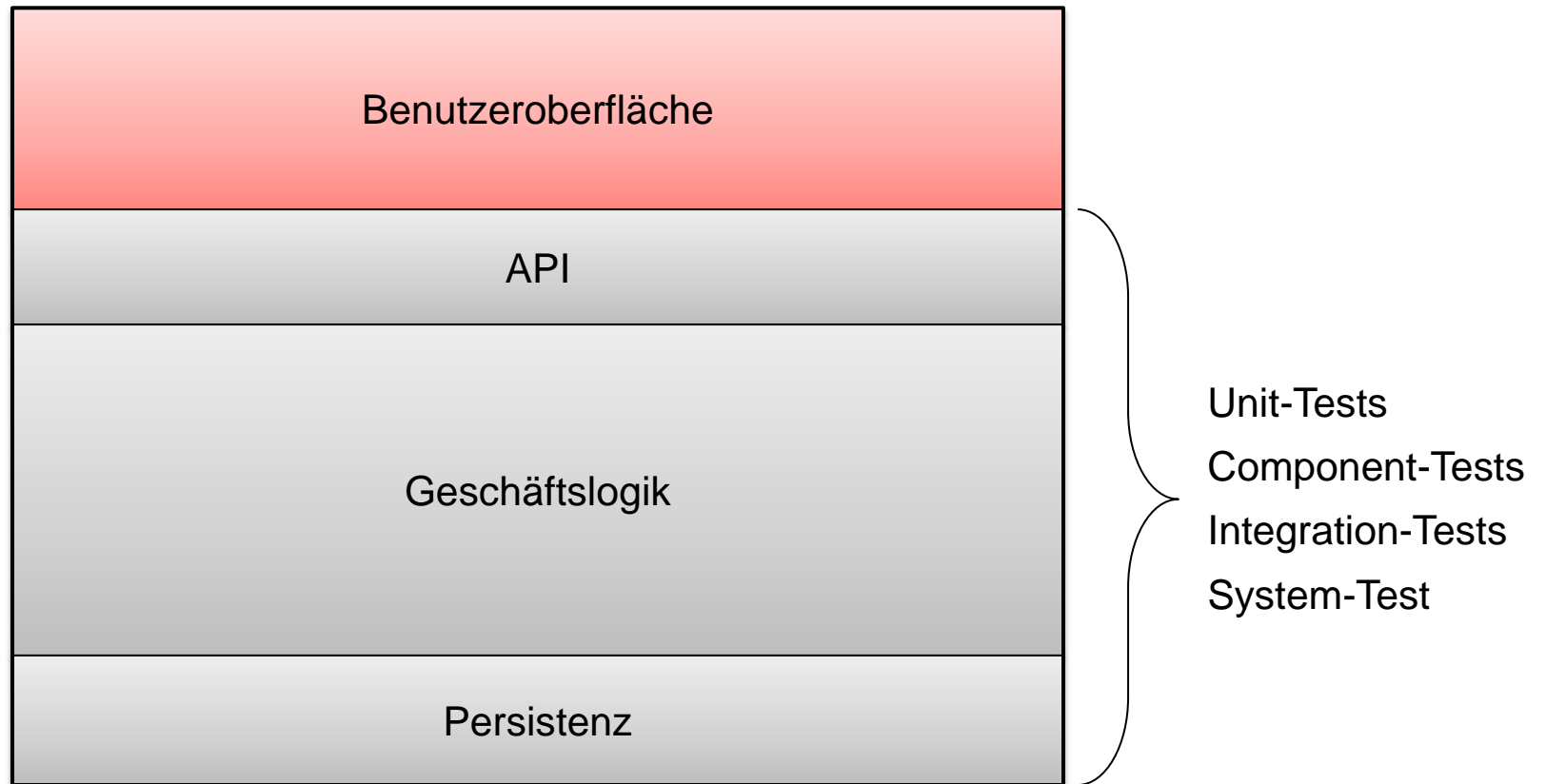
18.01.2016, PASCAL MOLL UND STEFAN LUDWIG

AGENDA

1. Testautomatisierung von Web-Anwendungen
2. Vorstellung verschiedener Ansätze
 1. Record & Replay
 2. Programmierung
3. Page-Object Pattern
4. Pause
5. Praxisübungen
6. Abschluss

TESTAUTOMATISIERUNG VON WEB-ANWENDUNGEN

Warum die Benutzeroberfläche testen?



Warum die Benutzeroberfläche testen?

Die Benutzeroberfläche ...

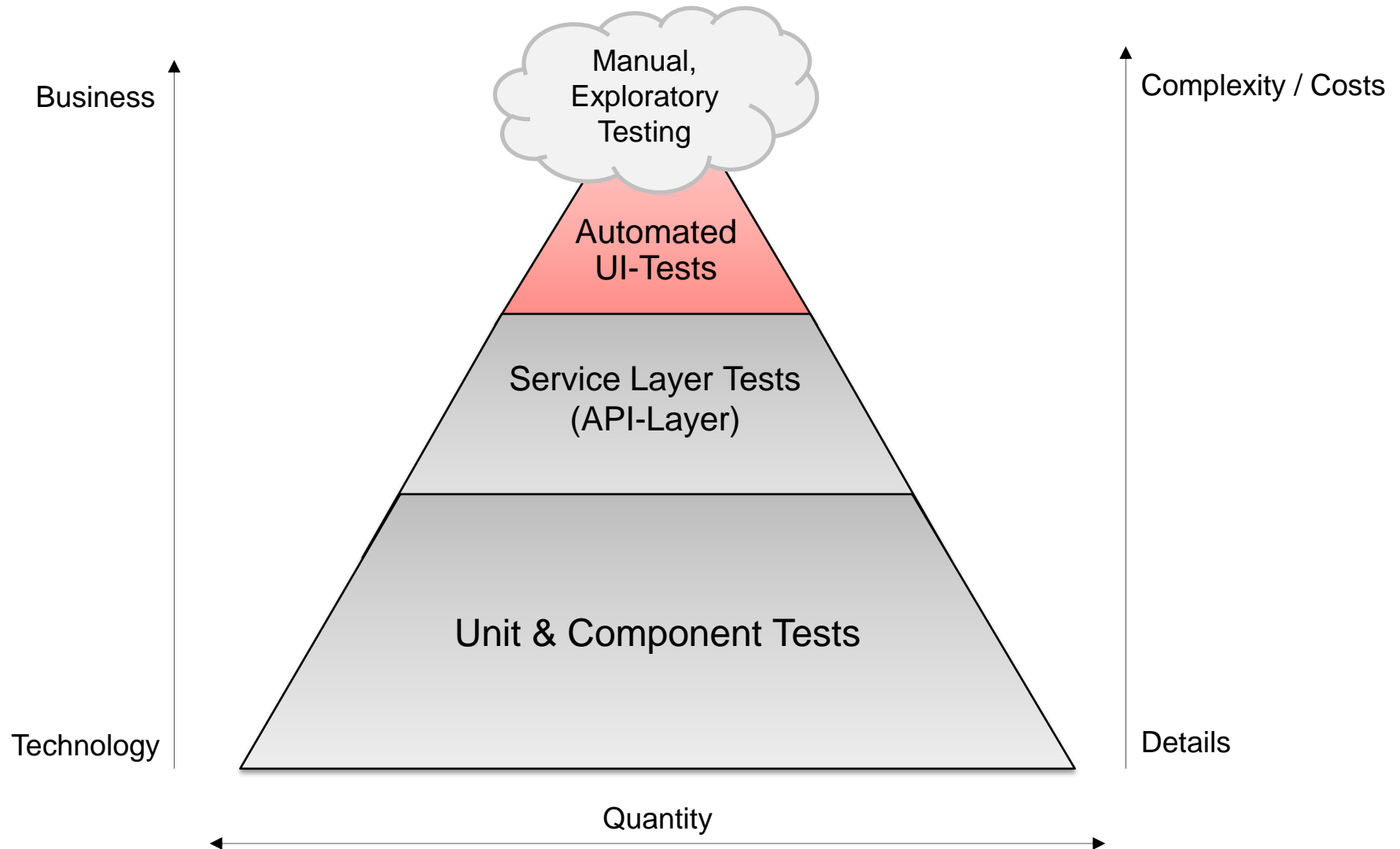
... ist Teil der Anwendung.

... ist ein Haupteinstiegspunkt in das System.

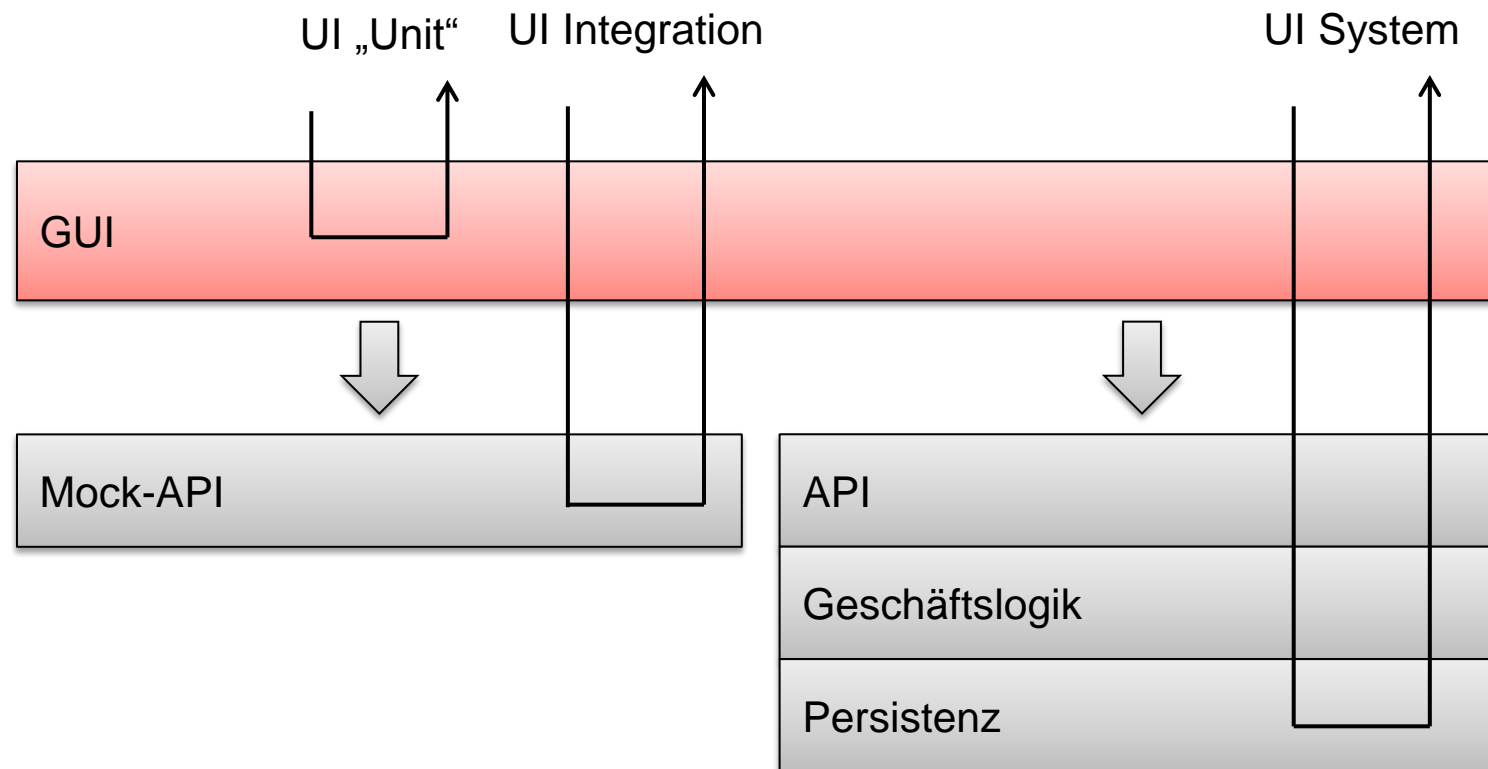
... ist, was der Kunde als DIE “Anwendung” wahrnimmt.

... beinhaltet ihre eigene Geschäftslogik.

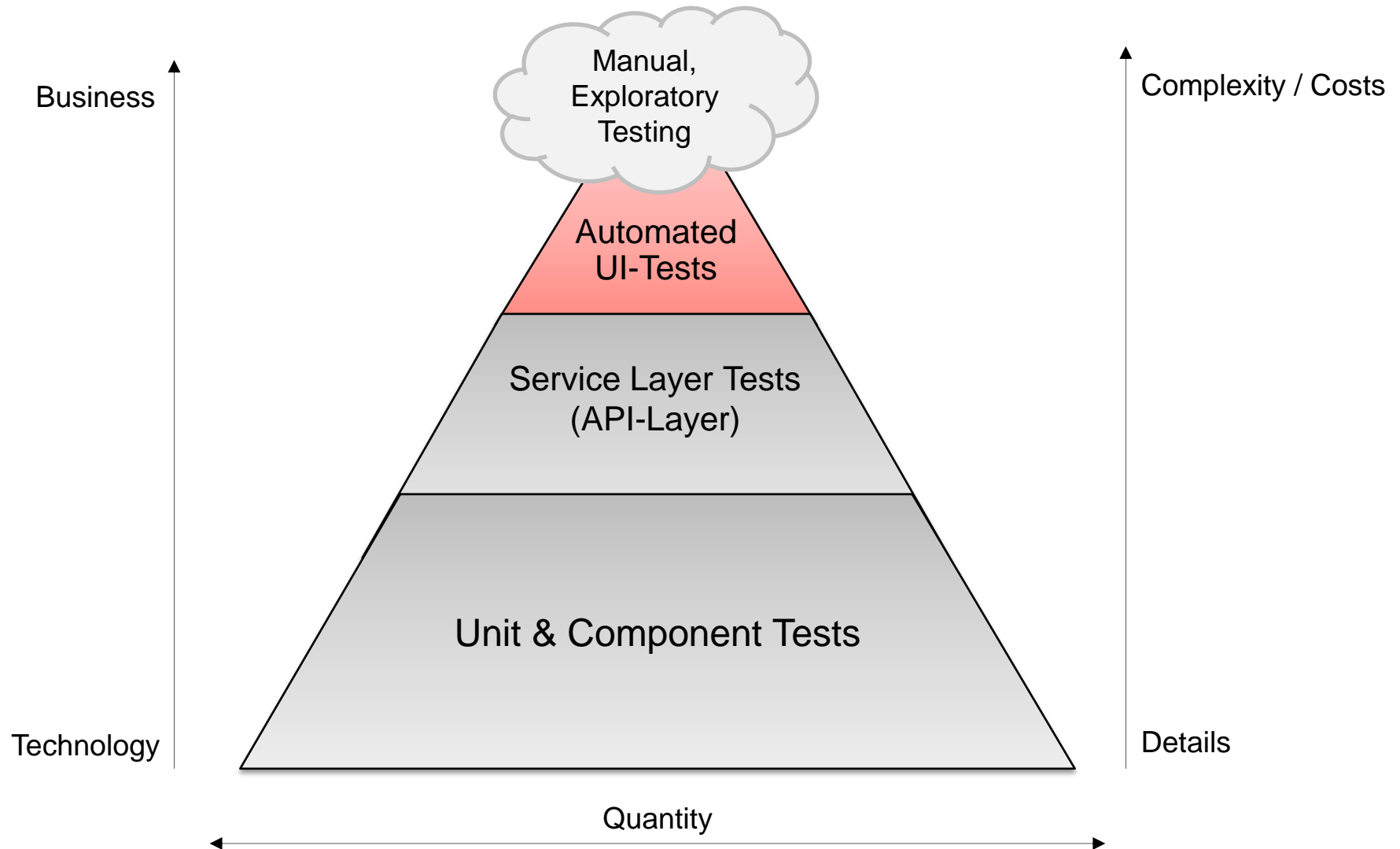
Testpyramide



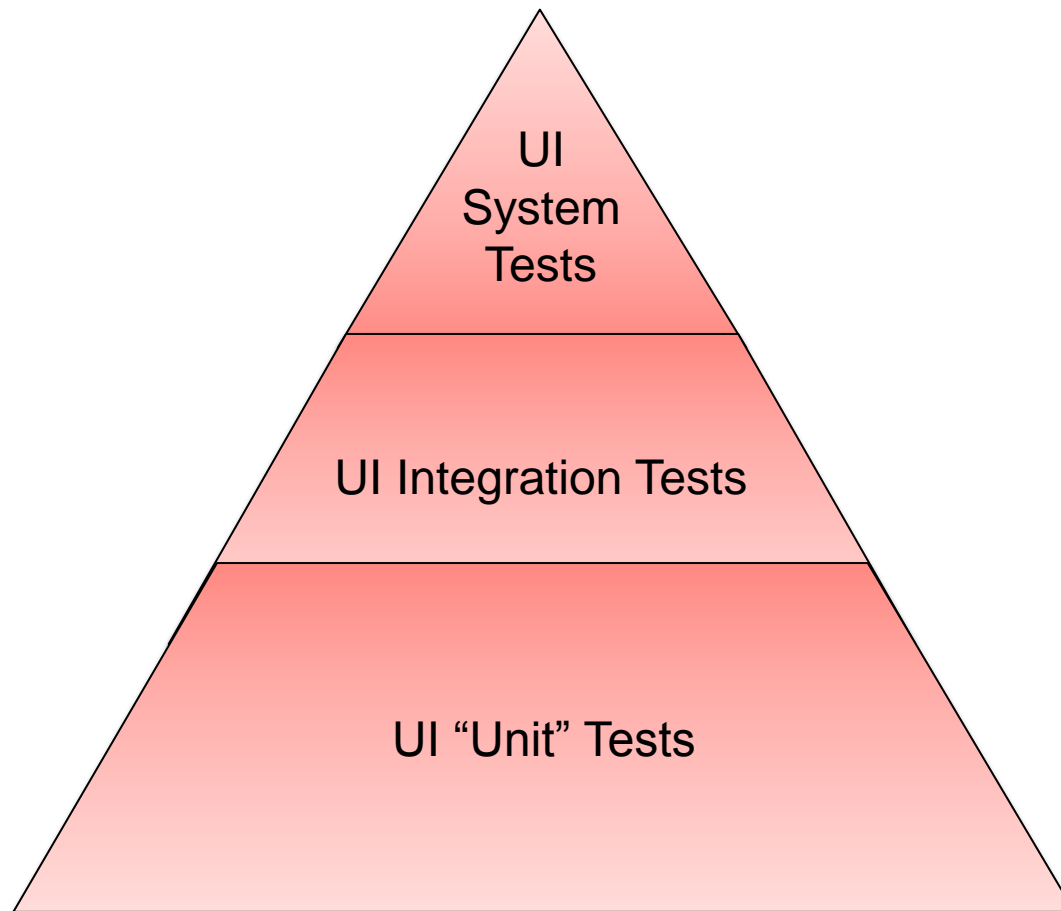
Unterschiedliche Arten von UI-Tests



Testpyramide



UI-Testpyramide



Was sollte über die UI getestet werden?



Was sollte über die UI getestet werden?

Geschäftslogik der Benutzeroberfläche (z.B. JavaScript)



Was sollte über die UI getestet werden?

Prozessabläufe

(z.B. Maskenreihenfolge, Navigationswege etc.)



Was sollte über die UI getestet werden?

Datenaustausch (UI <-> Backend)



Was sollte über die UI getestet werden?

Systemdurchstich
(~1 Gut- und ein Schlechtfall / Prozess)



Was sollte NICHT über die UI getestet werden?



Was sollte NICHT über die UI getestet werden?

Variationen von Systemdurchstichen



Funktionen die sich durch
Unit-, Component- oder Integrationstest
schneller testen lassen



VERSCHIEDENE ANSÄTZE ZUR UI-TESTAUTOMATISIERUNG

RECORD & REPLAY
PROGRAMMIERUNG

Record & Replay



Testfälle aufnehmen
und regelmäßig abspielen

Record & Replay



Demonstration

Record & Replay



Schnelle Umsetzung
(niedriger Initialaufwand)



Kein Programmierwissen nötig



Unterstützung von manuellen Tests



Reproduzierbarkeit von explorativen Tests



Record & Replay



Testdaten



Starke Browserbindung



Kein Durchgriff auf das
„System Under Test“



Instabil / Fehleranfällig



Redundanzen



Keine Trennung von
Test- und Fachlogik



Kostspielige Wartung von bestehenden Tests



Schwerfällige Einbindung in den „Continuous Integration“ Prozess



VERSCHIEDENE ANSÄTZE ZUR UI-TESTAUTOMATISIERUNG

RECORD & REPLAY
PROGRAMMIERUNG

Programmierung



Tests in der Programmiersprache des Projekts schreiben



Demonstration



Browserunabhängigkeit durch Abstraktion



Vielfältige Stabilisierungsmöglichkeiten



Durchgriff auf das „System Under Test“



Vielseitige Testdatenbereitstellung



Redundanzen
können vermindert werden



Test- und Fachlogik
können getrennt werden



Niedrigere Wartungskosten



Einbindung in den „Continuous Integration“ Prozess





Programmierwissen wird benötigt



Erstellung von Testcases dauert länger



Keine Unterstützung bei manuellen Testaufgaben



PAGE-OBJECT PATTERN

Page-Object Pattern



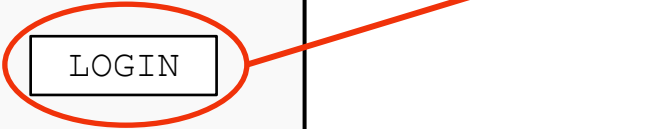
Trennung von
Test- und Interaktionscode
formalisieren

Szenario: Login

login.html

Username:

Password:



main.html

Hello testuser!

Szenario: Login – Testcode ohne Optimierungen

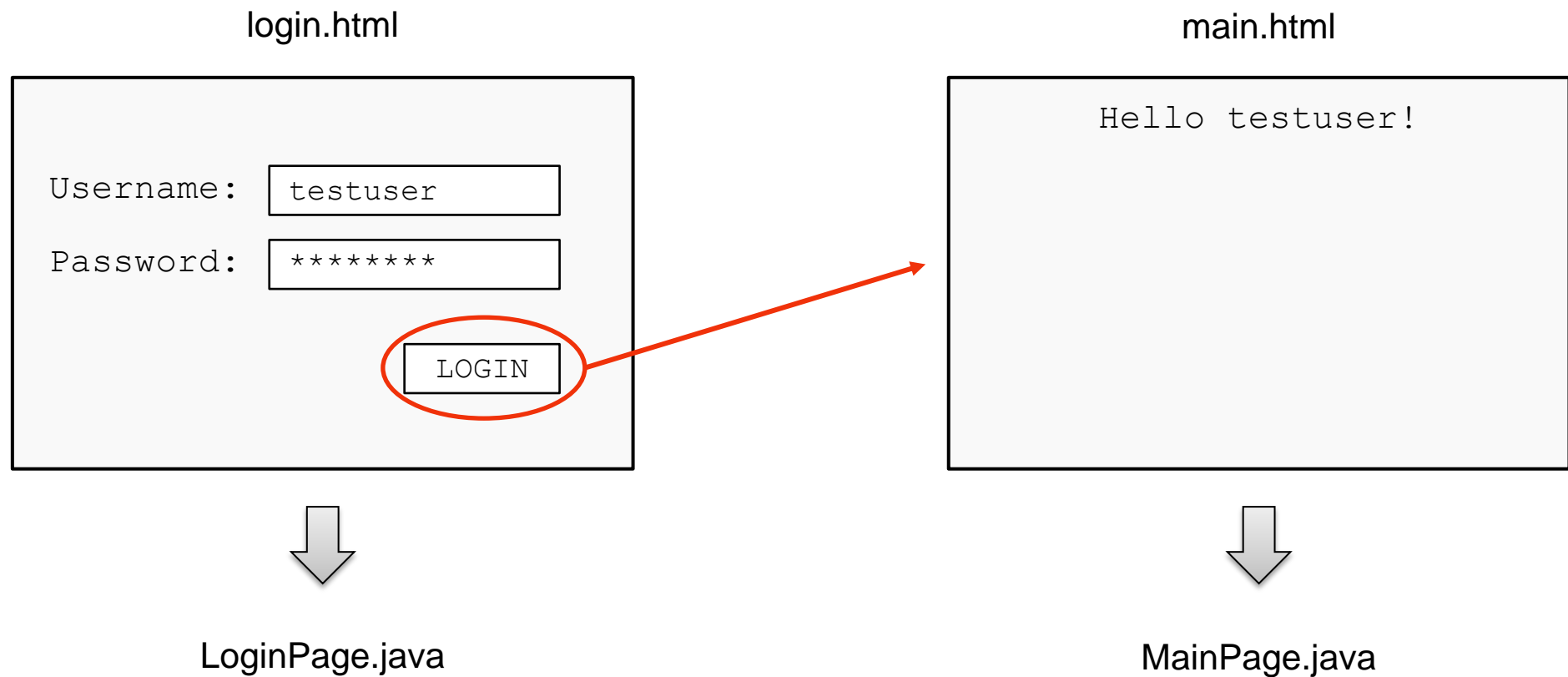
@Test

```
public void existingUserCanLogIn () {
    webDriver.findElement(By.id("username")).sendKeys("testuser");
    webDriver.findElement(By.id("password")).sendKeys("123456");
    webDriver.findElement(By.id("login")).click();
    String headline = webDriver.findElement(By.id("headline")).getText();
    assertThat(headline, is("Hello testuser!"));
}
```

@Test

```
public void unknownUserCantLogIn () {
    webDriver.findElement(By.id("username")).sendKeys("foo");
    webDriver.findElement(By.id("password")).sendKeys("123456");
    webDriver.findElement(By.id("login")).click();
    String errorMessage = webDriver.findElement (By.id("errorMessage")).getText();
    assertThat(errorMessage, is("Wrong Credentials!"));
}
```

Szenario: Login



Szenario: Login – Testcode mit Page-Object Pattern

@Test

```
public void existingUserCanLogIn () {  
    MainPage mainPage = loginPage.login("testuser", "123456");  
    assertThat(mainPage.getWelcomeMessage(), is("Hello testuser!"));  
}
```

@Test

```
public void unknownUserCantLogIn () {  
    loginPage = loginPage.loginExpectingError("foo", "123456");  
    assertThat(loginPage.getErrorMessage(), is("Wrong Credentials!"));  
}
```

Unterschiedliche Arten von Interaktionen

Aktion

Username:

Password:

Hello testuser!

Page-Object Pattern: Aktion

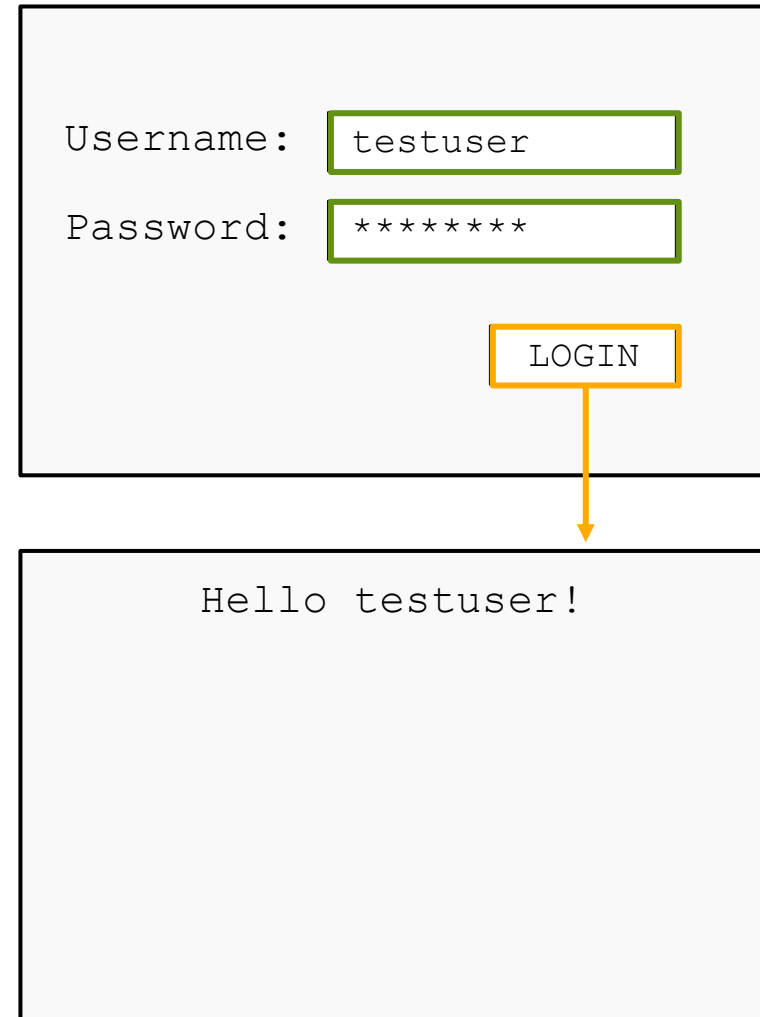
```
public LoginPage setUsername(String value) {  
    username.sendKeys(value);  
    return this;  
}
```

```
public LoginPage setPassword(String value) {  
    password.sendKeys(value);  
    return this;  
}
```

Unterschiedliche Arten von Interaktionen

Aktion

Navigation



Page-Object Pattern: Navigation

```
public MainPage clickLogin() {  
    login.click();  
    return createPageObject(MainPage.class);  
}
```

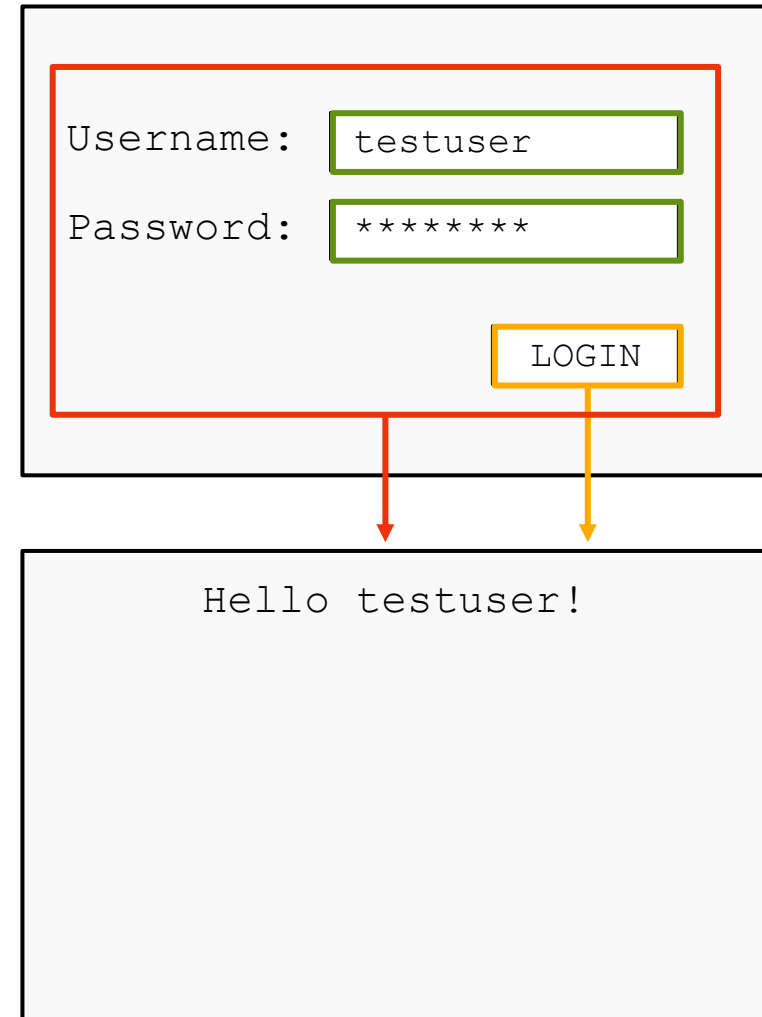
```
public LoginPage clickLoginExpectingError() {  
    login.click();  
    return createPageObject(LoginPage.class);  
}
```

Unterschiedliche Arten von Interaktionen

Aktion

Navigation

Workflow



Page-Object Pattern: Workflow

```
public MainPage login(String username, String password) {  
    return setUsername(username).setPassword(password)  
        .clickLogin();  
}
```

```
public LoginPage loginExpectingError(String username, String password) {  
    return setUsername(username).setPassword(password)  
        .clickLoginExpectingError();  
}
```

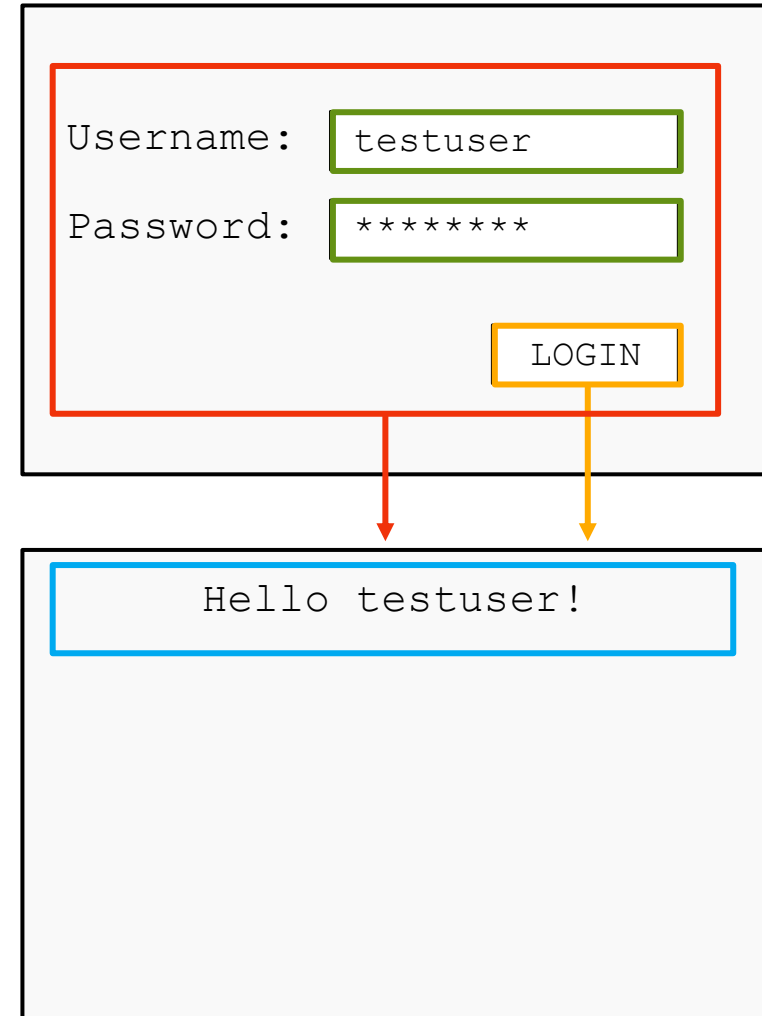
Unterschiedliche Arten von Interaktionen

Aktion

Navigation

Workflow

Information



Page-Object Pattern: Information

```
public String getErrorMessage () {  
    return errorMessage.getText();  
}
```

```
public List<String> getErrorMessages () {  
    List<String> returnValue = new LinkedList<>();  
    for (ErrorMessage message : errorMessages) {  
        returnValue.add(message.getText());  
    }  
    return returnValue;  
}
```

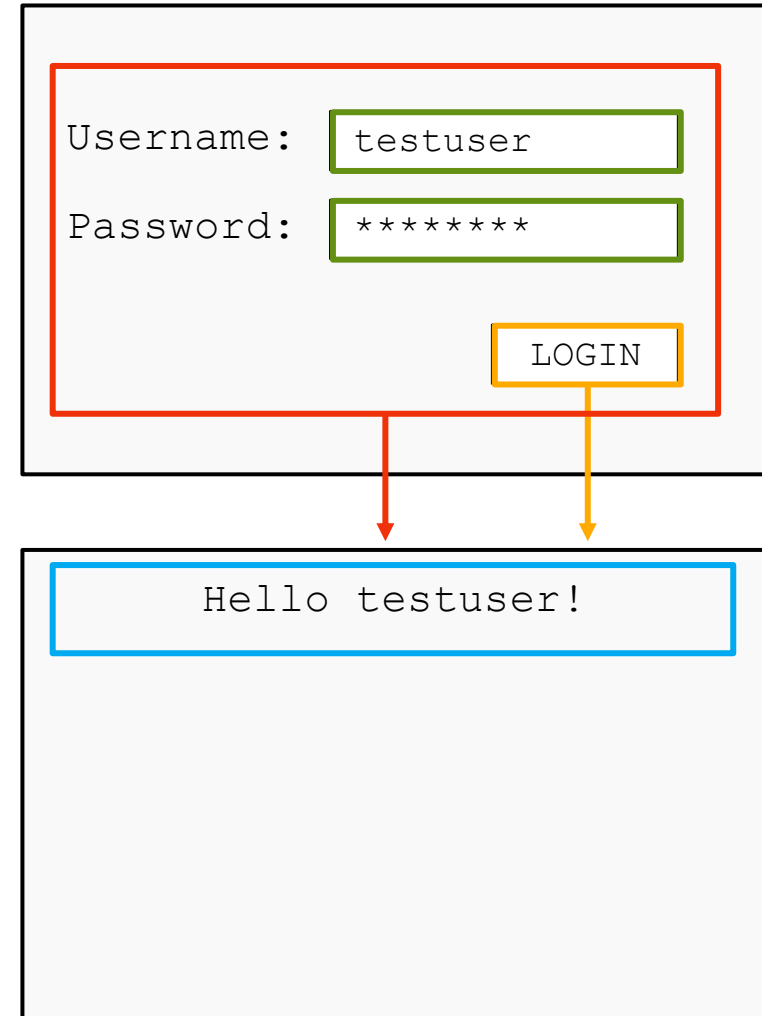
Unterschiedliche Arten von Interaktionen

Aktion

Navigation

Workflow

Information



Page-Object Pattern



Alle Vorteile von Programmierten Tests



Bewährtes Vorgehens- / Designmodell



Definierte Trennung von Test- und Interaktionscode



Minimaler
Wartungsaufwand



Maximale
Wiederverwendung



Page-Object Pattern



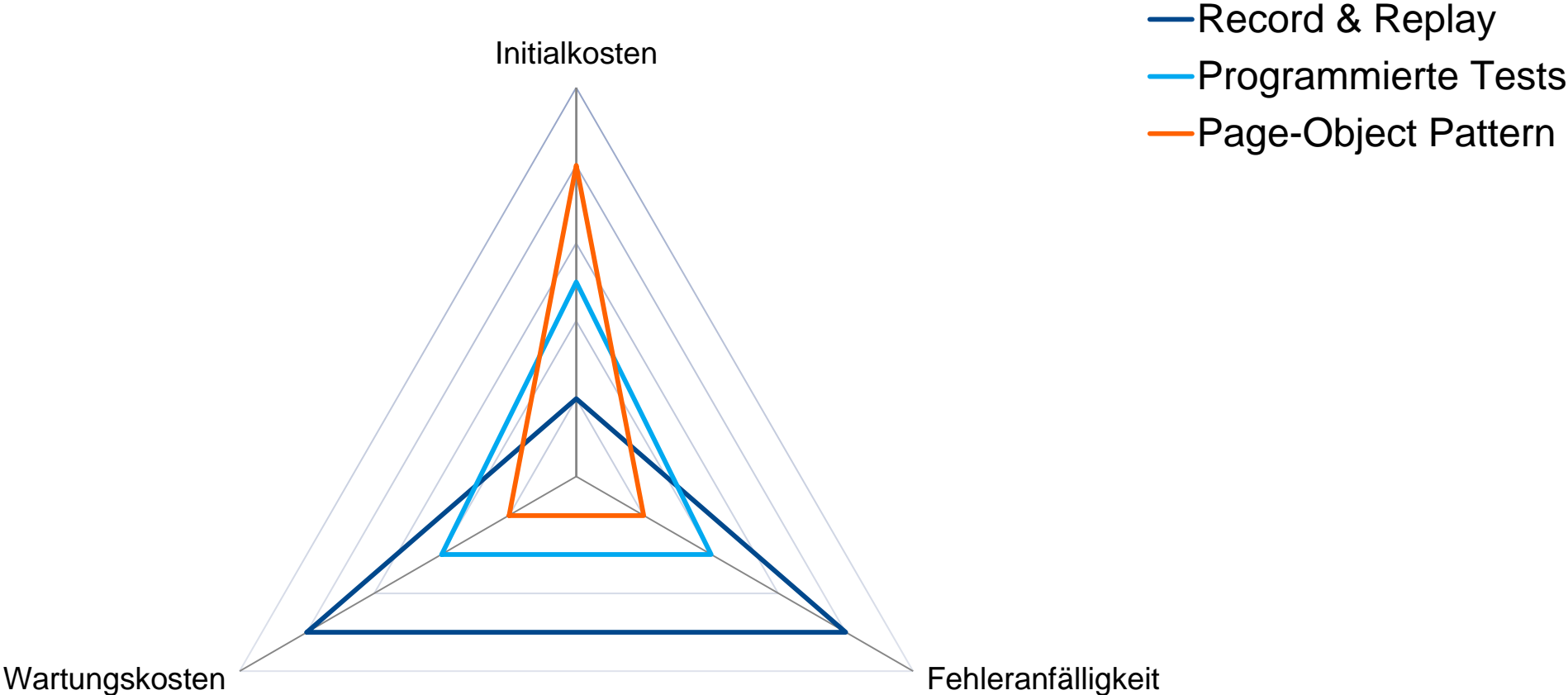
Initialaufwand



Höheres Maß an
Programmierwissen / Disziplin
erforderlich



Zusammenfassung



PRAXISÜBUNGEN

ABSCHLUSS

Erweiterte Projektanforderungen

- Projektspezifische und wiederverwendbare Klassen für einzelne Komponenten der Benutzeroberfläche
- Reporting der Testabläufe (Testprotokoll)
- Unterstützung beim Fehlerfindungsprozess (Screenshots, Sourcecode, Video etc.)
- Nachvollziehbarkeit beim Debuggen
- Browserneutralität von Tests

Open Source Projekt: testIT WebTester

- Open Source Framework auf Basis von Selenium
- Gewachsen aus jahrelanger Projekterfahrung
- Bietet eine verbesserte API zur Browserfernsteuerung
- Zusätzliche Hilfsklassen zur nachhaltigen Testautomatisierung
- <https://github.com/testIT-WebTester>



webtester

FRAGEN?

18.01.2016, PASCAL MOLL UND STEFAN LUDWIG



blog.novatec-gmbh.de



[@NT_AQE](https://twitter.com/NT_AQE)



aqe.novatec-gmbh.de



Agilität und Requirements Engineering

Agiles Requirements Engineering als Basis für das richtige Produkt!

Agilität und Testing-Strategie

Wie lassen sich Testaktivitäten effizient in Ihren agilen Prozess integrieren?

Effiziente Testautomatisierung

Wir setzen Testautomatisierung wartungsarm und nachhaltig um!

Tools für die Testautomatisierung

Welche Toolchain unterstützt Sie effizient?

Living Documentation

Anforderungen, Dokumentation und Tests aktuell und an einem Ort!

ALM* Tool-Auswahl

Welches ALM Tool passt zu Ihren Prozessen?