

README

Beschreibung

Zusammenfassung für Projektmanagement und Software Engineering auf Grundlage der Vorlesung FS 16 von Hans Heinrich Pletscher auf der Vorlage von H.Badertscher

Bei Korrekturen oder Ergänzungen wendet euch an einen der Mitwirkenden.

Modulschlussprüfung

Kompletter Stoff aus Skript, Vorlesung, Übungen und Praktikum

Schwerpunkt ist der in den Übungen behandelte Stoff, insbesondere Qt

Die Prüfung besteht aus 2 Teilen:

1.Teil	closed Book	Theoretische Fragen zum ganzen Prüfungsinhalt
2.Teil	semi-open book	Aufgaben im Stil der Übungen, Praktika und der in den Vorlesungen gelösten Aufgaben

Plan und Lerninhalte

Werkzeuge und Techniken

- Versionsverwaltung mit Subversion
- Unit-Testing mit CPPUnit
- Generierung der Dokumentation aus dem Source-Code mit Hilfe von Doxygen
- Erstellen von GUI-Programmen mit Hilfe der qt-Library.

Software Entwicklung

- Vorgehensmodelle
- Software Projektmanagement
- Testen von Software (u.a. Unit-Testing)
- Refactoring (Überarbeitung, Verbesserung bestehender Software)
- Allgemeine Entwurfsprinzipien: Design by Contract, defensives Programmieren
- Ereignisbasierte Programmierung, Entwurf von GUI-Programmen

Contributors

Michel Gisler	michel.gisler@hsr.ch
Stefan Reinli	stefan.reinli@hsr.ch
Luca Mazzoleni	luca.mazzoleni@hsr.ch
Hannes Badertscher	hannes.badertscher@hsr.ch

License

Creative Commons BY-NC-SA 3.0

Sie dürfen:

- Das Werk bzw. den Inhalt vervielfältigen, verbreiten und öffentlich zugänglich machen.
- Abwandlungen und Bearbeitungen des Werkes bzw. Inhaltes anfertigen.

Zu den folgenden Bedingungen:

- Namensnennung: Sie müssen den Namen des Autors/Rechteinhabers in der von ihm festgelegten Weise nennen.
- Keine kommerzielle Nutzung: Dieses Werk bzw. dieser Inhalt darf nicht für kommerzielle Zwecke verwendet werden.
- Weitergabe unter gleichen Bedingungen: Wenn Sie das lizenzierte Werk bzw. den lizenzierten Inhalt bearbeiten oder in anderer Weise erkennbar als Grundlage für eigenes Schaffen verwenden, dürfen Sie die daraufhin neu entstandenen Werke bzw. Inhalte nur unter Verwendung von Lizenzbedingungen weitergeben, die mit denen dieses Lizenzvertrages identisch oder vergleichbar sind.

Weitere Details: <http://creativecommons.org/licenses/by-nc-sa/3.0/ch/>

Projektmanagement und Softwareengineering

M. Gisler S. Reinli L. Mazzoleni

26. August 2016

Inhaltsverzeichnis

1 Software Entwicklung	2
1.1 Schwierigkeiten/Eigenschaften	2
1.2 Bewältigung der Komplexität	2
1.3 Magical Number Seven	2
1.4 Effekt des zweiten System	2
1.5 Vorgehen zur Softwareentwicklung	3
1.6 Vorgehensmodelle (Klassisch)	3
1.7 Vorgehensmodelle (Agil)	4
1.8 Objektorientierte Softwareentwicklung (OOAD)	5
1.9 Kriterien für die Wahl des Modells	5
1.10 The Big Bang Problem bei Wasserfallmodellen	5
2 Testing	6
2.1 Allgemeine Begriffe	6
2.2 Testmethoden	7
2.3 Testwerkzeuge	8
2.4 Wer testet?	8
2.5 Automatisiertes Testing	9
2.6 Unit-Test	9
3 Dokumentation & Doxygen	11
3.1 Dokumentation	11
3.2 Doxygen	11
4 Projektmanagement	13
4.1 Grundlagen	13
4.2 Projektablauf	14
4.3 Projektphasen	14
5 Versionskontrolle	17
5.1 Versionskontrollsysteme (VCS)	17
5.2 Git	18
6 Qt	24
6.1 Grundlagen zu Qt	24
6.2 QWidget (Widget = Window Gadget ("Fenster Ding"))	24
6.3 GUI-Programmierung	25
6.4 Layout	25
6.5 Interaktion	26
6.6 Zeichnen und Malen	26
7 Beispiele	27
7.1 CppUnit	27
7.2 Doxygen	28
7.3 Qt	30

1 Software Entwicklung

Die Softwareentwicklung ist schwierig, Hauptgrund: Komplexität. Ein Softwareprojekt kann sich aufgrund der Komplexität in einen Werwolf verwandeln. Dieser kann nur mit einer Silberkugel getötet werden.

Frederic Brooks hat dies analysiert:

Software Projects are still:

- LATE
- UNDER FEATURED
- OVER BUDGET
- FULL OF BUGS

- "No Silver Bullet" → Es gibt kein Wundermittel um komplexe Software zu entwickeln
- "Adding Manpower to a late Software project makes it later"
 - Neue Leute müssen sich einarbeiten und Kommunikationsaufwand steigt.

1.1 Schwierigkeiten/Eigenschaften

2 Arten von Schwierigkeiten führen zu Komplexität

1. Essentielle Schwierigkeiten	Verursacht durch Komplexität des Problems Essenz: Kern der Sache
Eigenschaften (essential)	Grundlegende Eigenschaften <u>Bsp.</u> <ul style="list-style-type: none"> • Auto muss einen Motor, Räder und Türen haben, sonst ist es kein Auto.
2. Akzidentelle Schwierigkeiten	Selbst verursacht durch falsche Methoden, Mittel Akzidenz: Art der Realisierung
Eigenschaft (accidental)	Zufällige Eigenschaft, die ein Objekt haben kann. <u>Bsp.</u> <ul style="list-style-type: none"> • Achtzylinder oder Vierzylindermotor

1.2 Bewältigung der Komplexität

1. Teile und herrsche "divide et impera"

- Problem in Teilprobleme aufteilen
- Jedes Teilproblem für sich alleine lösen, alle zusammen ergeben die Problemlösung
- Möglichst eigenständige Teile (hohe Kohäsion)
- Möglichst geringe Abhängigkeiten (kleine Kopplung)

2. Abstraktion

- Definition: Weglassen von Aspekten, die für den gegenwärtigen Zweck nicht wichtig sind.

3. Modelle

⇒ visuelle Darstellung des Sachverhalts

- Abstraktion der realen Welt, Es gibt zwei Typen von Modellen:
 - Design-Modelle (Lösungsdarstellung)
 - Domain-Modelle (Problemdarstellung)

1.3 Magical Number Seven

Ein Mensch kann nur gleichzeitig 7 ± 2 Informationseinheiten im Arbeitsgedächtnis präsent halten.

1.4 Effekt des zweiten System

Für die gewünschte Aufgabe gibt es bereits funktionsfähige Systeme, welche auch problemlos funktionieren. Das neue zweite System soll wesentlich mehr können und besser sein als das erste, ansonsten bräuchte es gar kein neues System. Dieser Ansatz führt zur "eierlegenden Wollmilchsau" welche alles können soll, aber sehr schwierig zu bauen ist. Dieser Effekt ist nicht nur in der Softwareentwicklung verbreitet.

1.5 Vorgehen zur Softwareentwicklung

Wenn die folgenden Schritte gleich nacheinander durchgeführt werden, spricht man von *Phasen* und *Phasenplänen*.

1. Analyse

- Ziel: Man weiss **WAS** entwickelt werden soll.
→ Man weiss nacher mehr als vorher
- Probleme, Ideen, Anforderungen aufnehmen
- Erstellen eines Lasten-, Pflichtenhefts und einer Anforderungsspezifikation
- *"doing the right things"* → Die richtigen Dinge tun

2. Design

- Ziel: Man weiss **WIE** das Produkt entwickelt werden soll
- Erstellen eines **Grobentwurfs** (Festlegung der Software-Architektur, Hilfsmittel) und eines **Detaillentwurfs** (UML-Diagramme). Es entsteht noch kein Code. Nur Baupläne für die Software
- *"doing things right"* → Die Dinge richtig tun

3. Implementierung

- Codieren
- Debuggen
- (Testen) ⇐ Nicht immer!

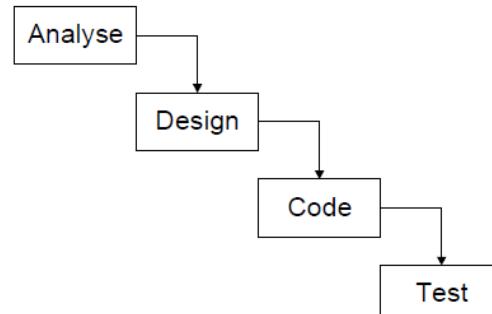
1.6 Vorgehensmodelle (Klassisch)

1.6.1 Begriffe

- Ein Vorgehensmodell dient dazu die Softwareentwicklung übersichtlicher zu gestalten
- Schwergewichtiger Prozess → Klassische Softwareentwicklung
- Leichtgewichtiger Prozess → Agile Softwareentwicklung

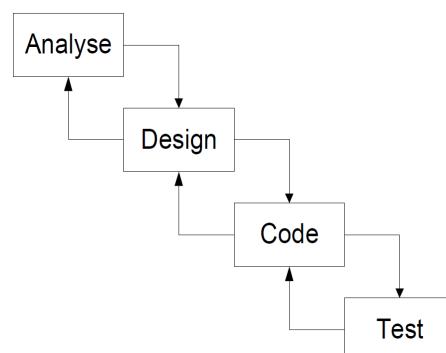
1.6.2 Wasserfall-Modell

- lineares Vorgehensmodell
- nicht iterativ
- in Realität nicht durchführbar, da keine Rückkopplung vorhanden ist
- Freigabe beim Abschluss jeder Phase
- Es gibt kein zurück, alles muss beim ersten Mal richtig gemacht werden.



1.6.3 Iterativer Wasserfall (Kaskade)

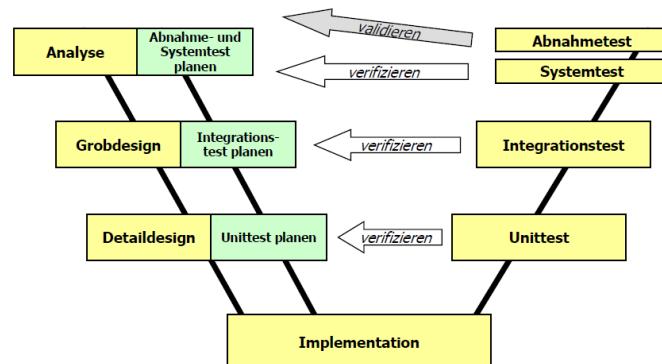
- Erweiterung des Wasserfalls um Korrekturschleifen
- In Realität durchführbar
- Man kann immer nur eine Phase zurück



1.6.4 V-Modell

Von der deutschen Bundeswehr für ihre eigenen IT-Projekte entwickelt.

- Grundidee: Korrespondierende Tests
 - x-Achse = Zeit
 - y-Achse = Detaillierungsgrad
 - Weiterentwicklung V-Modell XT,
immer projektspezifische Anpassungen nötig
(XT = Extreme Tailoring, engl. „to tailor“ =
schneidern)



1.6.5 Iterative / Inkrementelle Entwicklung / Spiralmodell

Es gibt zwei Modelle:

1. Spiralmodell
 2. RUP (Rational Unified Process)

Inkrementell:

- Resultat in Schritten entwickeln
 - Jeder Schritt ist Teil des Endresultats
⇒ wird nicht mehr angepasst

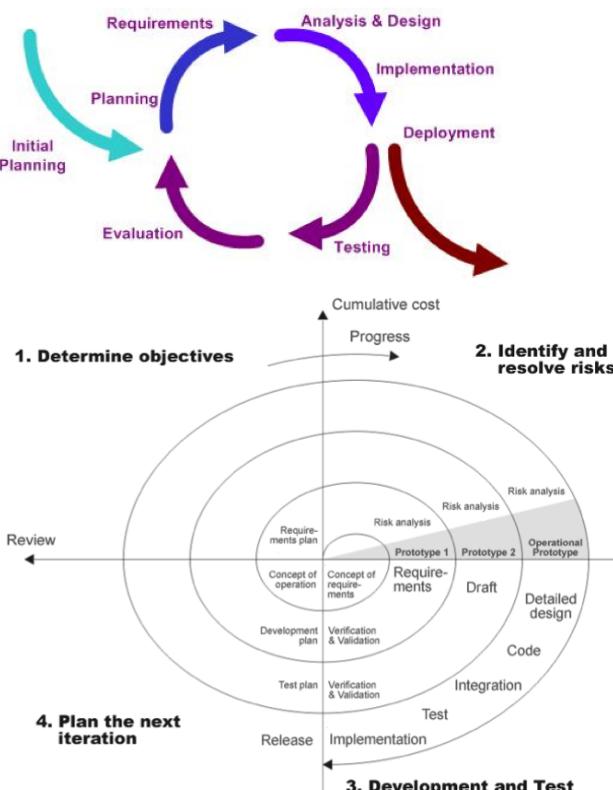
Iterativ:

- Erste Version entwickeln
 - In weiteren Schritten bessere Versionen erstellen

1.7 Vorgehensmodelle (Agil)

⇒ Ist 3 Mal erfolgreicher als das Wasserfall-Modell

- weniger formalisiert als schwerfällige, bürokratische klassische Softwareentwicklung
 - agil = flink, beweglich
 - Individuen und Interaktionen gelten mehr als Prozesse und Tools

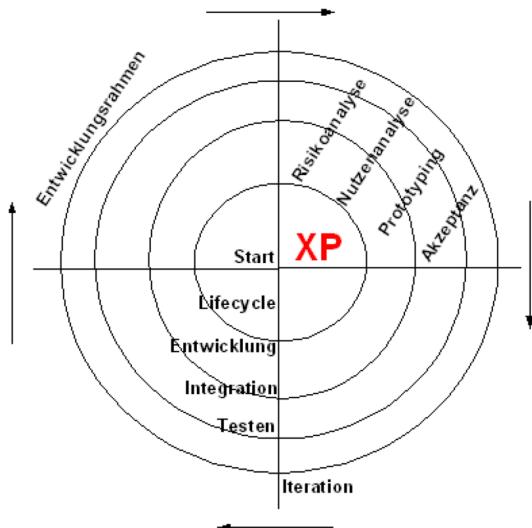


Agile Softwareentwicklung besteht aus:

1. **Agile Werte:** bilden das Fundament.
Menschen & Interaktionen wichtiger als Prozesse & Werkzeuge
Funktionierende Software steht über Dokumentation
Reagieren auf Veränderung anstelle von Regeln befolgen
 2. **Agile Prinzipien:** sind Handlungsgrundsätze, die auf agilen Werten basieren.
Zufriedenstellung des Kunden durch frühe, kontinuierliche Auslieferung von Software
Fortschrittsmaß gilt die Funktionsfähigkeit der Software
 3. **Agile Methoden:** sind konkrete Verfahren, die sich auf agile Werte und Prinzipien stützen.
 4. **Agile Prozesse:** sind die Zusammenfassungen angewandten Methoden.
Vorgehensmodelle wie XP, Scrum...

1.7.1 XP (Extreme Programming)

- Bekanntester agiler Prozess

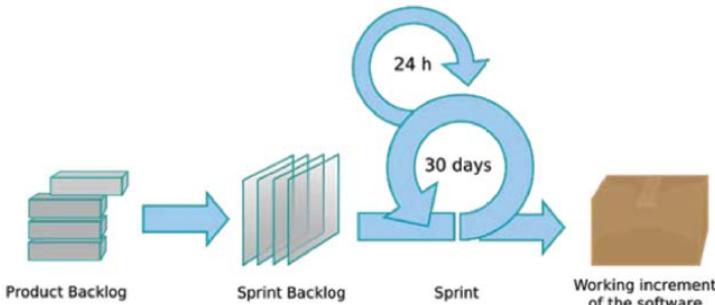


1.7.2 Scrum

⇒ Basiert auf sogenannten User Stories (Use Cases)

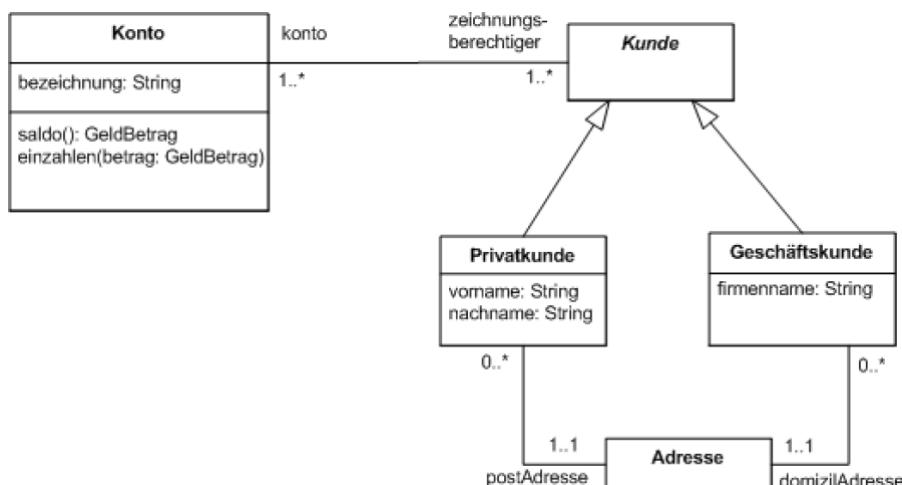
- Gedränge

- Basis:** Sprints von 15-30 Tagen, in welchen vom Team eine neue (bessere) Softwareversion erstellt wird.



1.8 Objektorientierte Softwareentwicklung (OOAD)

- Modelle auf allen Stufen: **OO-Analyse → OO-Design → OO-Implementation**
- Objekte sind Abstraktionen von realen Dingen (Modellen)
- Klassen entstehen durch Gruppieren und Zusammenfassen von Objekten mit gleicher Datenstruktur
- Gleiche Notation bei OOA, OOD, OOI
- Darstellung mit UML
- OOAD kann in jedem Vorgehensmodell verwendet werden



1.9 Kriterien für die Wahl des Modells

- Projektgrösse
- Projektkomplexität
- Verfügbarkeit der Ressourcen
- Zeitpunkt von Änderungen (diskret, laufend)
- Qualität der Anforderungsdefinitionen
- Stabilität bzw. Volatilität der Anforderungen

1.10 The Big Bang Problem bei Wasserfallmodellen

- zuerst lange Analyse-Phase mit dem Kunden
- dann lange Phase für Design, Codierung und Test
→ Kein Kundenkontakt
- Einführung des Systems beim Kunden ← **Big Bang**
- Folge:** Hektisches Nachbessern
⇒ Häufig wird dadurch das Vertrauen des Kunden zerstört.

2 Testing

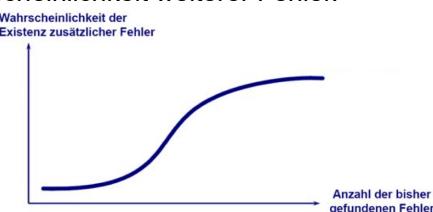
2.1 Allgemeine Begriffe

- **Definition:** Programm mit der Absicht ausführen, Fehler zu finden.
- Durch das Testen kann die Korrektheit eines Programms aber nicht bewiesen werden!
Ausnahme: triviale Programme
- Mit Testen kann nur die Anwesenheit von Bugs bewiesen werden, aber nicht die Abwesenheit.
- Erster Bug: 1947 wurde eine Motte im Relais eines Mark II Aiken Rechners entdeckt.

Nach sorgfältigem Testen steigt die Wahrscheinlichkeit, dass das Programm sich auch in nicht getesteten Fällen wunschgemäß verhält.

2.1.1 Wahrscheinlichkeit von Fehlern

Je mehr Fehler gefunden werden, desto höher ist die Wahrscheinlichkeit weiterer Fehler.



2.1.2 Testen & Debuggen

Ziel des Testing:

- Aufzeigen, dass Fehler existieren.
- *"Jeder gefundene Bug ist ein Gold Nugget"*

Ziel des Debugging:

Durch Testing gefundene Fehler lokalisieren und beheben.

2.1.3 Verifikation & Validierung

Validierung:

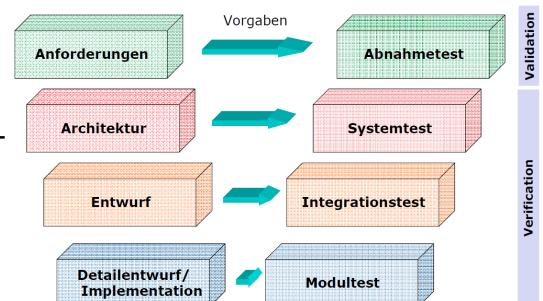
"Are we building the right product?"

Überprüft das Produkt ob es die Anforderungen des Auftraggebers erfüllt.

Verifikation:

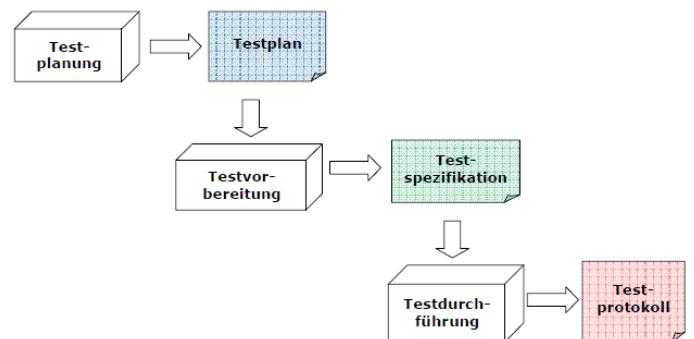
"Are we building the product right?"

Überprüft ob die erstellte Software funktioniert.

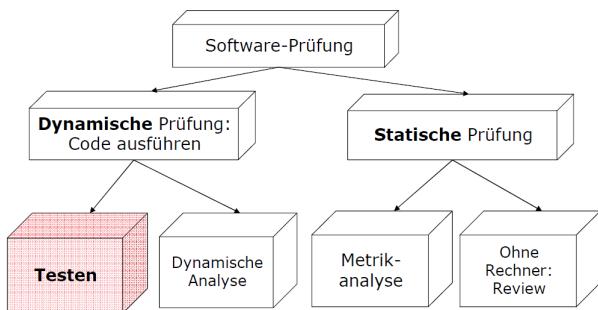


2.1.4 Anforderungen an Softwaretests

- Geplant: Testplanung → Testplan
- Systematisch spezifiziert → Testspezifikationen
- Testresultate festgehalten → Testprotokoll
- Reproduzierbarkeit:
 - Wissen, was getestet wurde
 - Unabhängig von testender Person
- wenn möglich automatisiert
- Testspezifikationen laufend erweitern,
⇒ Regressionstests



2.1.5 Massnahmen zur Softwareprüfung

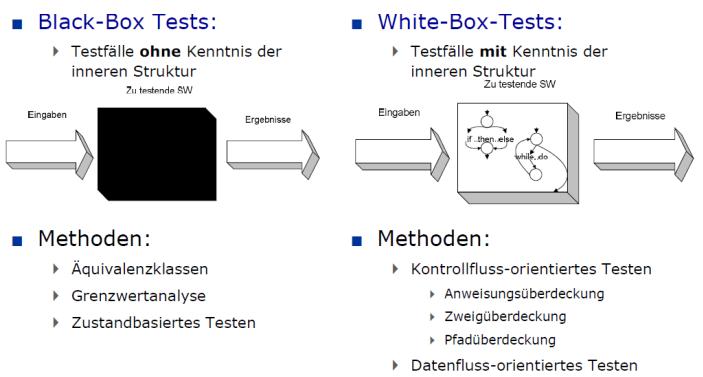
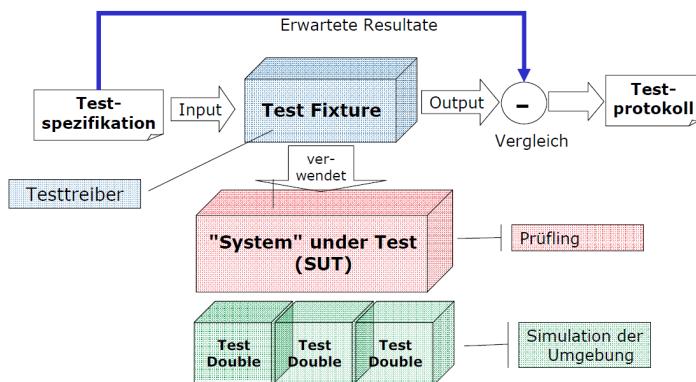


2.1.6 Arten von Tests

Anforderungskategorien	Testkategorien
<ul style="list-style-type: none"> Funktionale Anford. Nichtfunktionale Anford. <ul style="list-style-type: none"> Leistung Usability 	<ul style="list-style-type: none"> Funktionale Tests Nichtfunktionale Tests <ul style="list-style-type: none"> Leistungstests Usabilitytests

2.2 Testmethoden

2.2.1 Testumgebung



2.2.2 Blackbox Tests

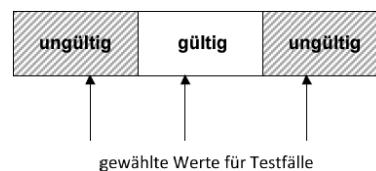
Testfälle **ohne** Kenntnis der inneren Struktur

Äquivalenzklassen:

Wertebereich, für welche das Programm voraussichtlich das gleiche Verhalten zeigt.

Methodik: 1 Testcase pro Äquivalenzklasse

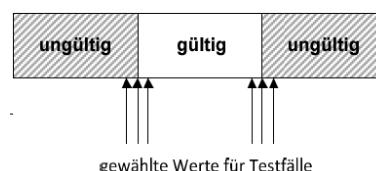
Beispiel: Quadratische Gleichung; Determinante < 0 ; $= 0$; > 0



Grenzwertanalyse:

Fehler liegen oft an Grenzen zulässiger Eingabewertbereiche.

Methodik: Testfälle auf Grenzen und knapp daneben



Zustandsbasiertes Testing:

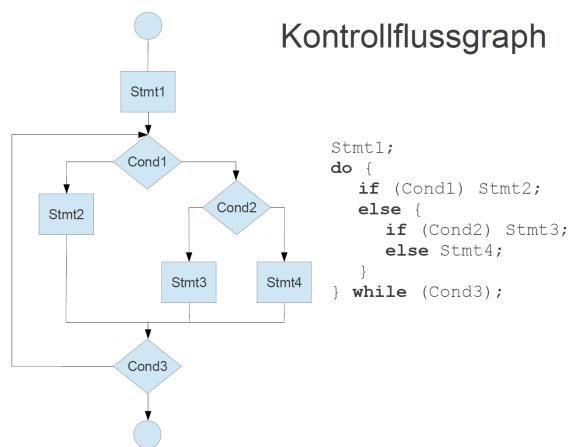
Beispiel Stack:

Testfall	Zustand	Stack leer	Stack halbvoll	Stack voll
Element hinzufügen				
Element entfernen				

2.2.3 Whitebox Tests

Tests mit Kenntnis der inneren Struktur

- Whitebox-Test werden mittels Kontrollfluss-Graphen durchgeführt.
 - Jedes Statement entspricht Knoten.
 - Zweige, stellen Schleifen und Verzweigungen dar, verbinden Knoten.
- Der Test wird so ausgelegt, dass die Überdeckung (Coverage) möglichst gut ist. (Test der Überdeckung mit Dynamic Analyzer messen)

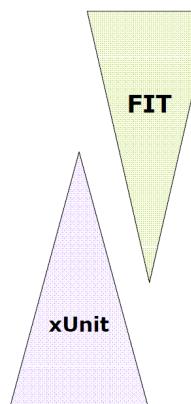
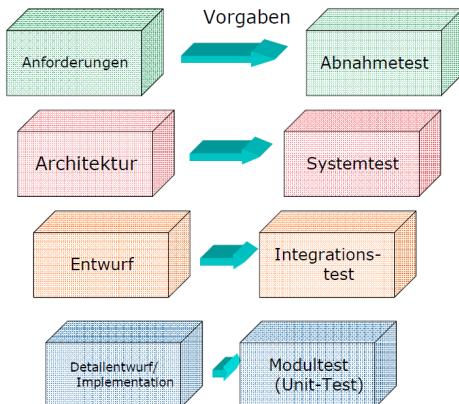


2.2.3.1 Testüberdeckung (test coverage)

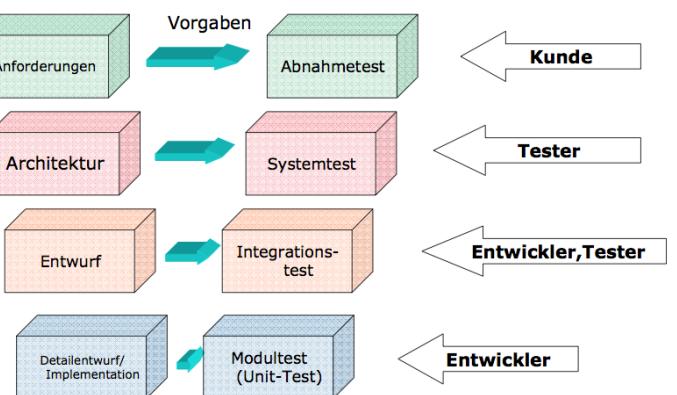
- Anweisungsüberdeckung (statement coverage)**
 - Prozentualer Anteil der Anweisungen welche im Test ausgeführt werden
 - 100% Anweisungsüberdeckung ist Minimum
- Zweigüberdeckung (branch coverage)**
 - Prozentualer Anteil der Zweige welche durchlaufen werden
 - 100% Zweigüberdeckung \Rightarrow 100% Anweisungsüberdeckung
- Bedingungsüberdeckung**
 - Verschiedene Kombinationen testen bei zusammengesetzten Bedingungen
 - mind. 1x true/false durchlaufen
- Pfadüberdeckung (path coverage)**
 - Prozentualer Anteil der Pfade welche im Test durchlaufen werden
 - Ein Pfad ist möglicher Weg durch Kontrollgraphen
- Funktionsüberdeckung**
 - "Tut es das, was der Kunde spezifiziert hat?"
 - 1 Szenario pro Use Case
 - Blackbox Tests

2.3 Testwerkzeuge

Blackbox: FIT
Whitebox: xUnit



2.4 Wer testet?



2.5 Automatisiertes Testing

Vorteile:

- Wiederholbarkeit
⇒ Regressionstests möglich
- Eindeutige Spezifikationen (Testcode ist Programmcode)

Nachteile:

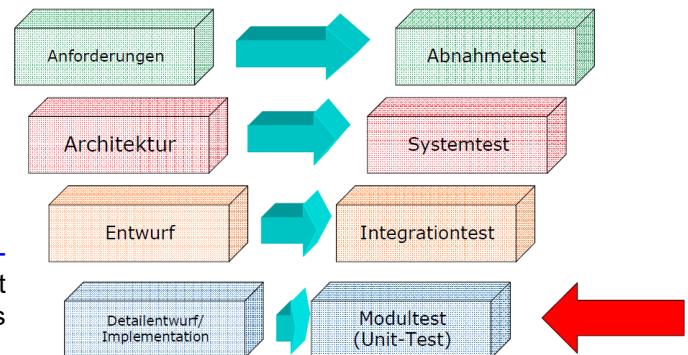
- Mehr Code zu schreiben und zu pflegen
- Testcode ist Programmcode: Wird überhaupt das Richtige getestet?

2.6 Unit-Test

2.6.1 Konzept

- Test häufig durch Programmierer selbst
- Test einer Komponente (Unit)
- Testen der Schnittstelle der Unit
- Im Voraus bekannte Ergebnisse → Assertions einfügen
- automatisch, wiederholbar
- heute mittels **Unit-Test-Frameworks**

Frameworks: Programmgerüst in welches das **Anwendungsprogramm** eingebettet wird. Die Funktionen der Unit werden aus dem Framework heraus aufgerufen, sogenanntes **Hollywood-Prinzip**:



"Don't call us, we'll call you"

2.6.2 Arbeitsweise

- Spezielle, möglichst einfache Testfunktion mit Zusicherungen
 - ASSERT(...) Vergleicht ein Soll- mit einem Ist-Wert
 - Stimmen beide überein, ist der Test erfolgreich
 - Sind sie unterschiedlich → Test fehlgeschlagen
- Tests laufen automatisiert ab
- *Test-Runner* = Programm, das die Test-Funktionen der Reihe nach ausführt
 - Test-Run endet mit "OK(...)" oder "FAILED: ...".
- **Achtung:**
 - ASSERT(...) - Anweisungen von Unit-Test nicht mit dem "assert(...)"-Makro aus "assert.h" verwechseln

2.6.3 Unit-Test Frameworks: Vorteile

- Übersichtliche Organisation der Testfälle
- Aufteilung der Tests auf mehrere Dateien in der Regel
 - Zum Beispiel eine Testklasse pro zu testende Klasse.
- Innerhalb der Datei: Organisation der Testfälle in Form von Testfunktionen und Testsuiten.
- Automatische Bereitstellung und Abbau einer Testumgebung ist möglich.
 - bspw. `setUp()`, `tearDown()`
- Testprotokolle: Ausgabe der Testergebnisse in verschiedenen Formen möglich.

2.6.4 CppUnit: Wichtige Begriffe

- **Assert-Anweisung**
→ Zum Vergleich von Ist- und Soll-Wert
- **TestKlasse**
→ selbst geschriebene Klasse, welche von "CppUnit::TestCase" erbt.
- **TestFunktion**
 - enthält Assert-Anweisungen
 - ist Memberfunktion einer Testklasse
- **TestSuite**
→ zur Zusammenfassung von Testfunktionen
- **TestFixture**
→ zur Bereitstellung und Abbau einer Testumgebung
- **Registry**
→ zur Zusammenfassung von Testklassen

2.6.5 CppUnit: Testklasse

```
// Filename: MyClassTest.cpp (no MyClassTest.h-File)
// Testklasse: enthält den Testcode
#include "MyClass.h" // <-- zu testende Klasse
#include <cppunit/extensions/HelperMacros.h>
class MyClassTest: public CppUnit::TestFixture
{
    CPPUNIT_TEST_SUITE( MyClassTest );
    CPPUNIT_TEST( testFunction1 );
    CPPUNIT_TEST( testFunction2 );
    CPPUNIT_TEST_SUITE_END();
    void testFunction1(void)
    {
        CPPUNIT_ASSERT(object1.mySqrt(4) == 2);
        CPPUNIT_ASSERT_EQUAL(4, 2*2);
    }
    void testFunction2(void)
    {
        // ...
    }
};
CPPUNIT_TEST_SUITE_REGISTRATION( MyClassTest );
```

2.6.6 CppUnit: Konventionen

- Für jede zu testende Klasse wird eine entsprechende Testklasse erstellt
- Der Name einer Testklasse beginnt mit dem Namen der zu testenden Klasse und endet mit "Test".
- Der Name einer Testfunktion beginnt mit "test"
Beispiel: "testAddition()"

2.6.7 Testprinzipien

Erst der Test, dann der Code dazu

2.6.8 Good Unit Tests are "A TRIP"

- **Automatic – automatisch**
 - Unit-Test müssen automatisch ablaufen: automatisch bezüglich a) dem Aufrufen der Tests und b) dem Prüfen der Ergebnisse.
- **Thorough – gründlich**
 - alles testen, was vermutlich schief gehen kann.
- **Repeatable – wiederholbar**
 - gleiche Ergebnisse unabhängig wie oft aufgerufen (Aufräumen nicht vergessen).
- **Independent – unabhängig**
 - keine (zeitlichen) Abhängigkeiten zwischen einzelnen Tests
- **Professional – professionell**
 - Gleiche Qualitätsanforderungen für die Unit-Tests wie beim anderen Code. Auch Tests überprüfen: Bedingungen negieren und prüfen, ob Test fehlschlägt. Absichtlich kurzfristig Fehler in Code einbauen.

2.6.9 CORRECT - Boundary Conditions

- **Conformance**
 - z.B. email Adresse prüfen: foo@bar.com
- **Ordering**
 - Ist Reihenfolge relevant? Was passiert wenn falsch?
- **Range**
 - Stimmt der Wertebereich der Ergebnisse
- **Reference**
 - Annahmen über Umfeld prüfen
- **Existence**
 - ist etwas da? nicht null?
- **Cardinality**
 - off-by one errors, 0,1,viele
- **Time**
 - Reihenfolge in der etwas passiert, Concurrency

2.6.10 CppUnit: Assert-Makros

- **CPPUNIT_ASSERT (condition)**
CPPUNIT_ASSERT_MESSAGE (msg, condition)
Test ist okay (d.h. wird bestanden), falls 'condition' true ist.
- **CPPUNIT_FAIL (msg)**
Test, der immer fehlschlägt.
- **CPPUNIT_ASSERT_EQUAL (expected, actual)**
CPPUNIT_ASSERT_EQUAL_MESSAGE (msg, expected, actual)
Test ist okay, falls 'expected' und 'actual' gleich sind. Dabei werden einfache Datentypen ('int' etc.) sowie 'std::string' unterstützt.
- **CPPUNIT_ASSERT_DOUBLES_EQUAL (expected, actual, delta)**
Test ist okay, falls 'expected' und 'actual' innerhalb einer Toleranz von 'delta' gleich sind. Die Argumente sind vom Datentyp 'double'.
- **CPPUNIT_ASSERT_THROW (expression, ExceptionType)**
Test ist okay, falls 'expression' eine Ausnahme vom Typ 'ExceptionType' auslöst.
- **CPPUNIT_ASSERT_NO_THROW (expression)**
Test ist okay, falls 'expression' keine Ausnahme auslöst.
- Bei den Varianten mit 'msg' (= message) wird diese zusätzlich ausgegeben, falls der Test fehlschlägt.

2.6.11 Typischer Output

Suite hat 2 Tests fehlerlos absolviert

...

OK (2)

Suite hat 10 Tests mit Fehler in Test 2 und 5 absolviert

...

```
MyClassTest.cpp:32:Assertion
Test name: MyClassTest::testCtorWithValues
equality assertion failed
- Expected: 0
- Actual : 15
```

```
MyClassTest.cpp:57:Assertion
Test name: MyClassTest::testEquals
assertion failed
- Expression: object.isValid()
```

Failures !!!

Run: 10 Failure total: 2 Failures: 2 Errors: 0

3 Dokumentation & Doxygen

3.1 Dokumentation

3.1.1 Zweck der Dokumentation

- Wissenssicherung / Wissenstransfer
- Kommunikation
- Sichtbarkeit des Projektfortschritts

3.1.2 Dokumentation von Software

- Benutzung von bestehenden Programmteilen
→ Schnittstellenbeschreibung (API, Application Program Interface)
- Zielgruppe ist Programmierer
- Dokumente werden oftmals nicht nachgeführt, wenn Änderungen am Sourcecode gemacht werden → Inkonsistenz
→ Programmdokument ist fehlerhaft und verliert dadurch an Wert.
- Im Sourcecode werden mittels speziell gekennzeichneten Kommentaren Beschreibungen erstellt.
- Mittels Dokumentationswerkzeugen kann Kommentar Source-Code extrahiert werden und daraus eine Beschreibung erstellen.

3.2 Doxygen

Doxygen ist ein solches Dokumentationswerkzeuge wie oben erwähnt. Es erstellt API-Beschreibungen. Es ist weit verbreitet und Open-Source. (Beispiel im Anhang)

- unterstützt Sprachen: C,C++,Java, C#,...
- Ausgabeformate: HTML, LaTeX, RTF, XML,...
- auch UML-Diagramme möglich
- Ist grundsätzlich ein Command Line Tool
- Hauptbefehl (command-line): \$ `doxygen`
- existiert aber auch ein GUI und ein Eclipse-Plugin (Eclox, in Eclipse das Symbol @)
- benötigt Konfigurationsdatei "Doxyfile"
→ "Doxygen" ist sehr umfangreich: rund 70.. 100 Optionen.
→ Wichtigster Parameter: PROJECT_NAME
- im File Doxyfile sind alle Einstellungen gespeichert was alles in die Dokumentation muss

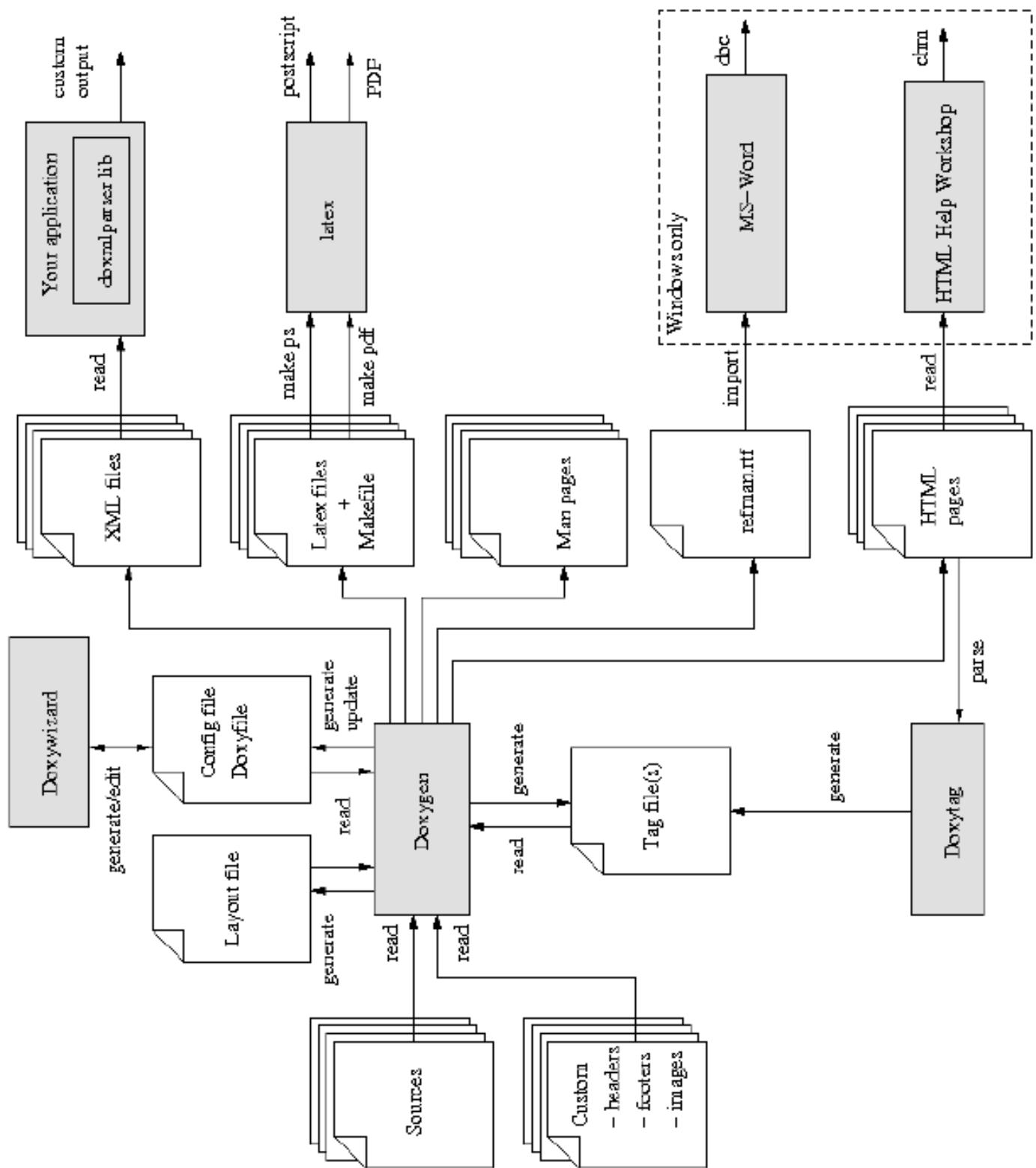
3.2.1 Doxygen Commands

<code>/** DoxyCmt */</code>	Doxygen Comment
<code>@file</code>	File Name. Next Line Description of the File.
<code>@author</code>	Author
<code>@version</code>	Version
<code>@date</code>	Datum
<code>@bug</code>	A known Bug
<code>@brief</code>	One Line description
<code>@extended</code>	Description over several Lines
<code>@param</code>	Description of your Parameter
<code>@return</code>	Description of your Returnvalue
<code>@warning</code>	Warnings
<code>@note</code>	Note

3.2.2 Aufruf-Beispiele

- `doxygen` // gibt Hilfe-Text aus
- `doxygen -g` // erzeugt Konfig.-Datei "Doxyfile"
- `doxygen Doxyfile` // erzeugt Dokumentation

3.2.3 Doxygen Architektur



4 Projektmanagement

4.1 Grundlagen

4.1.1 Definition eines Projektes

Ein Projekt ist ein zeitlich beschränktes Vorhaben zur Erzeugung eines einmaligen Produktes oder Dienstes.

4.1.2 Definition des Projektmanagement

Das Projektmanagement bezeichnet die Gesamtheit von Führungsaufgaben, -organisation, -techniken und -mitteln für die Initiierung, Definition, Planung, Steuerung und den Abschluss von Projekten.

4.1.3 Merkmale eines Projektes

- einmaliges Vorhaben
- klare Ziele
- hat Risiken
- zeitlich begrenzt
- Kostenrahmen
- hat innovativen Charakter
- es arbeiten mehrere Personen daran
- hat Projektleiter & Projektteam

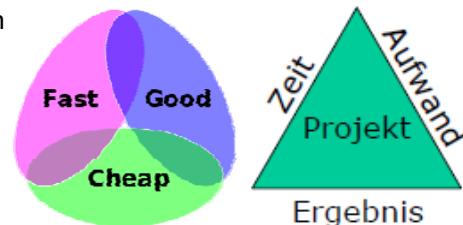
4.1.4 Kategorien von Projekten

- Forschungs- und Entwicklungsprojekte
 - Entwicklung neuer Produkte
 - Erstellung von Software
- Investitionsprojekte
 - Bau eines Flughafen
- Organisationsprojekte
 - Einführung Software
 - Vorbereitung Veranstaltung

4.1.5 Die 3 Eckpfeiler eines Projektes

Diese drei Größen sind voneinander abhängig und sind auch bekannt als "Fast, Good, Cheap"

- Ergebnis "Scope" → Spezifikation
- Zeit "Time" → Terminplan
- Aufwand "Cost" → Budget



4.1.6 Das Projektdreieck (Magisches Dreieck)

Diese drei Größen werden als Dreieck dargestellt. Die Aufgabe des Projektmanagements ist es, ein sinnvolles Verhältnis dieser 3 herzustellen und während der Projektdauer zu gewährleisten.

4.1.7 Projektbeteiligten (Stakeholder)

- Auftraggeber → bezahlt & befiehlt
- Projektleiter (PL) → Primär Manager, weniger Fachexperte
- Projektmitarbeiter (Projektteam) → Fachexperten, hohe Methoden und Sozialkompetenz

4.1.8 Projektorganisationsformen

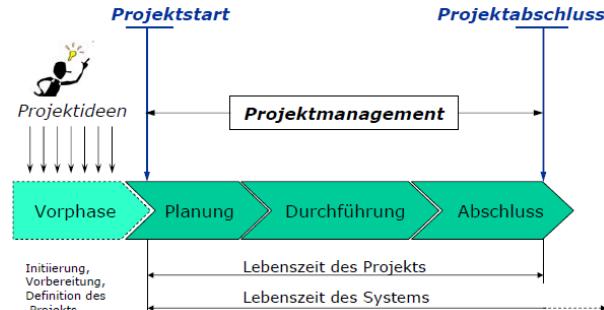
- | | | |
|--|---|---|
| <ul style="list-style-type: none"> • Reine Projektorganisation <ul style="list-style-type: none"> – Projektleiter & Projektteam arbeiten Vollzeit am Projekt – Mitarbeiter sind vollkommen dem Projektleiter unterstellt – eher selten | <ul style="list-style-type: none"> • Matrix-Projektorganisation <ul style="list-style-type: none"> – Mitarbeiter aus spezif. Bereichen arbeiten nur zeitweise am Projekt mit – Verantwortung ist aufgeteilt zwischen Projektleiter/Linieneinheiten – Häufigste Form | <ul style="list-style-type: none"> • Stabs-Projektorganisation <ul style="list-style-type: none"> – Hierarchie ist unverändert – ergänzt durch Projektkoordinator hat keine Weisungsbefugnisse – stimmt Zusammenarbeit mit Mitarbeiter ab |
|--|---|---|

4.2 Projektablauf

Jedes Projekt hat ein Anfang und ein Ende (Lebenszyklus). Der **vorzeitige Abbruch** ist ein Unfall, ein sogenanntes **ungeplantes Ende**. Jedes Projekt besteht aus 4 Phasen. Die 4 Phasen überlappen sich und laufen parallel ab. Während der Durchführung ist auch die Planung anzupassen.

Projektphasen:

1. Projektdefinition
(Vorphase, Projektvorbereitung)
2. Projektplanung
3. Projektdurchführung
4. Projektabschluss



4.3 Projektphasen

Verschiedene Teilprozesse können zeitlich parallel ablaufen.

4.3.1 Projektdefinition (Vorphase, Initiierung, Projektdefinition)

- Vorbereiten des Projektes
- evtl. Machbarkeitsstudie durchführen
- Ziele, Termine festlegen (Das Wichtigste eines Projektes)
- Grobe Aufwands- und Kostenschätzung, Projektorganisation festlegen
- Wird in formellen Projektauftrag festgehalten

Projektauftrag

Der Projektauftrag ist ein schriftliches Dokument wo Ziele und Rahmenbedingungen festgehalten sind. Das Dokument wird vom Auftraggeber unterzeichnet, damit ist Existenz des Projektes formell bestätigt.

Die Ziele des Projektes sollten gemäss **S.M.A.R.T.** formuliert sein.

Häufige Bestandteile:

- | | |
|-------------------------|---------------------|
| • Projektbezeichnung | • Projektergebnisse |
| • Auftraggeber | • Projektbudget |
| • Projektbeginn / -ende | • Projektleiter |
| • Kurzbeschreibung | • Terminvorgaben |

4.3.2 Projektplanung

Als Voraussetzung zur Projektplanung gilt der erstellte Projektauftrag. Anschliessend findet das "**Kick-Off-Meeting**" statt. Das "**Kick-Off-Meeting**" gilt als **offizieller Projektstart**, → **Hauptziel**: Alle sind arbeitsfähig!

Die Projektplanung ist das Wichtigste. Die Pläne werden periodisch überarbeitet da niemals alles nach Plan läuft.

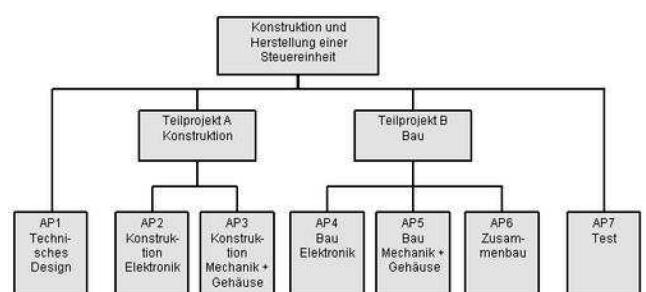
→ **Wichtig: Meilensteine** setzen

Grundsätze der Zielformulierung – SMART:

- S** "Specific" – spezifisch
 - konkret und präzise formuliert, ohne Spielraum für Interpretationen.
- M** "Measurable" – messbar
 - In absoluten oder relativen Werten quantifizierbar. (Es kommt eine Zahl samt zugehöriger Maßeinheit vor.)
- A** "Acceptable" – akzeptierbar, erreichbar
 - Von den Teammitgliedern akzeptierbar, da Ziel erreichbar.
- R** "Realistic, relevant" – realistisch
 - Die Umsetzung des Ziels ist realistisch, gemessen an den zur Verfügung stehenden Ressourcen.
- T** "Time-bound" – terminiert
 - Es existiert ein Endtermin für das Ziel.
- !** Ein Ziel ist nur dann S.M.A.R.T., wenn es alle diese fünf Bedingungen erfüllt.

1. Projektstrukturplan (PSP)

- Projekt in Teilprojekte zerlegen
- Kleinstmögliche Aufgaben sind Arbeitspakete
- Arbeitspakete sind **phasenorientiert** (nach Zeitabschnitten) oder **objektorientiert** (nach Komponenten) oder **funktionsorientiert** (nach Ähnlichkeit der Aufgaben) aufgeteilt
- Pro Arbeitspaket wird eine Arbeitsbeschreibung erstellt



APx = Arbeitspaket x; Teilprojekte A/B = Teilaufgaben A/B

Arbeitspakete

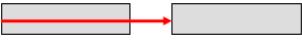
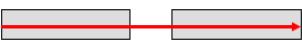
- Für jedes Arbeitspaket wird eine **Arbeitspaketbeschreibung** erstellt.
- Dabei wird ein **Verantwortlicher** bestimmt und der **Aufwand** für das AP geschätzt (→ Budget, → Ressourcen).
- Die Termine für das AP (Anfangs- und Endtermin) werden später festgelegt.

Arbeitspaketbeschreibung	
Projekt: Steuergerät für Hebelelektronik GmbH	
Arbeitspaket: AP 2: Konstruktion Elektronik	
Projektleiter: Herr Schneider	
Arbeitspaketverantwortlicher: Herr Maihoff	
Tätigkeiten:	
- Schaltungsentwurf gemäß Pflichtenheft	
- Versuchsaufbau und Tests	
- Optimierung Bauteileauswahl	
- Stückliste und Dokumentation erstellen	
Zu erarbeitende Ergebnisse: Erstellung einer funktionsfähigen Elektronik als Versuchsanordnung gemäß Spezifikation des Pflichtenhefts mit Unterlagen	
Budget: 4 MA-Tage*, 600 € für Material	
Termine:	
* MA-Tage = Mitarbeiter Tage oder „Mannstage“ dienen als Einheit für den Arbeitsaufwand. 4 MA-Tage bedeutet: 1 Mitarbeiter braucht 4 Tage oder 2 Mitarbeiter brauchen 2 Tage usw.	

2. Projektablaufplan (PAP)

Stellt die **logische Abhängigkeiten** der Arbeitspakete dar. Dazu wird eine Vorgangsliste erstellt, welche die Abhängigkeiten zwischen den Arbeitspaketen aufzeigt.

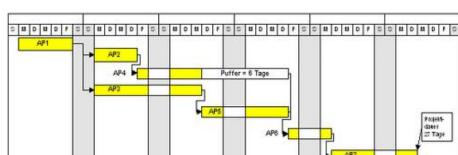
Ap.-NR.	Beschreibung	Vorgänger	Nachfolger
1	Technisches Design	0	2,3
2	Konstruktion Elektronik	1	4
3	Konstruktion Mechanik und Gehäuse	1	5
4	Bau Elektronik	2	6
5	Bau Mechanik und Gehäuse	3	6
6	Zusammenbau der Komponenten	4,5	7
7	Test	6	-

Abhängigkeit	Funktion	Bild
Normalfolge	Nachfolger kann erst beginnen, wenn Vorgänger beendet ist	
Anfangsfolge	Nachfolger kann erst beginnen, wenn Vorgänger begonnen hat	
Endfolge	Nachfolger kann erst enden, wenn Vorgänger beendet ist	
Sprungfolge	Nachfolger kann erst enden wenn Vorgänger begonnen hat	

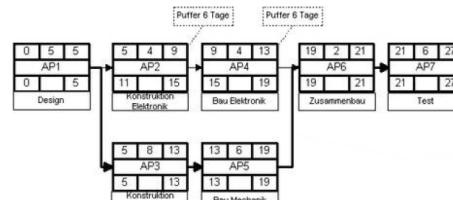
3. Projektterminplan

- Zuordnung von Terminen zu den Arbeitspaketen
- Festlegen der **Meilensteine/Milestones** (= wichtige Termine)
 - Meilensteine sind Etappenziele welche zur Projektkontrolle dienen
 - Meilensteine sind so formuliert, dass "Erfüllt" oder "Nicht Erfüllt" gilt
 - Meilensteine stehen am Ende jeder Projektphase
- Projektterminplan wird aufgrund des PSP und PAP erstellt

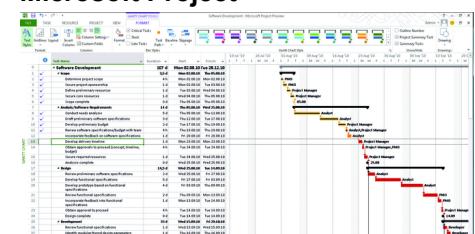
Balkendiagramm



Netzplan



Microsoft Project



3.1 Microsoft Project

In MS Project wird immer von Task (Vorgang) gesprochen.

MS Project hat als Grundlage das Balkendiagramm und die Netzplantechnik.

Wichtigster Zusammenhang: **Work = Duration · Units**

- **Einzelvorgang (Sub Task)** → Arbeitspaket
- **Sammelvorgang (Summary Task)**
→ mehrere Vorgänge
- **Vorgang mit Dauer 0** → Meilenstein
- **Fixed Units** → Ressourcen = konstant
- **Fixed Work** → Arbeit = konstant
- **Fixed Duration** → Dauer = konstant

Work	Arbeitsumfang	Einheit: Anzahl Personentage, Mannstunden
Duration	Zeitdauer	Einheit: Anzahl Stunden, Tage
Units	Intensität	Einheit: Angabe der Beteiligung in Prozent

4. Ressourcen- und Kapazitätsplan

Die benötigten Ressourcen ermitteln und mit den Kapazitäten abstimmen.

5. Kosten- und Budgetplan

Die Kosten für die Ressourcen schätzen und Budget erstellen.

4.3.3 Projektdurchführung

Die eigentliche Arbeit beginnt. Der Projektleiter hat die Aufgabe des Projekt-Controlling (to control → steuern).

[Projekt-Controlling = Projektkontrolle + Projektsteuerung](#)

Projektkontrolle	Rechtzeitiges Feststellen von Abweichungen gegenüber dem geplanten Soll-Ist-Vergleich
Projektsteuerung	Massnahmen um Projekt bei Abweichungen wieder auf Zielkurs zu bringen. Soll-Werte anpassen

4.3.4 Projektabschluss

- Eigentliche Projektarbeit wurde erfolgreich abgeschlossen.
 - Ziele sind erreicht
- Formaler Abschluss des Projekts mit dem Auftraggeber:
 - Projektpräsentation
 - Projektabnahme, Übergabe, Entlastung des PL.
 - Allfällige Nacharbeiten
- Projektteam interner Abschluss
 - Abschlussmeeting: Manöverkritik, "Lessons learned"
 - Abschlussfeier

5 Versionskontrolle

5.1 Versionskontrollsysteme (VCS)

5.1.1 Zweck

- Aufbewahrung, Verwaltung, Wiederherstellen von früheren Versionen in einem Archiv
- Koordination des Zugriffs, bekannt als File-Sharing
- erlaubt hervorholen von alten Versionen

5.1.2 Haupteinsatz

- Softwareentwicklung: zur Verwaltung des Source-Codes
- Dokument-Management-Systeme (DMS)
- Archiv = Repository = Lager

5.1.3 Hauptaufgaben von VCS

- **Protokollierungen** der Änderungen:
→ Es kann jederzeit nachvollzogen werden, wer wann was geändert hat.
- **Wiederherstellung** von alten Ständen (Milestones) einzelner Dateien:
→ Versehentliche Änderungen können jederzeit rückgängig gemacht werden.
- **Koordinierung** des gemeinsamen Zugriffs auf die Dateien eines Projektes.
→ Mehrere Mitarbeiter können gefahrlos auf die Dateien eines Projektes zugreifen.
- **Mehrere Entwicklungszweige** (engl. "Branches")
→ Es kann gleichzeitig an verschiedenen Entwicklungszweigen gearbeitet werden.
→ Ein "Branch" ist eine Abspaltung von der normalen Entwicklungslinie, bei der z.B. eine alternative Lösung für ein Teilproblem untersucht wird.

5.1.4 Einteilung von VCS

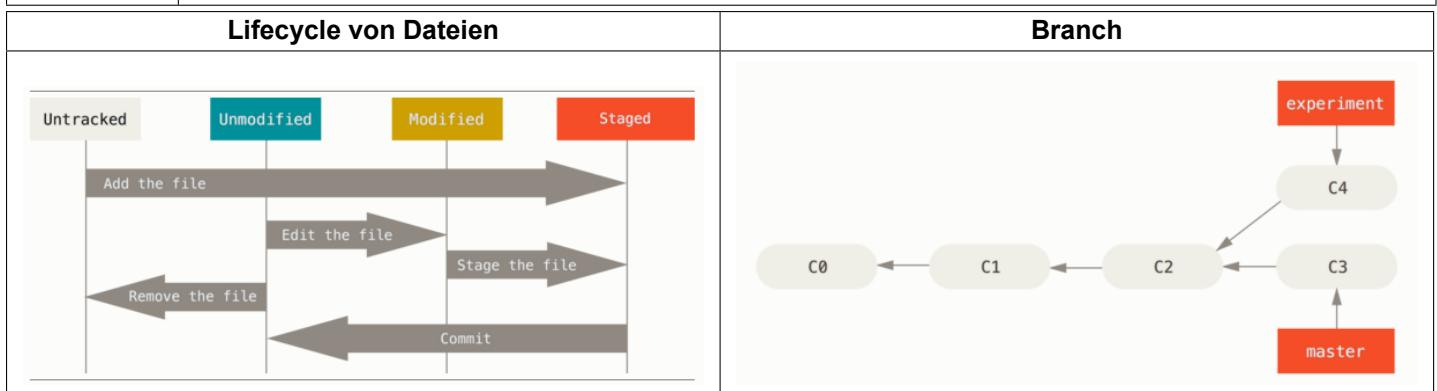
- **Lokale (dateibasierte) Versions-Kontrollsysteme**
 - "Local Version Control System" (LVCS)
 - Dateibasiert = **ein Archiv für jede** zu archivierende Datei
 - Der Name der Archiv-Datei wird dabei vom Namen des Source-Files abgeleitet.
→ hallo.cpp → hallo.cpp,v (Suffix ',v')
 - Die im Archiv "abc.xyz,v" abgespeicherte Versionen der Datei "abc.xyz" werden "Revisionen" genannt.
 - Wichtige Operationen:
 - * **"check-in"**: Datei zu Archiv hinzufügen (neue Revision)
→ \$ ci -m "first check-in" hallo.cpp # "einchecken"
 - * **"checkout"**: eine bestimmte Revision aus dem Archiv holen
→ \$ co -l hallo.cpp # "auschecken" und "locken"
- **Zentralisierte Versions-Kontrollsysteme**
 - "Centralized Version Control Systems" (CVCS)
 - **Ein** zentrales Archiv für alle zu archivierende Dateien.
- **Verteilte Versions-Kontrollsysteme**
 - "Distributed Version Control Systems" (DVCS)
 - **Mehrere, verteilte, gleichberechtigte** Archive für die zu archivierenden Dateien
 - **Git**, gehört in diese Sparte!

5.2 Git

- Git → (engl. Blödmann)
- Entwickelt von Linus Torvalds, Linux-Erfinder
- Ziele waren Geschwindigkeit, Einfachheit, Unterstützung von nichtlinearer Entwicklung, vollständig verteilt (distributed Repos), Verwaltung von grossen Projekten
- Ist ein spezielles Filesystem
- Git ist ein DVCS

5.2.1 Allgemeines

Repository	Ist ein Archiv für ein Projekt und enthält alle Änderungen und Versionen
Branch	Beschreibt zusammenhängende Änderungen in einem Projekt. Es gibt Minimum einen bis beliebig viele. Der Master-Branch ist der Produktivzweig
Commit	Beschreibt eine Änderung in einem Branch an einer Datei mit Änderungsinformation
Snapshot	Momentanes Zeitabbild des Projektes
Merge	Zusammenführen von Änderungen aus zwei Branches
Stagen	Datei zum Index hinzufügen

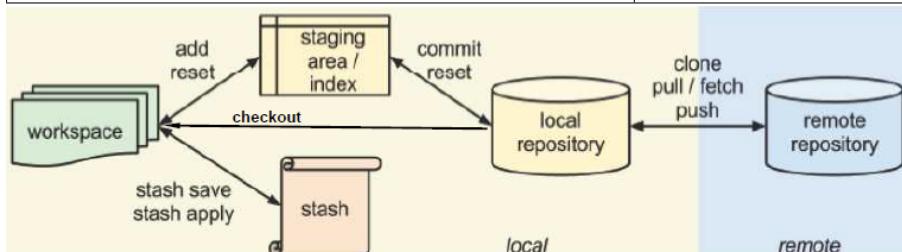


5.2.2 Git-Konzepte

- Git-Workspace ist ein Verzeichnis welches die zu bearbeitenden Dateien eines Projektes enthält
- In diesem Verzeichnis ist ein Unterverzeichnis ".git"
- Dieses Unterverzeichnis ".git" bildet das lokale Repository
- Im Branch zeigen die Pointer immer auf den vorhergehenden Commit

5.2.3 Datenbereiche

Workspace -Projektverzeichniss -enthält die Dateien mit denen man arbeitet	Repository -enthält die Versionsgeschichte in Form von Commits (Revision) -lokal/remote
Stash -Lager -zum temporären Speichern der Workspace-Daten	Staging Area/Index/Cache -sammelt Änderungen für Commits -Bereitstellungsraum



5.2.4 Git-Repository

- **lokales Repository** → ".git"-Verzeichniss im Workspace
- **remote Repository** → Ein normales Git-Repository ohne Projektverzeichnis an externem Ort
- **"bare" Repository** → Git-Repository ohne Projektverzeichnis zum teilen, Klon von lokalen Repositories

5.2.5 Git-Hashwerte

- Zur Identifizierung von Commits (und anderen Git-Objekten) werden sogenannte Hash-Werte verwendet
- Diese werden mittels SHA1-Algorithmus berechnet.
→ SHA-1 = "Secure Hash Algorithmus Version 1"
- Dieser liefert 160 Bits (40 Hex Ziffern), **eindeutiger Prüfwert** für beliebige Daten
- die Hash-Werte werden als Zweigerwerte (Adressen) verwendet um die Repository-Datenstruktur aufzubauen
- Der Hashwert wird aus dem Dateiinhalt berechnet und für den Dateinamen verwendet

5.2.6 Ein existierendes Verzeichnis als Repository initialisieren

Beispiel:

```
$ cd /_projekte/_projektHallo      # "projektHallo" ist leer
$ git init                         # erzeugt leeres Git Repository
Initialized empty Git repository in ...
$ tree -a -F
.
└── .git/
    ├── config
    ├── description
    ├── HEAD          # ist Referenz auf aktuellen Branch
    ├── hooks/
    │   └── # einige Samples
    ├── info
    │   └── exclude
    ├── objects/
    │   ├── info/
    │   └── pack/
    └── refs/
        ├── heads/      # fuer "Zeiger" (Branches)
        └── tags/        # fuer "Zeiger" (Tags)
9 directories, 12 files
$
```

5.2.7 Git-Objekte

- Git-Objekte sind einfache Dateien im Verzeichnis ".git/objects/"
- Dabei werden **die ersten zwei Hex-Ziffern als Namen für ein Unterverzeichnis** verwendet, die restlichen 38 für den Namen der Datei in diesem Unterverzeichnis.

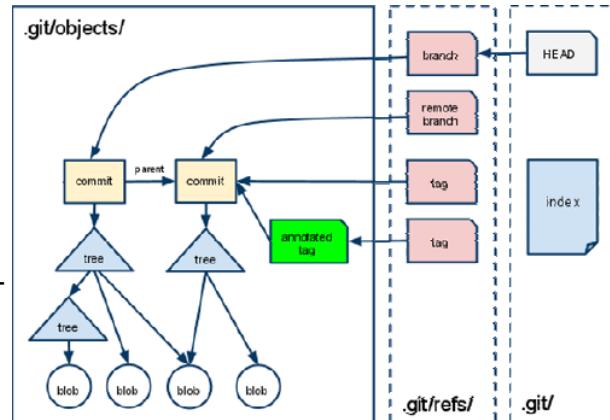
► Beispiel:

```
$ mkdir gruesse; cd gruesse;
$ echo 'Hello, world!' > greeting.txt
$ git hash-object greeting.txt
af5626b4a114abcb82d63db7c8082c3c4756e51b
$ git init                  # lokales Git Repository erzeugen
$ git add greeting.txt     # "stagen" (= Index erstellen!)
$ tree -a -F .git/objects
.git/objects
|-- af/
|   |-- 5626b4a114abcb82d63db7c8082c3c4756e51b
|-- info/
`-- pack/
```

<p>blob ("binary large Object")</p> <ul style="list-style-type: none"> -Inhalt einer Datei -enthält <i>keine</i> Zeiger auf andere Git-Objekte <p>5b1d3..</p> <table border="1" data-bbox="123 316 398 541"> <thead> <tr> <th>blob</th><th>size</th></tr> </thead> <tbody> <tr> <td>#ifndef REVISION_H #define REVISION_H #include "parse-options.h" #define SEEN (1u<<0) #define UNINTERESTING (1u #define TREESAME (1u<<2)</td><td></td></tr> </tbody> </table>	blob	size	#ifndef REVISION_H #define REVISION_H #include "parse-options.h" #define SEEN (1u<<0) #define UNINTERESTING (1u #define TREESAME (1u<<2)		<p>tree</p> <ul style="list-style-type: none"> -enthält Zeiger auf blob-, oder tree-objekte -enthält Referenzen über Git-Objekte, Ordner- & Dateinamen <p>c36d4..</p> <table border="1" data-bbox="847 316 1138 574"> <thead> <tr> <th>tree</th><th>size</th></tr> </thead> <tbody> <tr> <td>blob 5b1d3</td><td>README</td></tr> <tr> <td>tree 03e78</td><td>lib</td></tr> <tr> <td>tree cdc8b</td><td>test</td></tr> <tr> <td>blob cba0a</td><td>test.rb</td></tr> <tr> <td>blob 911e7</td><td>xdiff</td></tr> </tbody> </table>	tree	size	blob 5b1d3	README	tree 03e78	lib	tree cdc8b	test	blob cba0a	test.rb	blob 911e7	xdiff						
blob	size																						
#ifndef REVISION_H #define REVISION_H #include "parse-options.h" #define SEEN (1u<<0) #define UNINTERESTING (1u #define TREESAME (1u<<2)																							
tree	size																						
blob 5b1d3	README																						
tree 03e78	lib																						
tree cdc8b	test																						
blob cba0a	test.rb																						
blob 911e7	xdiff																						
<p>commit</p> <ul style="list-style-type: none"> -Startobjekt für einen Commit -enthält Zeiger auf tree und vorgängiges Commit-Objekt <p>ae668..</p> <table border="1" data-bbox="123 743 398 1046"> <thead> <tr> <th>commit</th><th>size</th></tr> </thead> <tbody> <tr> <td>tree c4ec5</td><td></td></tr> <tr> <td>parent a149e</td><td></td></tr> <tr> <td>author Scott</td><td></td></tr> <tr> <td>committer Scott</td><td></td></tr> <tr> <td>my commit message goes here and it is really, really cool</td><td></td></tr> </tbody> </table>	commit	size	tree c4ec5		parent a149e		author Scott		committer Scott		my commit message goes here and it is really, really cool		<p>tag</p> <ul style="list-style-type: none"> -Etikette mit Kommentar -enthält Zeiger auf ein Commit-Objekt <p>49e11..</p> <table border="1" data-bbox="847 743 1138 1001"> <thead> <tr> <th>tag</th><th>size</th></tr> </thead> <tbody> <tr> <td>object ae668</td><td></td></tr> <tr> <td>type commit</td><td></td></tr> <tr> <td>tagger Scott</td><td></td></tr> <tr> <td>my tag message that explains this tag</td><td></td></tr> </tbody> </table>	tag	size	object ae668		type commit		tagger Scott		my tag message that explains this tag	
commit	size																						
tree c4ec5																							
parent a149e																							
author Scott																							
committer Scott																							
my commit message goes here and it is really, really cool																							
tag	size																						
object ae668																							
type commit																							
tagger Scott																							
my tag message that explains this tag																							

5.2.8 Git-Referenzen

- Referenzen = Zeiger (nur Zeigerwerte)
- Zeigen nur auf Git-Objekte
- Hashwert-Zeiger → zeigt auf Branches/Tags
- Symbolischer-Zeiger → zeigt auf Datei mit Namen eines anderen Zeigers



5.2.9 Bashprints

Files changed but not staged.

\$ git status

On branch master

Your branch is up-to-date with 'origin/master'.

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout - <file>..." to discard changes in working directory)

modified: idiotenseite/IdiotenseiteInclude.tex

no changes added to commit
(use "git add" and/or "git commit -a")

\$ git log

commit 1b7ff854a491397087ff89689a535c275549e6ee

Author: Michel Gisler <Michel Gisler>

Date: Tue May 31 15:08:58 2016 +0200

Kleiner Ergänzungen und Anpassungen

5.2.10 Git Commands

Create	
git init <directory>	Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
git clone <repo>	Clone repo located at <repo> onto local machine. Original repo can be located on the local filesystem or on a remote machine
Local Changes	
git status	List which files are staged, unstaged, and untracked
git diff	Show unstaged changes between your index and working directory
git diff HEAD	Show difference between working directory and last commit.
git add <directory>	Stage all changes in <directory> for the next commit. Replace <directory> with a <file> to change a specific file or use <.> to Stage all.
git commit -m "<message>"	Commit the staged snapshot, but instead of launching a text editor, use <message> as the commit message.
git commit -a	Commit all local changes in tracked files.
git commit --amend	Change the last commit. Don't amend published commits!
Stash	
git stash	when you want to record the current state of the working directory
git stash list	Lists the modifications stashed away by this command
git stash pop	Restore your stashed files
Commit History	
git log	Display the entire commit history using the default format.
git log -p <file>	Show changes over time for a specific file
git log --author="<pattern>"	Search for commits by a particular author.
git blame <file>	Who changed what and when in <file>
Branches and Tags	
git branch	List all of the branches in your repo. Add a <branch> argument to create a new branch with the name <branch>.
git branch <new-branch>	Create a new branch based on your current HEAD
git branch -d <branch>	Delete a local branch
git branch -dr <remote/branch>	Delete a branch on the remote
git checkout -b <branch>	Create and check out a new branch named <branch>. Drop the -b flag to checkout an existing branch.
git merge <branch>	Merge <branch> into the current branch.
git tag <tag-name>	Mark the current commit with a tag (v1.0.1)
Update and Publish	
git remote -v	List all currently configured remotes.
git remote add <name> <url>	Add new remote repository, named <remote>
git fetch <remote>	Download all changes from <remote>, but don't integrate into HEAD
git pull <remote> <branch>	Download changes and directly merge/integrate into HEAD
git push <remote> <branch>	Publish local changes on a remote
Merge and Rebase	
git merge <branch>	Merge <branch> into your current HEAD
git rebase <branch>	Rebase your current HEAD onto <branch>! Don't rebase published commits!
git rebase --abort	Abort a rebase
git mergetool	Use your configured merge tool to solve conflicts
Undo	
git reset --hard HEAD	Discard all local changes in your working directory.
git checkout HEAD <file>	Discard local changes in a specific file
git revert <commit>	Revert a commit (by producing a new commit with contrary changes)
git reset <commit>	Move the current branch tip backward to <commit>, reset the staging area to match, but leave the working directory alone.

```

$ git status      // nichts wurde geändert
2 # On branch master
nothing to commit (working directory clean)
4 -----
$ git status    // es wurde hello.html verändert
6 # On branch master
# Changes not staged for commit:
8   # (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
10 #
#   modified:   hello.html
12 #
no changes added to commit (use "git add" and/or "git commit -a")
14 -----
$ git add hello.html // hello.html staged und dann git status
16 $ git status
# On branch master
18 # Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
20 #
#   modified:   hello.html
22 #

24 $ git status      // hello.html geändert nach dem staging und danach git status
# On branch master
26 # Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
28 #
#   modified:   hello.html
30 #
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
34 #
#   modified:   hello.html
36 #

38 $ git commit -m "Added standard HTML page tags" // commit und git status
[master 8c32287] Added standard HTML page tags
40 1 files changed, 3 insertions(+), 1 deletions(-)
$ git status
42 # On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
46 #
#   modified:   hello.html
48 #
no changes added to commit (use "git add" and/or "git commit -a")
50 #

$ git log --pretty=oneline
52 fa3c1411aa09441695a9e645d4371e8d749da1dc Added HTML header
53 8c3228730ed03116815a5cc682e8105e7d981928 Added standard HTML page tags
54 43628f779cb333dd30d78186499f93638107f70b Added h1 tag
55 911e8c91caeab8d30ad16d56746cbd6eef72dc4c First Commit
56 #
$ git checkout 911e8c9      // zu einem älteren Commit wechseln
58 Note: checking out '911e8c9'.
59 You are in 'detached HEAD' state. You can look around, make experimental
60 changes and commit them, and you can discard any commits you make in this
61 state without impacting any branches by performing another checkout.
62
64 If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:
66
68   git checkout -b new_branch_name
HEAD is now at 911e8c9... First Commit
70 #
$ git checkout master      // zum aktuellsten commit wechseln
72 Previous HEAD position was 911e8c9... First Commit
Switched to branch 'master'
74 #
git tag v1                // dem aktuellen Commit den Tag v1 geben

```

```

$ git checkout v1^      // wechselt zum vorherigen Commit
2 Note: checking out 'v1^'.
4 You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
6 state without impacting any branches by performing another checkout.
8 If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:
10
11   git checkout -b new_branch_name
12
13 HEAD is now at 8c32287... Added standard HTML page tags
14 -----
15 $ git reset HEAD hello.html          // unstage von hello.html (nach add befehl)
16 Unstaged changes after reset:
17   M hello.html
18 -----
19 $ git revert HEAD --no-edit        // Revert Commit (erstellt ein neuer Commit mit 'Revert' als Prefix)
20 [master 45fa96b] Revert "Oops, we didn't want this commit"
21   1 files changed, 1 insertions(+), 1 deletions(-)
22 $ git hist                         //
23 * 45fa96b 2011-03-09 | Revert "Oops, we didnt want this commit" (HEAD, master) [Alexander Shvets]
24 * 846b90c 2011-03-09 | Oops, we didnt want this commit [Alexander Shvets]
25 * fa3c141 2011-03-09 | Added HTML header (v1) [Alexander Shvets]
26 * 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]
27 * 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
28 * 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
29 -----
30 $ git tag oops                  // Markiert den Revert Commit mit dem Tag oops
31 -----
32 $ git reset --hard v1           // Setzt den Master Branch zum gewuenschten Commit
33 HEAD is now at fa3c141 Added HTML header
34 $ git hist
35 * fa3c141 2011-03-09 | Added HTML header (HEAD, v1, master) [Alexander Shvets]
36 * 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]
37 * 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
38 * 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
39 -----
40 $ git hist --all               // Commits werden aber immer noch in der History angezeigt!
41                               // Wird erst durch Garbage Collector aus History entfernt
42 * 45fa96b 2011-03-09 | Revert "Oops, we didn't want this commit" (oops) [Alexander Shvets]
43 * 846b90c 2011-03-09 | Oops, we didnt want this commit [Alexander Shvets]
44 * fa3c141 2011-03-09 | Added HTML header (HEAD, v1, master) [Alexander Shvets]
45 * 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]
46 * 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
47 * 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
48 -----
49 $ git tag -d oops             // Loescht den Ooops Tag und fuehrt den Garbage Collector aus
50 Deleted tag 'oops' (was 45fa96b)
51 $ git hist --all             // Commits wurden geloescht
52 * fa3c141 2011-03-09 | Added HTML header (HEAD, v1, master) [Alexander Shvets]
53 * 8c32287 2011-03-09 | Added standard HTML page tags (v1-beta) [Alexander Shvets]
54 * 43628f7 2011-03-09 | Added h1 tag [Alexander Shvets]
55 * 911e8c9 2011-03-09 | First Commit [Alexander Shvets]
56 -----
57 $ git add hello.html
58 $ git commit --amend -m "Add an author/email comment" // Ersetzt den letzten Commit
59 [master 6a78635] Add an author/email comment
60   1 files changed, 2 insertions(+), 1 deletions(-)
61 -----
62 $ mkdir lib                  // Ordner erstellen
63 $ git mv hello.html lib     // move hello.html in lib Ordner
64 $ git status
65 # On branch master
66 # Changes to be committed:
67 #   (use "git reset HEAD <file>..." to unstage)
68 #
69 #       renamed:    hello.html -> lib/hello.html
70 #
71 -----
72 $ git init
73 Initialized empty Git repository in /Users/alex/Documents/Presentations/githowto/auto/hello/.git/

```

6 Qt

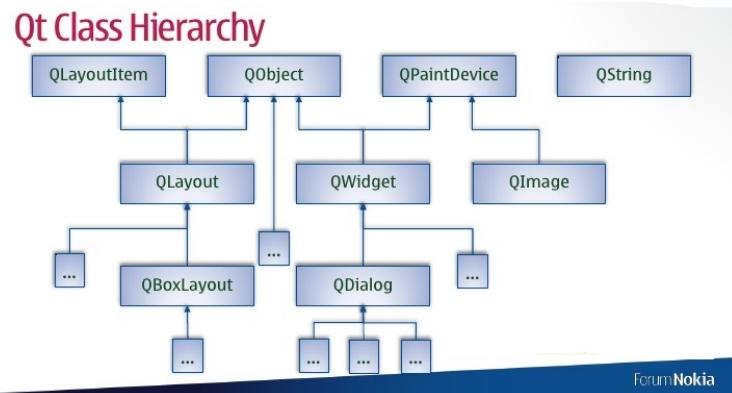
Qt (sprich cute = süß) ist eine plattformübergreifende Programmierungsapplikation für grafische Benutzeroberflächen (GUI = Graphical User Interface).

Qt wurde 1991 gegründet und gehörte der Firma Trolltech. Im Jahr 2008 übernahm Nokia die Firma Trolltech. Im Jahr 2014 wurde Qt als Open-Source-Projekt verwaltet.



6.1 Grundlagen zu Qt

- Qt C++ Klassenbibliothek stellt GUI-Elemente zur Verfügung
 - Das GUI wird als C++ Sourcefile geschrieben
 - Standardklasse QtGui und QtCore werden automatisch inkludiert
 - Alle Qt Widgets werden **auf dem Heap** erstellt
 - Jedes Programm enthält eine Instanz von QApplication
 - QApplication hält alles zusammen



6.1.1 QObject

- QObject ist die Basisklasse des Qt-Modells
 - stellt das Memorymanagement (automatische delete) für alle abgeleiteten Klassen von QObject zur Verfügung. Es entstehen Objektbäume.
 - stellt die connect-Funktion zur Verfügung
 - bearbeitet das Eventhandling
 - hat keine visuelle Representation

6.1.2 Qt Building: qmake

- qmake Makefile-Generator
 - macht aus plattformunabhängigem Projektfile (.pro) ein plattformspezifisches Projektfile.
 - die gewünschte Plattform wird qmake als Option mitgegeben
 - Bei Build-Problemen alle Dateien **ausser** .pro und Sourcefile löschen

6.1.3 Qt Konvention

Was	Beispiel	Konvention
Qt-Modulname	"QtCore"	<i>Qt</i>
Qt-Klassenname	"QString"	<i>Q</i>
Qt-Variablen-,Funktionsname	"qTranslator, qDebug()"	<i>q</i>
Qt-include	"#include <QtGui>"	<i>ohne.h</i>

6.2 QWidget (Widget = Window Gadget ("Fenster Ding"))

- abgeleitet von QObject, eine Instanz der Klasse stellt grafisches Element dar
 - Diese Klasse hat viele Methoden um Aussehen, Grösse, Position etc. zu verändern
 - mit show() wird das Widget angezeigt, mit hide() wird das Widget versteckt
 - Jedes Widget wird einem "*Parent-Widget*" (Eltern) zugewiesen, ein solches Widgets nennt man dann *Child-Widgets*
 - Durch die Parent-Child Funktion entsteht eine Verschachtelung
 - Ein Widget **ohne** Parent ist ein Top-Level-Widget oder Window
 - Das "*Parent-Widget*" übernimmt Verwaltungsfunktionen wie Memory-Management, weiterleiten von Ereignissen an "*Child-Widget*, anzeigen, verstecken von Widgets etc.

6.2.1 Top-Level-Widget

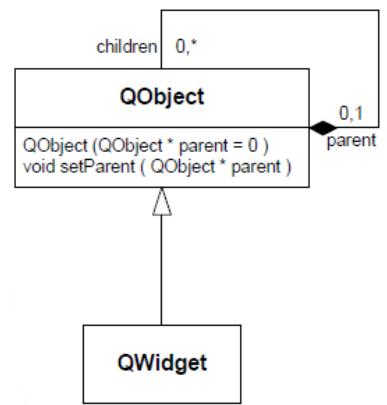
Die "Top-Level-Window" sind Widgets welche auf der obersten Hierarchiestufe liegen. Sie haben keine "Parent-Widget".

Widgets für "Top-Level-Window":

- **QMainWindow** → Applikations-Window (Hauptfenster)
- **QDialog** → Popup-Window für Abfragen ("Wirklich löschen?"), Hauptanwendung bleibt blockiert
- **QWidget** → Einfaches Fenster

6.2.2 Parent-Child-Widgets

- Jedes QObject kann **maximal ein** Eltern-QObject haben
- Jedes QObject kann beliebig viele Kind-QObject haben
- Kind muss Eltern über sich informieren
 - Mit Konstruktor `label->setParent(..)`
 - oder Layout Manager



6.3 GUI-Programmierung

Es gibt zwei grundlegende Aufgaben bei der GUI-Programmierung.

Das **Layout** legt Anordnung, Grösse und Farbe fest.

Die **Interaktion** legt die Reaktion des Programmes auf eine Eingabe fest.

6.4 Layout

Es gibt **drei Varianten** wie Widgets innerhalb eines Widget angeordnet werden können:

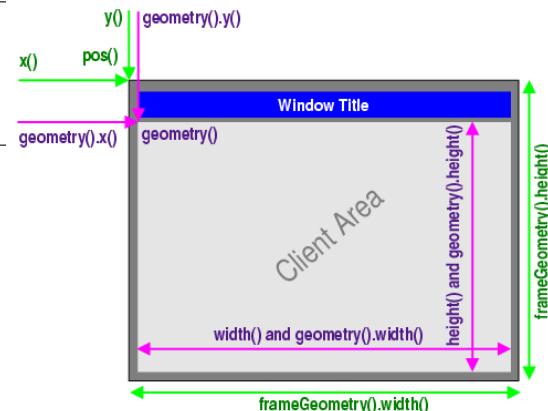
- Absolute Positionierung
- Layout Manager
- GUI-Designer (Qt Designer)

6.4.1 Absolute Positionierung

Mittels Methoden:

```

2 void QWidget::setFixedSize(int width, int height);
3 void QWidget::setGeometry(int x, int y, int w, int h);
4 void QWidget::resize(int w, int h);
5 void QWidget::move(int x, int y);
  
```



6.4.2 Layout Manager

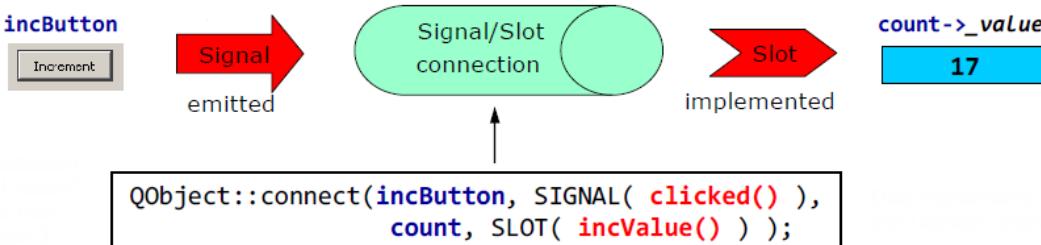
Klasse	Ort
QVBoxLayout	Elemente vertikal
QHBoxLayout	Elemente horizontal
QGridLayout	zweidimensionales Gitter
QFormLayout	Elemente zeilenweise
QStackedLayout	aufeinandergelegt Widgets

6.4.3 GUI-Designer / Qt Designer

Im Qt Designer mittels Drag-Drop können die Elemente angeordnet werden, anschliessend Umwandlung in Code.

6.5 Interaktion

In Qt werden die Eingaben als SIGNALS bezeichnet und die Ausgaben als SLOTS. Mittels der Funktion connect() werden die beiden miteinander verknüpft.



6.5.1 SIGNALS

- nur deklariert, nie definiert
- kein Zugriffsrecht vergeben
- Rückgabetyp immer void

6.5.2 SLOTS

- deklariert und definiert
- oftmals Memberfunktion, auch normal abrufbar
- Zugriffsrechte

6.5.3 connect

- Verbindungsfunction zwischen Input und Output
- ein Signal zu mehreren Slots
- mehrere Signale zu einem Slot

Beispiele connect-Funktion

```

1 //**** Qt4 Notation
2 QObject::connect( buDec, SIGNAL(clicked()), &counter, SLOT(decValue()) );
3 QObject::connect( &counter, SIGNAL(valueChanged(int)), laCount, SLOT(setNum(int)) );
4
5 //**** Funktionspointer ab Qt5
6 connect( button, &QPushButton::clicked, count, &Counter::incValue );
7
8 //**** Globaler Funktion ab Qt5
9 connect( button, &QPushButton::clicked, &printValue );

```

6.6 Zeichnen und Malen

6.6.1 QPainter

Der Maler mit Mal- & Zeichenwerkzeugen.

Klasse	Werkzeug
QPen	Zeichenstift
QBrush	Pinsel
QFont	Schriftart
draw	Formen malen

6.6.2 QPaintDevice

Oberfläche wo darauf gezeichnet & gemalt werden kann QPainter-Objekte zeigen auf QPaintDevice-Objekte. QPaintDevice ist eine abstrakte Basisklasse welche nicht von QObject abgeleitet ist und daher **kein automatisches delete**.

6.6.3 Vorgehen

```

1 //**** Malvorgang initialisieren
2 QPixmap pixmap (160, 160);           // Zeichenoberfläche
3 QPainter painter (&pixmap);           // 'painter' zeichnet auf 'pixmap'
4
5 //**** Zeichnungsgeräte initialisieren
6 QPen pen (Qt::blue, 2);               // Pinsel
7 painter.setPen(pen);
8 QBrush brush (Qt::green);             // Farbe
9 painter.setBrush(brush);
10
11 //**** Zeichenoperation durchführen
12 painter.drawEllipse(10, 10, 140, 140); // Kreis zeichnen

```

7 Beispiele

7.1 CppUnit

```

1 #include <cppunit/extensions/HelperMacros.h>
2 #include "../src/Author.h"
3
4 class AuthorTest: public CppUnit::TestCase // Test-Suite "AuthorTest"
5 {
6 private:
7     Author *author;
8
9     CPPUNIT_TEST_SUITE(AuthorTest);
10    CPPUNIT_TEST_EXCEPTION(testEmptyAuthorException, InvalidAuthor);
11    CPPUNIT_TEST(testCtors);
12    CPPUNIT_TEST(testOperatorEqual1);
13    CPPUNIT_TEST(testOperatorEqual2);
14    CPPUNIT_TEST(testOperatorEqual3);
15    CPPUNIT_TEST(testOperatorEqual4);
16    CPPUNIT_TEST_SUITE_END();
17
18 public:
19     void testCtors()
20     {
21         author = new Author("Samuel", "Beckett");
22         CPPUNIT_ASSERT(author->getFirstName() == "Samuel");
23         CPPUNIT_ASSERT(author->getLastName() == "Beckett");
24         delete author;
25     }
26     void testEmptyAuthorException()
27     {
28         author = new Author("", "");
29         delete author;
30     }
31     void testOperatorEqual1()
32     {
33         Author a1 ("Max", "Huber");
34         Author a2 ("Max", "Huber");
35         CPPUNIT_ASSERT ( a1 == a2 );
36     }
37     void testOperatorEqual2()
38     {
39         Author a1 ("Max", "Huber");
40         Author a2 ("Max", "Meier");
41         CPPUNIT_ASSERT ( ! (a1 == a2) );
42     }
43     void testOperatorEqual3()
44     {
45         Author a1 ("Max", "Huber");
46         Author a2 ("Fritz", "Huber");
47         CPPUNIT_ASSERT ( ! (a1 == a2) );
48     }
49     void testOperatorEqual4()
50     {
51         Author a1 ("Max", "Huber");
52         Author a2 ("Fritz", "Meier");
53         CPPUNIT_ASSERT ( ! (a1 == a2) );
54     }
55 }; // Alle Tests von 'TestAuthor' in der default-registry registrieren:
56 CPPUNIT_TEST_SUITE_REGISTRATION(AuthorTest);

```

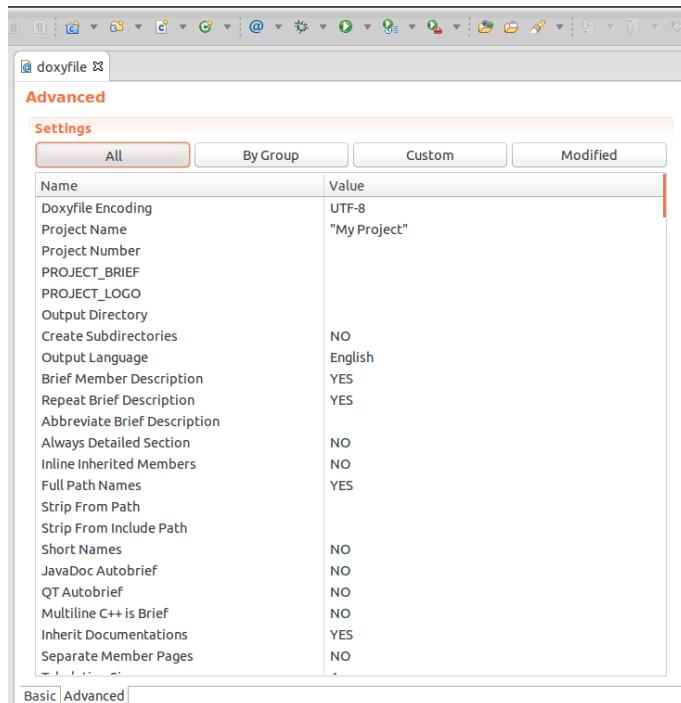
```

1 #include <cppunit/extensions/HelperMacros.h>
2 #include "../src/datum.h" // enthaelt die Deklaration
3 //    int anzTageSeit1900(int tag, int monat, int jahr);
4 //-----
5 class TestDatum: public CppUnit::TestCase
6 {
7 private:
8     CPPUNIT_TEST_SUITE(TestDatum);
9     CPPUNIT_TEST(testAnzTageSince1900);
10    CPPUNIT_TEST_SUITE_END();
11 public:
12     void testAnzTageSince1900()
13     {
14         CPPUNIT_ASSERT(anzTageSeit1900(31, 12, 1899) == 0);
15         CPPUNIT_ASSERT(anzTageSeit1900(30, 12, 1899) == 0);
16
17         CPPUNIT_ASSERT(anzTageSeit1900(1, 1, 1900) == 1);
18         CPPUNIT_ASSERT(anzTageSeit1900(31, 12, 1900) == 365);
19
20         CPPUNIT_ASSERT(anzTageSeit1900(1, 1, 1901) == 1+365);
21         CPPUNIT_ASSERT(anzTageSeit1900(31, 12, 1901) == 365+365);
22     }
23 };
24 // Alle Tests von 'TestDatum' in der default-registry registrieren:
25 CPPUNIT_TEST_SUITE_REGISTRATION(TestDatum);

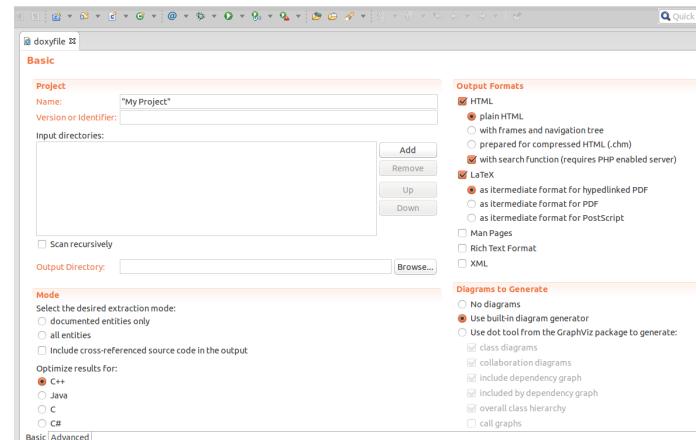
```

7.2 Doxygen

Eclipse Einstellungen



Eclipse-Plugin



```

1  /**
2   * @file TuerschlossNFC.ino
3   * @author Luca Mazzoleni
4   * @brief Doorlock with Keypad and NFC
5   *
6   * Board: Arduino Leonardo \n
7   * Library: \link{https://blabla.com/endlink} \n
8   */
9
10 /* Includes */
11 #include <Adafruit_PN532.h>
12
13 //set MACROS
14 /**Keypad Nr. 0 Pin 2*/
15 #define button0 (2)
16 //....
17
18 /**Check Adafruit library for more Information
19  * @param PN532_SCK Serial Clock
20  * @param PN532_MISO Master Input, Slave Output
21  * @param PN532_MOSI Master Output, Slave Input
22  * @param PN532_SS Slave Select
23  */
24 Adafruit_PN532 nfc(PN532_SCK,
25                     PN532_MISO,
26                     PN532_MOSI,
27                     PN532_SS);
28
29 //set Global Variable
30 /**Initialize a Array for the Inputs*/
31 char inputCode[maxIN];
32 //=====
33 void setup()
34 {
35     //....do setup Code...
36 }
37
38 void loop()
39 {
40     void checkCode(int p);
41     //.....do LOOP Code...
42 }
43
44 void checkCode(int p)
45 /** @brief Check if Code is correct
46  *
47  * Check if input code is the same as the secret code
48  * @param p Count of pressed Buttons
49  * @return Void
50  */
51 {
52     //....do checkCode...
53 }

```

Doorlock with Keypad and NFC. More...

#include <Adafruit_PN532.h>

Macros

#define button0 (2)

Functions

Adafruit_PN532 nfc(PN532_SCK, PN532_MISO, PN532_MOSI, PN532_SS)

void setup()

void loop()

void checkCode(int p)

Check if Code is correct. More...

Variables

char inputCode[maxIN]

Detailed Description

Doorlock with Keypad and NFC.

Author

Luca Mazzoleni Board: Arduino Leonardo

Library: <https://blabla.com>

Macro Definition Documentation

#define button0 (2)

Keypad Nr. 0 Pin 2

Function Documentation

void checkCode(int p)

Check if Code is correct.

Check if input code is the same as the secret code

Parameters

p Count of pressed Buttons

Returns

Void

void loop()

Adafruit_PN532 nfc(PN532_SCK ,

PN532_MISO ,

PN532_MOSI ,

PN532_SS)

Check Adafruit library for more Information

Parameters

PN532_SCK Serial Clock

PN532_MISO Master Input, Slave Output

PN532_MOSI Master Output, Slave Input

PN532_SS Slave Select

void setup()

Variable Documentation

char inputCode[maxIN]

Initialize a Array for the Inputs

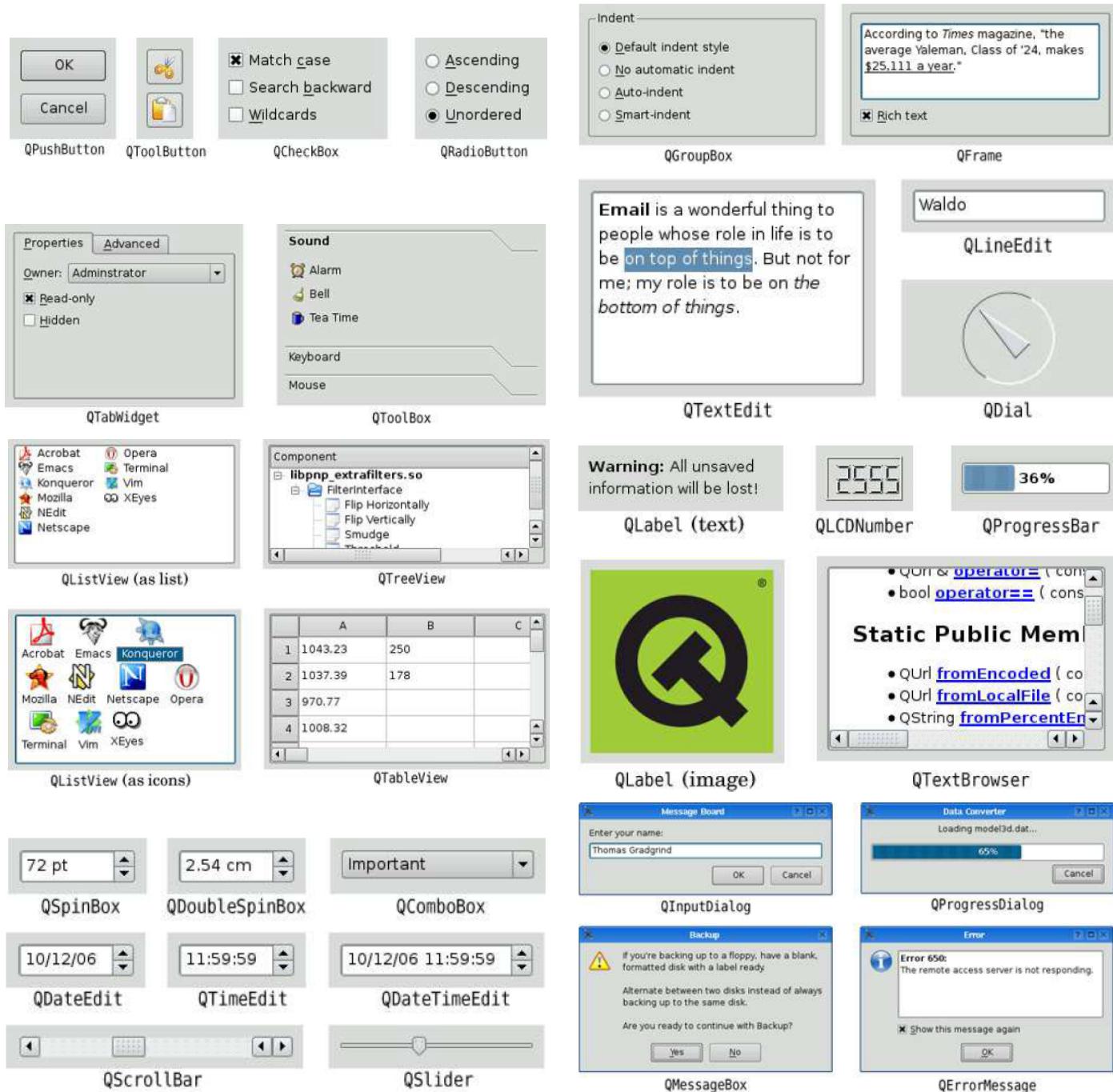
7.3 Qt

7.3.1 HTML-Tags

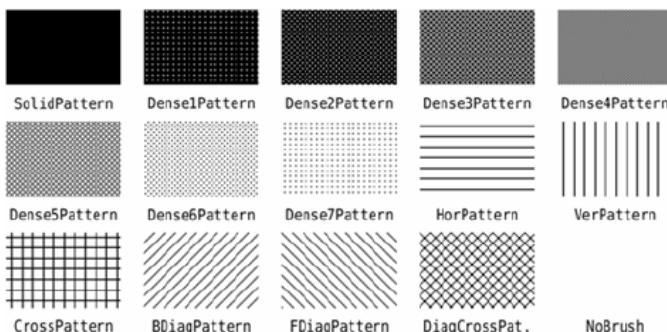
```
<b>Hello</b>
<i>Qt</i>
<p>Dies ist ein <br> Zeilenumbruch </p>
```

Tag	Description	Comment
a	Anchor or link	
address	Address	
b	Bold	Fett
big	Larger font	
blockquote	Indented paragraph	
body	Document body	
br	Line break	
center	Centered paragraph	
cite	Inline citation	Same as i
code	Code	Same as tt
dd	Definition data	
dfn	Definition	
div	Document divison	
em	Emphasized	Same as i
font	Font size, family, color	Support size, face, color (Qt::green) attributes
h1	Level 1 heading	Titel 1. Ordnung
h2	Level 2 heading	Titel 2. Ordnung
h3	Level 3 heading	Titel 3. Ordnung
h4	Level 4 heading	Titel 4. Ordnung
h5	Level 5 heading	Titel 5. Ordnung
h6	Level 6 heading	Titel 6. Ordnung
head	Document header	
hr	Horizontal line	
html	HTML document	
i	italic	<i>kursiv</i>
img	image	Support src, source, width, height attributes
kbd	User entered text	
meta	Meta-information	Text in Metacode in HTML
li	List-Item	Jeder einzelne Punkt muss umschlossen werden.
nobr	Non-breakable text	
ol	Ordered list	
p	paragraph	
pre	preformated text	
qt	Qt richt-text document	Synonym für HTML
s	Strikethrough	
samp	Sample Code	
small	Small font	kleine Schrift
strong	Strong	same as b
style	Style sheet	Allows styling information to be included with rich text
title	Document row	
u	underlined	
var	Variable	Same as i

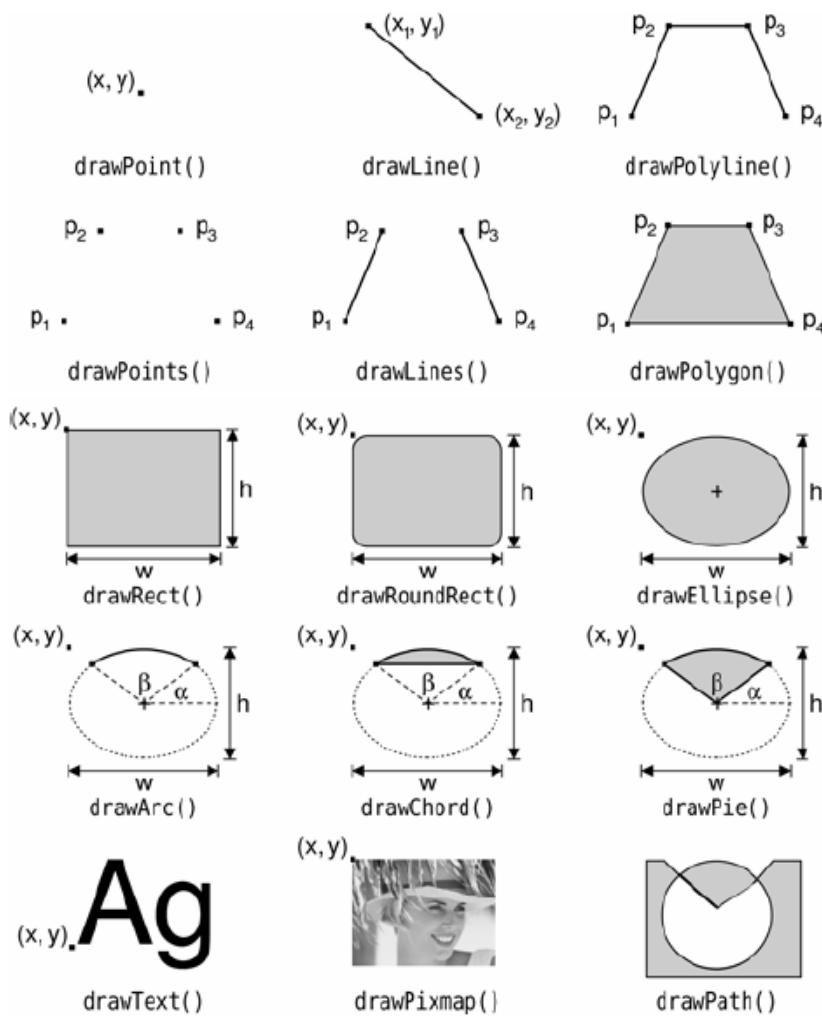
7.3.2 QWidgets



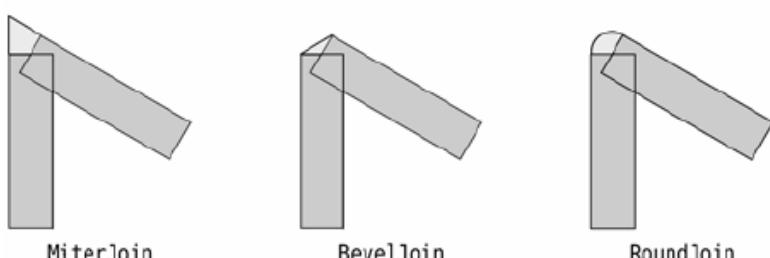
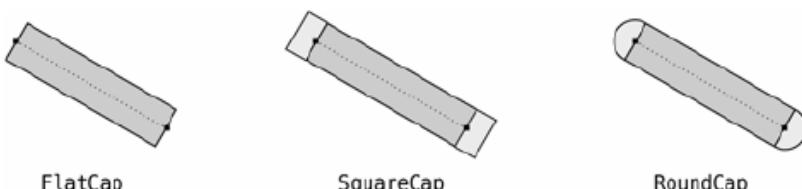
7.3.3 QBrush



7.3.4 QPainter draw-Methoden



7.3.5 QPen



	Line width			
	1	2	3	4
SolidLine	—	—	—	—
DashLine	- - -	- - -	- - -	- - -
DotLine
DashDotLine	- - . - -	- - . - -	- - . - -	- - . - -
DashDotDotLine	- - . - - . - -	- - . - - . - -	- - . - - . - -	- - . - - . - -
NoPen				

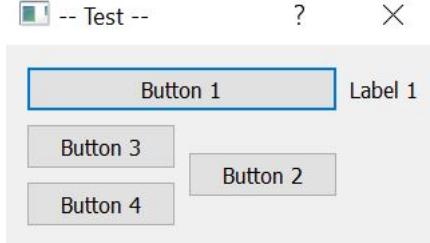
7.3.6 SpezUeb

GridLayout

```

1 #include <QtWidgets>
2
3 int main(int argc, char * argv[])
4 {
5     QApplication app(argc, argv);
6
7     QDialog mainWindow;
8     mainWindow.setWindowTitle(" -- Test -- ");
9     QGridLayout * layout = new QGridLayout();
10    mainWindow.setLayout(layout);
11
12    QPushButton * bu1 = new QPushButton("Button 1");
13    layout->addWidget(bu1, 0, 0, 1, 2);
14    QLabel * la1 = new QLabel("Label 1");
15    la1->setAlignment(Qt::AlignCenter);
16    layout->addWidget(la1, 0, 2);
17    QPushButton * bu2 = new QPushButton("Button 2");
18    layout->addWidget(bu2, 1, 1);
19
20    QVBoxLayout * lout2 = new QVBoxLayout;
21    layout->addLayout(lout2, 1, 0);
22    QPushButton * bu3 = new QPushButton("Button 3");
23    lout2->addWidget(bu3);
24    QPushButton * bu4 = new QPushButton("Button 4");
25    lout2->addWidget(bu4);
26
27    mainWindow.show();
28    return app.exec();
}

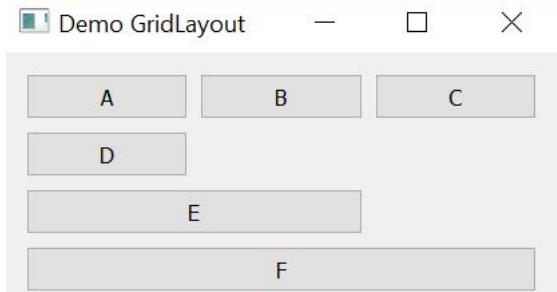
```



```

1 // Projekt: Qt5 Demo GridLayout
2 #include <QtWidgets> // fuer Qt5
3 int main(int argc, char *argv[])
4 {
5     // **** Qt-Prolog:
6     QApplication app(argc, argv);
7     // Main-widget:
8     QWidget *w = new QWidget();
9     // Grid layout:
10    QGridLayout *gridLayout = new QGridLayout;
11    // Set the grid layout as the main layout:
12    w->setLayout(gridLayout);
13    // Buttons:
14    QPushButton *b1 = new QPushButton("A");
15    QPushButton *b2 = new QPushButton("B");
16    QPushButton *b3 = new QPushButton("C");
17    QPushButton *b4 = new QPushButton("D");
18    QPushButton *b5 = new QPushButton("E");
19    QPushButton *b6 = new QPushButton("F");
20    // Einen Button zu 'gridlayout' hinzufuegen mit:
21    // 'addWidget(QWidget*, int row, int column)' oder
22    // 'addWidget(QWidget*, int row, int column, int rowspan, int
23    // colspan)'
24    // Zeile 0:
25    gridLayout->addWidget(b1, 0, 0); // Zeile 0, Spalte 0
26    gridLayout->addWidget(b2, 0, 1); // Zeile 0, Spalte 1
27    gridLayout->addWidget(b3, 0, 2); // Zeile 0, Spalte 2
28    // Zeile 1:
29    gridLayout->addWidget(b4, 1, 0); // Zeile 1, Spalte 0
30    // Zeile 2:
31    gridLayout->addWidget(b5, 2, 0, 1, 2); // Zeile 2; 2 Spalten, ab
32    // Spalte 0
33    // Zeile 3:
34    gridLayout->addWidget(b6, 3, 0, 1, 3); // Zeile 3; 3 Spalten, ab
35    // Spalte 0
36    w->setWindowTitle("Demo GridLayout");
37    w->show();
38    return app.exec();
}

```

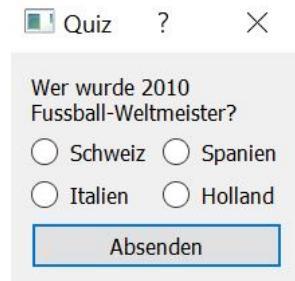


Quiz

```

1 #include <QtWidgets>
2
3 int main(int argc, char * argv[])
4 {
5     QApplication app(argc, argv);
6
7     QDialog mainWindow;
8
9     mainWindow.setWindowTitle("Quiz");
10    QGridLayout * layout = new QGridLayout(& mainWindow);
11
12    QLabel * label1 = new QLabel("Wer wurde 2010 <br>Fussball-");
13    mainWindow.setWindowTitle("Quiz");
14    mainWindow.setWindowTitle("Quiz");
15    mainWindow.setLayout(layout);
16
17    layout->addWidget(label1, 0, 0, 1, 2);
18
19    QRadioButton * rbSchweiz = new QRadioButton("Schweiz");
20    layout->addWidget(rbSchweiz, 1, 0);
21    QRadioButton * rbSpanien = new QRadioButton("Spanien");
22    layout->addWidget(rbSpanien, 1, 1);
23    QRadioButton * rbItalien = new QRadioButton("Italien");
24    layout->addWidget(rbItalien, 2, 0);
25    QRadioButton * rbHolland = new QRadioButton("Holland");
26    layout->addWidget(rbHolland, 2, 1);
27
28    QPushButton * button = new QPushButton ("Absenden");
29    layout->addWidget(button, 3, 0, 1, 2);
30
31    QObject::connect(button, SIGNAL(clicked()), qApp, SLOT( quit() ) );
32
33    mainWindow.show();
34    return app.exec();
35 }

```

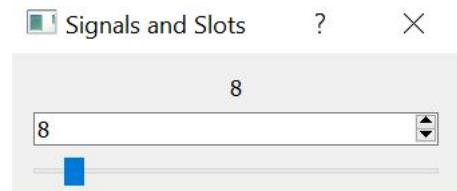


Signal and Slots

```

1 #include <QtWidgets>
2
3 int main(int argc, char * argv[])
4 {
5     QApplication app(argc, argv);
6
7     QDialog mainWindow;
8
9     mainWindow.setWindowTitle("Signals and Slots");
10    QVBoxLayout * layout = new QVBoxLayout(& mainWindow);
11
12    QLabel * label = new QLabel("0");
13    label->setAlignment(Qt::AlignCenter);
14    layout->addWidget(label);
15    QSpinBox * spinBox = new QSpinBox;
16    layout->addWidget(spinBox);
17    QSlider * slider = new QSlider(Qt::Horizontal);
18    layout->addWidget(slider);
19
20    QObject::connect(spinBox, SIGNAL(valueChanged(int)), label,
21                      SLOT(setNum(int)) );
22    QObject::connect(spinBox, SIGNAL(valueChanged(int)), slider,
23                      SLOT(setValue(int)) );
24    QObject::connect(slider, SIGNAL(valueChanged(int)), label,
25                      SLOT(setNum(int)) );
26    QObject::connect(slider, SIGNAL(valueChanged(int)), spinBox,
27                      SLOT(setValue(int)) );
28
29    mainWindow.show();
30    return app.exec();
31 }

```



7.3.7 Hello World

```

1 #include <QtGui>
2
3 int main(int argc, char **argv)
4 {
5     QApplication app(argc, argv);           // eine neue Instanz von QApplication
6     QWidget *window = new QWidget();         // eine neue Instanz von QWidget
7     QLabel *label = new QLabel("Hello World!"); // Ein Label mit einem Text erzeugen
8     app.setMainWidget(window);               // window wird zum Haupfenster der Applikation "app"
9     window->show();                      // Anzeigen des Fensters
10    return app.exec();                    // Ausführen der Applikation
}

```

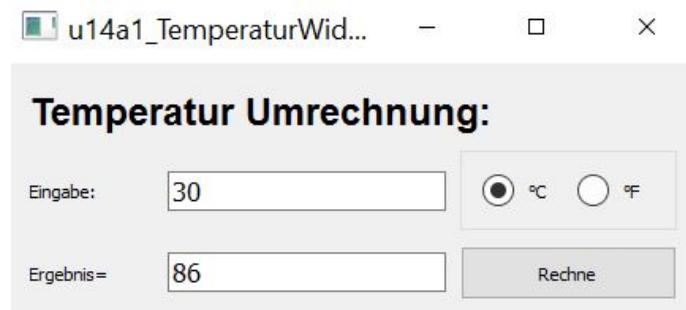
7.3.8 Temperaturwidget

main.cpp

```

1 //TemperaturWidget main.cpp
2 #include <QtWidgets>
3 #include "TemperaturWidget.h"
4 //-----
5 int main(int argc, char *argv[])
6 {
7     QApplication app(argc, argv);
8     TemperaturWidget mainWindow;
9
10    mainWindow.show();
11    return app.exec();
12 }

```



temperaturwidget.h

```

1 //TemperaturWidget TemperaturWidget.h
2 #ifndef TEMPERATURWIDGET_H
3 #define TEMPERATURWIDGET_H
4 #include <QtWidgets>
5 class TemperaturWidget : public QWidget
6 {
7     Q_OBJECT
8     private: // **** GUI-Stuff:
9     QGridLayout * mainLayout;
10    QLabel *label1, *label2, *label3;
11    QLineEdit *eingabeLineEdit, *resultLineEdit;
12    QGroupBox *groupBox;
13
14    QHBoxLayout * boxLayout;
15    QRadioButton * tCelsiusButton, * tFahrenheitButton;
16
17    QPushButton * rechneButton;
18 public:
19     TemperaturWidget(QWidget * parent = 0);
20     private slots:
21     void rechne();
22 };
23 #endif // MYDIALOG_H

```

temperaturwidget.cpp

```

//TemperaturWidget TemperaturWidget.cpp
2 #include <QtWidgets>
3 #include "TemperaturWidget.h"
4 //-----
5 TemperaturWidget::TemperaturWidget(QWidget * parent)
6   : QWidget(parent)
7 {
8   // **** Darstellung festlegen:
9   mainLayout = new QGridLayout;
10  this->setLayout(mainLayout);
11  // Obere Zeile:
12  label1 = new QLabel("<b>Temperatur Umrechnung:</b>");
13  mainLayout->addWidget( label1, 0, 0, 1, 3 );
14
15  label1->setAlignment(Qt::AlignLeft);
16  label1->setAlignment(Qt::AlignVCenter);
17  label1->setAutoFillBackground(true);
18  label1->setFont( QFont("Arial", 12) );
19  // Mittlere Zeile:
20  label2 = new QLabel("Eingabe:");
21  mainLayout->addWidget( label2, 1, 0 );
22  eingabeLineEdit = new QLineEdit();
23  mainLayout->addWidget( eingabeLineEdit, 1, 1 );
24  boxLayout = new QHBoxLayout;
25  groupBox = new QGroupBox;
26  groupBox->setLayout(boxLayout);
27  tCelsiusButton = new QRadioButton("\u00b0C", groupBox);
28  tCelsiusButton->setChecked(true);
29  boxLayout->addWidget(tCelsiusButton);
30  tFahrenheitButton = new QRadioButton("\u00b0F", groupBox);
31  boxLayout->addWidget(tFahrenheitButton);
32  mainLayout->addWidget(groupBox, 1, 2);
33  // Untere Zeile:
34  label3 = new QLabel("Ergebnis= ");
35  mainLayout->addWidget( label3, 2, 0 );
36  resultLineEdit = new QLineEdit();
37  resultLineEdit->setReadOnly(true);
38  mainLayout->addWidget( resultLineEdit, 2, 1 );
39
40  rechneButton = new QPushButton("Rechne");
41  mainLayout->addWidget( rechneButton, 2, 2 );
42  // **** Interaktives Verhalten initialisieren:
43  QObject::connect(rechneButton, SIGNAL(clicked()),
44                   this, SLOT(rechne()) );
45  QObject::connect(eingabeLineEdit, SIGNAL(editingFinished()),
46                   this, SLOT(rechne()) );
47 }
48 //-----
49 void TemperaturWidget::rechne()
50 // Slot: wird aufgerufen falls Button 'Rechne' betoetigt.
51 {
52   QString s = eingabeLineEdit->text();
53   bool okay;
54   double x = s.toDouble(&okay);
55   if (!okay)
56   {
57     resultLineEdit->setText("Eingabe falsch");
58     eingabeLineEdit->setText("0.0");
59   }
60   else
61   {
62     double res = 0;
63     if (tCelsiusButton->isChecked() )
64     {
65       // Umrechnung Celsius --> Fahrenheit
66       res = x * 1.8 +32;
67     }
68     else
69     {
70       // Umrechnung Fahrenheit --> Celsius
71       res = (x - 32) * 5.0 / 9.0;
72     }
73     QString r = QString("%1").arg(res, 0, 'g', 5 );
74     resultLineEdit->setText(r);
75   }
76 }

```

7.3.9 Counter-implementation

Implementation mit Klassen

main (Klassen).cpp

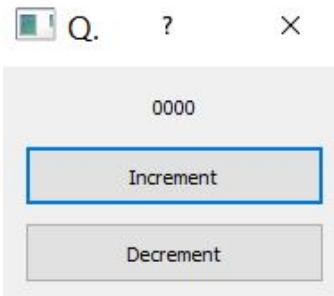
```
//countermain2.cpp counter in mehreren Files
2 #include <QtWidgets>
3 #include "mydialog.h"
4
5 int main(int argc, char *argv[])
6 {
7     QApplication app(argc, argv);
8     MyDialog windowTop(0);
9
10    windowTop.show();
11    return app.exec();
12 }
```

mydialog.h

```
//mydialog.h counter in mehreren Files
2 ifndef MYDIALOG_H
3 define MYDIALOG_H
4 #include "counter.h"
5 #include <QtWidgets> // fuer Qt5
6
7 class MyDialog : public QDialog
8 {
9     Q_OBJECT
10 private:
11     Counter counter;
12     QVBoxLayout *layoutTop;
13     QPushButton *buDec;
14     QPushButton *buInc;
15     QLabel *laCount;
16
17 public:
18     MyDialog(QWidget *parent = 0);
19 };
20
21 #endif // MYDIALOG_H
```

mydialog.cpp

```
//mydialog.cpp counter in mehreren Files
2 #include "counter.h"
3 #include "mydialog.h"
4
5 MyDialog::MyDialog(QWidget *parent):
6     QDialog(parent)
7 {
8     this->resize(200, 100); // w, h
9
10    layoutTop = new QVBoxLayout;
11    this->setLayout(layoutTop);
12
13    laCount = new QLabel("0000");
14    laCount->setAlignment(Qt::AlignHCenter);
15    layoutTop->addWidget(laCount);
16
17    buInc = new QPushButton("&Increment");
18    layoutTop->addWidget(buInc);
19
20    buDec = new QPushButton("&Decrement");
21    layoutTop->addWidget(buDec);
22
23    // **** Interaktives Verhalten initialisieren:
24    QObject::connect(buInc, SIGNAL(clicked()), 
25                      &counter, SLOT(incValue()));
26    QObject::connect(buDec, SIGNAL(clicked()), 
27                      &counter, SLOT(decValue()));
28
29    QObject::connect(&counter, SIGNAL(valueChanged(int)),
30                      laCount, SLOT(setNum(int)) );
31 }
```



Implementation "nur" im main.cpp

```
// CCounter main.cpp Alles im mainfile
2 #include <QtWidgets>
3 #include "counter.h"
4 int main(int argc, char *argv[])
5 {
6     // **** Qt-Prolog:
7     QApplication app(argc, argv);
8
9     // **** Problemdomain:
10    Counter counter;
11
12    // **** GUI-Darstellung aufbauen:
13    QDialog wTop;
14    wTop.resize(200, 100); // w, h
15
16    QVBoxLayout * layoutTop = new QVBoxLayout();
17    wTop.setLayout(layoutTop);
18
19    QPushButton * buDec = new QPushButton ("&Decrement");
20    layoutTop->addWidget(buDec);
21
22    QLabel * laCount = new QLabel("0000");
23    laCount->setAlignment(Qt::AlignCenter);
24    layoutTop->addWidget(laCount);
25
26    QPushButton * buInc = new QPushButton ("&Increment");
27    layoutTop->addWidget(buInc);
28
29    // **** Interaktives Verhalten initialisieren:
30    QObject::connect(buInc, SIGNAL(clicked()), 
31                      &counter, SLOT(incValue()));
32    QObject::connect(buDec, SIGNAL(clicked()), 
33                      &counter, SLOT(decValue()));
34
35    QObject::connect(&counter, SIGNAL(valueChanged(
36                      int)), 
37                      laCount, SLOT(setNum(int)) );
38
39    // **** Epilog:
40    wTop.show();
41
42    return app.exec();
43 }
```

counter.h

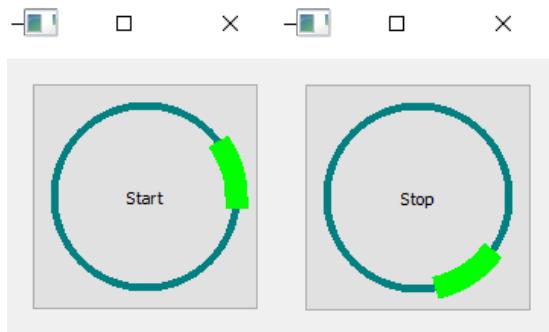
```
// File: counter.h
2 #ifndef _COUNTER_H
3 #define _COUNTER_H
4 #include <QObject>
5 class Counter : public QObject
6 {
7     Q_OBJECT
8     private:
9         int _value;
10    public:
11        Counter();
12        int value() const;
13    public slots:
14        void incValue();
15        void decValue();
16        void setValue(int value);
17        void clearValue();
18    signals:
19        void valueChanged(int newValue);
20};
#endif
```

counter.cpp

```
// File: counter.cpp
2
3 #include <QDebug>
4 #include "counter.h"
5 Counter::Counter()
6 {
7     _value = 0;
8 }
9 int Counter::value() const
10 {
11     return _value;
12 }
13 void Counter::setValue(int value) // slot
14 {
15     if (value != _value)
16     {
17         _value = value;
18         emit valueChanged(_value);
19     }
20 }
21 void Counter::incValue() // slot
22 {
23     _value++;
24     qDebug() << "incremented, new Value= " << _value;
25     emit valueChanged(_value);
26 }
27 void Counter::decValue() // slot
28 {
29     _value--;
30     qDebug() << "decremented, new Value= " << _value;
31     emit valueChanged(_value);
32 }
33 void Counter::clearValue() // slot
34 {
35     if (_value != 0)
36     {
37         _value = 0;
38         qDebug() << "Value cleared, new Value= " << _value;
39         emit valueChanged(_value);
40     }
41 }
```

CircleButton

```
//Circlebutton main.cpp
1 #include <QtWidgets>
2 #include "CircleButton.h"
3 #include "Demo1CircleButton.h"
4
5 int main(int argc, char * argv[])
6 {
7     QApplication app(argc, argv);
8
9     Demo1CircleButton demo1Window;
10    demo1Window.move(100, 100);
11
12    demo1Window.show();
13    return app.exec();
14 }
```



```
//Circlebutton Demo1CircleButton.h
1 ifndef DEMO1CIRCLEBUTTON_H
2 define DEMO1CIRCLEBUTTON_H
3
4 #include <QWidget>
5 #include "CircleButton.h"
6
7 class Demo1CircleButton : public QWidget
8 {
9     Q_OBJECT
10 private:
11     QVBoxLayout * layout;
12     CircleButton * circleButton;
13
14 public:
15     explicit Demo1CircleButton(QWidget *parent = 0);
16
17 private slots:
18     void onClicked();
19 };
20
21 #endif // DEMO1CIRCLEBUTTON_H
```

```
//Circlebutton Demo1CircleButton.cpp
1 #include "Demo1CircleButton.h"
2
3 Demo1CircleButton::Demo1CircleButton(QWidget *parent)
4     : QWidget(parent)
5 {
6     setWindowTitle("Demo 1");
7     layout = new QVBoxLayout(this);
8
9     circleButton = new CircleButton("Start");
10    layout->addWidget(circleButton);
11
12    QObject::connect(circleButton, SIGNAL(clicked()), this, SLOT(onClicked()));
13 }
14
15 void Demo1CircleButton::onClicked()
16 {
17     if (circleButton->isRotating())
18     {
19         circleButton->setRotating(false);
20         circleButton->setText("Start");
21     }
22     else
23     {
24         circleButton->setRotating(true);
25         circleButton->setText("Stop");
26     }
27 }
```

```

//Circlebutton CircleButton.h
2 #ifndef CIRCLEBUTTON_H
# define CIRCLEBUTTON_H
4
# include <QtWidgets>
6
class CircleButton : public QPushButton
8 {
10
private:
    Q_OBJECT
12
private:
    // Size of the Ring:
    enum { ringRadius = 60 };
16    enum { ringWidthSmall = 5 };
    enum { ringWidthWide = 15 };
18
    // Circular sector data:
    int _sectorPosition; // Kreissektor-Position in 1/16 Grad (0 <= 3 Uhr)
    int _sectorAngle; // Kreissektor-Winkel in 1/16 Grad
    int _rotationSpeed; // Umdrehungsgeschwindigkeit in 1/16 Grad pro 50ms
    bool _isRotating; // Rotation is enabled / disabled
24
    // Other data:
    QTimer * _timer;
26
private:
    void paintEvent(QPaintEvent *);
28
public:
    explicit CircleButton(const QString & text, QWidget * parent = 0);
    explicit CircleButton(QWidget * parent = 0);
34
    bool isRotating();
    // returns if ring rotating is enabled (true) or disabled (false)
36
    void setRotating(bool doRotate);
    // enable/disable ring rotating
38
private slots:
    void onTimeout();
40
public slots:
    void setSectorPosition(int position);
    // Kreissektor-Position in 1/16 Grad setzen
    // pos 0 entspricht der Uhrposition 3 Uhr
44
    void setSectorAngleDegrees(int angle);
    // Grösse des Sektorwinkel in Grad setzen
48
    void setRotationSpeed(int speed);
    // Rotationsgeschwindigkeit in 1/16 Grad pro 50ms setzen
50
};
52
#endif // CIRCLEBUTTON_H

```

```

//Circlebutton CircleButton.cpp
2 #include "CircleButton.h"
4
CircleButton::CircleButton(QWidget *parent)
    : QPushButton(parent)
6 {
    this->setFixedSize(150, 150);

    _sectorAngle = 60*16; // 30 Grad
    _sectorPosition = 0*16; // 0 Grad = 3 Uhr
    _rotationSpeed = (int)(90.0* 16.0 / 20); // 90 Grad/sec = (90.0 * 16.0)/ 20
    _isRotating = false;

    _timer = new QTimer(this);
    QObject::connect(_timer, SIGNAL(timeout()), this, SLOT(onTimeout()));
}
16
CircleButton::CircleButton(const QString &text, QWidget *parent)
    : QPushButton(text, parent)
18 {
    this->setFixedSize(150, 150);
20

```

```

22     _sectorAngle = 30*16;      // 30 Grad
23     _sectorPosition = 0*16;    // 0 Grad = 3 Uhr
24     _rotationSpeed = (int)(90.0* 16.0 / 20); // 90 Grad/sec = (90.0 * 16.0)/ 20
25     _isRotating = false;
26
27     _timer = new QTimer(this);
28     QObject::connect( _timer, SIGNAL( timeout() ), this, SLOT( onTimeout() ) );
29 }
30 void CircleButton::paintEvent(QPaintEvent * pe)
31 {
32     QPushButton::paintEvent(pe);
33     QPainter painter(this);
34
35     // schmaler Ring:
36     QPen pen(Qt::darkCyan, ringWidthSmall );
37     painter.setPen(pen);
38     painter.setBrush(Qt::NoBrush);
39
40     QRect rect ( (width()-2*ringRadius)/2, (height()-2*ringRadius)/2, 2*ringRadius, 2*ringRadius);
41     painter.drawEllipse(rect);
42
43     // Ringsegment (breit) zeichen:
44     painter.setPen( QPen(Qt::green, ringWidthWide) );
45
46     painter.drawArc(rect, _sectorPosition, _sectorAngle);
47 }
48 bool CircleButton::isRotating()
49 {
50     return _isRotating;
51 }
52 void CircleButton::setRotating(bool doRotate)
53 {
54     if (doRotate)
55     {   if (! _timer->isActive() )
56         {
57             _timer->start(50); // 50ms
58         }
59     }
60     else
61     {
62         _timer->stop();
63     }
64     _isRotating= doRotate;
65     update(); // update (draw) this widget
66 }
67 void CircleButton::onTimeout()
68 {
69     _sectorPosition -= _rotationSpeed;
70     update(); // update (draw) this widget
71 }
72 void CircleButton::setRotationSpeed(int speed)
73 {
74     if (_rotationSpeed != speed)
75     {   _rotationSpeed = speed;
76         update(); // update (draw) this widget
77     }
78 }
79 void CircleButton::setSectorPosition(int position)
80 {
81     if (_sectorPosition != position)
82     {
83         _sectorPosition = position;
84         update(); // update (draw) this widget
85     }
86 }
87 void CircleButton::setSectorAngleDegrees(int angle)
88 {
89     angle = angle * 16;
90     if (_sectorAngle != angle)
91     {
92         _sectorAngle = angle;
93         update(); // update (draw) this widget
94     }
95 }
96 }
```


Glossar

API	Application Program Interface
CVCS	Central Version Control System
DMS	Document Management System
DVCS	Distributed Version Control System
FIT	Framework for Integrated Test
GUI	Graphical User Interface
LVCS	Local Version Control System
moc	Meta Object Compiler
OOA	Objekt Orientierte Analyse
OOD	Objekt Orientiertes Design
OOI	Objekt Orientierte Implementation
PAP	Projekt Ablauf Plan
PC	Projektleiter
PSP	Projekt Struktur Plan
RUP	Rational Unified Process
SHA	Secure Hash Algorithmus
SUT	System Under Test
UML	Unified Modeling Language
VCS	Version Control System
XP	Xtreme Programming