

Inhaltsverzeichnis

LS3 – Infos – Funktionen.....	2
LS3 – Infos – Module.....	4
LS3 – Infos – Datenbanken.....	7
LS3 – Infos – Aufbau einer Datenbankanwendung.....	9
LS3 – Infos – Ressourcenverbrauch ermitteln.....	11
LS3 – Infos – Datenbankzugriff modular isolieren.....	12
LS3 – Infos – Versionsmanagement.....	13
LS3 – Infos – Testen (I).....	15
LS3 – Infos – Testen (II).....	18

LS3 – Infos – Funktionen

Quelle:

Leicht modifizierter und
ergänzter Chat-GPT Text



In der Welt der Programmierung sind Funktionen ein grundlegendes Konzept, das uns dabei hilft, unseren Code zu organisieren und wiederverwendbar zu machen. Funktionen ermöglichen es uns, eine bestimmte Aufgabe oder Operation zu definieren und sie dann an verschiedenen Stellen in unserem Programm aufzurufen, ohne den Code immer wieder neu schreiben zu müssen.

Was ist eine Funktion?

Eine **Funktion** ist eine benannte Gruppe von Anweisungen, die eine bestimmte Aufgabe erfüllen. Sie kann **Parameter** (auch **Argumente** genannt) akzeptieren, die als Eingabe für die Funktion dienen, und einen **Rückgabewert** liefern, der das Ergebnis der Funktion darstellt. Funktionen können auch ohne Parameter oder Rückgabewert definiert werden, je nachdem, welche Aufgabe sie erfüllen sollen.

Warum sind Funktionen nützlich?

Code-Organisation: Durch die Verwendung von Funktionen können wir unseren Code in kleinere, überschaubare Teile aufteilen. Dies erleichtert die **Lesbarkeit** und **Wartbarkeit** des Codes, da wir uns auf eine spezifische Aufgabe konzentrieren können, anstatt von Anfang bis Ende des Programms zu scrollen.

Wiederverwendbarkeit: Wenn wir eine Funktion erstellt haben, können wir sie an verschiedenen Stellen in unserem Programm aufrufen, um die gleiche Aufgabe zu erfüllen. Dies spart uns Zeit und vermeidet die Notwendigkeit, den gleichen Code mehrmals schreiben zu müssen.

Fehlerbehebung: Wenn wir eine Funktion haben, die eine bestimmte Aufgabe erfüllt, können wir diese Funktion isoliert testen und debuggen. Dadurch wird die Fehlersuche erleichtert und der Code wird stabiler.

Wie verwendet man Funktionen?

Um eine Funktion zu verwenden, müssen wir sie zuerst definieren. Die Definition einer Funktion besteht aus dem Namen der Funktion, den Parametern, die sie akzeptiert, und dem Codeblock, der die Anweisungen enthält, die ausgeführt werden sollen. Hier ist ein Beispiel für die Definition einer Funktion in Python:

```
def gruessen(name):  
    print("Hallo, " + name + "!")
```

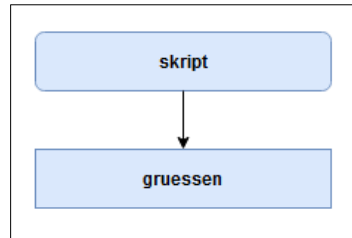
Um die Funktion aufzurufen, verwenden wir einfach den Funktionsnamen gefolgt von Klammern, in denen wir die Argumente angeben. Hier ist ein Beispiel für den Aufruf der Funktion `gruessen`:

```
gruessen("Max")
```

Dies würde die Ausgabe "Hallo, Max!" erzeugen.

Wie kann man die Aufrufe graphisch darstellen?

Durch die Funktionsaufrufe wird eine **Funktionshierarchie** gebildet. Diese kann man graphisch darstellen. Betrachten wir das aufrufende Python-Skript selber auch als eine Funktion, so ergibt sich folgende Funktionshierarchie:



Um den inneren Aufbau einer Funktion graphisch darzustellen kann man Struktogramme und Programmablaufpläne nutzen.

Zusammenfassung

Funktionen sind ein wichtiges Konzept in der Programmierung, das uns hilft, unseren Code zu organisieren und wiederverwendbar zu machen. Sie ermöglichen es uns, bestimmte Aufgaben oder Operationen zu definieren und sie an verschiedenen Stellen in unserem Programm aufzurufen. Funktionen verbessern die Lesbarkeit, Wartbarkeit und Fehlerbehebung unseres Codes und sind ein unverzichtbares Werkzeug für jeden Programmierer.

Wo finde ich weitere Information?

Mehr zu Python und Funktionen findet man:

1. w3schools: https://www.w3schools.com/python/python_functions.asp
2. Herdt-Verlag: Herdt-Python-1.pdf (Logineo) – Kapitel 8
3. Herdt-Verlag: Herdt-Python-3.pdf (Logineo) – Kapitel 6
4. Python-Dokumentation: <https://docs.python.org/3/tutorial/controlflow.html#defining-functions>

LS3 – Infos – Module

Quelle:

Leicht modifizierter und
ergänzter Chat-GPT Text



Modularisierung und Module in der Programmierung

In der Welt der Programmierung ist Modularisierung ein wichtiger Ansatz, um unseren Code zu organisieren und zu strukturieren. Modularisierung bedeutet, unseren Code in kleinere, unabhängige Module aufzuteilen, die jeweils eine spezifische Aufgabe erfüllen. Diese Module können dann miteinander interagieren und zusammenarbeiten, um ein größeres Programm zu erstellen.

Was ist Modularisierung?

Modularisierung ist der Prozess, bei dem ein Programm in separate Module aufgeteilt wird. Jedes **Modul** enthält einen bestimmten Satz von Funktionen, Variablen und Anweisungen, die eng miteinander verbunden sind und eine bestimmte Aufgabe erfüllen. Durch die Modularisierung können wir unseren Code in überschaubare Teile aufteilen, was die Wartbarkeit und Lesbarkeit verbessert.

Vorteile der Modularisierung:

Die Verwendung von Modulen bietet mehrere Vorteile:

1. **Code-Wiederverwendung:** Durch die Aufteilung des Codes in Module können wir Funktionen oder Codeblöcke, die wir bereits geschrieben haben, wiederverwenden. Dies spart Zeit und vermeidet die Notwendigkeit, den gleichen Code mehrmals zu schreiben.
2. **Code-Organisation:** Durch die Verwendung von Modulen können wir unseren Code in logische Einheiten aufteilen. Dies erleichtert die Lesbarkeit und Wartbarkeit des Codes, da wir uns auf eine spezifische Aufgabe konzentrieren können, anstatt den gesamten Code zu durchsuchen.
3. **Kollaboration:** Wenn mehrere Entwickler an einem Projekt arbeiten, können Module als Schnittstelle dienen, um unabhängig voneinander an verschiedenen Teilen des Codes zu arbeiten. Dies ermöglicht eine effiziente Zusammenarbeit und erleichtert die Integration der einzelnen Module zu einem Gesamtprogramm.

Zusammenhang zwischen Modulen und Funktionen:

Module bestehen aus Funktionen, Variablen und Anweisungen. Funktionen sind ein wichtiger Bestandteil von Modulen, da sie die eigentliche Arbeit erledigen. Funktionen in Modulen können Parameter akzeptieren, um Eingaben zu erhalten, und einen Rückgabewert liefern, der das Ergebnis der Funktion darstellt. Durch die Verwendung von Funktionen in Modulen können wir unseren Code weiter in kleinere Teile aufteilen und die Wiederverwendbarkeit verbessern.

Wie sieht ein Modul in Python aus?

In Python wird ein Modul durch eine Datei mit der Erweiterung ".py" repräsentiert. Das Modul enthält Funktionen, Variablen und Anweisungen, die in der Datei definiert sind. Hier ist ein Beispiel für ein einfaches Modul in Python:

```
# Beispielm modul "rechner.py"

def addiere(a, b):
```

```
    return a + b

def subtrahiere(a, b):
    return a - b

def multipliziere(a, b):
    return a * b
```

In diesem Beispiel enthält das Modul `rechner.py` drei Funktionen: `addiere`, `subtrahiere` und `multipliziere`. Diese Funktionen können in anderen Teilen des Programms aufgerufen werden, indem das Modul importiert wird.

Wie kann man andere Module in Python nutzen?

Um ein Modul in Python zu nutzen, muss es zuerst importiert werden. Hier ist ein Beispiel, wie man das Modul `rechner.py` in einem anderen Python-Skript verwendet:

```
# Beispielm modul "nutzer.py"

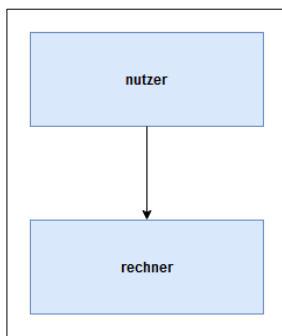
import rechner

ergebnis = rechner.addiere(5, 3)
print(ergebnis)
```

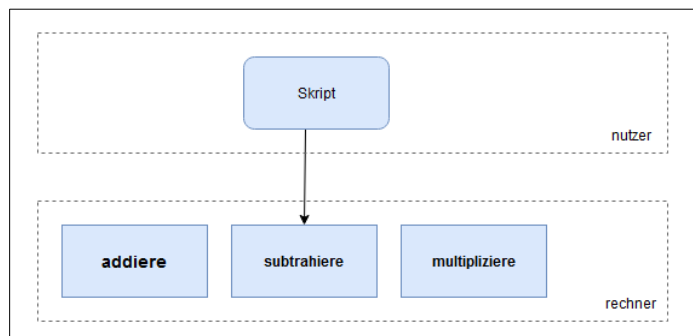
In diesem Beispiel importieren wir das Modul `rechner` und rufen dann die Funktion `addiere` mit den Argumenten 5 und 3 auf. Das Ergebnis wird dann ausgegeben.

Wie kann man die Zusammenhänge graphisch darstellen?

Modulhierarchie



Funktionshierarchie



Zusammenfassung

Modularisierung ist ein wichtiger Ansatz in der Programmierung, um unseren Code zu organisieren und zu strukturieren. Durch die Aufteilung des Codes in unabhängige Module können wir Funktionen, Variablen und Anweisungen wiederverwenden und unseren Code besser organisieren. Module bestehen aus Funktionen, die spezifische Aufgaben erfüllen, und können in anderen Teilen des Programms verwendet werden, indem sie importiert werden. Die Verwendung von Modulen verbessert die Wartbarkeit, Lesbarkeit und Wiederverwendbarkeit unseres Codes und ist ein grundlegendes Konzept, das jeder Programmierer verstehen sollte.

Wo finde ich weitere Information?

Mehr zu Python und Module findet man:

1. Python-Dokumentation: <https://docs.python.org/3/tutorial/modules.html>

LS3 – Infos – Datenbanken

Quelle:

Leicht modifizierter und
ergänzter Chat-GPT Text



Was sind Datenbanken?

Datenbanken sind strukturierte Sammlungen von Daten, die in einer organisierten und effizienten Art und Weise gespeichert, verwaltet und abgerufen werden können. Sie dienen dazu, große Mengen von Daten zu speichern und darauf zuzugreifen, um Informationen zu organisieren und zu verwalten.

Vorteile von Datenbanken

Die Verwendung von Datenbanken bietet mehrere Vorteile:

1. **Datenintegrität:** Datenbanken ermöglichen die Durchsetzung von Regeln zur Datenintegrität, um sicherzustellen, dass die gespeicherten Daten konsistent und korrekt sind. Dadurch wird die Genauigkeit und Zuverlässigkeit der Daten verbessert.
2. **Effizienter Datenzugriff:** Durch die Verwendung von Datenbanken können Daten schnell und effizient abgerufen werden. Datenbanken verwenden spezielle Algorithmen und Indexstrukturen, um den Zugriff auf Daten zu beschleunigen und die Leistung zu optimieren.
3. **Datenkonsistenz:** Datenbanken ermöglichen es, Daten konsistent zu halten, auch wenn mehrere Benutzer gleichzeitig darauf zugreifen. Durch die Verwendung von Transaktionen und Sperren können Datenbanken sicherstellen, dass Änderungen atomar, konsistent, isoliert und dauerhaft durchgeführt werden.
4. **Skalierbarkeit:** Datenbanken können große Mengen von Daten effizient verwalten und skalieren. Sie können mit wachsenden Datenmengen umgehen und die Leistung beibehalten, ohne dass größere Änderungen am System erforderlich sind.

Relationale Datenbanken

Eine der häufigsten Arten von Datenbanken ist die relationale Datenbank. In einer relationalen Datenbank werden Daten in Tabellen organisiert, die miteinander verknüpft sein können. Beziehungen zwischen den Tabellen werden durch Schlüssel definiert, die in den Tabellen gespeichert sind. Diese Beziehungen ermöglichen komplexe Abfragen und Analysen der Daten.

Einige weit verbreitete relationale Datenbanken

Es gibt viele verschiedene relationale Datenbanken, die in der Industrie weit verbreitet sind. Hier sind einige Beispiele:

1. **MySQL:** MySQL ist eine Open-Source-Datenbank, die für ihre Geschwindigkeit, Skalierbarkeit und Zuverlässigkeit bekannt ist. Sie wird häufig in Webanwendungen eingesetzt.
2. **MariaDB:** MariaDB ist eine Open-Source-Datenbank, die aus einer Abspaltung von MySQL entstanden ist. Sie bietet eine hohe Kompatibilität zu MySQL und wird als Alternative zu MySQL verwendet. MariaDB wird ebenfalls in Webanwendungen und anderen Projekten eingesetzt.
3. **Oracle Database:** Oracle Database ist eine kommerzielle relationale Datenbank, die für ihre Leistung, Sicherheit und Skalierbarkeit geschätzt wird. Sie wird in großen Unternehmen und Organisationen eingesetzt.

4. **Microsoft SQL Server:** Der Microsoft SQL Server ist eine kommerzielle relationale Datenbank von Microsoft. Er bietet eine breite Palette von Funktionen und Integrationen mit anderen Microsoft-Produkten.
5. **PostgreSQL:** PostgreSQL ist eine Open-Source-Datenbank, die für ihre Flexibilität und Erweiterbarkeit bekannt ist. Sie unterstützt erweiterte Funktionen und wird häufig in Anwendungen mit hohem Datenvolumen eingesetzt.

Diese Datenbanken sind nur einige Beispiele für die Vielfalt an relationalen Datenbanken, die auf dem Markt verfügbar sind. Jede Datenbank hat ihre eigenen Stärken und Einsatzbereiche, und die Wahl der richtigen Datenbank hängt von den spezifischen Anforderungen des Projekts ab.

SQL (Structured Query Language)

SQL ist eine spezielle Programmiersprache, die für die Kommunikation mit relationalen Datenbanken verwendet wird. Mit SQL können Datenbanken erstellt, verwaltet und abgefragt werden. Die Sprache ermöglicht es, Daten in Tabellen zu speichern, Abfragen durchzuführen, Daten zu aktualisieren und zu löschen, sowie komplexe Operationen wie Joins und Aggregationen durchzuführen.

Was ist XAMPP?

XAMPP ist eine All-in-One-Software (Bundle), die eine vollständige Entwicklungsumgebung für die Erstellung und den Test von Webanwendungen bereitstellt. Es enthält den Apache Webserver, die MySQL oder MariaDB-Datenbank, PHP und Perl, die alle notwendigen Komponenten für die Entwicklung von dynamischen Websites und Anwendungen bieten. XAMPP bietet eine einfache Installation, ist plattformübergreifend und bietet eine flexible und anpassbare Entwicklungsumgebung. Durch die Verwendung von XAMPP können Entwickler ihre Datenbankanwendungen effizient entwickeln und testen, bevor sie sie auf einen Produktionsserver hochladen.

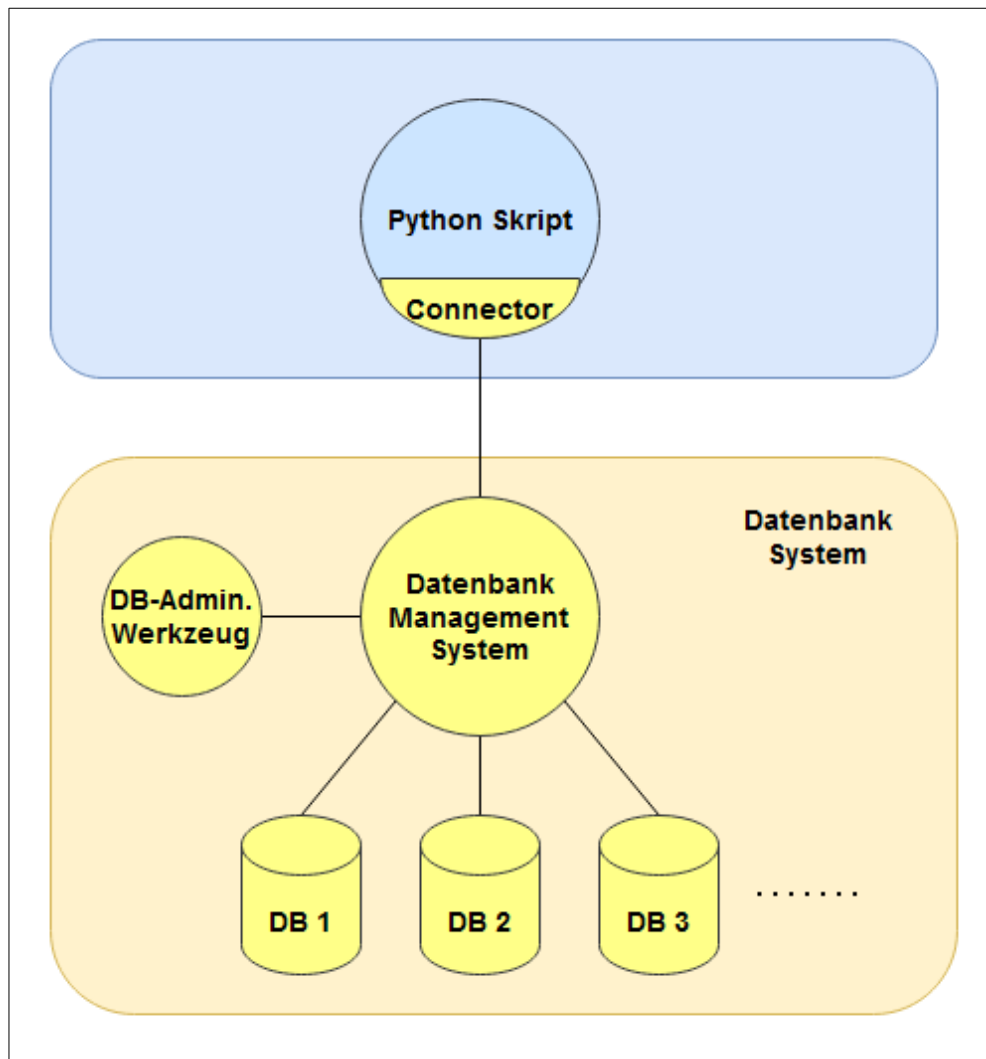
Zusammenfassung

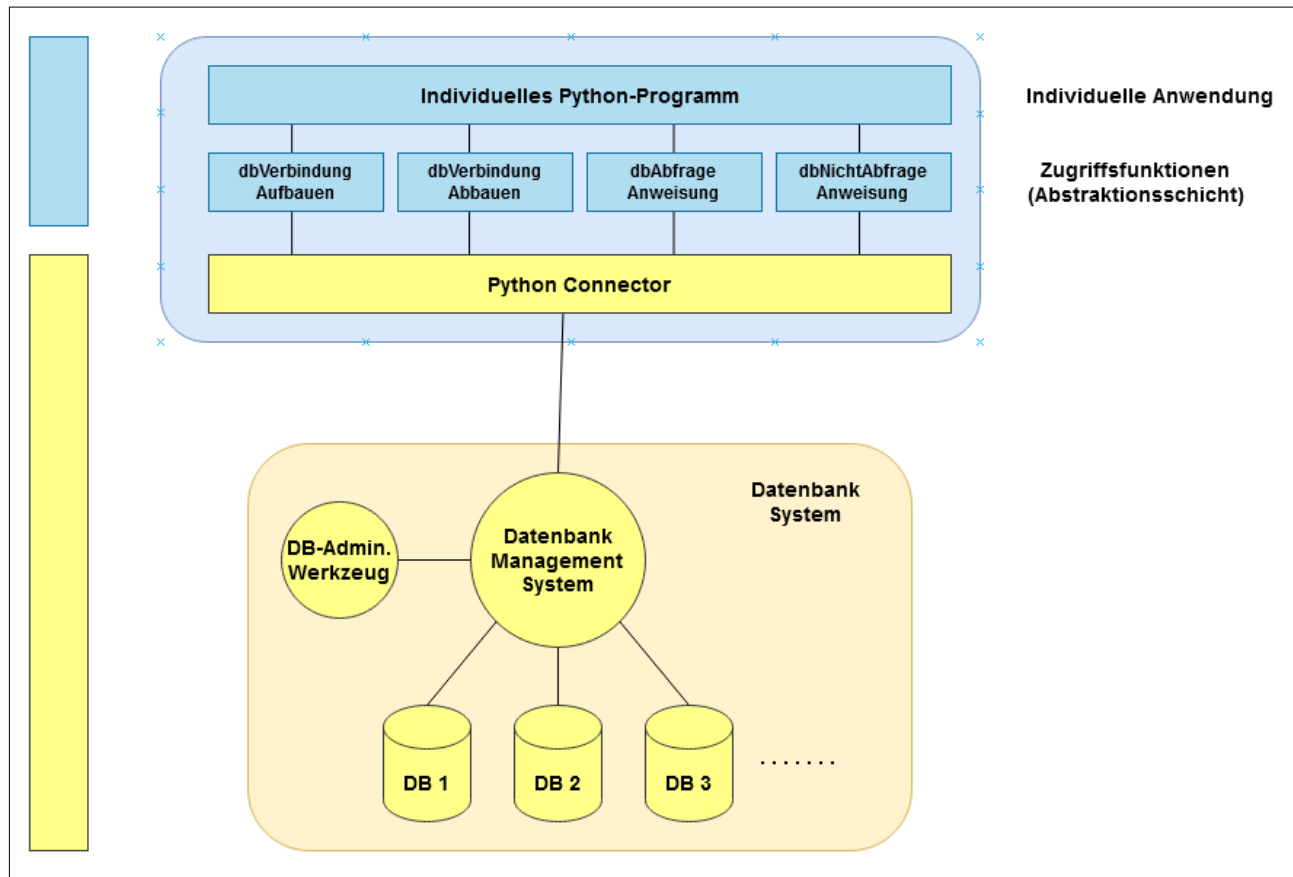
Datenbanken sind strukturierte Sammlungen von Daten, die in einer organisierten und effizienten Art und Weise gespeichert, verwaltet und abgerufen werden können. Sie bieten Vorteile wie Datenintegrität, effizienten Datenzugriff, Datenkonsistenz und Skalierbarkeit. Eine der häufigsten Arten von Datenbanken ist die relationale Datenbank, die Daten in Tabellen organisiert und Beziehungen zwischen den Tabellen herstellt. Es gibt viele weit verbreitete relationale Datenbanken wie MySQL, MariaDB, Oracle Database, Microsoft SQL Server und PostgreSQL. SQL ist die Sprache, die verwendet wird, um mit relationalen Datenbanken zu kommunizieren und Operationen wie Datenabfragen, -aktualisierungen und -lösungen durchzuführen.

Wo finde ich weitere Information?

- XAMPP: <https://www.apachefriends.org/index.html>

LS3 – Infos – Aufbau einer Datenbank-anwendung





LS3 – Infos – Ressourcenverbrauch ermitteln

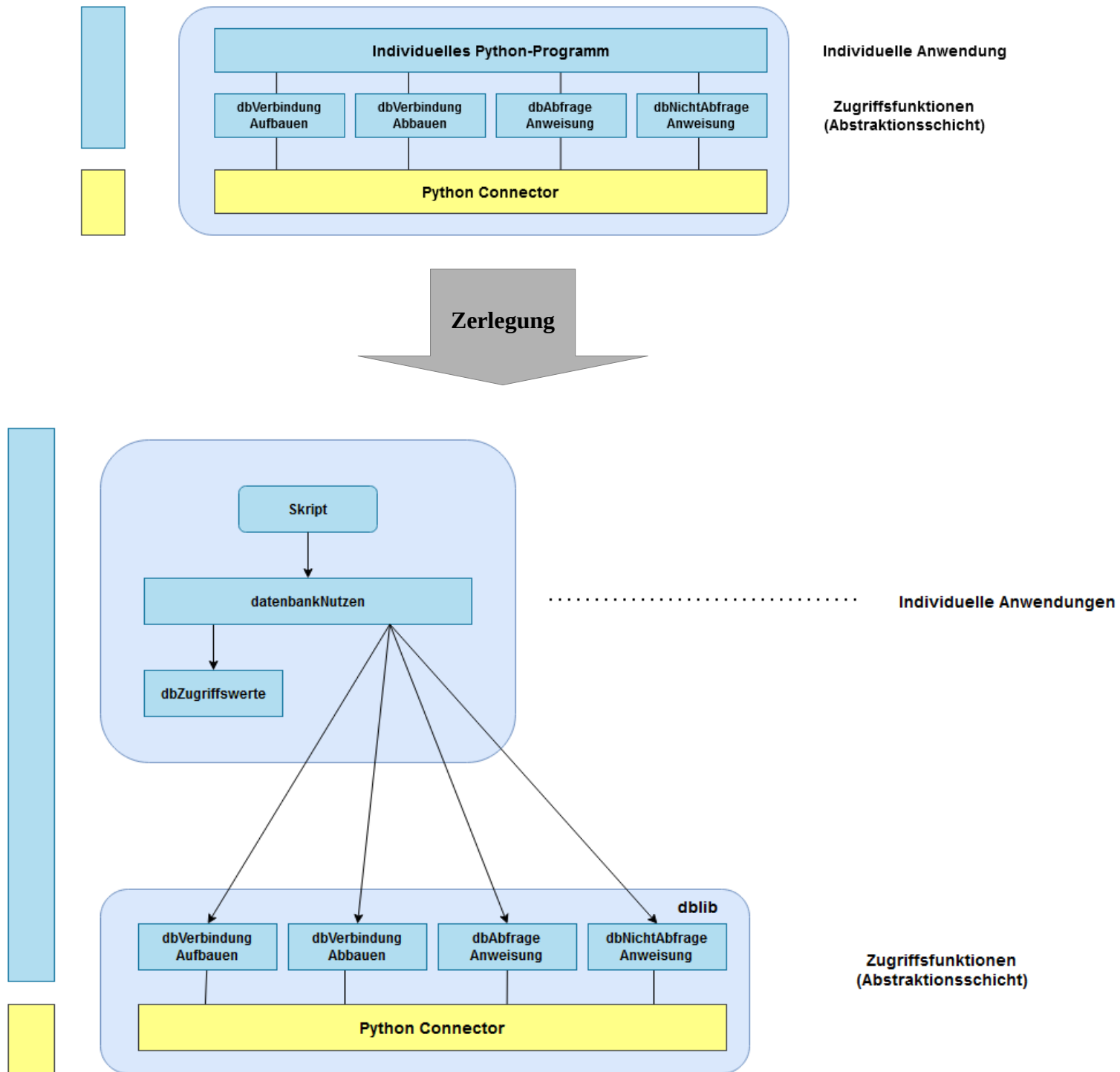
`beispiel_ls3_05.py`

```
import psutil as ps

def anzeigenResourcebVerbrauch():
    while True:
        cpuAuslastung = ps.cpu_percent(interval=1)
        print( "CPU-Auslastung:", cpuAuslastung, "%")
    return

anzeigenResourcebVerbrauch()
```

LS3 – Infos – Datenbankzugriff modular isolieren



LS3 – Infos – Versionsmanagement

Quelle:
www.wikipedia.de

Eine Versionsverwaltung (Versionsmanagement) ist ein System, dass zur Erfassung von Änderungen an Dokumenten und Dateien verwendet wird.

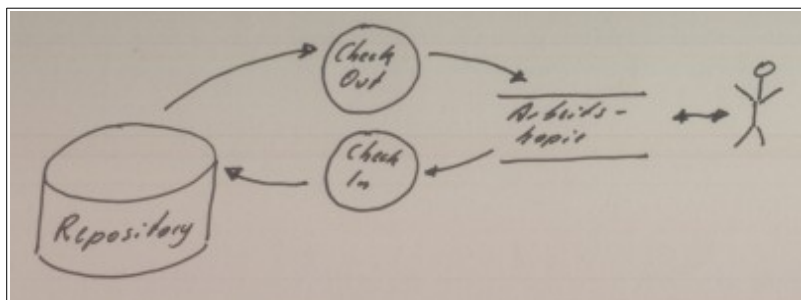
Aufgaben eines Versionsmanagementsystems

- Protokollierung von Änderungen; wer hat was wann geändert
- Wiederherstellung alter Stände
- Archivierung spezieller Stände
- Koordinierung gemeinsamer Zugriffe mehrerer Entwickler
- Gleichzeitige Entwicklung mehrerer Entwicklungszweige
- Zusammenfassung mehrerer Elemente zu einer Konfiguration
- Build-Unterstützung; automatische Generierung des Systems aus dem Repository

Ziele des Versionsmanagements

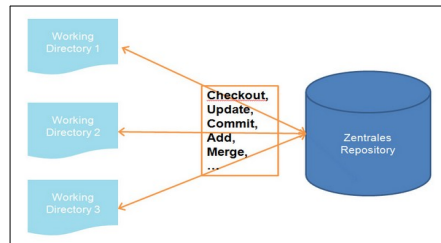
- Änderungen kontrollieren
- Qualität sicherstellen
- Produktivität steigern
- Transparenz verbessern

Prinzipielle Arbeitsweise

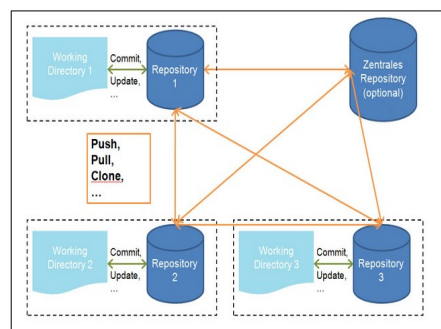


Typen von Versionsmanagementsystemen

1. **Lokale Versionsverwaltungssysteme** (z.B.: SCCS)
2. **Zentrale Versionsverwaltungssysteme** (z.B.: CVS, SVN(Subversion))



3. **Verteilte Versionsverwaltungssysteme** (z.B.: GIT)



Zugriffskonzepte

1. **Lock-Modify-Write/Unlock (Pessimistische Versionsverwaltung)**

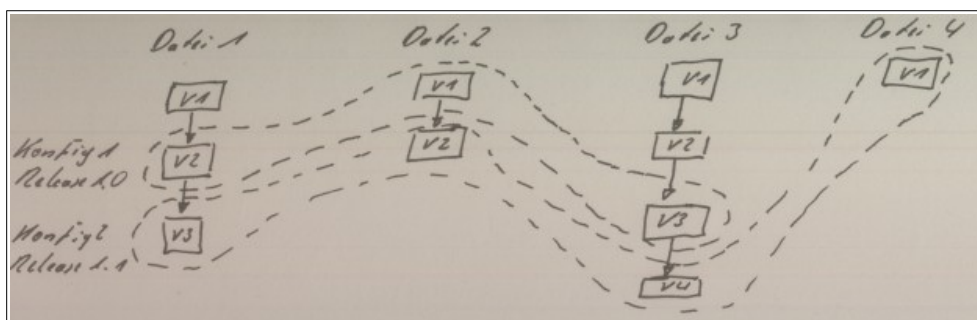
- Sequentielles Arbeiten
- kein Zusammenmischen (Merge) notwendig
- Warten auf Zugriff

2. **Copy-Modify-Merge (Optimistische Versionsverwaltung)**

- Gleichzeitiger Zugriff möglich
- kein Warten auf Zugriffe
- Automatisches oder manuelles Zusammenmischen notwendig

Versionsmanagement vs. Konfigurationsmanagement

- Oft als Synonym gebraucht
- Versionsmanagement eher Verwaltung eines einzelnen Elements
- Konfigurationsmanagement eher Zusammenspiel vieler Elemente zu einem Gesamtsystem



LS3 – Infos – Testen (I)

Quelle:

Leicht modifizierter und
ergänzter Chat-GPT Text



Ziel von Software Tests

Das Ziel von Software Tests ist es, die Qualität und Zuverlässigkeit von Software zu gewährleisten. Durch Tests können Fehler und Mängel in der Software erkannt und behoben werden, bevor sie in den produktiven Einsatz geht. Software Tests helfen dabei, die Funktionalität, Performance, Sicherheit und Benutzerfreundlichkeit einer Anwendung zu überprüfen und sicherzustellen, dass sie den Anforderungen und Erwartungen der Benutzer entspricht.

Typische Testinhalte

Bei Software Tests werden typischerweise verschiedene Aspekte der Software überprüft. Dazu gehören unter anderem:

- **Funktionale Tests:** Überprüfung, ob die Software die erwarteten Funktionen korrekt ausführt.
- **Nicht-funktionale Tests:** Überprüfung von Performance, Sicherheit, Benutzerfreundlichkeit und anderen nicht-funktionalen Aspekten.
- **Akzeptanztests:** Überprüfung, ob die Software die Anforderungen und Erwartungen der Benutzer erfüllt.

Blackbox- und Whitebox-Tests

Bei **Blackbox-Tests** wird die Software als "Blackbox" betrachtet, bei der nur die Eingaben und Ausgaben betrachtet werden. Es wird keine Kenntnis über die interne Funktionsweise der Software vorausgesetzt. Bei **Whitebox-Tests** hingegen wird die interne Struktur und Logik der Software berücksichtigt und getestet.

Komponententests, Integrationstests, Systemtests und Abnahmetests

1. **Komponententests** überprüfen einzelne Komponenten (Funktionen, Module, Klassen) einer Software auf ihre Funktionalität.
2. **Integrationstests**testen die Zusammenarbeit von verschiedenen Komponenten oder Modulen.
3. **Systemtests**überprüfen das gesamte System, einschließlich der Interaktion zwischen den Komponenten.
4. **Abnahmetests**werden durchgeführt, um sicherzustellen, dass die Software die Anforderungen und Erwartungen der Benutzer erfüllt und bereit für den produktiven Einsatz ist.

Performancetests und Regressionstests

Performancetests überprüfen die Leistungsfähigkeit einer Software, indem sie die Reaktionszeiten, die Skalierbarkeit und die Stabilität unter verschiedenen Lastbedingungen testen. **Regressionstests** werden durchgeführt, um sicherzustellen, dass nach Änderungen oder Erweiterungen an der Software keine neuen Fehler eingeführt wurden und bestehende Funktionen weiterhin wie erwartet funktionieren.

Testspezifikation und Testdokumentation

Testspezifikation bezieht sich auf die Planung und Definition der Tests. Hier werden die Testfälle und -szenarien festgelegt, die während der Tests durchgeführt werden sollen. **Testdokumentation** bezieht sich auf die Dokumentation der Testergebnisse, Fehler und Testabdeckung. Eine gute Dokumentation ermöglicht es, die Testergebnisse nachvollziehbar zu machen und Fehler zu analysieren.

Pytest und Testautomatisierung in der Python-Entwicklung

Pytest ist ein beliebtes Python-Testframework, das zur Automatisierung von Tests verwendet wird. Es bietet eine einfache und intuitive Syntax zum Schreiben von Tests und unterstützt verschiedene Arten von Tests wie Komponententests, Integrationstests und funktionale Tests.

Ein Beispiel für die Verwendung von Pytest könnte wie folgt aussehen:

```
# test_example.py
def add_numbers(a, b):
    return a + b

def test_add_numbers():
    assert add_numbers(2, 3) == 5
    assert add_numbers(5, 7) == 12
    assert add_numbers(10, -3) == 7
```

In diesem Beispiel wird die Funktion `add_numbers` getestet. Es werden verschiedene Testfälle definiert, die überprüfen, ob die Funktion die erwarteten Ergebnisse liefert. Mit Pytest können diese Tests einfach ausgeführt werden, indem das Kommando `pytest` in der Kommandozeile ausgeführt wird.

Pytest bietet auch viele nützliche Funktionen und Plugins, um die Testautomatisierung zu erleichtern, z. B. das Erfassen von Testabdeckung, das Erzeugen von Testberichten und das parallele Ausführen von Tests.

Test Driven Development (TDD)

Test Driven Development ist eine Entwicklungspraktik, bei der die Tests vor der eigentlichen Implementierung des Codes geschrieben werden. Durch das Schreiben der Tests zuerst wird sichergestellt, dass der Code die gewünschte Funktionalität erfüllt. Dieser Ansatz fördert auch die Modularität und Wartbarkeit des Codes.

Bedeutung von Testen bei Continuous Integration und DevOps

Continuous Integration (CI) und DevOps sind Entwicklungsansätze, die darauf abzielen, den Entwicklungsprozess effizienter und schneller zu gestalten. Tests spielen dabei eine entscheidende Rolle, um sicherzustellen, dass Softwareänderungen und -aktualisierungen reibungslos integriert und bereitgestellt werden können.

- **Continuous Integration** bezieht sich auf den Prozess des regelmäßigen Zusammenführens von Codeänderungen aus verschiedenen Entwicklungsbranchen in eine gemeinsame Codebasis. Bei CI werden Tests automatisiert durchgeführt, um sicherzustellen, dass die eingeführten Änderungen keine unerwünschten Nebenwirkungen haben und die bestehende Funktionalität nicht beeinträchtigen. Durch die Integration von Tests in den CI-Prozess können Probleme frühzeitig erkannt und behoben werden, was zu einer höheren Codequalität und einer schnelleren Bereitstellung führt.
- **DevOps** ist ein Ansatz, der die Zusammenarbeit zwischen Entwicklung (Development) und Betrieb (Operations) fördert, um die Softwarebereitstellung effizienter zu gestalten. Tests sind ein integraler Bestandteil des DevOps-Ansatzes und unterstützen die kontinuierliche Bereitstellung und Integration von Software. Durch die Automatisierung von Tests können Entwickler und Betriebsteams sicherstellen, dass Softwareänderungen reibungslos und zuverlässig in Produktionsumgebungen überführt werden können. Dies ermöglicht eine schnellere Time-to-Market und eine höhere Stabilität der bereitgestellten Software.

Testautomatisierung spielt eine entscheidende Rolle bei CI und DevOps. Durch die Automatisierung von Tests können Entwickler und Betriebsteams sicherstellen, dass Änderungen in der Software ohne manuellen Aufwand und mit hoher Geschwindigkeit getestet werden können. Dies ermöglicht es, Fehler frühzeitig zu erkennen und zu beheben, was zu einer höheren Codequalität und einer schnelleren Bereitstellung führt.

Darüber hinaus wird bei CI und DevOps oft auch der Ansatz der "Infrastruktur als Code" verwendet. Das bedeutet, dass die gesamte Infrastruktur, einschließlich der Testumgebungen, automatisiert und versioniert wird. Dadurch wird sichergestellt, dass Tests in einer konsistenten und reproduzierbaren Umgebung durchgeführt werden können, was zuverlässige und aussagekräftige Testergebnisse liefert.

Die Integration von Tests in den CI- und DevOps-Prozess ermöglicht es Entwicklern und Betriebsteams, schneller und effizienter zu arbeiten, indem sie die Qualität der Software sicherstellen und gleichzeitig die Bereitstellung beschleunigen. Es fördert auch eine Kultur der kontinuierlichen Verbesserung und Zusammenarbeit zwischen den verschiedenen Teams.

LS3 – Infos – Testen (II)

Quelle:

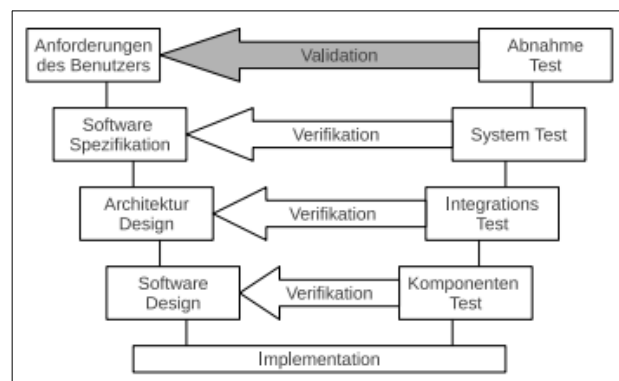
Florian Ehmke: Softwaretests
(Text tlw. leicht vereinfacht
und ergänzt)

Blackbox / Whitebox

Blackbox- und Whitebox-Tests bezeichnen zwei unterschiedliche Betrachtungen des zugrundeliegenden Systems während des Testens. Bei einem **WhiteboxTest** kennt man die Struktur und innere Funktionsweise der Software und macht Gebrauch von diesem Wissen. Es wird also der Quellcode mit berücksichtigt. Qualitätskriterien für Whitebox-Tests sind (unter anderem) die verschiedenen Formen der Codeabdeckung. Bei einem **BlackboxTest** kennt man die Struktur und innere Funktionsweise der Software nicht und das System wird gegen seine Spezifikation getestet.

Testebenen

Das Testen von Software findet auf unterschiedlichen Ebenen des Softwareentwicklungsprozesses statt. Bei höheren Ebenen spricht man von System- oder Abnahmetests, auf niedrigeren Ebenen finden Komponententests sowie Integrationstests statt. Grafisch dargestellt wird dies im populären V-Modell, welches den jeweiligen Ebenen des Softwareentwicklungsprozesses entsprechende Testformen zuordnet.



Die unterste Testebene — der **Komponententest** — testet die kleinsten testbaren Einheiten der Software. Das kann eine Klasse, eine Methode, eine Funktion oder ein Skript sein. Diese Komponenten werden später zusammengefügt (integriert) und mittels Integrationstests getestet. Bei test-getriebener Entwicklung (TDD – Test-driven development) wird der Komponententest vor der eigentlichen Komponente entwickelt und implementiert. Komponententests sind im Idealfall unabhängig von einander.

In der Phase der **Integrationstests** werden Komponenten kombiniert und in Gruppen getestet. Bei der Integrationsphase können Fehler entstehen, die selbst durch ausgiebige Komponententests nicht aufgedeckt werden. So kann beispielsweise vorkommen, dass 2 Komponenten jeweils eine temporäre Datei benötigen. Liegen diese temporären Dateien mit dem selben Namen im Verzeichnis entsteht ein Konflikt, der bei den jeweiligen Komponententests nicht entdeckt worden wäre. Integrationstests decken auch Fehler auf, die durch nicht eindeutige Spezifikationen entstanden sind. So ist die Maßeinheit Meilen nicht eindeutig. Es könnten sowohl Seemeilen als auch Landmeilen gemeint sein. Da diese unterschiedliche Werte haben kommt es bei der Interaktion zweier Komponenten, die unterschiedliche Meilen implementieren, zu Konflikten.

Der **Systemtest** ist die Teststufe in der gegen die Softwarespezifikation getestet wird. Das System befindet sich in dieser Phase in einem kompletten, integrierten Zustand. Die Testumgebung sollte der Produktivumgebung möglichst ähnlich sein oder ihr entsprechen. Da in dieser Phase keine Einblicke in den Code benötigt werden, handelt es sich um einen Blackboxtest. Im Zuge eines Systemtests werden unter anderem Performance-, Installations-, Wiederinbetriebnahme-, Stress- und Usability-Tests durchgeführt (vgl. Abschnitt 3.6).

Der **Abnahmetest** unterscheidet sich dadurch grundlegend von allen anderen Tests, dass er vom Benutzer durchgeführt wird und nicht vom Entwickler/Tester. Der Test findet außerdem in der (simulierten) Produktivumgebung statt und nicht länger in einer Testumgebung. Es wird getestet, ob das fertige Produkt aus Sicht des Benutzers alle Anforderungen erfüllt. Zu den geprüften Anforderungen gehören:

- Ist der Leistungsumfang vollständig? Wurden alle Spezifikationen (aus Nutzersicht) eingehalten?
- Ist die (Grafische) Benutzeroberfläche verständlich und benutzbar?
- Ist die Dokumentation angemessen umfangreich und verständlich?

Weitere Testformen

- Ein **Performancetest** ist ein Test, der gezielt die Performance eines System testet/misst und so den Entwickler dabei unterstützt Engpässe / Flaschenhalse aufzudecken. Mit den erlangten Erkenntnissen lässt sich die Performance des Systems leicht(er) optimieren.
- **Installationstests** simulieren eine Installation in der vorgesehenen Produktivumgebung und sollen Fehler, die durch die veränderte Umgebung (Entwicklungsumgebung anders als Produktivumgebung) entstanden sind aufdecken. Üblicherweise gehört auch ein Test der Deinstallations-Routinen zum Installationstest.
- Ein **Wiederinbetriebnahmetest** testet, wie sich die Software nach einem Ausfall bedingt durch z.B. einen Stromausfall verhält.
- Ein **Stresstest** untersucht, wie sich das System unter extremer Last verhält? Es wird kontrolliert ob es nach wie vor möglich ist Eingaben vorzunehmen oder anderweitig das Programm zu kontrollieren, wenn große Last auf dem System anliegt. Man kann hierbei zwischen 2 verschiedenen Lasten unterscheiden: 1. Last, die durch das zu testende Programm selbst erzeugt wird. 2. Last, die durch externe Programme entsteht und somit nicht immer vorhersehbar ist.

Wann immer Teile der Software verändert werden, sei es durch Erweiterung oder Veränderung der Funktionen oder Behebung eines Fehlers, ist es notwendig alle oder eine Teilmenge der vorhandenen Tests neu durchzuführen. Denn selbst kleinste Änderungen können zu unvorhergesehenen Fehlfunktionen führen. Man spricht dann von **Regressionstest**. Im Idealfall lassen sich diese, wie auch andere Testformen automatisieren.

Testphasen

