HSLU Hochschule Luzern

Verteilte Systeme und Komponenten

Integrations- und Systemtest

Martin Bättig

Letzte Aktualisierung: 27. Oktober 2022

FH Zentralschweiz

Inhalt

- Einleitung und Überblick
- Teststrategie
- Integrationstest
- Systemtest
- Testing in Scrum

Lernziele

- Sie können Komponenten entwerfen, dokumentieren, in Java realisieren, **testen** und deployen.
- Sie kennen die Zusammenhänge zwischen Analyse/Design und **Test/Abnahme** von Softwarekomponenten.
- Sie können geeignete Systemtests definieren, diese dokumentieren und die Durchführung protokollieren.
- Sie wissen, welche Informationen über die zu entwickelnde Software wann, wie und wo dokumentiert werden sollen.
- Sie wissen, welche Informationen aus dem Entwicklungsprozess gemäss Scrum wann, wie und wo dokumentiert werden sollen.
- Sie können Sprintbacklogs für ein kleines Team formulieren, schätzen und **geeignete Abnahmekriterien** festlegen.

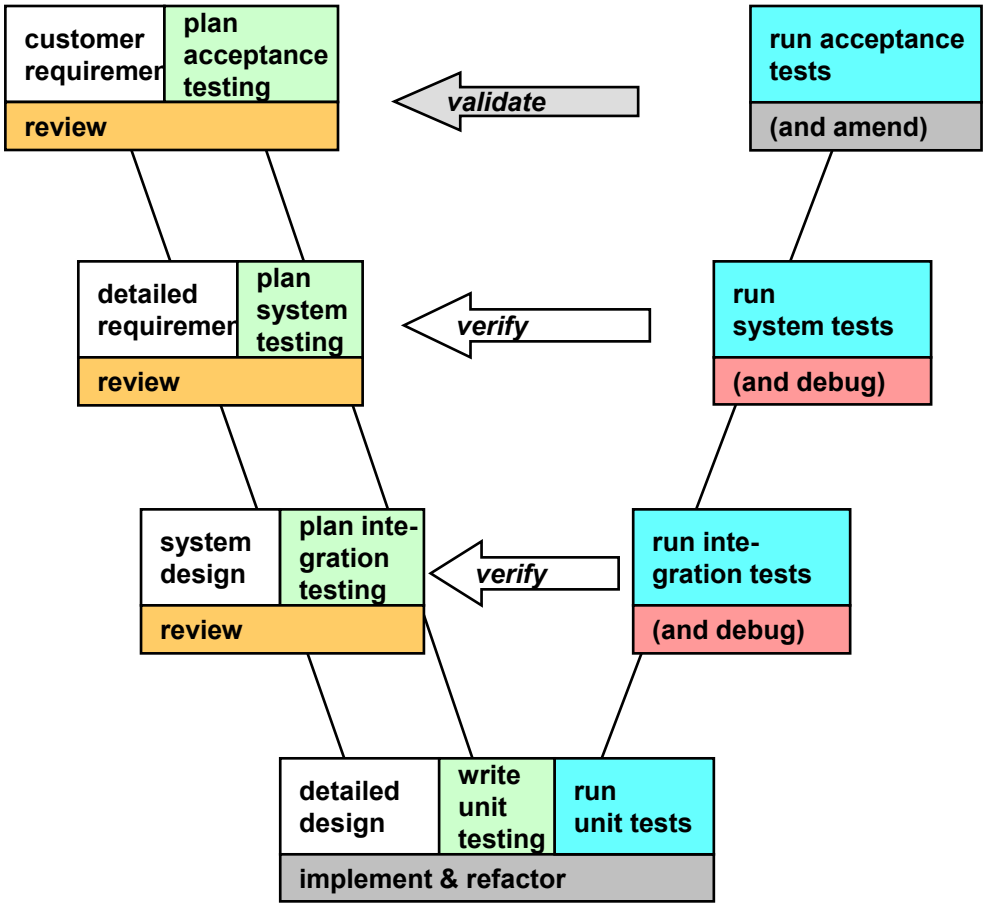
Einleitung und Überblick

Test Design

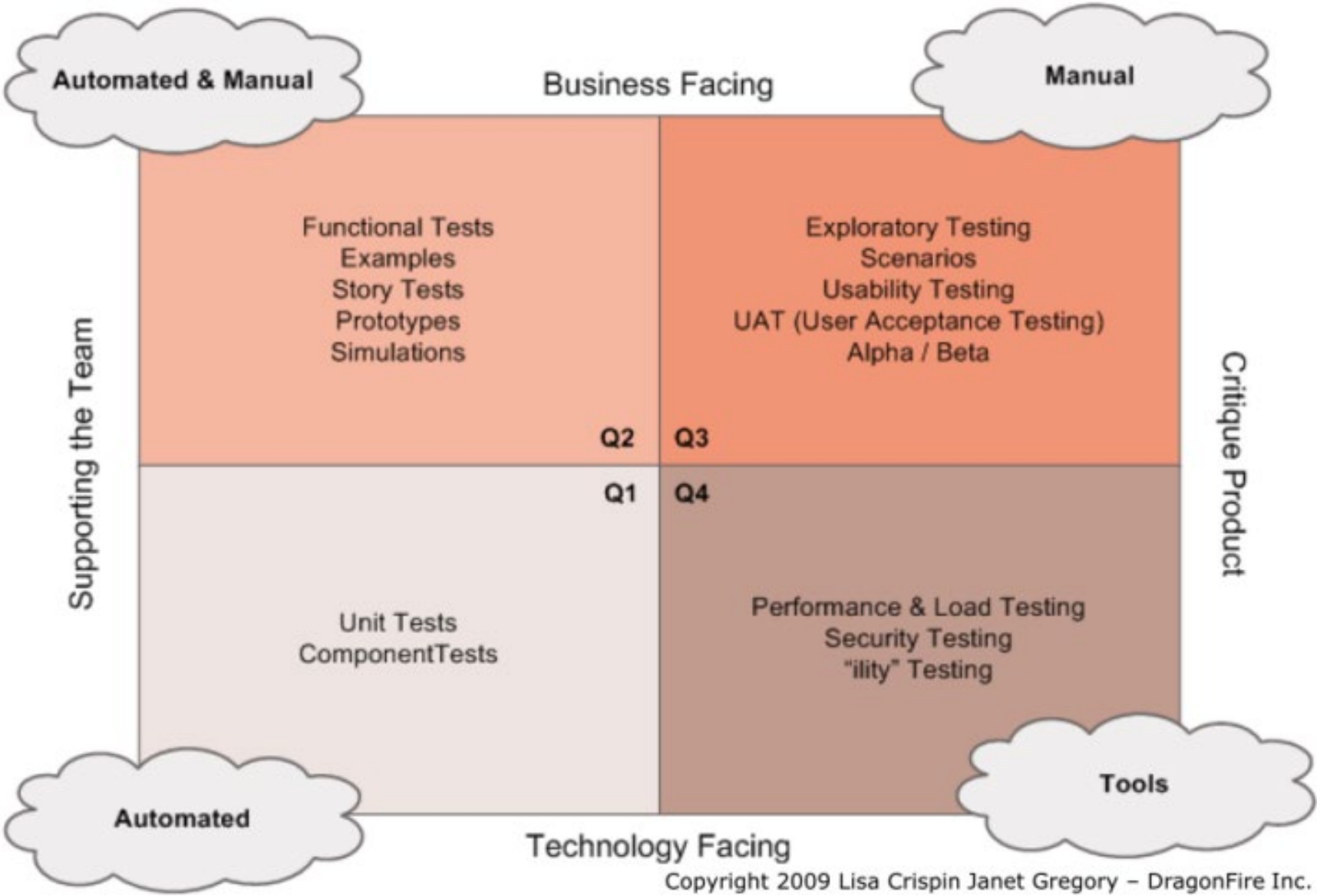
- "Testing by poking around is a waste of time" –
  (Zitat aus *Testing Object-Oriented Systems* von Robert Binder).
- Nur dokumentierte oder automatisierte Tests lassen sich **wiederholen** (=> Regressionstests!).

Klassisches Testen mit dem V-Modell

- Zu jeder Disziplin gibt es eine Entsprechung auf der Test-Seite.
- Anforderungen und Spezifikationen sind Grundlage für weitere Arbeiten => Review!
- Validieren (haben wir das Richtige entwickelt?) **und** verifizieren (haben wir es richtig gemacht?)



Agiles Testen



Ein Test genügt nicht

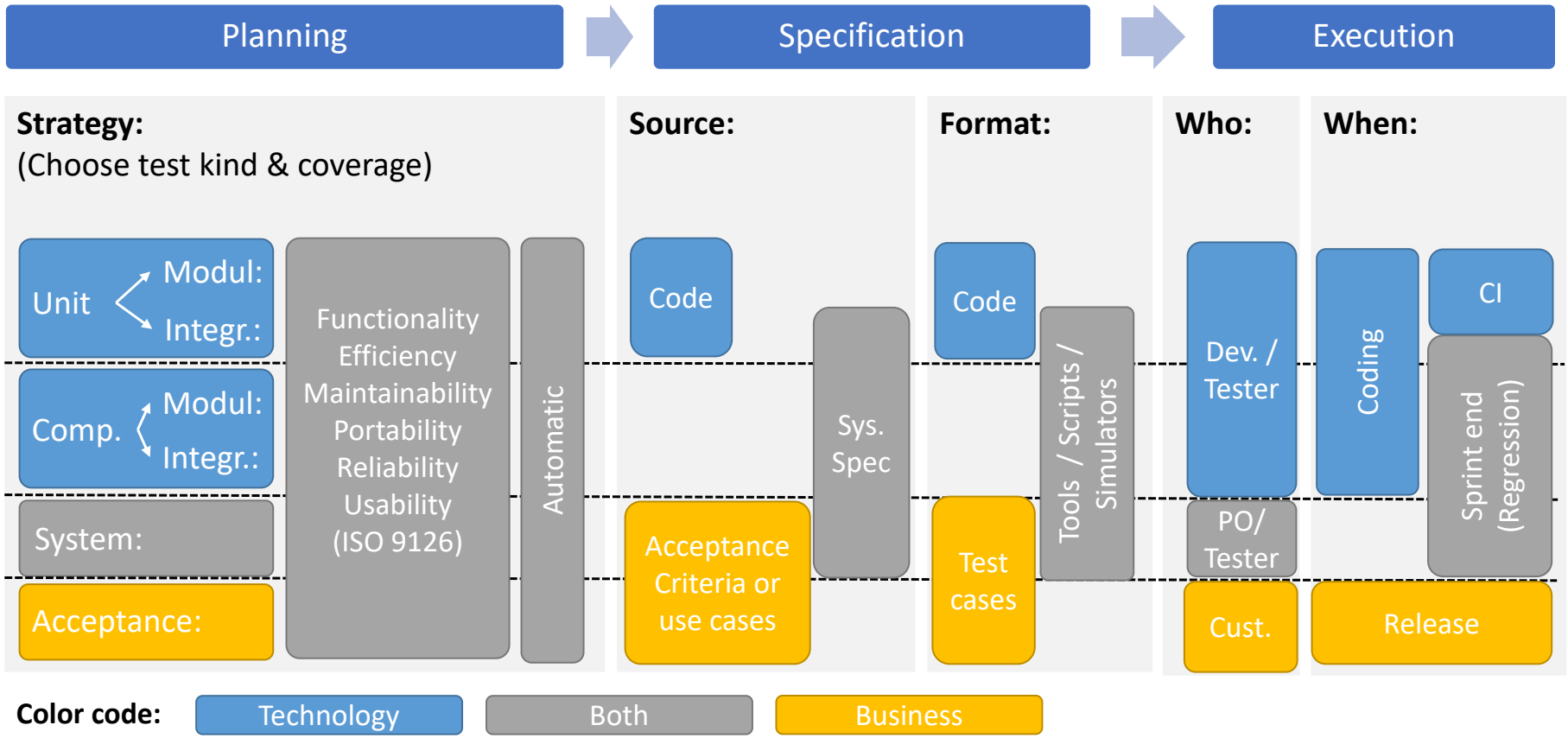
- Je nach Statistik findet man mit jeder Testart und jedem Review etwa 33% bis 67% der Fehler, welche in einer Software stecken.
- Für ein reales Beispiel (Betriebssystem mit 40 Mio. LOC) und einer durchschnittlichen Testausbeute von 50% pro Testart, würde dies so aussehen:

Test	Bugs found	Bugs remaining
-	-	2’000’000
Code reviews	1’000’000	1’000’000
Unit test	500’000	500’000
Integration test	250’000	250’000
System test	125’000	125’000

Alles herausholen!

- Mit keiner Testart und mit keinem Review findet man alle Fehler.
- Nur im Zusammenwirken der unterschiedlichen Techniken findet man ein Maximum an Fehlern.
- Vollständiges Testen ist schlicht nicht machbar.
- Es stellt sich deshalb die Frage: Welche Testarten und welche Tests haben die besten Chancen ein Maximum an Fehlern mit einem Minimum an Kosten aufzudecken? **=> Teststrategie.**

Testing im Überblick



Teststrategie

Festlegung der Teststrategie

Eigenschaften einer soliden Teststrategie:

- **Fachgerecht:** Passende Testziele, Testmethoden, Testarten.
- **Risikoorientiert:** Nicht alle Systeme bzw. Systemkomponenten sind gleich kritisch. Abdeckung entsprechend anpassen.
- **Wirtschaftlich:** In der Regel beschränktes Testbudget.

unter Berücksichtigung mehrerer Dimensionen:

- Qualitätskriterien des Produkts.
- Modul-Hierarchieebenen.
- Automatisierung.

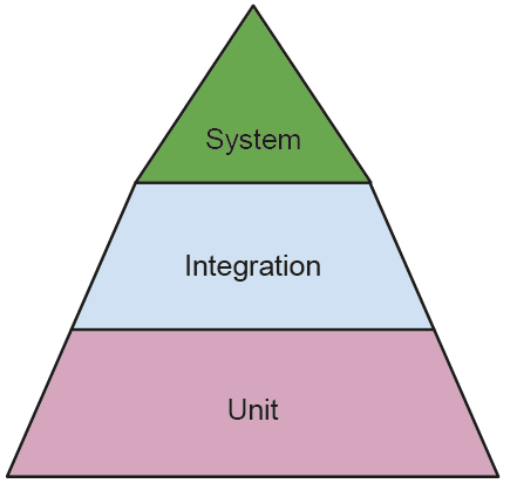
Qualitätskriterien des Produkts

Teilweise unterschiedliche Kriterien pro Modul.

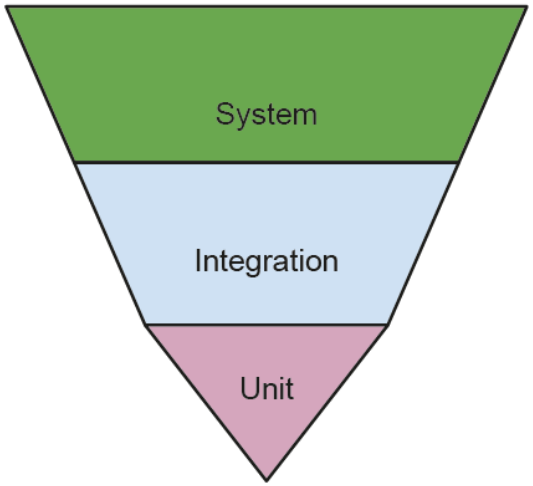
Quelle: Wikipedia (Sae1962 / CC BY-SA 4.0)

Modul-Hierarchieebenen

- Klassische agile Testpyramide:
  - Idee: Integrations- und Systemtests sind aufwändig, da schwer automatisierbar.
  - Unittests stellen Basis da.



- Alternative Testpyramide:
  - Idee: Mittels Tools und Virtualisierung sind Integrations- und Systemtests gut automatisierbar.
  - Unittest nur bei besonders kritischen Funktionalitäten.



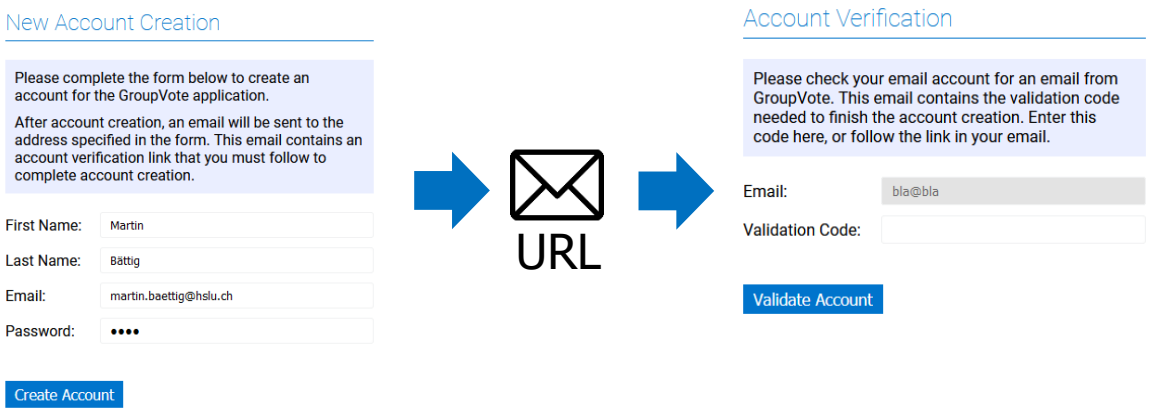
Quelle: Modulare Softwarearchitektur, Herbert Dowalil

Automatisierung

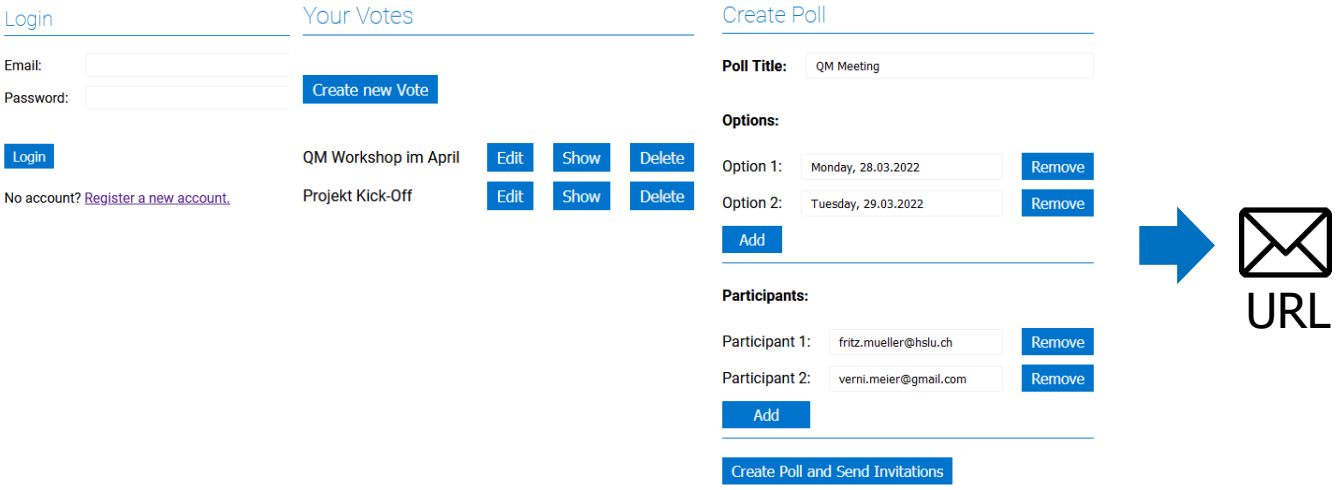
- Was lässt sich automatisieren?
- Steht der Nutzen im Verhältnis zum Aufwand?

Übung: Teststrategie für ein Doodle-Klon (Funktionsweise)

Erstellen eines Accounts:

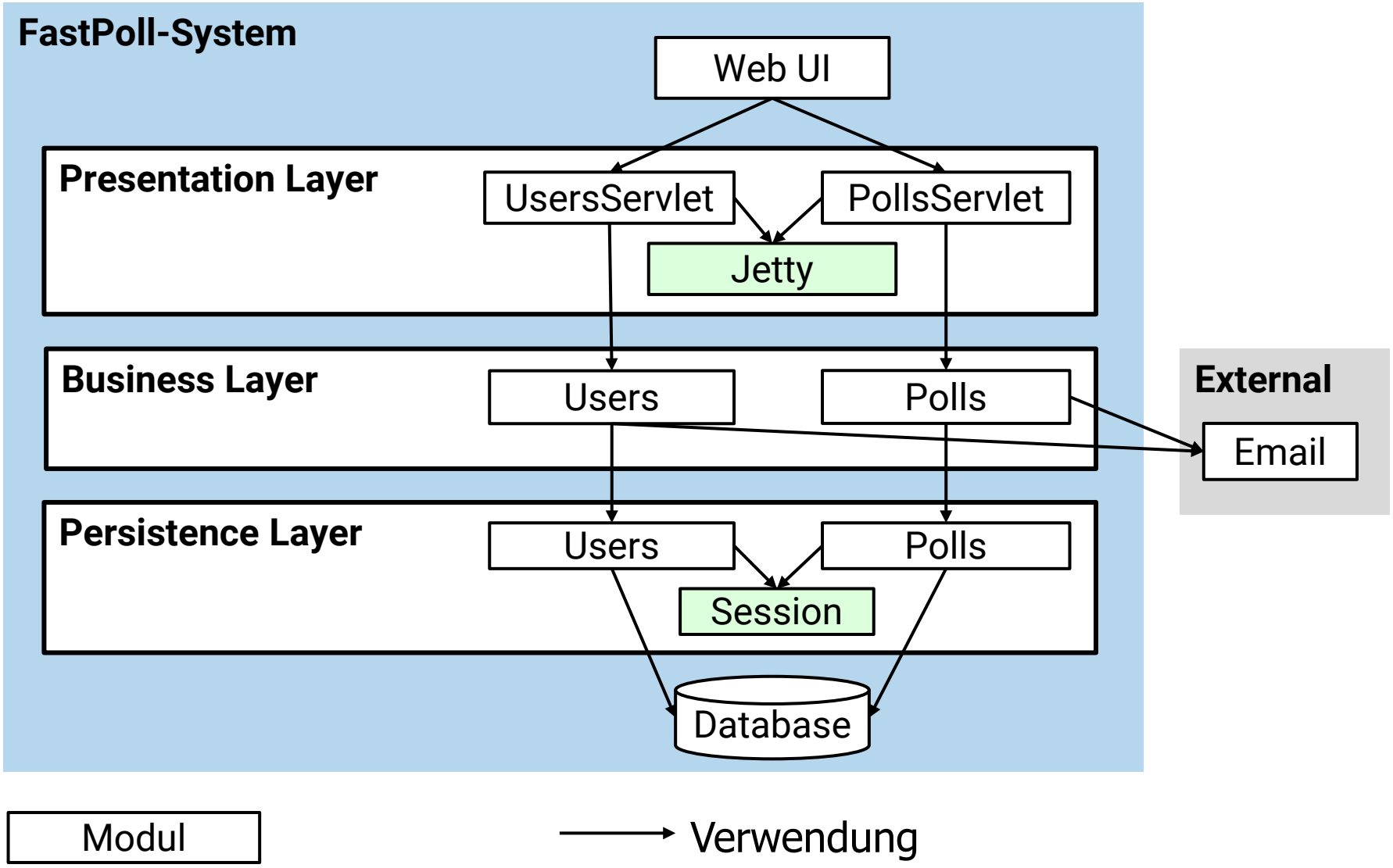


Erstellen einer Umfrage mit Einladung per Email:



Teilnahme an Umfrage

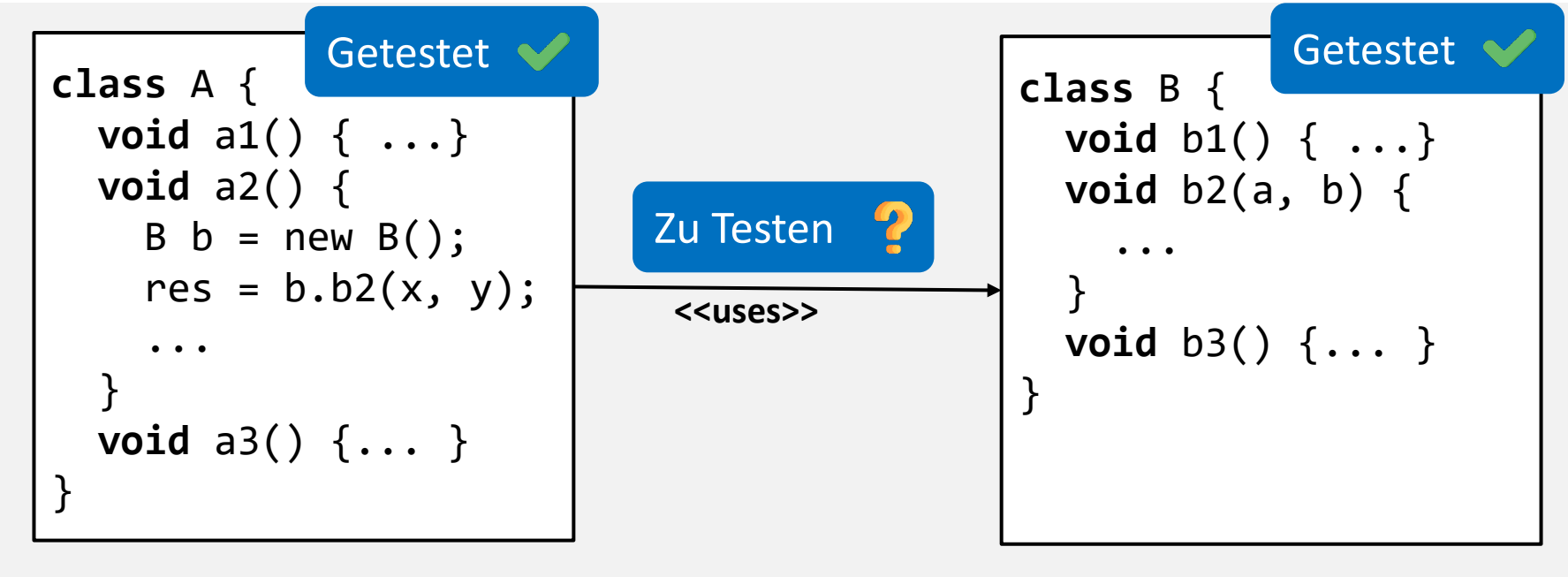
Übung: Teststrategie für ein Doodle-Klon (Architektur)



Integrationstest

Integrationstest

Prüfe **die Schnittstellen** und **das Zusammenspiel** der Systemkomponenten (Unit, Komponente, etc.):



Vorbedingung: Alle zu integrierenden Module sind bereits (soweit möglich) erfolgreich getestet.

Integrationstests testen folgendes:

- **Schnittstellen (direkte Abhängigkeiten)**
  - Objekt-Kompatibilität (Typen und Wertebereiche).
  - Aufruf-Sequenzen.
  - Wer validiert Inputs – der Aufrufende oder der Aufgerufene.
- **Datenabhängigkeiten (indirekte Abhängigkeiten)**
  - Für jede Komponente ermitteln, welche Datenabhängigkeiten bestehen und diese testen.
  - Gemeinsam genutzte Ressourcen (z.B. Dateien, Shared-Memory, Datenbanktabellen).
- **CallGraph-Abdeckung**
  - Bei Komponenten, die von verschiedenen Aufrufern genutzt werden, auch alle Aufrufvarianten testen.

Schwierigkeiten: Gemeinsam genutzte Ressourcen

- **Reentrance:** Eine Komponente wird mehrfach gleichzeitig ausgeführt.
- **Interrupts:** Eine Komponente A wird von einer Komponente B unterbrochen. Komponente B führt auf einer gemeinsam genutzten Ressource eine Schreiboperation aus. Anschliessend muss Komponente A wieder das korrekte Ergebnis zurückliefern.
- **Race Conditions:** Zwei Komponenten greifen (concurrent) ohne ausreichende Synchronisierung auf die gleiche Ressource zu.

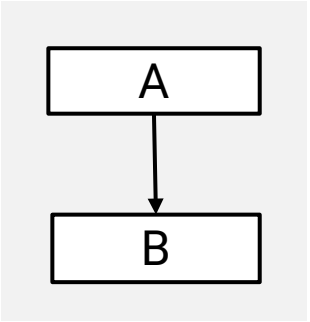
Code, der im statischen Testumfeld erfolgreich lief, kann in dynamischer Umgebung immer noch Fehler auslösen!

Stellvertreter

- Um ein Testobjekt zu testen, müssten alle Komponenten, die das Testobjekt benötigt, in der Testumgebung installiert und im Testlauf mit verwendet werden können.
- Bei inkrementellen Entwicklung ist es eher die Regel als die Ausnahme, dass nicht alle benötigten Komponenten fertig bzw. vorhanden sind.
- **Möglichkeit fehlende Komponenten durch Stellvertreter (z.B. Mock-Objekte oder Simulatoren) zu ersetzen.**
- Der Einsatz von Stellvertretern erlaubt auch **selektivere Tests** zu erstellen.

Integrationstestfälle entwerfen: Step-By-Step

1. Wechselwirkung analysieren: Welche Operationen von B ruft A auf? Diese Aufrufe sind zu testen.
2. Parametersätze der Aufrufe von B ermitteln (ggf. Äquivalenzklassenanalyse).
3. Testfallinput ermitteln: Welche Aufrufsequenzen von A sind notwendig um im Aufruf von B die in Schritt 2 ermittelten Parametersätze auszulösen?
4. Soll-Ist-Vergleich: Wie kann die Sollreaktion von B ermittelt werden? Direkt aus dem Output von A? Oder muss der Übertragungsweg mitgeschnitten werden?
5. Bei asynchroner Kommunikation: Zusätzlich Timing, Durchsatz, Kapazität, Performance sowie Datenübertragungsfehler prüfen.



Fallbeispiel: Integrationstest für Accounterstellung

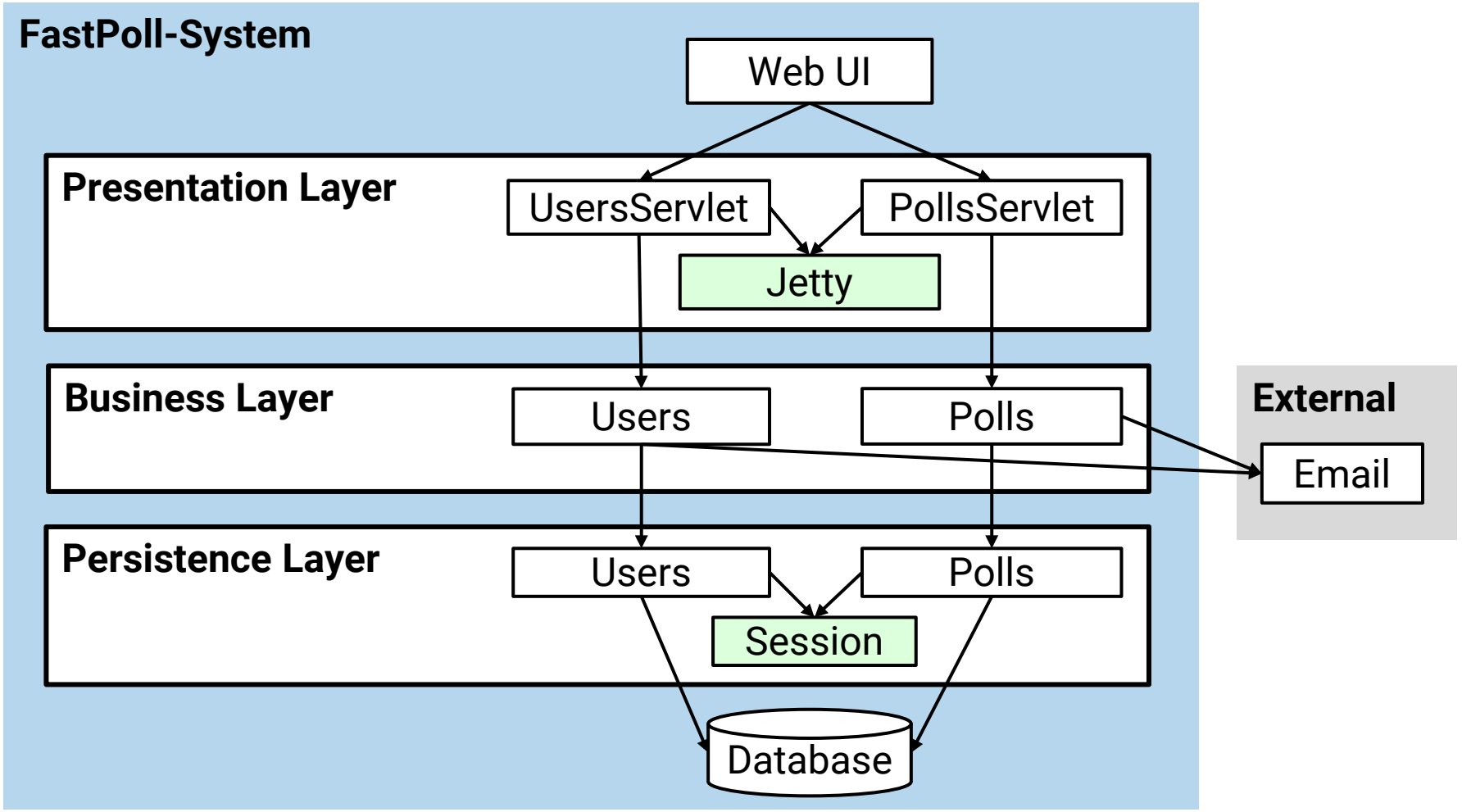
```
private static final String SENDER_ADDRESS = "noreply@fastpoll";
Properties emailProperties = ...; // email connection settings
AccountData accountData = ...; // data transfer object

public void createAccount() {
    accountData.store();
    Session session = Session.getInstance(emailProperties);
    try {
        Message message = new MimeMessage(session);
        message.setFrom(new InternetAddress(SENDER_ADDRESS));
        message.setRecipients(Message.RecipientType.TO,
            InternetAddress.parse(accountData.getEmail()));
        message.setSubject("FastPool: Account Verification");
        message.setText("Dear user, please use the code to...");
        Transport.send(message);
    } catch (MessagingException e) {
        throw new RuntimeException(e);
    }
}
```

Integrationsstrategien

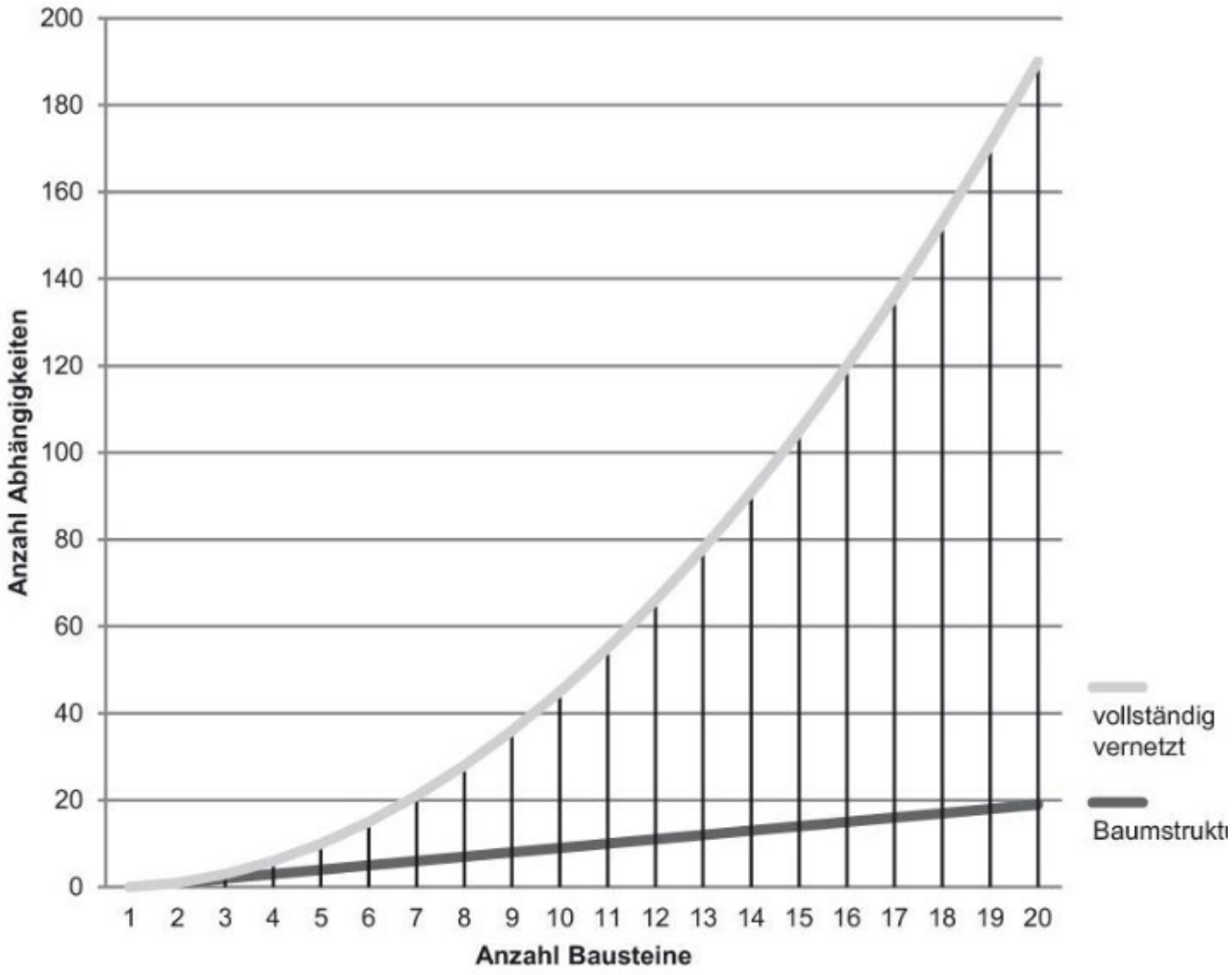
- **Bottom-Up the Small:** Kleinere Teilsysteme lassen sich bottom-up integrieren.
- **Top-Down the Controls:** Bei aufwändigen Kontrollstrukturen mit Hilfe von Stubs top-down vorgehen und dann die richtigen Komponenten integrieren.
- **Big-Bang the Backbone:** Was für den weiteren Testablauf benötigt wird in einem Aufwisch zusammenführen.
- **Continuous Integration:** Bei iterativ-inkrementeller Entwicklung neu dazu gekommenes laufend integrieren und testen.

Fallbeispiel: FastPoll-System ("Mini-Doodle")



Baustein **Modultest** → Verwendung: **Integrationstest**

Auswirkung von Modulvernetzung auf Abhängigkeiten

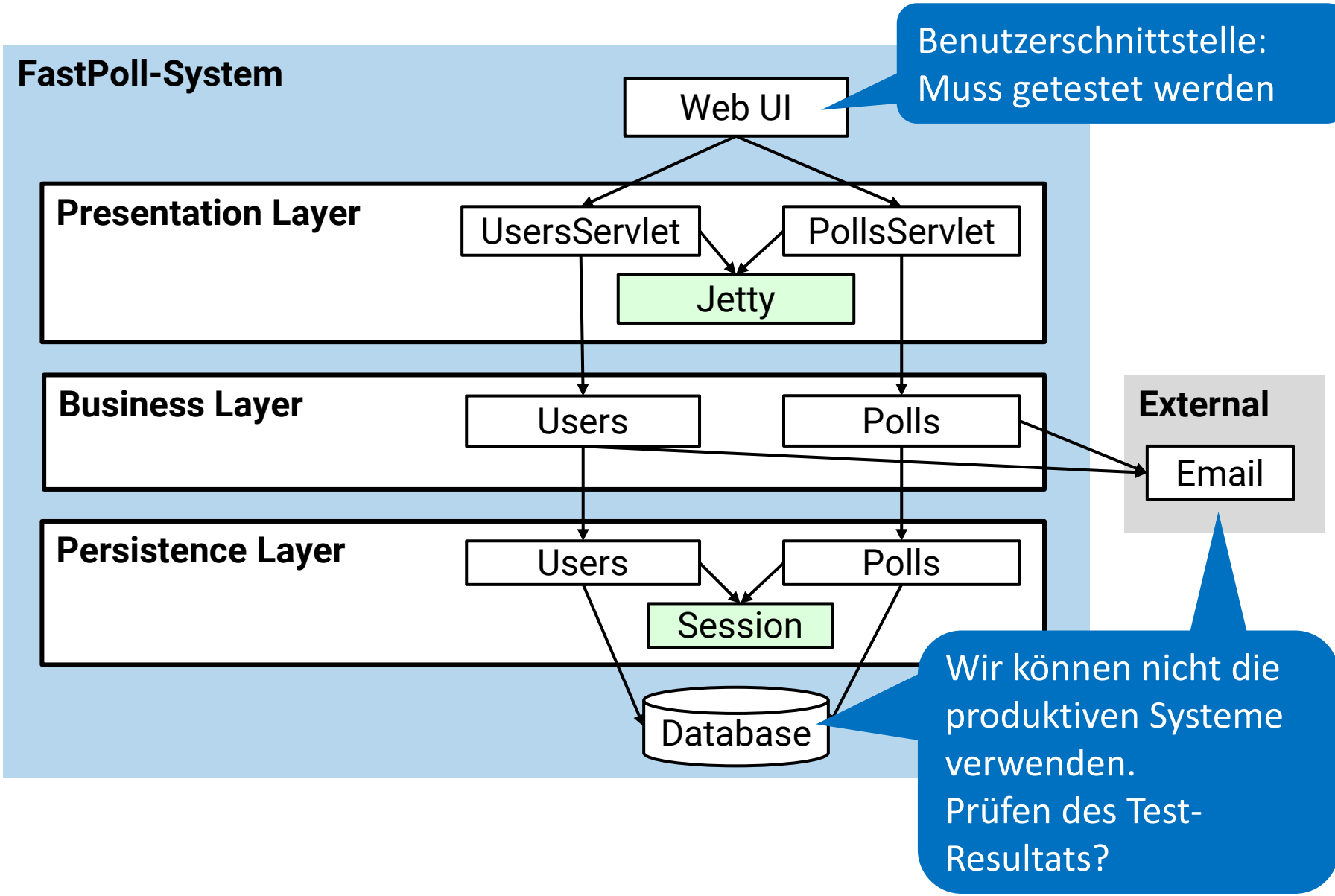


Systemtest

Systemtests

- Systemtests prüfen die **gesamte Wirkungskette im Softwareprodukt**, also Aspekte, die mit Unit-Tests und Integrationstests nicht abgedeckt werden.
- Ein potenziell lieferbares Softwareprodukt muss in der Regel:
  - ausserhalb der Entwicklungsumgebung lauffähig sein,
  - über eine Bedienschnittstelle verfügen
  - mit anderen Applikationen und Systemen interagieren.
- Es sind also Testfälle nötig, die in einer Testumgebung ablaufen, welche der **späteren Einsatzumgebung möglichst nahe** kommt.

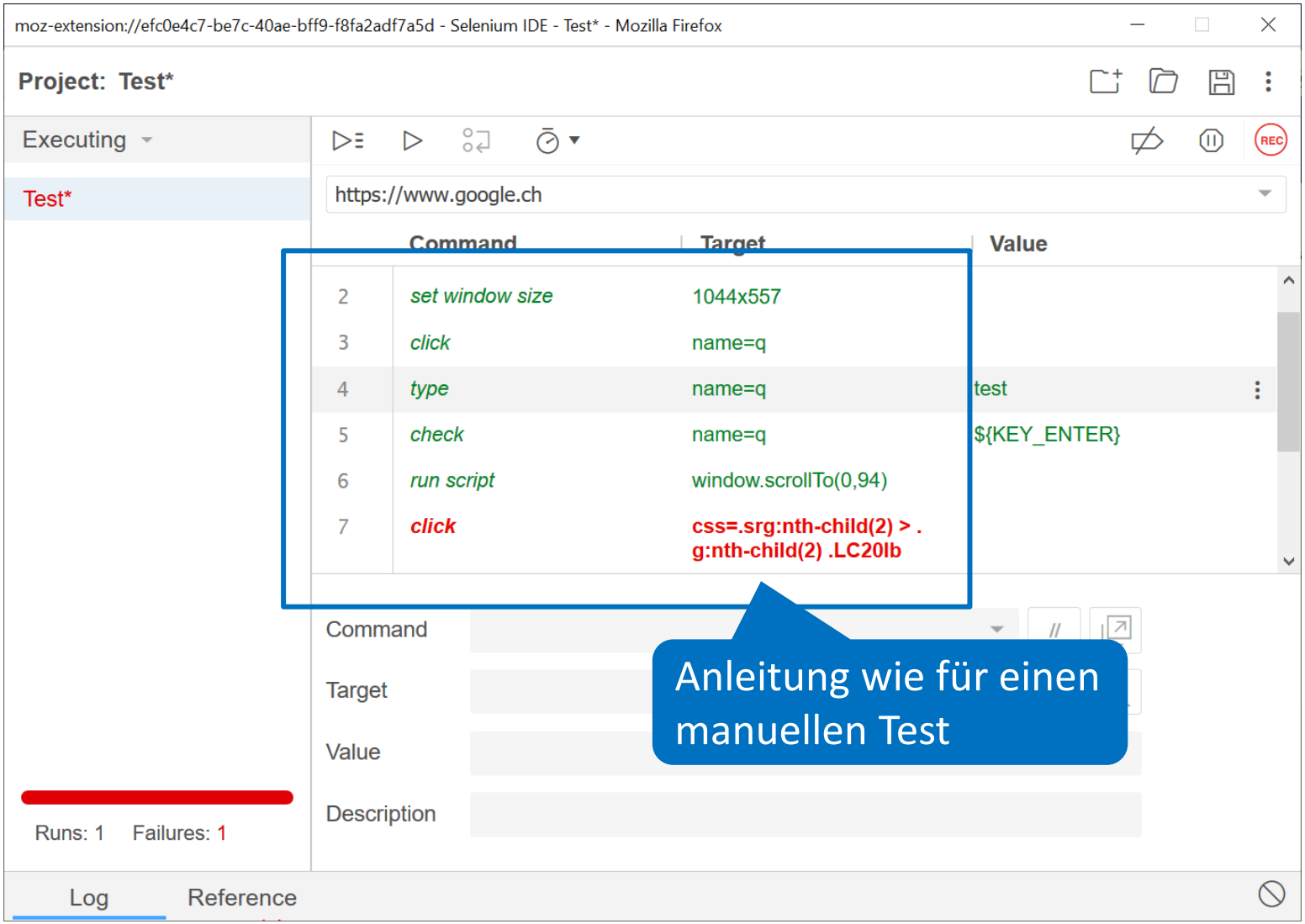
Fallbeispiel: FastPoll-System ("Mini-Doodle")



Herleitung der Systemtestfälle

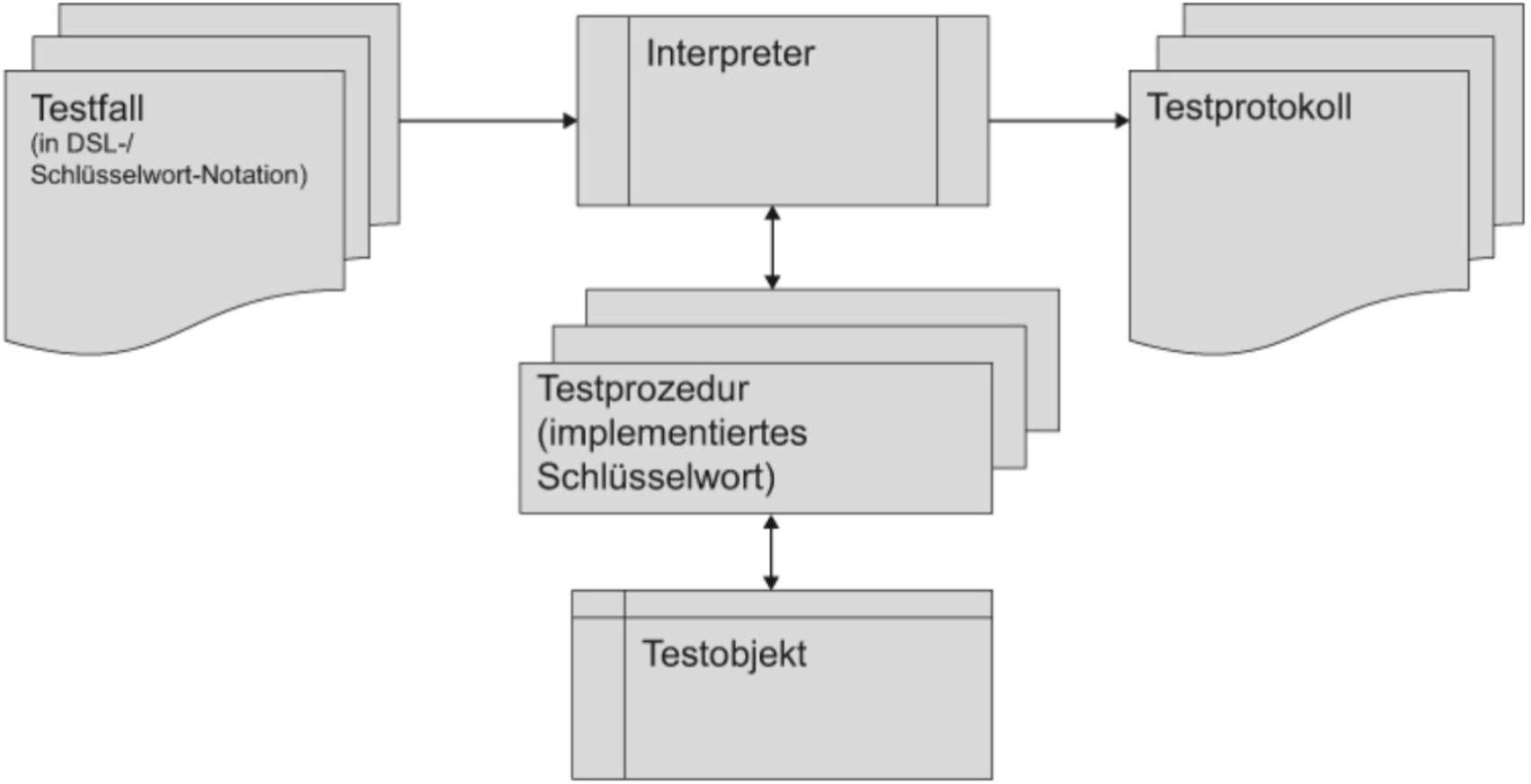
- Systemtestfälle können grundsätzlich abgeleitet werden aus:
  - den Anforderungen.
  - zugehörigen, detaillierteren Use-Case-Beschreibungen.
  - den Akzeptanzkriterien.
- Nicht-funktionale Anforderungen werden oft wenig explizit festgehalten, entsprechend kommen auch nicht funktionale Tests zu kurz:
  Last- / Performance- / Stress- / Security- / Robustness-Tests sind ebenfalls wichtige Systemtests.
- Wie beim Test-First-Ansatz auf Unit-Test-Ebene fördert auch das Formulieren der Systemtests das Verständnis der Anforderungen und bringt Unklarheiten und Inkonsistenzen frühzeitig zu Tage.

Test von Benutzerschnittstellen



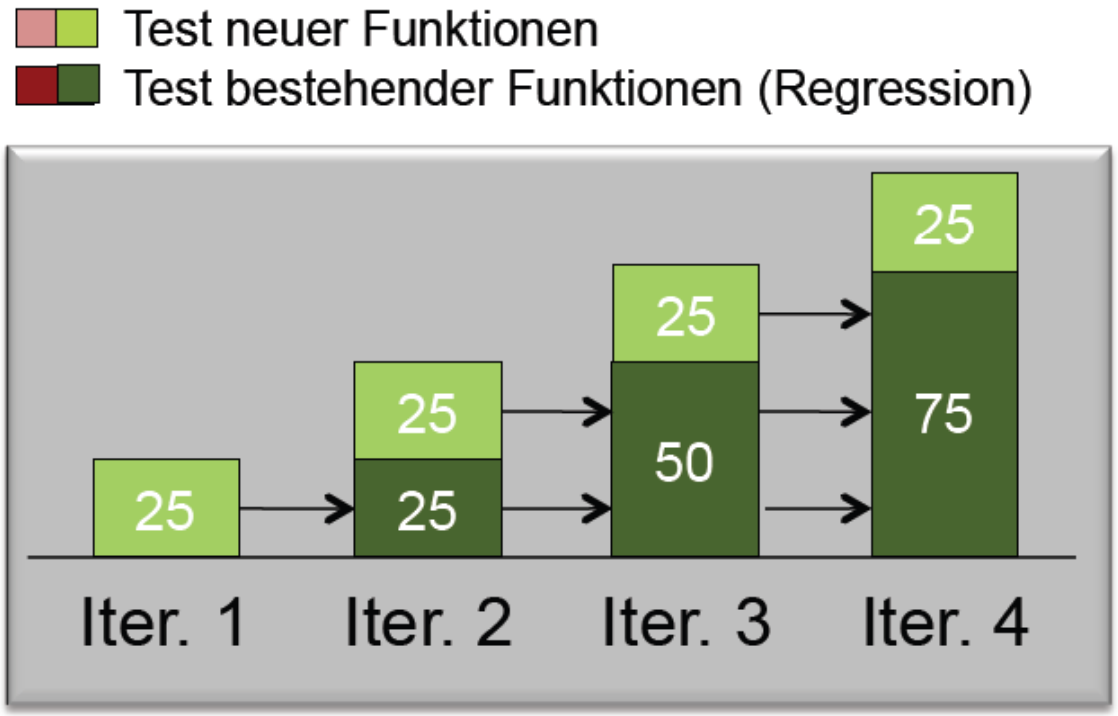
Entkopplung mittels dreischichtiger Testarchitektur

- Entkopplung der Testumgebung (z.B. Timeouts).
- Entkopplung der Testschnittstelle (verschiedene Oberflächen).



Regressionstests

– Bereits realisierte und getestete Features müssen nach jeder Änderung / Erweiterung der Software erneut getestet werden (➔ Regressionstest).



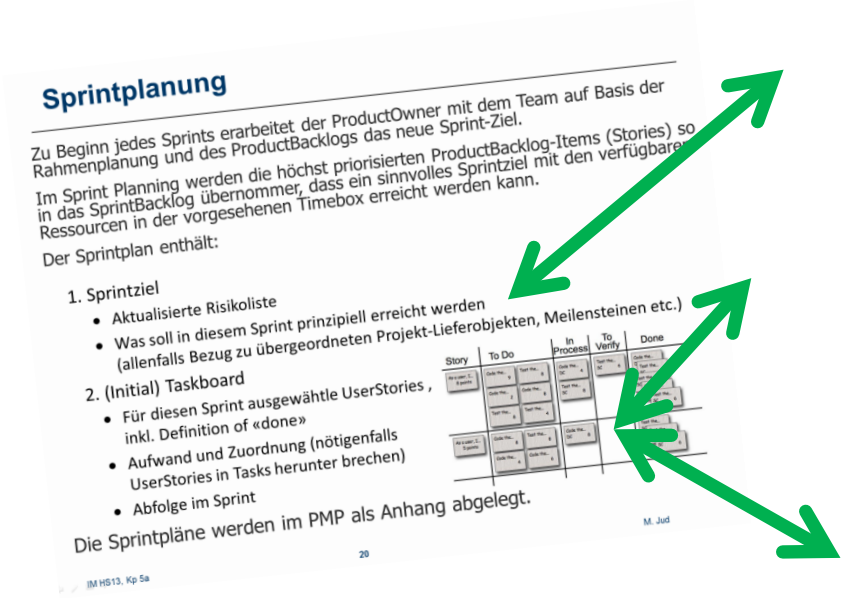
Bildquelle: https://www.slideshare.net/swissq/scrum-rocks-testing-sucks-de

Dokumentation

- Integrations- und Systemtests werden **als Regressionstest** auch in weiteren Entwicklungsschritten **immer wieder gebraucht** und genutzt.
- Damit Integrations- und Systemtests wiederholbar sind, müssen sie **nachvollziehbar dokumentiert** werden.
- Wichtige Bestandteile der Beschreibung eines Testfalls sind:
  - die Vorbedingungen für die Testausführung,
  - die Handlungen und Eingaben für die Durchführung des Tests,
  - die erwarteten Ergebnisse und Nachbedingungen.
- Durch Automatisierung wie bei Unittests sind auch Integrations- und Systemtests am besten dokumentiert.
- **(Teil-)Automatisierung lohnt sich!**

Testing in Scrum

Testplanung und -Organisation



- Ein Sprintziel muss es sein, die Fertigstellung von Features zu mit Hilfe von Abnahmetests prüfen zu können.
- Konsequenz davon ist, dass im Taskboard zu den in diesem Sprint geplanten User-Stories auch die erforderlichen Integrationstests, Systemtests und Abnahmetests (-> Sprint-Reviews) eingeplant werden müssen.
- Natürlich müssen auch die jedes Mal zunehmenden Regressionstests eingeplant werden.

Testaufgaben im Scrum-Team

- **Im Planning-Meeting:** Abschätzen wieviel Zeit zum Testen von User-Stories benötigt wird und dafür sorgen, dass diese bei der Aufwandschätzung berücksichtigt werden
- **Während dem Sprint:** Tests möglichst rasch durchführen, Anhäufung von pendenten Testfällen vermeiden
- **Product-Owner:** Nach Sprint aber vor Abnahme: Führt Akzeptanztests aus (Ist das Sprintziel erreicht?)
- **Sprint-Review:** getestete Features demonstrieren => inkrementelle Validierung
- **Retrospektive:** Wo waren die Stolpersteine aus Tester-Sicht, was lief besonders gut? Was kann man neu/anders machen.

Zusammenfassung

- V-Modell - Verifizieren und validieren.
- Agile Testing: supporting the Team / critique the Product.
- Poking around is a waste of time.
- Grössere Testausbeute bei Verwendung verschiedener Testarten.
- Integrationstest: Fokus auf Schnittstellen und Zusammenspiel der Komponenten.
- Systemtest: Fokus auf gesamte Wirkungskette des Produkts.
- Regression Testing: nach jeder Änderung / Erweiterung die Software erneut testen: nur machbar wenn dokumentiert / automatisiert.

Auftrag: Testing im Projekt "verteiltes Logging-System"

- Erstellen Sie eine Teststrategie für das Logger-Projekt. Diese ist im Kapitel 8 der Architekturbeschreibung festzuhalten. Umfang ca. ½ - 1 A4-Seite.
- Teil der Teststrategie sind mindestens drei manuelle Systemtests sowie automatisierte Unittests in geeignetem Umfang.
  - Ideen für Systemtests erhalten Sie aus den Akzeptanzkriterien der Userstories.
- Erstellen Sie für die drei manuellen Systemtests ein Drehbuch.
- [Ab Sprint 3] Separate Testberichte pro Sprint.
- Zeit für Testen (Regression!) in Sprintplanung mitplanen.

Quellen / Literatur

- Testen in Scrum Projekten von Tilo Linz, dpunkt-Verlag.
- Agile Testing von Lisa Crispin & Janet Gregory, Addison-Wesley.
- Testing Object-Oriented Systems von Robert Binder, Addison-Wesley.
- Scrum Rocks, Testing Sucks?! von Adrian Stoll SwissQ, Testing Day 2011.

Fragen?