



Agiles Software- Entwicklungsprojekt

Prof. Dr. Gerd Beneken

Kapitel 07

Toolbox zu Qualität und Produktrisiken

Spezielle
Inhalte für
Agile Projekte
am Ende des
Foliensatzes

Was ist Qualität?

Begriffe aus den Normen

- Software quality: Capability of a software product ***to satisfy stated and implied needs*** when used ***under specified conditions***

[ISO 25000-2005]

- D.h. ***Ohne*** festgelegte (Qualitäts-) ***Anforderungen*** (Erfordernisse) an Eigenschaften eines Produktes ist ***keine Qualitätsdefinition*** und -messung möglich
- Es gibt keine allgemeine Definition von „hoher“ Qualität



Agiles Software- Entwicklungsprojekt

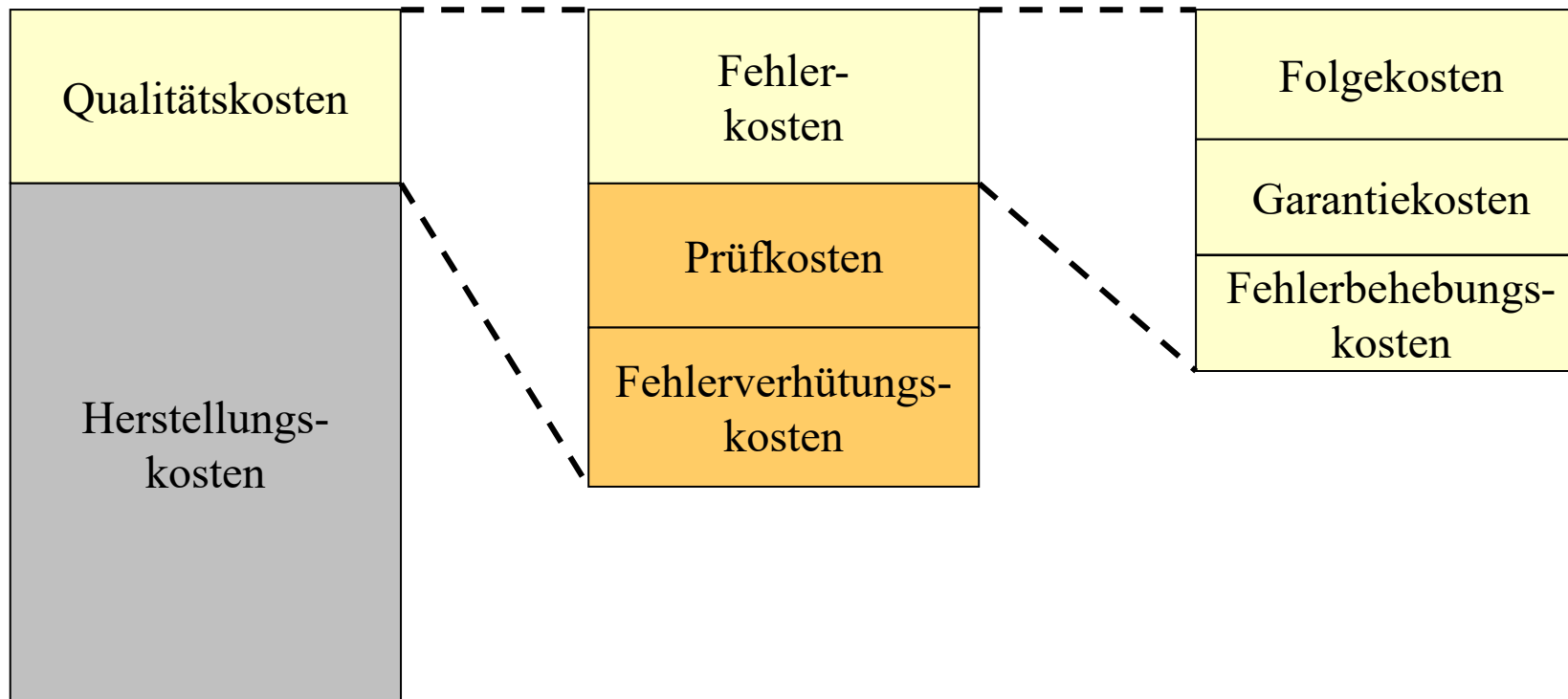
Prof. Dr. Gerd Beneken

Kapitel 7.1

Qualitätskosten

*Siehe Skriptum
SEP*

Qualitätskosten



Qualitätskosten Zusammensetzung

Prüfkosten

= Analytische Qualitätssicherung

- Durchführung Reviews
- Definition Testkonzept, Testfälle
- Durchführung Tests
- Werkzeuge, Testinfrastruktur, Testumgebung

Fehlerverhütungskosten

- = Konstruktive Qualitätssicherung
- Definition eines Entwicklungsprozesses (Checklisten, Dokumentvorlagen, Prozess)
- Definition eines Testprozesses (Checklisten, Dokumentvorlagen, Prozess)
- Ausbildung der Entwickler
- Kaffeeküche (!) = informelle Kommunikation

gegen

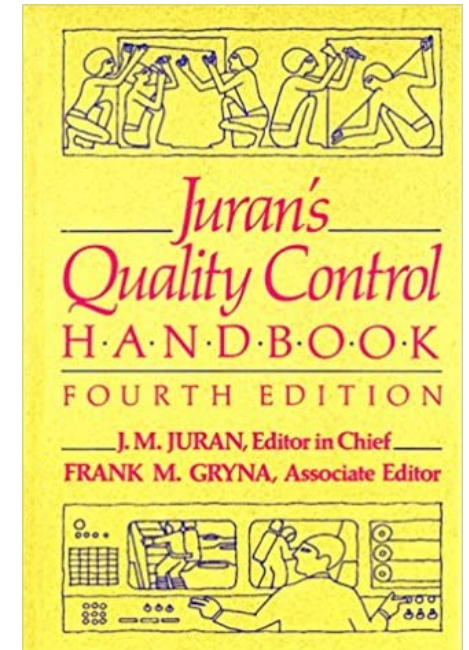
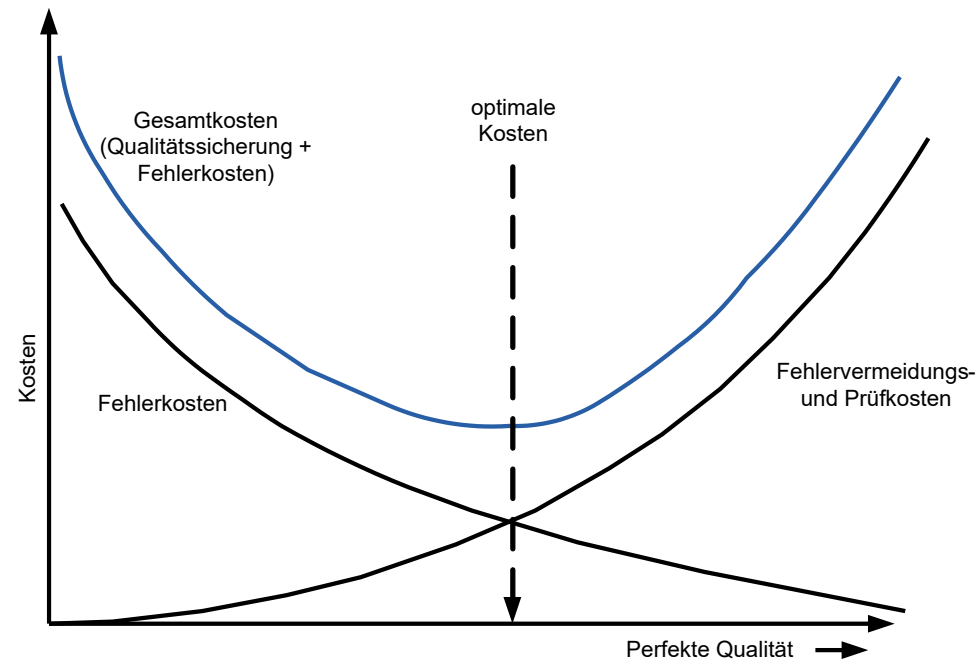
Fehlerkosten

- Folgekosten
 - Z.B. Arbeitszeitausfall / entgangener Umsatz bei Produktionsstillstand
 - Z.B. Stornokosten bei falschen Berechnungen
 - Z.B. Prämienausfälle bei falschen Prognosen
- Fehlerbehebungskosten
 - Debugging, Fixing, Nachtest, einspielen eines Hotfixes

Spannungsfeld Qualitätskosten

Ein Optimierungsproblem:

- „Quality is free“ (gutes QM senkt die Entwicklungskosten)
- Zu viel QM steigert die Entwicklungskosten wieder



Risiko-Orientiertes Vorgehen

- Risiko = *Möglichkeit, dass Ereignis mit negativen Auswirkungen eintritt*
= *Eintrittswahrscheinlichkeit * Schadenhöhe*
- Auslöser von Ereignissen
 - Menschliches Handeln (= unvorhersehbar)
 - z.B. Programmierfehler, Fehler im Fachkonzept, Fehler in der Planung
 - z.B. Hacker-Angriff, Sabotage
 - Technische Ursachen / Unfälle / Katastrophen (= unvorhersehbar)
 - Z.B. Plattencrash, Router kaputt, Kabel gebrochen, ...
 - Z.B. Brand im Rechenzentrum, Überflutung, ...
- Risikotypen: *Produkttrisiken* und *Projektrisiken*

Schäden: Beispiele

/1

■ **Verlust von Menschenleben**

- Software / System führt dazu, dass Menschen schwer verletzt/getötet werden
- Jetzt: Staatsanwaltschaft ermittelt wegen fahrlässiger Tötung
- Häufig sehr hohe Geldstrafen evtl. Haftstrafen

■ **Schwere Schäden an der Umwelt**

- Software / System verursacht Umweltschäden, z.B. Verschmutzungen von Luft / Wasser /Boden
- Kosten, um diese Schäden wieder zu „reparieren“

■ **Image-Schäden**, “Marke“ der Firma ist weniger wert (z.B. Wenn Kundendaten im Internet auftauchen ...)

- Kosten für Marketing Maßnahmen um Marke wieder aufzuwerten
- Umsatzausfall wenn Kunden sich für eine andere Versicherung entscheiden ...

Schäden: Beispiele

/2

- **Strafen** von Gerichten, von Aufsichtsbehörden, oder Partnern
 - Verstöße gegen IT-Compliance (Datenschutzgesetz, ...)
 - Weil Gesetzesänderungen nicht rechtzeitig umgesetzt sind
 - Vertragsstrafen von Partnern
- **Umsatzausfälle**
 - Wenn Systeme sind nicht verfügbar sind
(z.B. weil Backup eingespielt werden muss, weil System Durchsatz nicht schafft)
 - Wenn Fehler die Verwendung wichtiger Funktionen ausschließen
 - Wenn Nutzerakzeptanz fehlt (Robustheit, Performance, Usability)
 - Wenn Antwortzeiten zu groß sind

Schäden: Beispiele

/3

- **Vermögensschäden** (durch zusätzlichen Aufwand, Cacheflow, ...)
 - Wenn Systeme falsch rechnen
(z.B. Prämienneuberechnung bei Versicherung + Abwicklung)
 - Wenn Batches nicht im Zeitfenster fertig werden
(z.B. Rechnungen mehrere Tage zu spät)
 - Wenn Systeme Daten zerstören (z.B. Neuerfassung der Tagesdaten)
 - Wenn unnötige Bescheide, ungültige Rechnungen an Kunden versendet werden
- **Reparaturkosten**
 - Fehlerdiagnose
 - Erstellung, Test und Einspielen eines Patches
 - Änderung der Dokumentation

Wo sind Fehler wahrscheinlich?

- Bei **umfangreichen Änderungen**
 - = Änderungen am Code, der Parametrierung, dem Datenbankschema, der Datenbasis
 - Neu entwickelte Module
 - (Stark) geänderte / restrukturierte Module
- Bei Änderungen in der Umgebung (**Anpassungen**)
 - = Änderungen an Plattform, Bibliotheken / Frameworks, Programmiersprachen-Version, Hardware, Betriebssystem, ...
 - = Änderungen an Konfiguration wie Netzwerkinfrastruktur / -konfiguration
 - = Schnittstellen zu Nachbarsystemen, die neu released wurden
- Bei reparierten Fehlern und Optimierungen
- Stellen mit (historisch bedingter) **hoher Fehlerdichte***



Agiles Software- Entwicklungsprojekt

Prof. Dr. Gerd Beneken

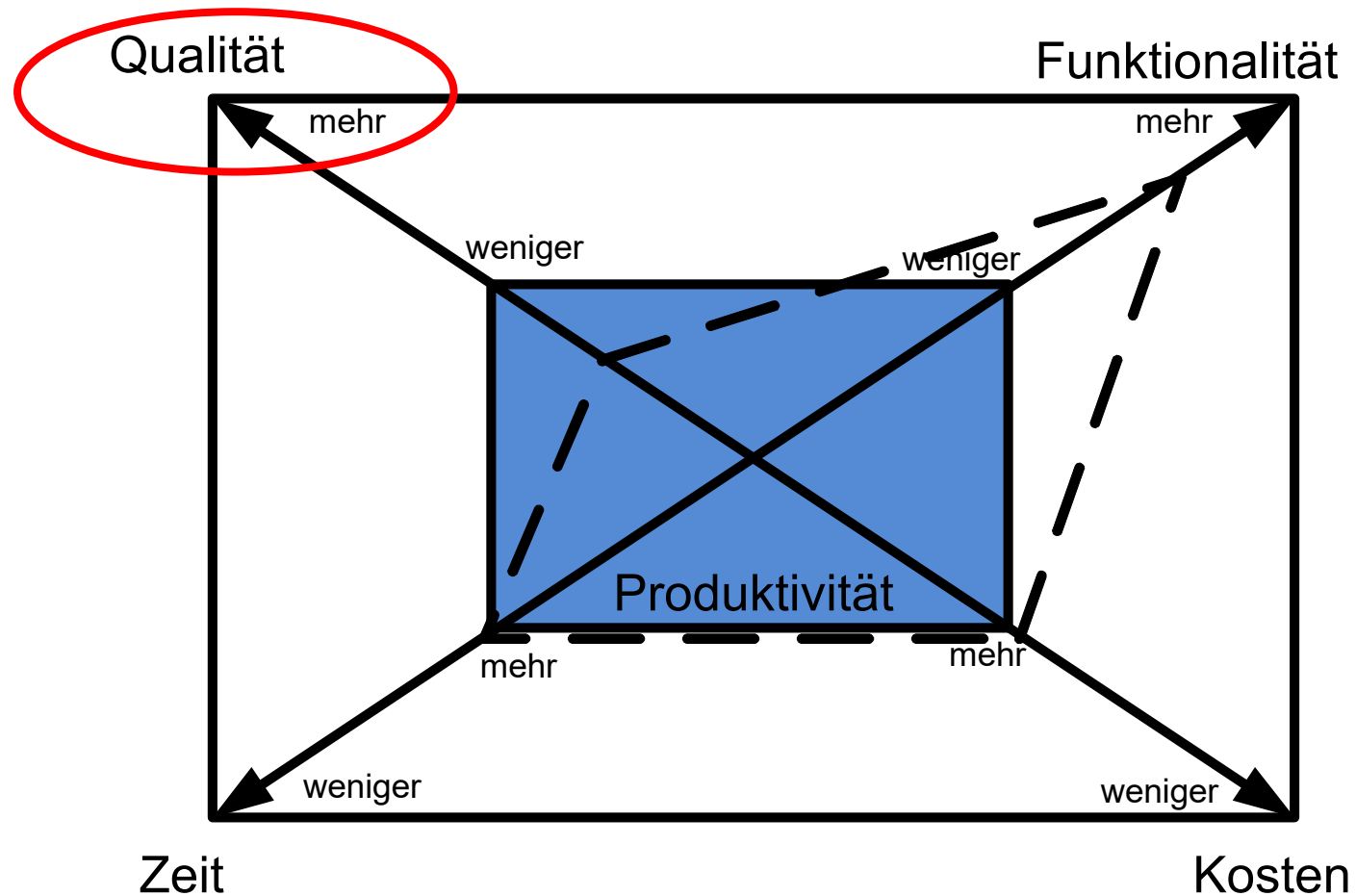
Kapitel 7.2

Qualitätssicherung: Vertrauen erhöhen

*Siehe Skriptum
SEP*

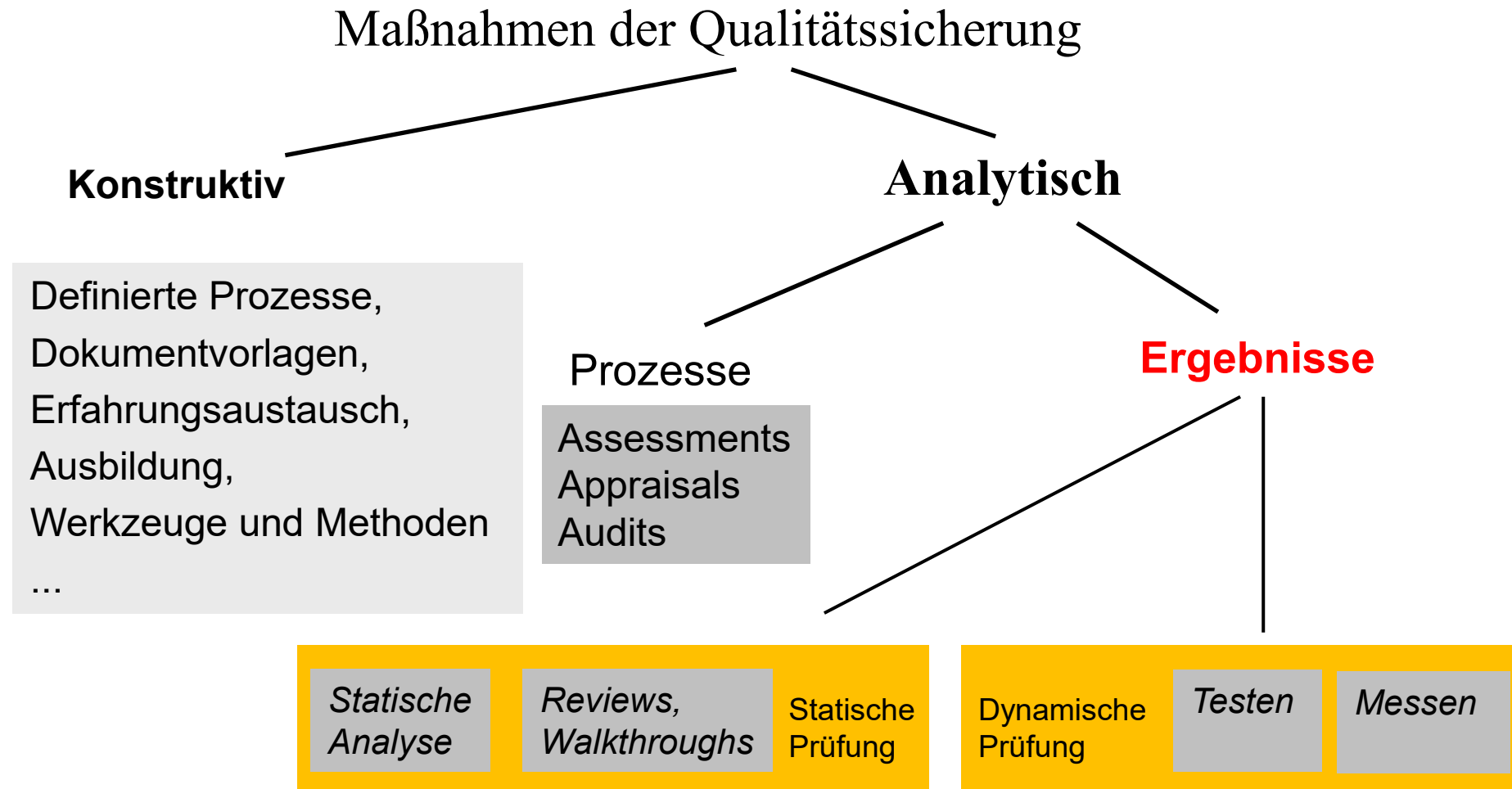
Teufelsquadrat nach Sneed

Sie ziehen hier!



Qualitätssicherung, Fokus hier auf analytische Maßnahmen

Analytische und Konstruktive Maßnahmen



Analytische Maßnahmen der Qualitätssicherung

Statische Verfahren (Anschauen/Analysieren)

- Prozesse
 - Audits / Assessments / Appraisals
(Interviews + Inspektion der Prozess Dokumentation, z.B. nach SPICE, CMMI)
- Anforderungen, Modelle, Dokumente, Code, ...
 - **Walkthroughs** (informell = durchgehen, drüberschauen)
 - **Peer Reviews** (mit definiertem Prüfprozess: Rollen, Meetings, ...)
 - **Inspektionen**, Methodisch ausgearbeitete Review (z.B. ATAM)
- Code: Statische Analyse
 - **Metriken**
 - Untersuchung auf **Fehlermuster**, Codingstyles der Quelltexte
 - Programmverifikation (Beweis von Programmeigenschaften)

*) Architecture Tradeoff Analysis Method, Verfahren des Software-Engineering Institute, <http://www.sei.cmu.edu/reports/00tr004.pdf>

Analytische Maßnahmen der Qualitätssicherung

Dynamische Verfahren (= Ausführen der SW, Testen und Messen)

■ Software / Code

- Blackbox Test
(Zufallstest, Grenzwertanalyse, Anwendungsfall-Test, ...)
- Whitebox/Glassbox Test
(Datenfluss-orientiert, Kontrollfluss-orientiert, ...)
- Lasttest, Stresstest, Langzeitmessungen *)
- Usability/Benutzbarkeits Test
- Zuverlässigkeitstest, Verfügbarkeitstest, Robustheitstest

■ Modelle / Prototypen

- Simulation (Modell des Systems wird getestet und analysiert)
- Model Checking (Zustandsraum der Software wird analysiert)

*) z.B. in der Testliteratur vernachlässigt aber wichtig, z.B. mit Software EKG,
<http://www.spring-framework.org/de/veroeffentlichungen>

Heuristiken zur Testkonzeption und -planung

- Maximieren der **Vielfältigkeit**: Verschiedene Testverfahren und Überdeckungsmaße einsetzen
- **Überspezifikation vermeiden**: Mit informellen Testspezifikationen beginnen, Testern Freiheiten lassen (für unvorhergesehene Probleme, situationsabhängig)
- Testen gegen die Absicht von Anforderungen, nicht gegen deren wörtliche Aussage
- **Zusammenarbeit stärken**: mit Fachabteilung, Betriebsorganisation, IT-Betrieb
- **Testbarkeit einbauen** / einfordern: Code und Architektur so bauen, dass diese testbarer werden (siehe entsprechende Vorlesung)
- Testkonzept-Dokument:
Gutes „Toner zu Information“ Verhältnis sicherstellen



Agiles Software- Entwicklungsprojekt

Prof. Dr. Gerd Beneken

Kapitel 7.3

Automatisierte Analytische QS Build Pipeline

Kontinuierliche Integration nach Fowler

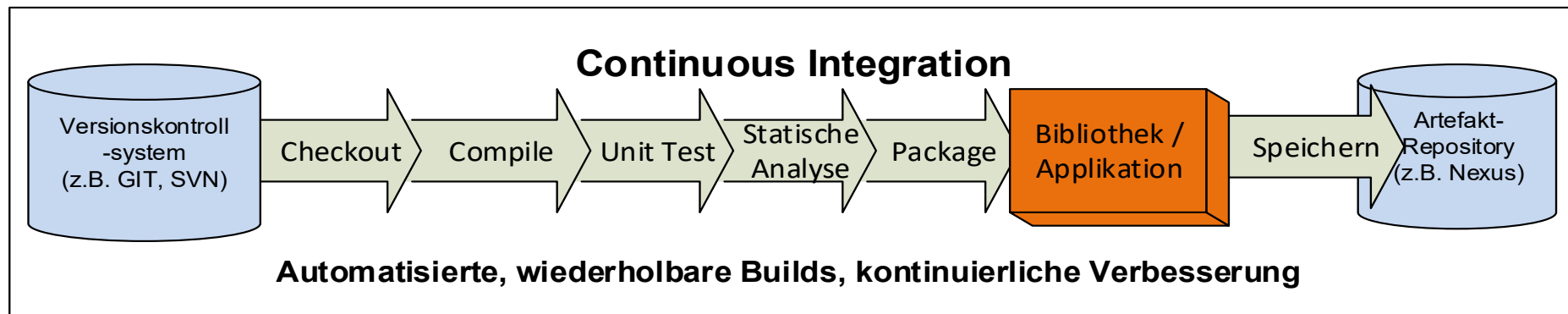
An important part of any software development process is getting **reliable builds** of the software. Despite its importance, we are often surprised when this isn't done. We stress a **fully automated** and **reproducible build, including testing**, that **runs many times a day**. This allows each developer to integrate daily thus reducing integration problems.

[Martin Fowler]

Continuous Integration doesn't get rid of bugs, but it does make them dramatically easier to find and remove [M. Fowler]

Continuous Integration: Automatisierung

Computer übernehmen Routinetätigkeiten, schnelles Feedback zur Qualität



- Checkout aus Versionskontrolle dann **automatisierter Build mit einem Knopfdruck** Inklusive: Datenbankschema (-Migration), Testdaten, ggf. Infrastruktur wie Docker-Container
- **Automatisiertes Testen**, soweit möglich und sinnvoll (TDD)
- Automatisierte Qualitätsmetrik / Statische Analyse
- Werkzeuge: z.B. Gitlab CI, Jenkins, Teamcity,..., Teamscale, Sonar

Bei uns Gitlab CI (da kein Zusatzaufwand)

GitLab Pipelines · vorlesungen / apt2019

https://inf-git.fh-rosenheim.de/vorlesungen/apt2019/pipeline

GitLab Projects Groups Activity Milestones Snippets Search or jump to...

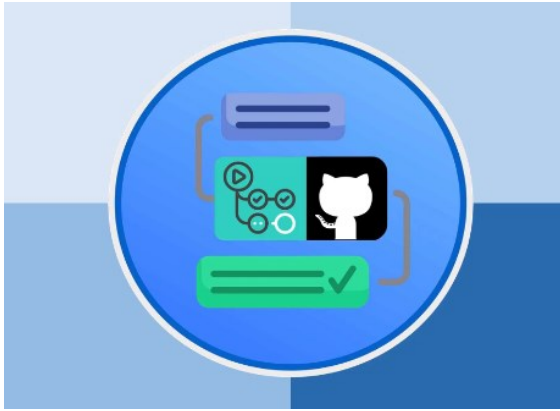
vorlesungen > apt2019 > Pipelines

All 26 Pending 0 Running 0 Finished 26 Branches Tags Run Pipeline Clear Runner Caches CI Lint

Status	Pipeline	Triggerer	Commit	Stages	Duration	Time
passed	#22541 latest	tach	master - 06148039 success Build	✓ ✓	00:01:40	14 hours ago
failed	#22540	tach	master - d7f256b5 tach	✓ ✗	00:01:40	14 hours ago
failed	#22539	tach	master - 8b0c10af tach	✓ ✗	00:01:40	14 hours ago
failed	#22538	tach	master - 31ab885a tach	✓ ✗	00:01:39	14 hours ago
failed	#22537	tach	master - a0ff29f2 tach	✓ ✗	00:01:41	14 hours ago
failed	#22536 yml invalid error	tach	master - addc649a Update .gitlab-ci.yml			15 hours ago

https://inf-git.fh-rosenheim.de/vorlesungen/apt2019/pipeline_schedules

Alternative CI-Werkzeuge



Github Actions



Jenkins



TeamCity

SonarQube zur kontinuierlichen Qualitätsüberwachung

The screenshot displays the SonarQube web interface at the URL <https://inf-sonarqube.th-rosenheim.de/projects/favorite>. The interface includes a navigation bar with tabs for Projects, Issues, Rules, Quality Profiles, and Quality Gates. A search bar is present in the top right. The main content area shows a list of projects under the 'My Favorites' tab. Two projects are listed: 'BenekenGerd_Beispiele' (Failed) and 'BenekenGerd_Studienarbeit01' (Passed). Each project card displays various quality metrics: Bugs, Vulnerabilities, Hotspots Reviewed, Code Smells, Coverage, Duplications, and Lines. The 'BenekenGerd_Beispiele' project has 0 bugs (A), 0 vulnerabilities (A), 0.0% hotspots reviewed (E), 30 code smells (A), 0.0% coverage, 17.8% duplications, and 1.2k lines (S). The 'BenekenGerd_Studienarbeit01' project has 1 bug (E), 0 vulnerabilities (A), 0.0% hotspots reviewed (E), 8 code smells (A), 26.4% coverage, 0.0% duplications, and 78 lines (XS). The left sidebar contains filters for Quality Gate (Passed, Failed) and Reliability (A, B, C, D, E).

Project	Status	Bugs	Vulnerabilities	Hotspots Reviewed	Code Smells	Coverage	Duplications	Lines
BenekenGerd_Beispiele	Failed	0 (A)	0 (A)	0.0% (E)	30 (A)	0.0%	17.8%	1.2k (S)
BenekenGerd_Studienarbeit01	Passed	1 (E)	0 (A)	0.0% (E)	8 (A)	26.4%	0.0%	78 (XS)

Sonarqube zur kontinuierlichen QS

The screenshot shows the Sonarqube project overview for 'BenekenGerd_Studienarbeit01'. The 'QUALITY GATE STATUS' is 'Passed' with the message 'All conditions passed.' The 'MEASURES' section displays various metrics:

- New Code:** Since April 19, 2022, Started 2 days ago.
- Overall Code:**
- Bugs:** 1 bug, Reliability grade E.
- Vulnerabilities:** 0 vulnerabilities, Security grade A.
- Security Hotspots:** 1 hotspot, 0.0% Reviewed, Security Review grade E.
- Debt:** 1h 32min, 8 Code Smells, Maintainability grade A.
- Coverage:** 26.4% (Coverage on 51 Lines to cover), Unit Tests: 1.
- Duplications:** 0.0% (Duplications on 78 Lines), Duplicated Blocks: 0.

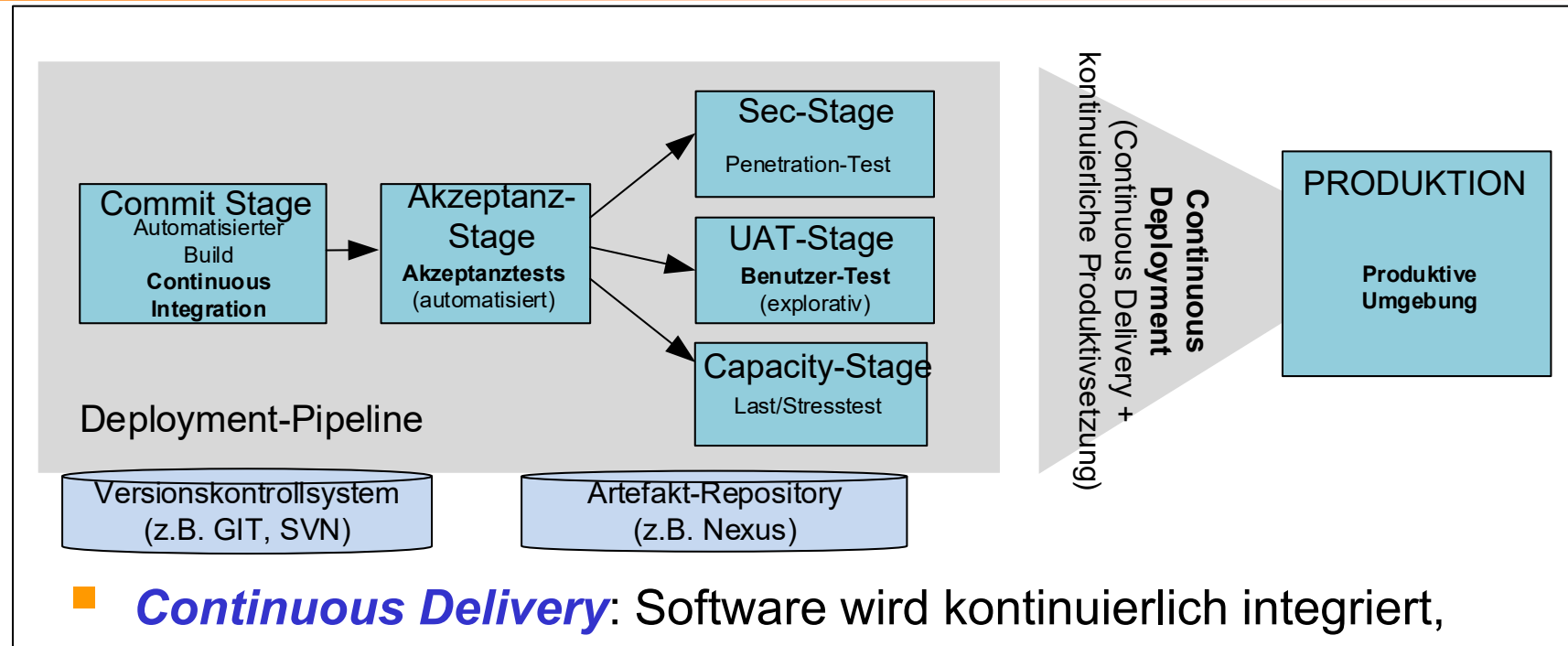
The screenshot shows the 'Issues' tab for the project 'BenekenGerd_Studienarbeit01'. The 'Filters' section on the left shows 'Type' as 'CODE SMELL' (8 items) and 'Severity' as 'Major' (6 items). The 'Issues' list on the right shows several issues, including:

- Define and throw a dedicated exception instead of using a generic one.** (Code Smell, Major, Open, Not assigned, 20min effort)
- Use the built-in formatting to construct this argument.** (Code Smell, Major, Open, Not assigned, 5min effort)
- Replace this use of System.out or System.err by a logger.** (Code Smell, Major, Open, Not assigned, 10min effort)
- Define and throw a dedicated exception instead of using a generic one.** (Code Smell, Major, Open, Not assigned, 20min effort)
- Catch Exception instead of Throwable.** (Code Smell, Major, Open, Not assigned, 20min effort)
- Add at least one assertion to this test case.** (Code Smell, Blocker, Open, Not assigned, 10min effort)
- Remove this 'public' modifier.** (Code Smell, Info, Open, Not assigned, 2min effort)

8 of 8 shown

Continuous Delivery / Deployment: Automatisierung

Computer übernehmen Routinetätigkeiten, BuildQualityIn, Transparenz



- **Continuous Delivery:** Software wird kontinuierlich integriert, und automatisiert getestet, sodass sie zu jedem Zeitpunkt released werden kann
- **Continuous Deployment:** jede Änderung läuft durch Deployment Pipeline und wird automatisiert produktiv gesetzt
- Hilfreich: Infrastructure as code, Container-Techniken, Cloud
- Werkzeuge: Jenkins, Gitlab CI, Github Actions

Die „Standard“-Folie

<https://www.youtube.com/watch?v=dxk8b9rSKOo>

Amazon May Deployment Stats
(production hosts & environments only)

11.6 seconds
Mean time between deployments (weekday)
1,079
Max # of deployments in a single hour
10,000
Mean # of hosts simultaneously receiving a deployment
30,000
Max # of hosts simultaneously receiving a deployment

Velocity 2011: Jon Jenkins, "Velocity Culture"

47.857 Aufrufe • 20.06.2011

👍 310 🗨 MAG ICH NICHT ➦ TEILEN ✂ CLIP ➕ SPEICHERN ...

Aufwendigere Tests eventuell nur auf den Liefer-Banches

- Aufwendige Analysen der Bibliotheken / Container
 - Security Prüfungen: SAST, CVE-Prüfungen, Secret Scan, ...
 - Lizenz – Scanning
- Aufwendige Tests, bei denen mehr Infrastruktur erzeugt werden muss, als Service in Gitlab-CI, über Docker Compose oder K8S
 - Akzeptanztest mit GUI (über Cypress / Selenium)
 - API-Tests z.B. mit Newman (Postman)
 - Lasttest / Stresstest, Kapazitätstest
 - Penetrationstest (DAST)

Beispiel für ein Finding in einem Studentenprojekt

- Gitlab sammelt Vulnerabilities in einer Liste
- Hier der Dependency Check
- Findings immer mit Verweis auf CVE-Quelle und ggf. Solution

<input type="checkbox"/>	2021-01-08	Needs Triage	High	Improper Input Validation in pip cv/requirements.txt	CVE-2018-20225 + 1 more	Dependency Scanning	
<input type="checkbox"/>	2020-10-27	Needs Triage	High	Object Prototype Pollution in Iodash gui/package-lock.json	CVE-2020-8203 + 1 more	Dependency Scanning	
<input type="checkbox"/>	2020-10-27	Needs Triage	High	Type checking vulnerability in kind-of gui/package-lock.json	CVE-2019-20149 + 1 more	Dependency Scanning	
<input type="checkbox"/>	2020-10-26	Needs Triage	High	Object Prototype Pollution in Iodash application/cv_innfactory_gui/package-lock.json	CVE-2020-8203 + 1 more	Dependency Scanning	
<input type="checkbox"/>	2020-06-17	Needs Triage	High	Improper Input Validation in pip application/requirements.txt	CVE-2018-20225 + 1 more	Dependency Scanning	
<input type="checkbox"/>	2020-06-17	Needs Triage	High	Injection Vulnerability in serialize-javascript application/cv_innfactory_gui/package-lock.json	CVE-2020-7660 + 1 more	Dependency Scanning	
<input type="checkbox"/>	2020-06-17	Needs Triage	High	Type checking vulnerability in kind-of application/cv_innfactory_gui/package-lock.json	CVE-2019-20149 + 1 more	Dependency Scanning	

Klaarte

Pickle and modules that wrap it can be unsafe when used to deserialize

Needs triage Detected 1 year ago in pipeline 36076

Status Needs triage ▾

Integer Overflow or Wraparound in elliptic

Description

The Elliptic package for allows ECDSA signature malleability via variations in encoding, leading to bytes, or integer overflows. This could conceivably have a security-relevant impact if an application relied on a single canonical signature.

Severity: Critical**Tool:** Dependency Scanning**Scanner:** Gemnasium

Location

File: [application/cv_innfactory_gui/package-lock.json](#)

Links

- <https://nvd.nist.gov/vuln/detail/CVE-2020-13822>

Identifiers

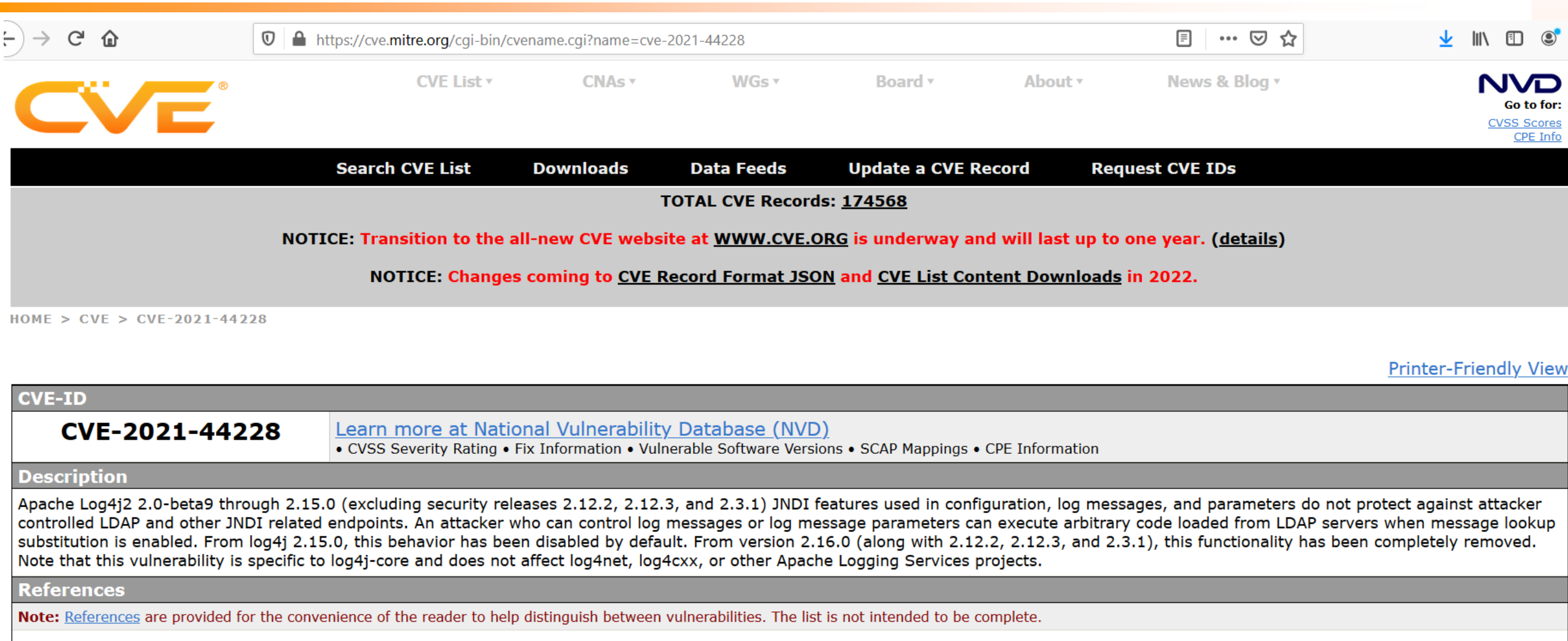
- [Gemnasium-0a6fe6be-860e-4b1d-b16f-665a77c09dac](#)
- [CVE-2020-13822](#)

Solution

Unfortunately, there is no solution available yet.

Prüfung von Bibliotheken gegen die CVE-Datenbanken

Vulnerability am Beispiel log4j



The screenshot shows the CVE Mitre website interface. The browser address bar displays the URL <https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2021-44228>. The website header includes the CVE logo, navigation links (CVE List, CNAs, WGs, Board, About, News & Blog), and the NVD logo with links to CVSS Scores and CPE Info. A black navigation bar contains links: Search CVE List, Downloads, Data Feeds, Update a CVE Record, and Request CVE IDs. Below this, a grey banner displays 'TOTAL CVE Records: 174568' and two notices in red text regarding the website transition and format changes in 2022. The breadcrumb trail reads 'HOME > CVE > CVE-2021-44228'. A 'Printer-Friendly View' link is in the top right. The main content area is a table with three sections: CVE-ID, Description, and References.

CVE-ID	
CVE-2021-44228	Learn more at National Vulnerability Database (NVD) • CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information
Description	
Apache Log4j2 2.0-beta9 through 2.15.0 (excluding security releases 2.12.2, 2.12.3, and 2.3.1) JNDI features used in configuration, log messages, and parameters do not protect against attacker controlled LDAP and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled. From log4j 2.15.0, this behavior has been disabled by default. From version 2.16.0 (along with 2.12.2, 2.12.3, and 2.3.1), this functionality has been completely removed. Note that this vulnerability is specific to log4j-core and does not affect log4net, log4cxx, or other Apache Logging Services projects.	
References	
Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.	

Beispiel für Vulnerability Report in Gitlab

The screenshot shows the GitLab interface for a project named 'cv-innfactory'. The 'Vulnerability report' section is active in the left sidebar. The main content area displays a 'Vulnerability Report' for the project's default branch. It indicates that the report shows results from the latest successful pipeline and container scan. The report is last updated 1 year ago with ID #46341.

Summary of vulnerabilities:

Severity	Count
Critical	76
High	31
Medium	50
Low	96
Info	5
Unknown	68

Filters: Status (Needs triage +1 more), Severity (All severities), Tool (All tools), Activity (All activity).

Detected	Status	Severity	Description	Identifier	Tool	Activity
2021-01-26	Needs Triage	Critical	Generic Object Injection Sink gui/src/sidebar/ButtonSegment.tsx:70	ESLint rule ID security/dete-ct-object-injection + 1 more	SAST	
2021-01-12	Needs Triage	Critical	Generic Object Injection Sink gui/src/store/slice/symbol.ts:157	ESLint rule ID security/dete-ct-object-injection + 1 more	SAST	
2021-01-12	Needs Triage	Critical	Generic Object Injection Sink gui/src/store/slice/symbol.ts:148	ESLint rule ID security/dete-ct-object-injection + 1 more	SAST	

- Beispiel für Zusammenfassung mehrerer Analyse-Werkzeuge
- SAST (Statische Code Analyse)
- CVE-Check
- Secret Scanning
- ...

Lizenz-Scanning

Sie müssen sich an die Lizenzauflagen halten

License Compliance ?

Displays licenses detected in the project, based on the [latest successful](#) scan • 1 year ago

Detected in Project 3

Policies 0

Name	Component
unknown	Flask (1.1.2) , Flask-RESTful (0.3.8) , and 8 more
MIT License	apispec (1.2.0) , marshmallow (2.19.0) , and 4 more
Apache License 2.0	sortedcontainers (2.3.0)

Dependencies ?

Based on the [latest successful](#) scan • 1 year ago

Severity

Component	Packager	Location ?	License
jsonschema 3.1.1	Python (pip)	cv/requirements.txt	
marshmallow 2.19.0	Python (pip)	cv/requirements.txt	MIT License
packaging 19.2	Python (pip)	cv/requirements.txt	
pipdeptree 0.13.2	Python (pip)	cv/requirements.txt	
pipenv 2018.11.26	Python (pip)	cv/requirements.txt	
pyparsing 2.4.6	Python (pip)	cv/requirements.txt	
pyrsistent 0.15.7	Python (pip)	cv/requirements.txt	
python-magic 0.4.18	Python (pip)	cv/requirements.txt	MIT License
pytz 2020.5	Python (pip)	cv/requirements.txt	MIT License
redis 3.5.3	Python (pip)	cv/requirements.txt	MIT License
rq 1.7.0	Python (pip)	cv/requirements.txt	unknown

- An welche Auflagen müssen sie sich halten?
- Virale Lizenzen beteiligt (GPL, AGPL, LGPL)?
- Ggf. schließen sich Lizenzmodelle gegenseitig aus

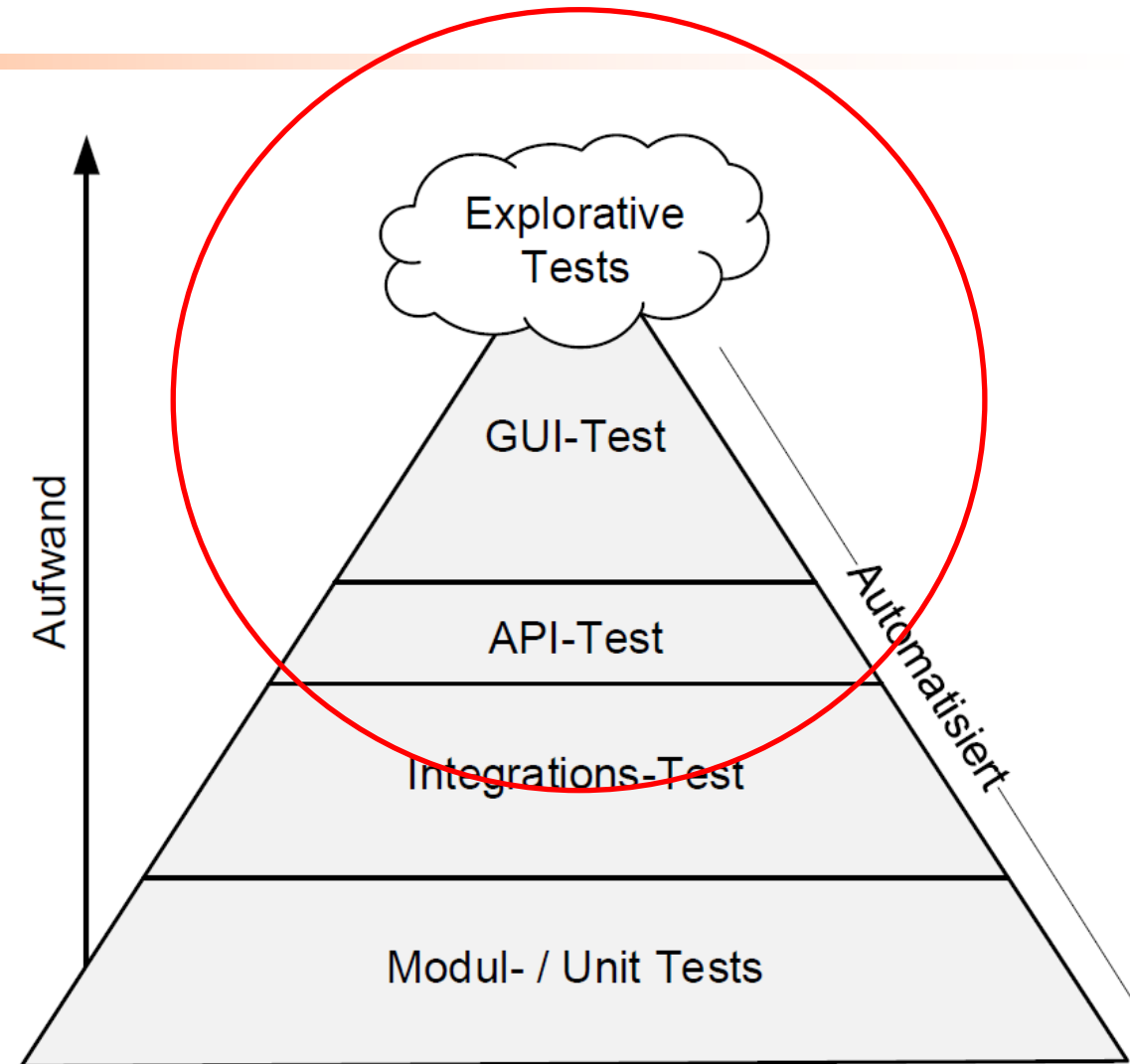
Test Pyramide

■ Commit-Stage

- Läuft auf allen Branches
- Schnell
- Modul und Unittests (Umgebung ist gemockt, Hauptspeicher-DB, ...)
- Integrationstest ggf. noch mit Unittest-Werkzeug und ggf. ATDD

■ Akzeptanztest-Stage

- Nur auf den Liefer-Branches (Develop / Main / Master)
- Darf langsam sein
- API-Tests, GUI-Tests, ...

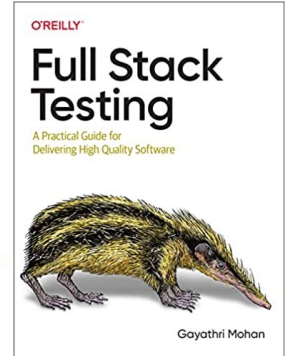


Akzeptanztest-Stage

Werkzeug-Integration wird aufwendiger

- Bei verteiltem System mehrere Container starten
 - DBMS, Mindestens noch der Server
 - ggf. größerer Datenbestand (zurücksetzbar?)
 - = Aufwand mit Docker-Compose oder K8S oder vorhandenen Servern
- Für API-Tests
 - Start von Testwerkzeug, das REST- / GraphQL- / gRPC / WebSocket Aufrufe absetzt
 - Beispiel Newman mit „Postman-Collections“ als Testskript
 - Eventuell auch einfache shell-Skripte
- Für GUI-Test
 - Ggf. Start eines Headless-Browsers
 - Start des Testwerkzeugs als eigenes Programm, z.B. Selenium oder Cypress

Beispiele fehlen hier noch ...



- **GUI-Test** mit Selenium, wird nachgereicht im Programmierteil
- **API-Test** mit Newman, aus VV kopieren: © Thomas Mildner
- **Integrationstest** mit Pact oder ähnlichem Werkzeug, wird eventuell nachgereicht



Agiles Software- Entwicklungsprojekt

Prof. Dr. Gerd Beneken

Kapitel 7.4

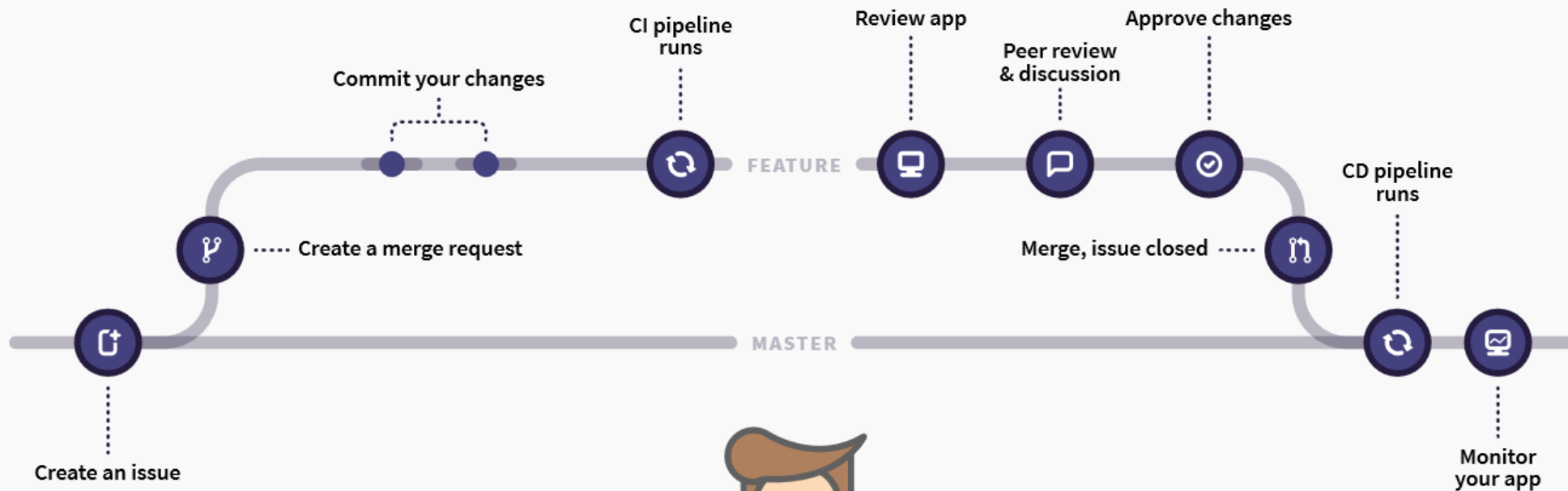
Manuelle Analytische QS

Merge Requests

Qualitätssicherung kommt häufig zu kurz

- Code Reviews werden häufig weggelassen
 - Oder nicht systematisch genug durchgeführt
- Dadurch leichter Wissensmonopole
 - Bzw. Truck-Faktor „1“
- Features später nicht mehr eindeutig im Code erkennbar
 - Entwickler nicht gut genug von einander entkoppelt
 - Zu frühes Mergen unfertiger Teilergebnisse, dadurch gegenseitiges Behindern
 - Was passiert wenn Feature beim Sprint-Review „durchfällt“

Merge Requests und der Gitlab Flow



- = Mittel um Qualitätssicherung zu erzwingen und zu dokumentieren
 - Approval durch Teammitglieder; Änderungen werden genehmigt
 - Approval sollte Code Review beinhalten, Diskussionspunkte sollten erledigt sein
 - Build-Pipeline sollte vor und nach dem Merge-Request durchlaufen

Feature Branches in gitlab

Ticket erklärt den Sinn und Zweck des Branches

The screenshot shows the GitLab web interface for an issue titled "Next Feature (#2)". The issue is in the "Issues" section of the "apt2019" project. The issue description is "@be changed weight to 20 just now". The issue has 1 discussion and 0 designs. A dropdown menu is open, showing options to "Create merge request and branch" and "Create branch". The "Create branch" option is selected, and the "Create branch" button is highlighted with a red circle. The dropdown menu also shows the "Branch name" as "2-next-feature" and the "Source (branch or tag)" as "master".

Related issues 0

Discussion 1 Designs 0

be @be changed weight to 20 just now

Write Preview

Write a comment or drag your files here...

Markdown and quick actions are supported

Comment Close issue

Create merge request and branch

✓ Create branch

Branch name

2-next-feature

Source (branch or tag)

master

Create branch

To Do Mark as done

Assignee be @be Edit

Epic None Edit

Milestone None Edit

Time tracking No estimate or time spent

Due date None Edit

Labels None Edit

Weight 20 - remove weight Edit

Confidentiality Not confidential Edit

Merge Request und Merge

The screenshot shows a GitLab Merge Request page. At the top, a notification states "You pushed to 2-next-feature just now". Below this, a commit message "Readme Datei repariert" by "be authored just now" is shown. A table lists branches, with "2-next-feature" selected. A "Create merge request" button is circled in red. A red arrow points from this button to the "Description" field of the "New Merge Request" form. In the form, the title is "Readme Datei repariert" and the description is "Closes #2", both of which are circled in red. The text "Issue wird automatisch ge - closed" is placed next to the "Closes #2" description.

vorlesungen / apt2019 / repository

You pushed to 2-next-feature just now

2-next-feature apt2019 / + v

History Find file Web IDE

Readme Datei repariert
be authored just now

17017dc0

Name	Last commit	Last update
katas	Undo test	

vorlesungen > apt2019 > Merge Requests > New

New Merge Request

From 2-next-feature into master

Title: Readme Datei repariert

Start the title with **WIP:** to prevent a **Work In Progress** merge request from being merged before it's ready.
Add [description templates](#) to help your contributors communicate effectively!

Description: [Write](#) [Preview](#)

Closes #2

Markdown and [quick actions](#) are supported

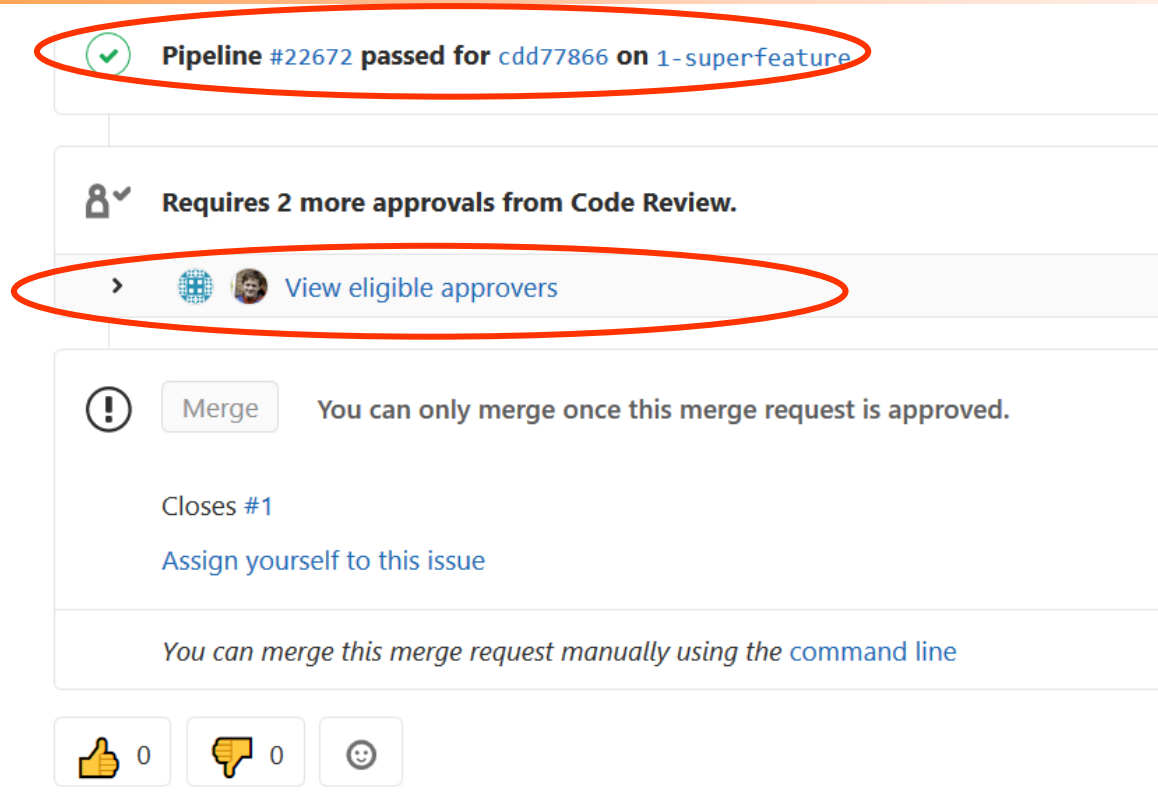
Issue wird automatisch ge - closed

Merge Request selbst durchführen

The screenshot displays the GitLab web interface for a Merge Request. The browser address bar shows the URL `https://inf-git.fh-rosenheim.de/vorlesungen/apt2019/merge/`. The GitLab navigation bar at the top includes links for Projects, Groups, Activity, Milestones, Snippets, and a search bar. The left sidebar shows the project structure for 'apt2019', with 'Merge Requests' selected and showing 1 item. The main content area is titled 'Readme Datei repariert' and indicates it 'Closes #2'. It shows a 'Request to merge 2-next-feature into master' with buttons for 'Open in Web IDE', 'Check out branch', and a download icon. A green checkmark indicates 'Pipeline #22670 passed for b7017dc0 on 2-next-feature'. Below this, it states 'No approval required'. A green 'Merge' button is visible, with a red arrow pointing to the merge confirmation message. The confirmation message states: 'Merged by be just now', 'The changes were merged into master with e36c5287', 'The source branch has been deleted', and 'Closed #2'. It also includes 'Revert' and 'Cherry-pick' buttons. At the bottom, a yellow status bar shows 'Pipeline #22671 pending for e36c5287 on master'. The right sidebar contains various settings: 'To Do' (Add a To Do), '0 Assignees' (Edit), 'Milestone' (None, Edit), 'Time tracking' (No estimate or time spent, ?), 'Labels' (None, Edit), 'Lock merge request' (Unlocked, Edit), '1 participant' (user profile), 'Notifications' (toggle on), and 'Reference: vorlesungen/apt2019!1'.

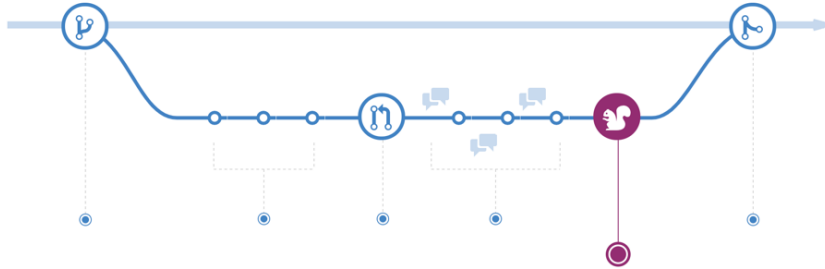
Merge Request Code Review erzwingen:

1) Pipeline und 2) Approval Rule

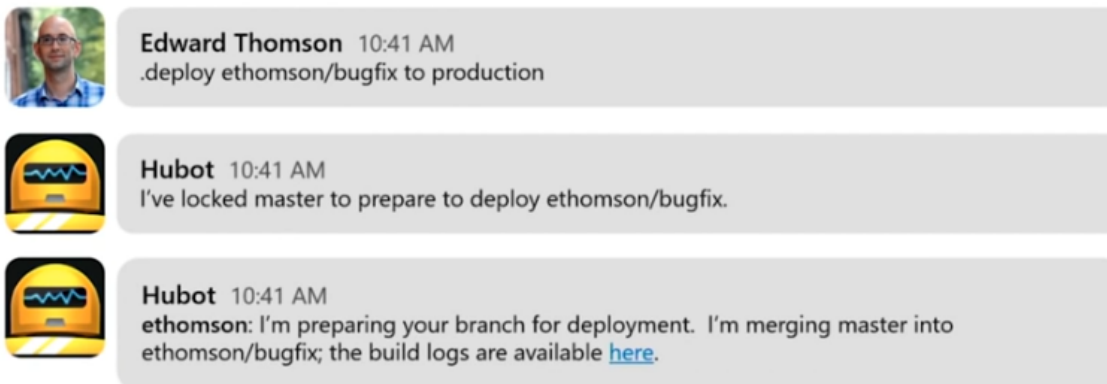


Github Flow

- Entspricht Trunkbased Development
- Mit Merge-/Pullrequests UND Code Reviews
- Spannend: Eichhörnchen – Deploy in Produktion



- Deploy über Chatbot (Koordination im Team)





Agiles Software- Entwicklungsprojekt

Prof. Dr. Gerd Beneken

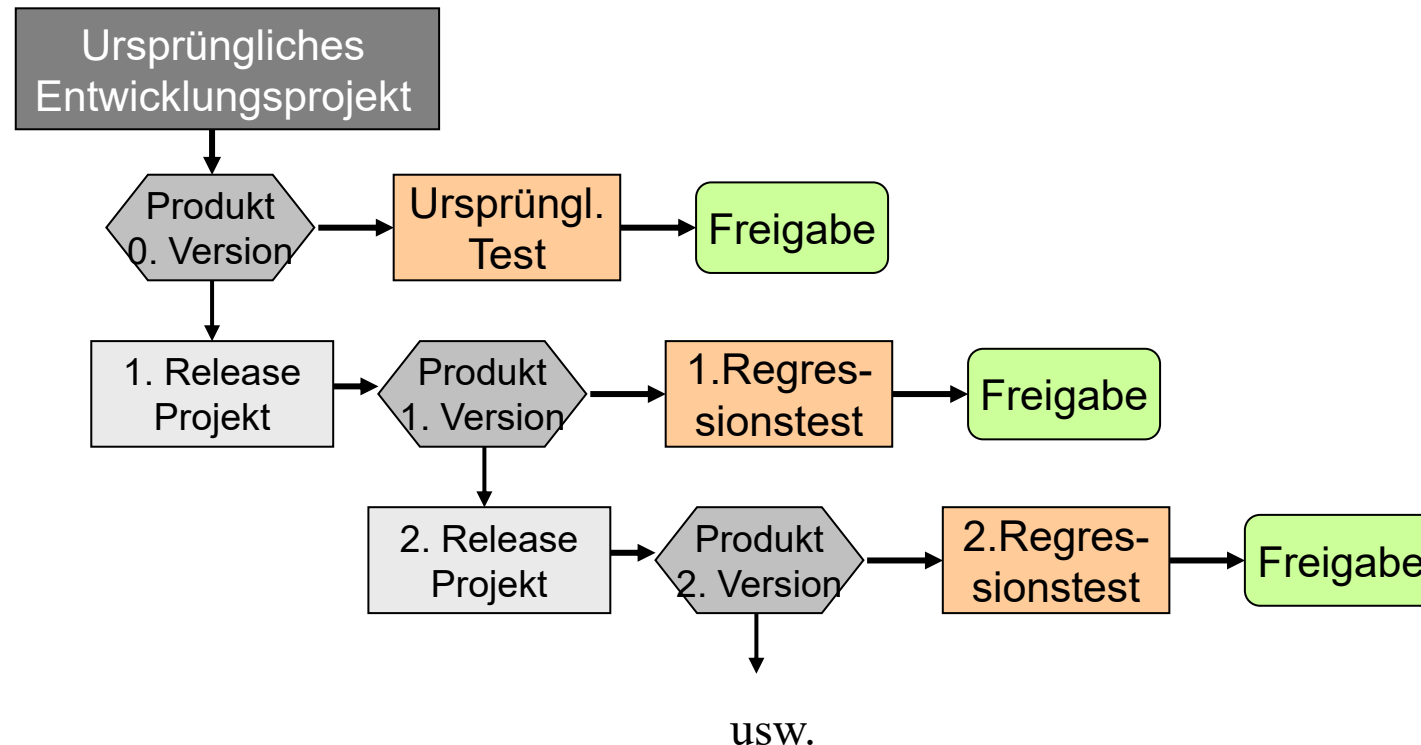
Kapitel 7.5

Automatisierte Analytische QS

Testautomatisierung

Regressionstest im Software-Lebenszyklus

Testen aus permanente Aufgabe ...

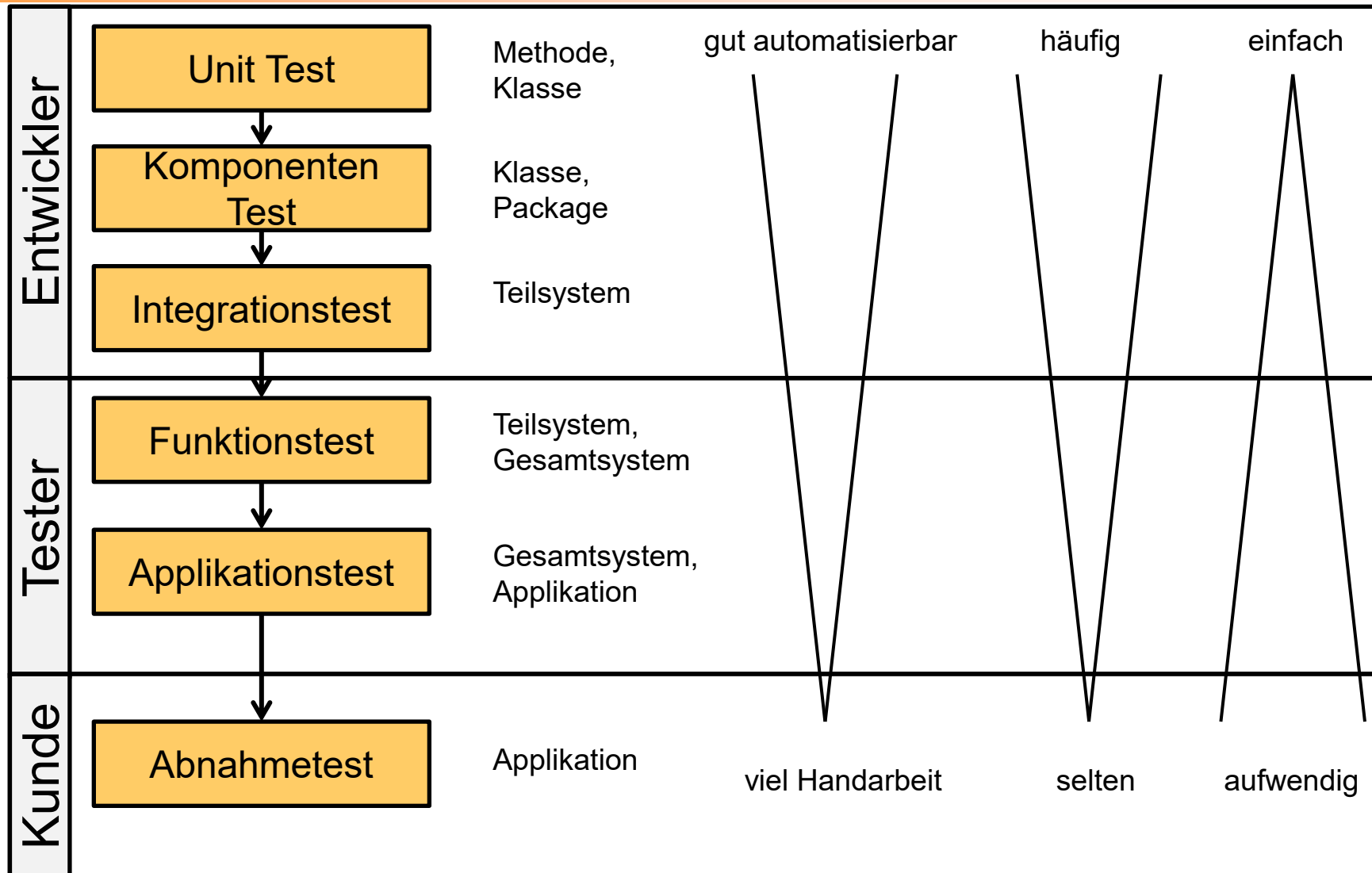


(Regeessions-) Testaufwand in lange laufenden Projekten: 60-80% [Sneed]

Investition in Testautomatisierung lohnt sich idR.

Regressionstest – Ebenen

[nach Inden: Der Weg zum Java Profi]



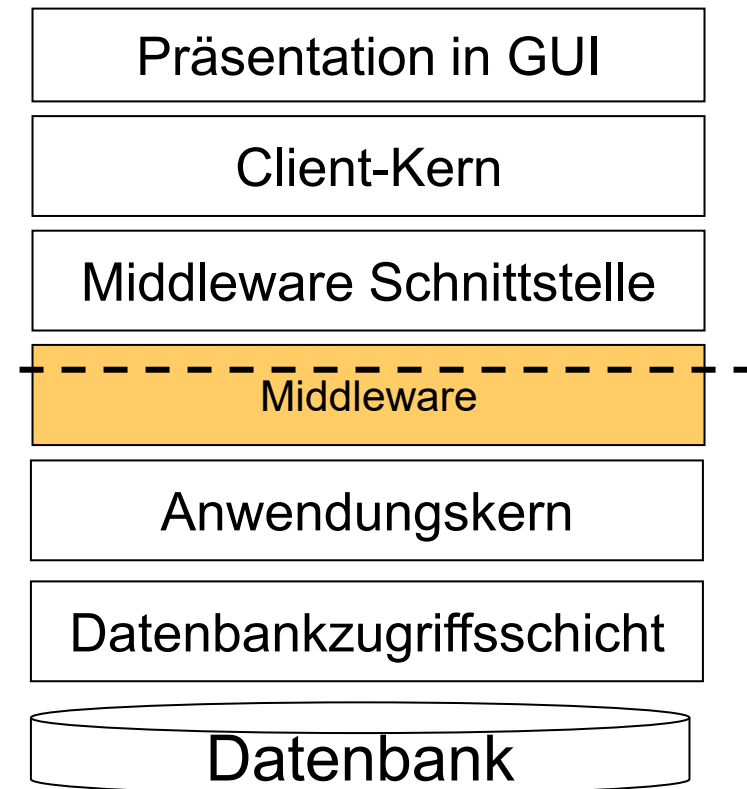
Was ist ein automatisierter Testtreiber?

- = Klasse in einem xUnit Framework
 - Java: JUnit + Zusatzframeworks wie DBUnit, EasyMock, ...
 - .NET: Einbauten in VisualStudio, NUnit, NMock, ...
 - CppUnit, phpUnit, ...
 - Gatling-Skript in Scala
 - Selenium-Programm zur Fernbedienung von Oberflächen
- = vollständiges Programm oder Skript
 - Perl / Ruby / Python-Skript
- = Skript eines Capture-Replay Tools
 - LoadRunner/JMeter/WAPT:
Zeichnet HTTP-Requests und anderen Netzwerkverkehr auf
 - WinRunner/Selenium: Zeichnet Events in einer Native-GUI auf
 - Skript = Bearbeitete Aufzeichnung eines Testfalls

Wo können Testtreiber implementiert werden?

Der Testtreiber verwendet:

1. die grafische Oberfläche direkt von außen (Typisch = Capture/Replay Tools, z.B. WinRunner, Silktest, Selenium)
2. den „Kern“ des Clients, z.B. Model oder Controller bei MVC, Dialogsteuerung (= JUnit / JBehave / FitNesse-Tests)
3. die Middleware testet eigentlich den Anwendungskern (= Postman, Swagger)
4. die Services / Schnittstellen des Anwendungskerns (= A-Verwalter, Anwendungsfälle) (Online und Batch)
5. die Datenbankzugriffsschicht



Was tut ein Testtreiber?

Arrange – Act - Assert

1. *Initialisieren* (häufig auch separates Skript)

- Erzeugung der **Testdaten** / Lesen der **Testdaten**
- Umgebung konfigurieren / initialisieren (Real und/oder Test-Dummy)

2. *Durchführen* der Testfälle

- Aufrufen der zu testender Features, Methoden, ...
- Prüfen: Liefert Testfall erwartetes Ergebnis?
- Protokollieren

3. *Aufräumen* und Start-Zustand wieder herstellen

- Angelegte Testdaten löschen
- Ressourcen der Umgebung freigeben

Was tut ein Testtreiber?

/2

Durchführen der Testfälle für jedes zu testende Methode / ...

- **Aufruf der Methode**, der Operation, des Dialogs, des Webservice, des ...
- **Vergleich** (Java: z.B. **assertTrue** (...), bei Dateien evtl. **grep / diff**
 - des **tatsächlichen Ergebnisses** (Rückgabewert, OUT-Parameter, Systemfehler, Exception, Antwort-Maske, Datei) mit dem
 - **erwarteten Ergebnis**
- Bei Übereinstimmung: positiven Ausgang melden
- Bei Abweichung: Fehler melden, protokollieren (z.B. im Trace, assert), weitermachen bei nächstem Testfall
- Möglichst: Keine Ausgabe auf die Konsole oder in ein Trace, die Interpretation seitens eines Entwicklers erfordert!

Hier fehlen noch folgende Folien

- GUI-Test Automatisierung am Beispiel Cypress oder Selenium
- API-Test über Newman
- Ggf. Vertragstest mit Pact oder etwas ähnlichem