Projektmanagement & Software Engineering

H. Badertscher bearbeitet von Lukas Vassalli

28. Juli 2016

Inhaltsverzeichnis

1 Software Entwicklung

1.1 Allgemeine Begriffe

1.1.1 Schwierigkeiten

Essentielle Schwierigkeiten:	verursacht durch Komplexität des Problems
Kontingente Schwierigkeiten:	selbst verursacht durch falsche Methoden, Mittel

1.1.2 Komplexität

- Hauptproblem in der Softwareentwicklung
- **1. Gebot**: Halte die Komplexität im Griff.

Bewältigung der Komplexität:

1. Teile und herrsche *"divide et impera"*
   - Problem in Subprobleme aufteilen
   - Unterteilung nach Problembereich (fachlich), Tätigkeiten und Zielen oder zeitlicher Abfolge
   - Ansätze: möglichst eigenständige Teile (hohe Kohäsion) und möglichst geringe Abhängigkeiten (kleine Kopplung)
2. Abstraktion
   - Definition: Weglassen von Aspekten und Details, die für den gegenwärtigen Zweck nicht wichtig sind.
3. Modelle
   - Abstraktionen der realen Welt, z.B. Design-Modelle oder Domain-Modelle

1.1.3 Brook'sche Regel

Adding manpower to a late project makes it later - Problem: Kommunikationsaufwand wird massiv erhöht.

1.2 Vorgehensmodelle

1.2.1 Projektphasen

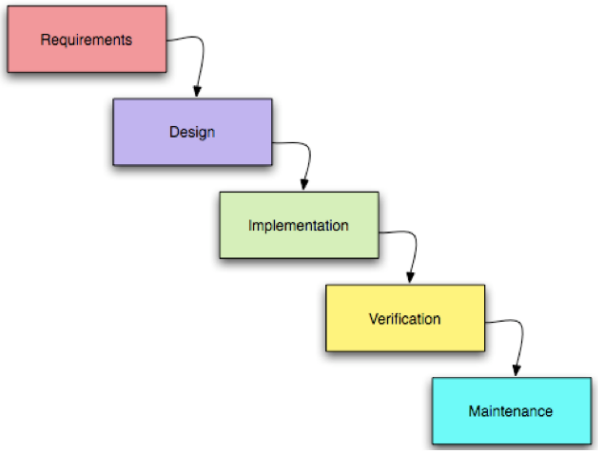
Alle Modelle basieren mehr oder weniger auf folgenden Projektphasen: (Werden diese Phasen nacheinander durchgeführt spricht man auch von Phasen und Phasenplänen)

- Analyse
  - Ziel: Man weiss mehr als vorher, Man weiss WAS entwickelt werden soll.
  - Probleme, Ideen, Anforderungen aufnehmen
  - Erstellen eines Pflichtenhefts
  - Was soll gebaut werden? ->*doing the right things*
  - Das Ergebniss der Analyse ist eine Anforderungsspezifikation, Lasten- und Pflichtenheft
- Design
  - Ziel: Man weiss WIE das Softwaresystem gebaut werden soll. Man hat ein Lösungskonzept
  - Erstellen eines Grobentwurfs, Softwarekonzepts und von Detailentwürfen
  - Wie soll es gebaut werden? ->*doing things right*
  - Das Ergebniss aus der Desigen Phase sind Baupläne für die Software (noch kein Code)

- Implementierung
  - Entwicklung des Sourcecodes anhand des Designs
- Test

1.3 Wasserfall-Modell

- linear (nicht iterativ)
- 1970 als fehlerhaftes, nicht funktionierendes System beschrieben
- Es gibt kein Zurück, alles muss beim ersten Mal richtig gemacht werden. (Dieses Modell funktioniert nicht!)
- Freigabe beim Abschluss jeder Phase

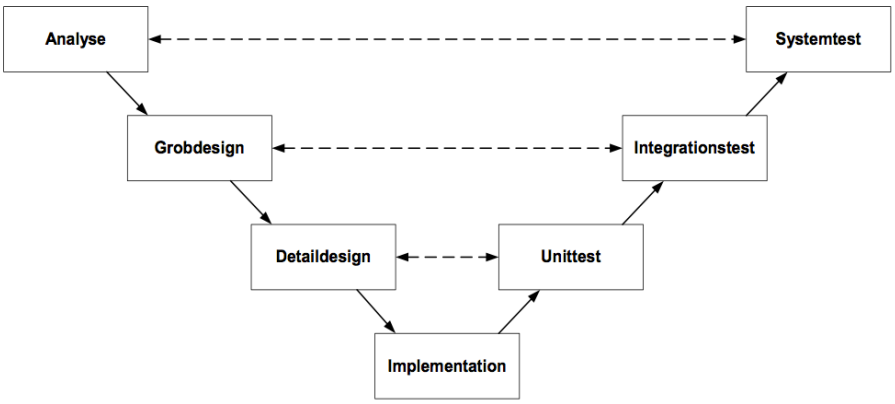


1.3.1 Iterativer Wasserfall

- = Kaskade
- Erweiterung des Wasserfalls um Korrekturschleifen
- In Realität durchführbar (nicht wie der reine Wasserfall)
- Man kann immer nur eine Phase zurück

1.4 V-Modell

Grundidee: korrespondierende Tests
Eigentlich eine (durch Tests) erweitertes Wasserfall-Modell (x-Achse=Zeit, y-Achse =Detallierungsgrad)

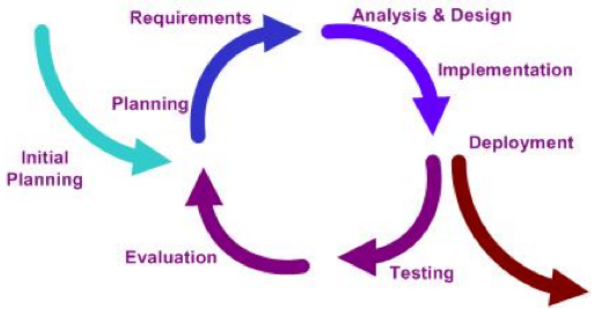


1.5 Iterative / Inkrementelle Entwicklung

Grundidee: Software wird in einzelnen Schritten erstellt.

Inkrementell:

- Resultat in Schritten entwickeln
- Jeder Schritt ist Teil des Endresultats (werden nicht mehr angepasst)

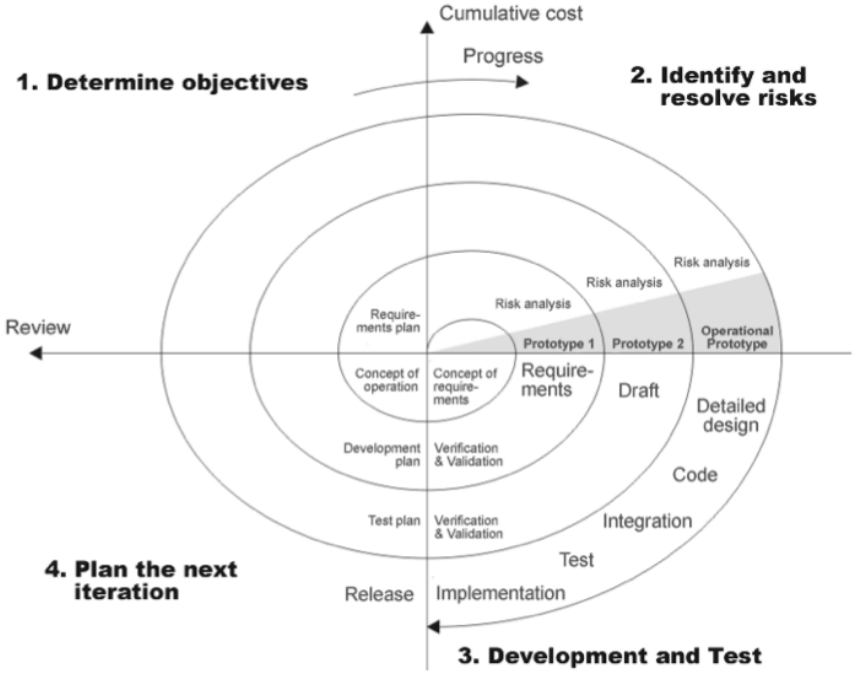


Iterativ:

- Erste Version entwickeln
- In weiteren Schritten bessere Versionen erstellen

1.5.1 Spiralmodell

Entwickelt von Barry Böhm, 1988



1.6 Agile Softwareentwicklung

- Gegenbewegung zur herkömmlichen schwerfälligen, bürokratischen Softwareentwicklung
- agile = flink, beweglich
- Individuen und Interaktionen gelten mehr als Prozesse
- Grundlagen: Extreme Programming (Kent Beck, 1999); Agiles Mainfest (2001)

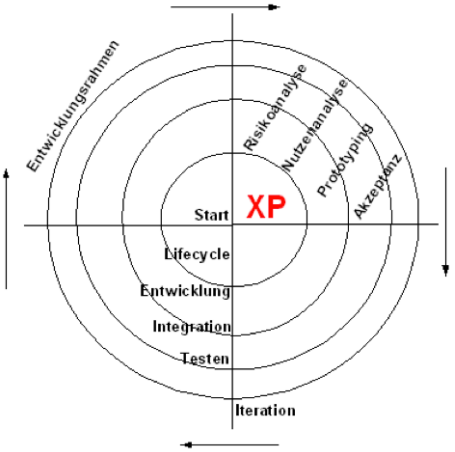
Agile Softwareentwicklung besteht aus:

1. Agile *Werte* bilden das Fundament.
2. Agile *Prinzipien* sind Handlungsgrundsätze, die auf agilen Werten basieren.

3. Agile *Methoden* sind konkrete Verfahren, die sich auf agile Werte und Prinzipien stützen.
4. Agile *Prozesse* sind die Zusammenfassung agiler Methoden.

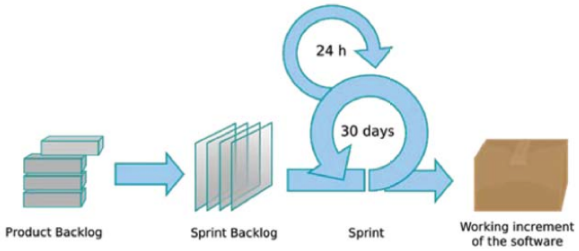
1.6.1 XP (Extreme Programming)

- Bekanntester agiler Prozess
- Nach Buch "Extreme Programming" von Kent Beck (1999)



1.6.2 Scrum

- = "Gedränge"
- Vorgestellt auf OOPSLA '95
- Basis: Sprints von 15-30 Tagen dauernden Perioden, in welchem von Team eine neuer (brauchbarer) Software-Release erstellt wird.



1.7 Objektorientierte Softwareentwicklung

Grundidee: Modelle auf allen Stufen: OO-Analyse → OO-Design → OO-Implementation. Objekte entsprechen in der Regel realen Dingen; Änderungen werden daher weniger häufig und einfacher.

Kriterien für die Wahl des Modells:

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

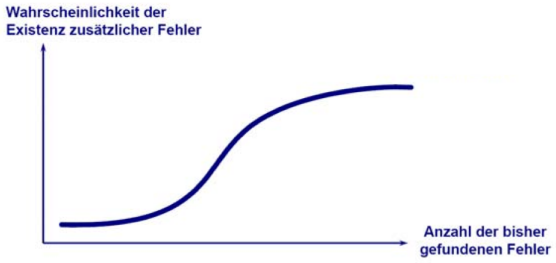
2 Testing

2.1 Allgemeine Begriffe

- Definition: Programm mit der Absicht auszuführen, Fehler zu finden
- Durch das Testen kann die Korrektheit eines Programms nicht bewiesen werden!
- Erster Bug: 1947 wurde eine Motte im Relais eines Mark II Aiken Rechners entdeckt.

2.1.1 Wahrscheinlichkeit von Fehlern

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



2.1.2 Ziel

Ziel des Testing:
Aufzeigen, dass Fehler existieren

Ziel des Debugging:
Durch Testing gefundene Fehler lokalisieren und beheben.

2.1.3 Anforderungen

- Geplant → Testplan erstellen
- Systematisch → Testspezifikationen erstellen
- Festgehalten → Testprotokoll erstellen
- Reproduzierbarkeit:
  - Wissen, was getestet wurde
  - Unabhängig von testender Person
- Automatisierung, wenn möglich
- Testspezifikationen laufend erweitern, Regressionstests

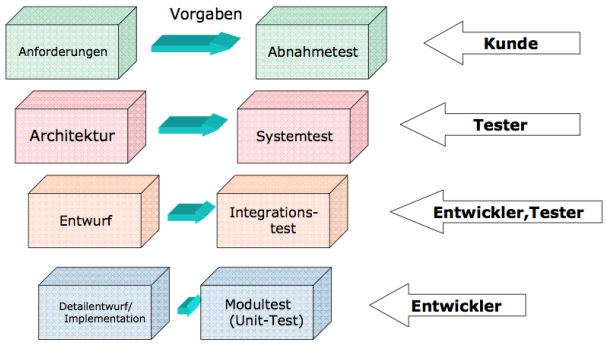
2.1.4 Arten von Tests

- Funktionale Tests
- Nicht-Funktionale Tests
  - Leistungstests
  - Usability Tests
  - ...

2.1.5 Massnahmen zur Softwareprüfung

- Dynamische Prüfung
  - Testing
  - Dynamische Analyse
- Statische Prüfung
  - Metrikanalyse
  - Code Reviews

2.1.6 Wer Testet



2.1.7 Verifikation und Validierung

Validierung:
"Are we building the right product?"
Überprüft das Software-Produkt, ob es die Anforderungen des Auftraggebers erfüllt.
Verifikation:
"Are we building the product right?"
Überprüft die Artifacts (Dokumente ect.) während der Entwicklung ob sie ihre Vorgaben erfüllen

2.2 Methoden

2.2.1 Blackbox Tests

Tests ohne Kenntnis der inneren Struktur; Anwendung: Abnahme-, System-, Integrationstests

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:
Zustände: Stack leer, Stack halb-voll, Stack voll
Testfälle: Element hinzufügen, Element entfernen

2.2.2 Whitebox Tests

Tests mit Kenntnis der inneren Struktur; Anwendung: Modul-, Integrationstests

Kontrollfluss-orientiertes Testen:
Erstellen eines Kontrollfluss-Graphen, in welchem jedes Statement einem Knoten entspricht. Zweige, welche Schleifen und Verzweigungen entsprechen, verbinden die Knoten. Der Test wird so ausgelegt, dass die Überdeckung (Coverage) möglichst gut ist. (Test der Überdeckung mit Dynamic Analyzer)

- Anweisungsüberdeckung
  - Prozentualer Anteil der Anweisungen im Test ausführen
  - 100% Anweisungsüberdeckung ist Minimum
- Zweigüberdeckung
  - Prozentualer Anteil der Zweige wird durchlaufen
  - 100% Zweigüberdeckung ⇒ 100% Anweisungsüberdeckung
  - Da 100% kaum erreichbar werden oft Äquivalenzklassen u/o Grenzwerte auf Kontrollstrukturen angewandt
- Bedingungsüberdeckung
  - Verschiedene Kombinationen testen bei Verzweigungen
  - mind. 1x true/false durchlaufen
- Funktionsüberdeckung
  - "Tut es das, was der Kunde spezifiziert hat?"
  - 1 Szenario pro Use Case
  - Blackbox Tests

2.3 Automatisiertes Testing

Vorteile:

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

Nachteile:

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

2.4 Testwerkzeuge

Blackbox:	FIT
Whitebox:	xUnit

TDD = Test Driven Developement (Test getriebene Software-Entwicklung)

1. Methode nur als Stub
2. Test dafür schreiben
3. Methode implementieren und Testen

3 Unit-Testing mit CppUnit

Konzept: Testen einer Komponente (Unit=Modul,Übersetzungseinheit,Klasse,Funktino)
Getestet wird das Interface(Schnittstelle) einer Unit
Heute wird dies mittel Unit-Test-Frameworks gemacht.

3.1 Frameworks allgemein

- Framework = Programmgerüst(wird vom Programmierer ins Prorgramm eingebettet)
- Hollywodd Prinzip "Dont call us, we'll call you"

3.2 Frameworks Arbeitsweise

- Spezielle Test-Funktionen mit Zusicherungen
- Tests laufen automatisiert ab
- Test-Runner=Programm, das die Test-Funktionen der Reihe nach ausführt

3.3 CppUnit Wichtige Begriffe

- **Assert-Anweisungn:** Zum Vergleich von Ist- und Soll-Werten
- **TestKlasse:** selbst geschriebene Klassse, die von CppUnit::TestCase erbt
- **TestFunktion:** enthält Assert-Anweisungen, ist eine Meberfunktion einer Testklasse
- **TestSuite:** zur Zusammenfassung von Testfunktionen
- **TestFixture:** zur Bereitstellung und Abbau einer Testumgebung
- **Registry:** zur Zusammenfassung von Testklassen

3.4 CppUnit Konvention

- Für jede zu testenden Klasse eine entsprechende Testklasse erstellen
- Der Name einer Testklasse beginnt mit dem Namen der zu testenden Klasse und endet mit "Test"Bsp. StackTest, AuthorTest
- Der Name einer Testfunktion beginnt mit "test"Bsp. "testAddition()"

4 Dokumentation mit Doxygen

4.1 Allgemeines zu Dokumentation Zweck

- Wissenssicherung
- Kommunikation: Reden allein genügt nicht => Schreiben zwing zu Genauigkeit
- Sichtbarmachen des Projektfortschrittes

4.2 Eigenschaften von Doxygen

- Weit verbreitetes Dokumentationswerkzeug
- Formate: HTML, LaTex, Unix man pages, RTf, XML
- Erstellt primär API-Beschreibungen (Schnittstellenbeschreibungen), aber mit Zusätzen auch Klassendiagramme und andere Diagramme
- Doxygen ist grundsätzlich ein Command-Line-Tool
- Hauptbefehl (command-line): $ doxygen
- Aufruf Beispiele:
  - doxygen //gibt Hilfe-Text aus
  - doxygen -g //erzeugt Konfig.-Datei "Doxyfile"
  - doxygen Doxyfile //erzeugt Dokumentation

5 Projektmanagement

Merke 1: Planung ersetzt den Zufall durch den Irrtum!
Merke 2: Plane- und du wirst irren. Plane nicht - und du wirst nicht wissen, wo du geirrt hast
Merke 3: Plans are nothing; planning is everything

5.1 Grundlagen Projektmanagement

5.1.1 Die Merkmale eines Projekts

- Es ist ein einmaliges Vorhaben
- Es hat klare Ziele
- Das Projekt hat innovativen Charakter.
- Es ist mit Risiken behaftet
- Es ist Zeitlich begrenzt
- Es gibt einen Kostenrahmen
- Es wird dafür eine spezielle Organisation etabliert.
- Es arbeiten mehrere Personen (>2) daran
- Es gibt einen Projektleiter und ein Projektteam
- Stakeholder: Auftraggeber, Projektleiter, Projektmitarbeiter

5.1.2 Die drei Eckpfeiler eines Projekts

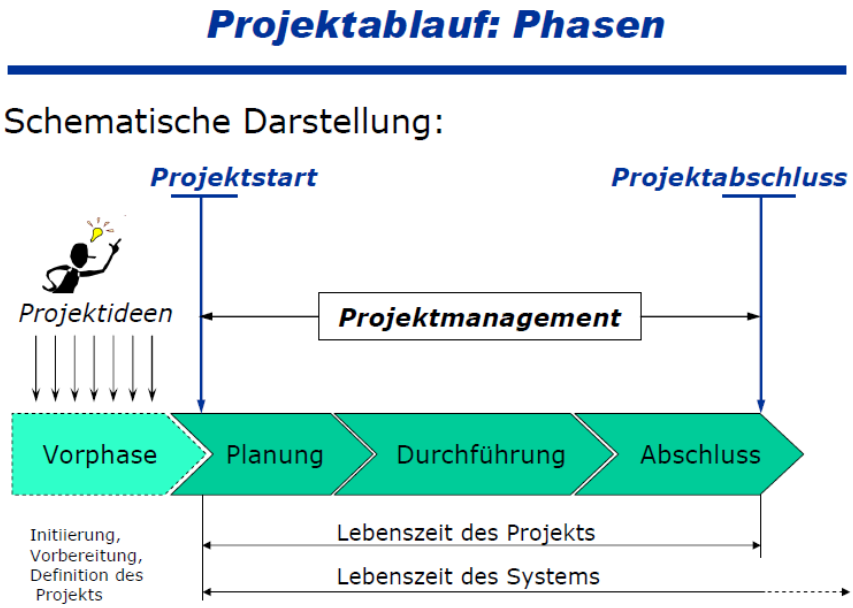
- Ergebnis-SScope": Sachziele, inhaltliches Ergebnis Was genau , mit welchen Eigenschaften, soll erstellt, "gebaut"werden?
- Zeit-"Time": Bis wann muss das Projekt abgeschlossen sein ->Terminplan
- Aufwand - "Cost": Was kostet es bzw. was darf es kosten? ->Budget

5.2 Die Projektbeteiligten

- Auftraggeber
- Projektleiter (PL)
- Projektmitarbeiter

5.3 Projektorganisationsformen

- **Reine Projektorganisation:** PL und Teammitglieder arbeiten Vollzeit, Mitarbeiter sind PL unterstellt
- **Matrix-Projekorganisation:** Projektmitarbeiter zu einem Teil an diesem Projekt beteiligt, Verantwortung und Kompetenzen sind zwischen PL aufgeteilt, weit häufigste Organisation
- **Stabs-Projektorganisation:** Unternehmenshirarchie bleibt unverändert, wird ergänzt durch Projektkoordinator der keine Weisungsbefugnisse besitzt



5.4 Projektphasen

1. **Initiierung,Vorphase,Projektdefinition**
   - Das Projekt wird initiiert, d.h. definiert
   - Tätigkeiten finden vor dem eigentlichen Projekt statt
   - Das Ergebnis dieser Phase ist der Projektauftrag (Ziele,Termine, Budget, Projektorganisation)
   - Projektziel = normative Aussagen über den gewünschten anzustrebenden künftigen Zustand
   - Zielformulierung: S (specific) M (Measurable) A (Acceptable) R (Realistic) T (Time-bound) = SMART
2. **Projektplanung**
   - Eigentlicher Projektstart (Kick-off-Meeting)
   - **Planungstätigkeiten**
     a) **Projektstrukturplan(PSP):** Hierarchische Strukturierung aller vorkommenden Arbeiten in Arbeitspakete
     b) **Projektablaufplan(PAP):** Darstellung der logischen Abhängigkeiten der im PSP vorhandenen Arbeitspakete.
        - **Mögliche Fälle von Abhängigkeiten**
          * Normalfoge: Nachfolger kann erst beginnen, wenn der Vorgänger beendet ist (häufigste Abhängigkeit).
          * Anfangsfolge: Nachfolger kann erst beginnen, wenn der Vorgänger begonnen hat (seltenere Abhängigkeit)

        * Endfolge: Nachfolger kann erst enden, wenn der Vorgänger beendet ist (seltenere Abhängigkeit)
        * Sprungfolge: Nachfolger kann erst enden, wenn der Vorgänger begonnen hat (äusserst ungewöhnliche Abhängigkeit)
    c) **Projektterminplan:** Zuordnung von Terminen zu den einzelnen Arbeitspaketen, Festlegung von Meilensteinen
    d) **Ressourcen und Kapazitätsplan:** Benötigte Ressourcen ermitteln un vorhandene Ressourcen mit Bedarf abstimmen
    e) **Kosten und Budgetplan:** Die Kosten für die benötigten Ressourcen schätzen Kosten ermitteln und Budget erstellen

3. **Projektdurchführung**
    - Projektkontrolle: Soll-Ist-Vergleich, bezweckt das rechtzeitige Feststellen von Abweichungen gegenüber dem geplanten
    - Projektsteuerung: Korrekturmassnahmen, umfasst diejenigen Massnahmen, welche das Projekt auf Zielkurs halten oder bei Abweichungen wieder darauf zurückbringen soll
    - Projekt-Controlling=Projektkontrolle + Projektsteuerung
4. **Projektabschluss**
    - Projektabschluss ist nicht ein Zeitpunkt, sondern eine Phase
    - Abschlussmeeting: Lessons Learned , Reflektieren

6 Versionskontrollsysteme mit Git

6.1 Versionskontrollsysteme

- Aufbewarug(Speicherung), Verwaltung und Wiederherstellung von früheren Fassungen
- Koordinierung des gemeinsamen Zugriffs von mehreren Personen auf Dateien
- Hauptaufgaben
    - Protokollierung
    - Wiederherstellung
    - Koordinierung
    - Mehrere Entwicklungszweige (Branches)
    - Ein Branch ist eine Abspaltung von der normalen Entwicklungslinie bei der alternative Lösungen untersucht werden können.
- Repository = Lager, Depot, Zentraler Speicher für Daten
- to commit = Daten in das Repository abspeichern est entsteht eine neue Revision im Repository
- Revision= Zustand der Daten in einem Repository zu einem bestimmten Zeitpunkt

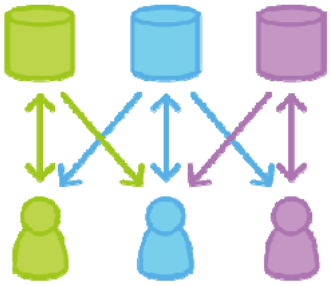
6.2 Centralized vs. distributed VCS

Centralized VCS:

- Beispiel "Subversion":
- es gibt (pro Projekt) genau ein zentrales Repository. (Client Server-Architektur.)

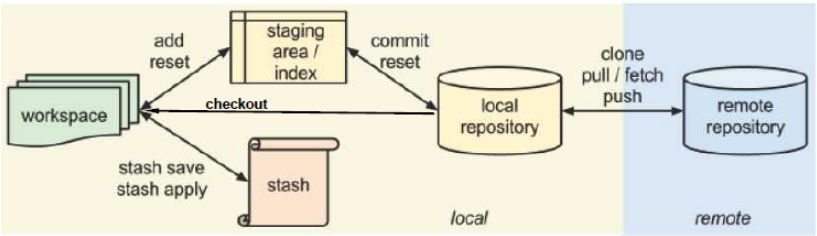
Distributed VCS:

- Beispiel "Git":
- es gibt (für das gleiche Projekt) mehrere, grundsätzlich gleichberechtigte Repositories.



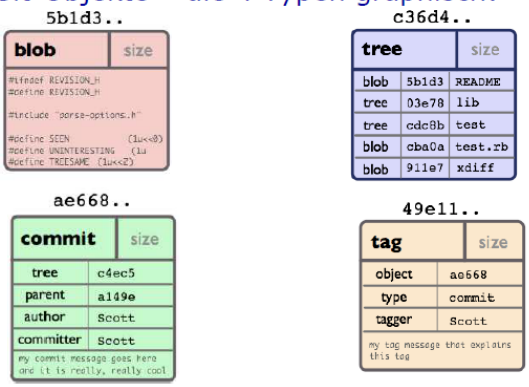
6.3 Git Grundkonzept

- **Workspace**
  - Projektverzeichnis.
  - enthält die Dateien mit denen man arbeitet.
- **Staging Area/Index/Cache**
  - "Bereitstellungsraum".
  - sammelt Änderungen für nächsten Commit.
- **Repository**
  - enthält die Versionsgeschichte in Form von Commits.
  - Unterscheidung: "local" und "remote" Repositories
- **Stash**
  - "Versteck", "Lager".
  - zum temporären Speichern der Workspace-Daten.



- **Git-Workspace:** Ist ein gewöhnliches Verzeichnis mit den bearbeitenden Dateien eines Projekts.
- **Git-Repository:** Das Unterverzeichnis ".git"mit den seinen Daten bildet das lokale Git-Repository
- **Remote Repository:** Ist in der Regel ein normales Git-Repository, jeoch OHNE zugehöriges Projektverzeichnis. Wird auch "bare"genannt.S
- **SHA1-Hashwert:** Diese Hashwerte werden als Zeigerwerte (Adresse) verwendet um die Repository Datenstruktur aufzubauen 40stelliger Hex-Code
- **Hashwert:** Dieser wird als Dateinamen verwendet. Ersten zwei Hex-Ziffern für den Namen des Unterverzeichnis, die restilichen 38 für den Namen der Datei
- **Git-Objekte-die 4 Typen:**
  - **blob = binary large Object:** besteht aus dem Inhalt einer Date, enthält keine Referenzen
  - **tree:** enthält Referenzen auf blob oder tree Objekte

  - **commit:** Startobjekt für einen Commit, enthält einen Zeiger auf tree, enthält eine Zeiger auf das Eltern Commit zeigen kann
  - **tag:** Für Etiketten mit Kommentar, beinhaltet einen Zeiger auf ein commit-Objekt
    - Git-Objekte – die 4 Typen graphisch:

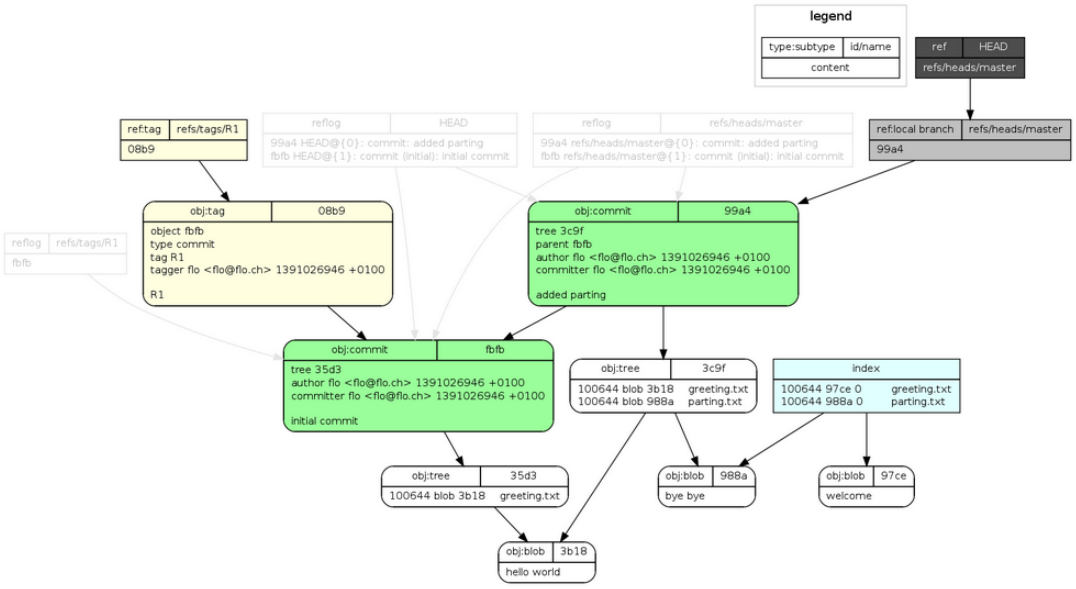


- **Git-Referenzen:** Sind einfach Dateien mit einem normalen Namen, Sind Zeiger auf ein Git-Objekt oder Git-Referenz
  - Hashwert-basierte Zeiger
  - Symbolische Zeiger

6.3.1 Konkretes Beispiel

```
git init
echo 'hello world' > greeting.txt
git add greeting.txt
git commit -m 'initial commit'
git tag R1 -m R1
echo 'bye bye' > parting.txt
git add parting.txt
git commit -m 'added parting'
echo 'welcome' > greeting.txt
git add greeting.txt
```

`git-draw` will display the following image



6.4 Git Befehle (vor den Befehl kommt immer "git"Bsp. "git init")

```
These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Reapply commits on top of another base tip
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local ↩
      branch
   push       Update remote refs along with associated objects

 git help -a  and  git help -g  list available subcommands and some
concept guides. See  git help <command>  or  git help <concept>
to read about a specific subcommand or concept.
```


7 Qt

7.1 2 Arten von QT-Programmen

- **QT Widget Programme:** Basierend auf der Qt C++ Klassenbibliothek
- **Qt-Quick oder QML Programm:** Basierend auf QML, legt das Aussehen fest (QML=Qt Meta Language oder Qt Modelling Language)

7.2 Qt-Namen Konvention

- **Qt-Modul-Namen:** beginnend mit Qt..., QtGui, QtWidgets
- **Qt-Klassen-Namen:** beginnend mit Q..., QObjekt, QWidget
- **Qt-Variablen und Funktionen-Namen:** beginnend mit q..., qApp, qTraslator

7.3 Zwei Arten von Qt-Klassen

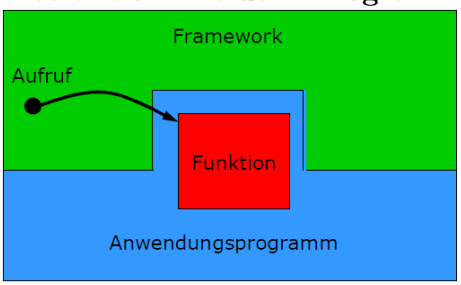
- **QObject und davon abgeleitete Klassen**: alle Widgetsklassen ect.
- **Nicht von QObject abgeleitete Klassen**: Hilfsklassen

7.4 Qt Building

Bei einem Build-Problem können alle Dateien ausser der .pro-Datei und den Source-Dateien gelöscht werden.

7.5 GUI Frameworks 1 und 2 (Grund Probleme)

- **Ansicht festlegen (Layout)**
  1. Absolute Positionierung
  2. Verwendung von Layout-Managern "von Hand"
  3. Verwendung eines des Gui-Designer (Qt Layout Mangager)
- **Interaktion mit dem Programmbenutzer (Ereignisbehandlung)**



7.6 Qt Layout Manager

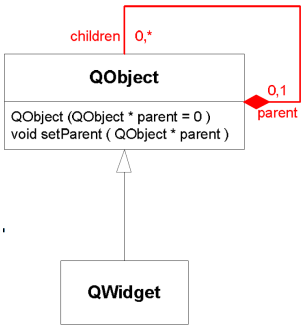
Das Koordinatensystem bei Widgets hat den Ursprung in der oberen linken Ecke (0;0). X-Werte wachsen nach rechts y-Werte nach unten

- **QVBoxLayout:** Teile vertikal anordnen
- **QHBoxLayout:** Teile horizontal anordnen
- **QGridLayout:** Teile innerhalb eines Zweidimensionalen Gitters anordnen
- **QFormLayout:** um die Teile in Form von Zeilen anzuordnen, wobei jede Zeile aus einem QLabel gefolgt von einem anderen QWidget (häufig einem QLineEdit) besteht

- **QStackedLayout:** Widgets Stapel, nur das oberste ist sichtbar

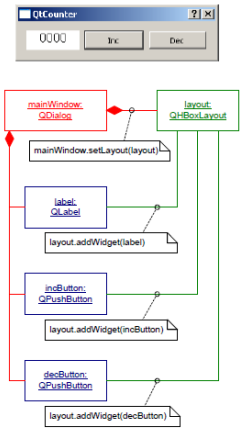
7.7 Memory Management bei Qt

- **Dynamische Speichermanagement:** Objekte werden praktisch alle auf dem Heap angelegt dies erfolgt mit **new** und **delete**
- **Speicherfreigabe:** Mit Hilfe der Klasse QObject werden Objektäume aufgebaut.
- **parent/child -Relationship:**
  Jedes QObject kann maximal ein Eltern-QObject haben
  Jedes QObject kann beliebig viel Kind-QObject haben



- Beim Löschen eines Eltern QObjekct werden automatisch deren Kinder QObject und auch deren Kinder, also dern gesamte QObject Baum gelöscht
- Realisierung Bsp. :
  - In der regel wir das Eltern-QObject im Konstruktor des Child-QObject angegeben
  - **im .h File:** LcdStopWatch(QWidget * parent = 0); //Die LcdStopWatch ist das Child- und QWidget das Eltern-Object
  - **im .cpp File:** LcdStopWatch::LcdStopWatch(QWidget *parent): QWidget(parent)
- Bsp. QLayout-Objekte und Parent Child
  - Ein *QLayout-Objekt* ist bei einem *"Parent-Widget"* zuständig für das Layout (Anordnung) von dessen *"Child-Widgets"*.
  - Das *QLayout-Objekt* ist selbst ein "Kind" vom *"Parent-Widget"*.
    - Es ist zweckmässig, sich das *QLayout-Objekt* als ein spezielles Geschwister vorzustellen, das als Kindermädchen ("Nanny") der *"Child-Widgets"* wirkt.
    - *QLayout-Objekte* haben *nie* QWidget-Objekte als eigene Kinder!
  - Das *QLayout-Objekt* informiert das *"Parent-Widget"* automatisch über die Existenz der *"Child-Widgets"*.

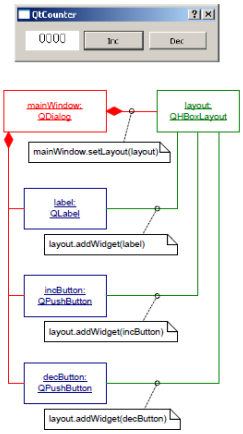
    → Das *"Parent-Widget"* muss bei den *"Child-Widgets"* nicht explizit angegeben werden!



- Bsp. QLayout-Objekte2 und Parent Child

```
int main(int argc, char *argv[])
{
   QApplication app(argc, argv);
   QDialog mainWindow;
   QHBoxLayout* layout = new QHBoxLayout;
   mainWindow.setLayout(layout);
   QLabel * label = new QLabel("0000");
   layout->addWidget(label);
   QPushButton* incButton= new QPushButton("Inc");
   layout->addWidget(incButton);
   QPushButton* decButton= new QPushButton("Dec");
   layout->addWidget(decButton);
   // ......
   mainWindow.show();
   return app.exec();
}
```

Das ist in Ordnung so! Das *"Parent-Widget"* muss bei den *"Child-Widgets"* ***nicht explizit*** angegeben werden!

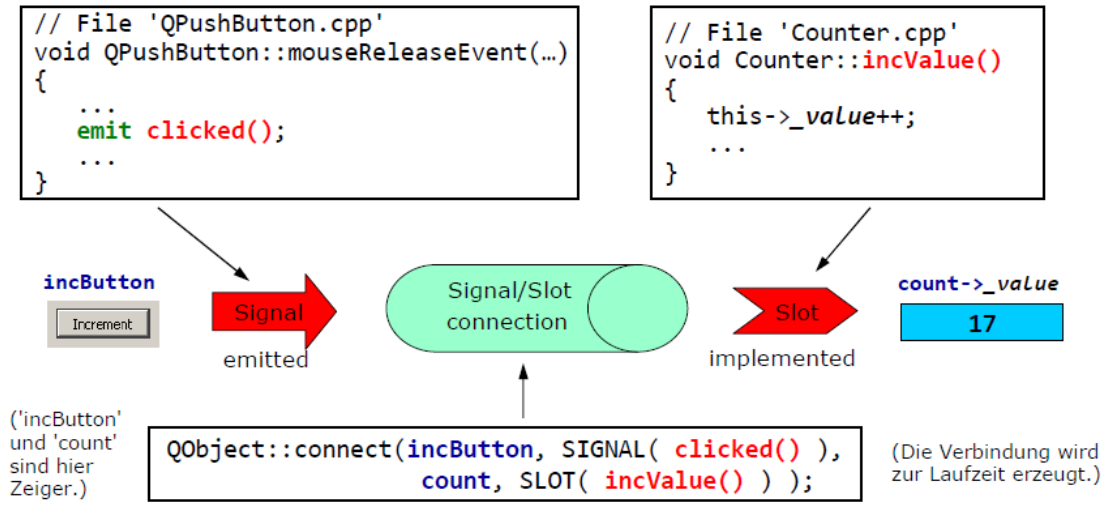


7.8 Top-Level-Windows

Sind Widgets auf der obersten Hierarchiestufe. Sie werden automatisch zu Top-Level-Windows da sie keine Eltern haben. // Bsp. "Hello World in Qtïst das QLabel ein Top-Level-Widget. // Übliche Widgets für "Top-Level-Windows"

- **QMainWindow:** Typisch für Applikations-Window (Hauptfenster)
- **QDialog:**QDialog: Popup Window Ökey", Äbbrechen", Hauptanwendung bleibt so lange blockiert "modal"
- **QWidget:** Einfaches fenster "non-modal"

7.9 Signals and Slots



➔ Beim Aufruf von `incButton->clicked()` wird `count->incValue()` aufgerufen.
⚠ Die Funktionsparameter von `clicked()` und `incValue()` müssen korrespondieren!

In den .cpp Dateien darf nur ëmitßtehen da Signals and Slots nicht zum Standard von C++ gehören

- **Signals:**
  - Sie werden nur deklariert
  - keine Zugriffsrechte werden angegeben
  - Rückgabetyp ist in der Praxis immer Void
  - Signal senden: ëmit valueChanged(17)
- **Slots:**
  - Werden deklariert und müssen auch definiert werden
  - Sind gewöhnliche Memberfunktionen die auch als solche verwendet ("normaläufgerufen) werden können und die üblichen Zugriffsrechte aufweisen können.
  - Sie können aber noch zusätzlich indirekt via den Signal/Slot-Mechanismus aufgerufen werden
- **n:m** Eine Signal kann mit mehreren Slots verbunden werden. Mehrere Signale können mit dem gleichen Slot verbunden werden.
- **Qt5 Bsp:** connect(button, &QPushButton::clicked count, &Counter::incValue);

7.10 Drawing and Painting

- **Mal-Utensilien:** QPainter,Qpen,QBrush,QFont
- **Oberflächen:** Von QpaintDevice (ist die Basisklasse aller Objekte auf die gezeichnet werden kann)abgeleitete Klassen wie QWidget, QImage, QPixmap, QPrinter
- QPainter-Objekte zeichnen auf QPaintDevice-Objekte

7.10.1 Arbeiten mit QPainter

- **Malvorgang Initialisieren:**
  - QPixmap pixmap(160,160); //Zeichenoberfläche erstellen
  - QPainter paint (&pixmap); // painter zeichnet auf pixmap
- **Zeichnungsgeräte initialisieren:**
  - QPen pen (Qt::blue, 2); painter.setPen(pen);
    QBrush brush (Qt::green); painter.setBrush(brush)
- **Zeichenoperation durchführen:**
  - painter.drawEllipse(10, 10, 140, 140); // Kreis zeichnen
    Dabei werden evt. Stift ("QPen") und Pinsel ("QBrush") benutzt!

8 Examples

8.1 CppUnit

```
// File: "tests/AuthorTest.cpp", 22.3.2012 ple, Loesung: 28.3.2012

#include <cppunit/extensions/HelperMacros.h>
#include "../src/Author.h"

class AuthorTest: public CppUnit::TestCase
// Test-Suite "AuthorTest"
{
private:
    Author *author;

    CPPUNIT_TEST_SUITE(AuthorTest);
    CPPUNIT_TEST(testCtors);
    CPPUNIT_TEST_EXCEPTION(testEmptyAuthorException, InvalidAuthor);
    CPPUNIT_TEST(testOperatorEqualEqual);
    CPPUNIT_TEST_SUITE_END();
public:
    void testCtors()
    {
        author = new Author("Samuel", "Beckett");
        CPPUNIT_ASSERT(author->getFirstName() == "Samuel");
        CPPUNIT_ASSERT(author->getLastName() == "Beckett");
        delete author;
    }
    void testEmptyAuthorException()
    {
        author = new Author("", "");
        delete author;
    }
    void testOperatorEqualEqual()
    {
        Author a1 ("Max", "Frisch");
        Author a2 ("Johann", "Frisch");
        Author a3 ("Friedrich", "Glauser");
        Author a4 ("Friedrich", "Duerrenmatt");
        CPPUNIT_ASSERT(   a1 == a1  );  // beide Namen gleich
        CPPUNIT_ASSERT(! (a3 == a4) );  // nur Vorname gleich
        CPPUNIT_ASSERT(! (a1 == a2) );  // nur Nachname gleich
        CPPUNIT_ASSERT(! (a1 == a3)) ;  // beide Name ungleich
    }

};
// Alle Tests von  TestAuthor  in der default-registry registrieren:
CPPUNIT_TEST_SUITE_REGISTRATION(AuthorTest);
```

main Programm für CppUnit Test des AuthorTest

```
// File  testmain.cpp , 18.3.2104 ple

#include <cppunit/BriefTestProgressListener.h>
#include <cppunit/CompilerOutputter.h>
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/TestResult.h>
#include <cppunit/TestResultCollector.h>
#include <cppunit/TestRunner.h>
#include <iostream>


int main( int argc, char* argv[] )
{
  // Create the event manager and test controller:
  CppUnit::TestResult controller;

  // Add a listener that colllects test result:
  CppUnit::TestResultCollector result;
  controller.addListener( &result );

  // Add a listener that print some info as test run:
  CppUnit::BriefTestProgressListener progress;
  controller.addListener( &progress );

  // Add the top suite in the registry to the test runner
  CppUnit::TestRunner runner;
  runner.addTest( CppUnit::TestFactoryRegistry::getRegistry().makeTest() );

  // Run the tests:
  runner.run( controller );
  std::cout << "----  runer.run()  done. " << std::endl;

  // Print test in a compiler compatible format.
  CppUnit::CompilerOutputter outputter( &result, std::cout );
  outputter.write();

  return result.wasSuccessful() ? 0 : 1;
}
```

8.2 Bsp. Chusep Ohne CppUnit nur mit assert.h testMyDate.cpp

```
//=============================================================================
// Modul PmSwe, Uebung 2, Aufgaben 1-3: Klasse "MyDate" mit Tests
// File "testMyDate.cpp", 20.2.2013, 4.6.2012, H. Pletscher
//=============================================================================

#include <iostream>
#include <string>
```

```
#include <assert.h>

#include "MyDate.h"

using namespace std;
//============================================================================
bool equalDates (const MyDate & date, int day, int month, int year)
// Hilfsfunktion, nur zum Testen. Lliefert  true  falls beide Daten gleich
// (sonst  false ).  bool _isValid  wird nicht beruecksichtigt.
{
    if ( date.getDay()   != day   ) return false;
    if ( date.getMonth() != month ) return false;
    if ( date.getYear()  != year  ) return false;
    return true;
}
//============================================================================
void testDefaultCtor()
{
    cout << "--- Test of Default-Constructor:  MyDate d;        :" << flush;
    MyDate d1;
    assert (d1.getDay() == 0);
    assert (d1.getMonth() == 0);
    assert (d1.getYear() == 0);
    cout << " -- OK" << endl;
}
//----------------------------------------------------------------------------
void testCtorWithValues()
{
    cout << "--- Test of Constructor:  MyDate d1 (25, 2, 2010); :" << flush;
    MyDate d1 (25, 2, 2010);
    assert (d1.getDay() == 25);
    assert (d1.getMonth() == 2);
    assert (d1.getYear() == 2010);
    cout << " -- OK" << endl;
}
//----------------------------------------------------------------------------
void testCopyCtor()
{
    cout << "--- Test of Copy-Constructor:  MyDate d2 = d1;     :" << flush;
    MyDate d1 (25, 2, 2010);
    MyDate d2 = d1;
    assert ( d2.getDay()   == d1.getDay() );
    assert ( d2.getMonth() == d1.getMonth() );
    assert ( d2.getYear()  == d1.getYear() );
    cout << " -- OK" << endl;
}
//----------------------------------------------------------------------------
void testAssignment()
{
    cout << "--- Test of Assignment, op=():  d3 = d2 = d1;      :" << flush;
    MyDate d1 (25, 2, 2010);
```



```
    MyDate d2; MyDate d3;
    d3 = d2 = d1;
    assert ( d2.getDay()   == d1.getDay() );
    assert ( d2.getMonth() == d1.getMonth() );
    assert ( d2.getYear()  == d1.getYear() );
    assert ( d3.getDay()   == d2.getDay() );
    assert ( d3.getMonth() == d2.getMonth() );
    assert ( d3.getYear()  == d2.getYear() );
    cout << " -- OK" << endl;
}
//=============================================================================
void testIsValid()
{
    cout << "--- Test of  isValid()                                  :" << flush;
    // valid dates:
    {   MyDate date1(25, 2, 2010);  assert ( date1.isValid() );     }
    {   MyDate date1(1, 1, 2000);   assert ( date1.isValid() );     }
    {   MyDate date1(31, 1, 2000);  assert ( date1.isValid() );     }
    {   MyDate date1(31, 3, 2000);  assert ( date1.isValid() );     }
    {   MyDate date1(30, 4, 2000);  assert ( date1.isValid() );     }
    {   MyDate date1(29, 2, 2000);  assert ( date1.isValid() );     }
    {   MyDate date1(28, 2, 2001);  assert ( date1.isValid() );     }
    {   MyDate date1(29, 2, 2004);  assert ( date1.isValid() );     }

    // invalid dates:
    {   MyDate date1(0, 0, 0);      assert ( !date1.isValid() );    }
    {   MyDate date1(0, 5, 2000);   assert ( !date1.isValid() );    }
    {   MyDate date1(15, 0, 2000);  assert ( !date1.isValid() );    }
    {   MyDate date1(15, 12, -1);   assert ( !date1.isValid() );    }
    {   MyDate date1(31, 4, 2000);  assert ( !date1.isValid() );    }
    {   MyDate date1(29, 2, 2100);  assert ( !date1.isValid() );    }
    {   MyDate date1(32,12, 2000);  assert ( !date1.isValid() );    }

    cout << " -- OK" << endl;
}
//-----------------------------------------------------------------------------
void testIsLeapYear()
{
    cout << "--- Test of  isLeapYear()                               :" << flush;
    {   MyDate d1 (25, 2, 2010);   assert ( d1.isLeapYear() == 0);  }
    {   MyDate d1 (25, 2, 2008);   assert ( d1.isLeapYear() != 0);  }
    {   MyDate d1 (26, 2, 1600);   assert ( d1.isLeapYear() != 0);  }
    {   MyDate d1 (26, 2, 1700);   assert ( d1.isLeapYear() == 0);  }
    {   MyDate d1 (26, 2, 1701);   assert ( d1.isLeapYear() == 0);  }
    {   MyDate d1 (26, 2, 1702);   assert ( d1.isLeapYear() == 0);  }
    {   MyDate d1 (26, 2, 1703);   assert ( d1.isLeapYear() == 0);  }
    {   MyDate d1 (26, 2, 1704);   assert ( d1.isLeapYear() != 0);  }
    cout << " -- OK" << endl;
}
//=============================================================================
void testEqualComparison()
```



```
{
    cout << "--- Test of Comparison, op==():  d2 == d1;          :" << flush;
    MyDate d1 (25, 2, 2010);
    MyDate d2 (25, 2, 2010);
    MyDate d3 (26, 2, 2010);
    MyDate d4 (26, 3, 2010);
    MyDate d5 (26, 3, 2011);
    MyDate d6 (25, 3, 2010);
    MyDate d7 (25, 3, 2011);
    MyDate d8 (25, 2, 2011);
    assert ( (d1 == d2) != 0 );
    assert ( (d1 == d3) == 0 );
    assert ( (d1 == d4) == 0 );
    assert ( (d1 == d5) == 0 );
    assert ( (d1 == d6) == 0 );
    assert ( (d1 == d7) == 0 );
    assert ( (d1 == d8) == 0 );

    // ungueltige Daten
    MyDate date1(15, 10, 1986);
    MyDate date2;
    assert( !(date2 == date1) );
    assert( !(date1 == date2) );

    // fragliche Tests mit ungueltigen Daten
    assert( date2 == date2 );                // ???
    assert( MyDate() == MyDate() );          // ???
    assert(date2 == MyDate(15, 0, 1900));    // ???

    cout << " -- OK" << endl;
}
//-----------------------------------------------------------------------------
void testNotEqualComparison()
{
    cout << "--- Test of Comparison, op!=():  d2 != d1;          :" << flush;

    MyDate date1(15, 10, 1986);
    assert( (date1 != date1) == false );
    assert( date1 != MyDate(16, 10, 1986) );

    // ungueltige Daten:
    MyDate date2;
    assert(date2 != date1);
    assert(date1 != date2);

    // fragliche Tests mit ungueltigen Daten:
    assert( !(date2 != date2) );                // ???
    assert( !(MyDate() != MyDate()) );          // ???
    assert( !(date2 != MyDate(15, 0, 1900) ));  // ???

    cout << " -- OK" << endl;
```

```
}
//------------------------------------------------------------------------------
void testLessThanComparison()
{
    cout << "--- Test of Comparison, op< ():  d2 < d1;          :" << flush;

    MyDate date1(15, 10, 1986);
    assert( (date1 < date1) == false );
    assert( date1 < MyDate(16, 10, 1986) );

    // ungueltige Daten
    MyDate date2;
    assert( !(date2 < date1) );
    assert( !(date1 < date2) );

    // fragliche Tests mit ungueltigen Daten
    assert( !(date2 < date2) );          // ???
    assert( !(MyDate() < MyDate()) );       // ???
    assert( !(date2 < MyDate(15, 0, 1900) ));   // ???

    cout << " -- OK" << endl;
}
//==============================================================================
void testAddDays()
{
    cout << "--- Test of  addDays()                              :" << flush;
    {
        MyDate d1 (28, 2, 2010); // kein Schaltjahr
        //assert ( d1.getDaysInMonth() == 28);
        MyDate d2, d3;
        d2 = d1.addDays(1);
        assert ( equalDates( d2,  1,3,2010) );

        d3 = d2.addDays(-1);
        assert ( equalDates( d3,  28,2,2010) );
    }
    {
        MyDate date1(1, 1, 1999);
        assert( equalDates( date1.addDays(1), 2, 1, 1999 ) );
        assert( equalDates( date1, 1, 1, 1999 ) );

        assert( equalDates( date1.addDays(0),     1, 1, 1999 ) );
        assert( equalDates( date1.addDays(30),   31, 1, 1999 ) );
        assert( equalDates( date1.addDays(31),    1, 2, 1999 ) );
        assert( equalDates( date1.addDays(-31),   1,12, 1998 ) );
        assert( equalDates( date1.addDays(31+28), 1, 3, 1999 ) );
    }
    {
        MyDate date1(31, 12, 1999);
        assert( equalDates( date1.addDays(1),     1, 1, 2000 ) );
    }
```



```
    {
        MyDate date1(1, 3, 2008);
        assert( equalDates( date1.addDays(-1),   29, 2, 2008 ) );
        assert( equalDates( date1.addDays(-366),  1, 3, 2007 ) );
        assert( equalDates( date1.addDays(-367), 28, 2, 2007 ) );
    }
    {
        MyDate date1(28, 2, 2000);
        assert( equalDates( date1.addDays(2),     1, 3, 2000 ) );
    }
    cout << " -- OK" << endl;
}
//-----------------------------------------------------------------------------
void testDaysTil()
{
    cout << "--- Test of  daysTil()                                  :" << flush;

    MyDate date1(5, 1, 2010);
    assert(date1.daysTil(date1) == 0);
    assert(date1.daysTil(MyDate(6, 1, 2010)) == 1);
    assert(date1.daysTil(MyDate(4, 1, 2010)) == -1);
    assert(date1.daysTil(MyDate(15, 1, 2010)) == 10);
    assert(date1.daysTil(MyDate(30, 12, 2009)) == -6);

    // fraglicher Test mit ungueltigem Datum:
    MyDate date2;
    assert( date1.daysTil( date2 )  == 0);  // ???

    cout << " -- OK" << endl;
}
//=============================================================================
int main(int argc, char **argv)
{
    testDefaultCtor();
    testCtorWithValues();
    testCopyCtor();
    testAssignment();
    cout << endl;

    testIsValid();
    testIsLeapYear();
    cout << endl;

    testEqualComparison();
    testNotEqualComparison();
    testLessThanComparison();
    cout << endl;

    testAddDays();
    testDaysTil();
}
```



```
//============================================================================
```

8.3 MyDate.h

```
#ifndef MYDATE_H_
#define MYDATE_H_
//============================================================================
// Modul PmSwe, Uebung 2, Aufgaben 1-3: Klasse "MyDate" mit Tests
// File "MyDate.h", 21.2.2013, 4.6.2012, H. Pletscher
//============================================================================

class MyDate
{
private:
    int _day;    // 1..31
    int _month;  // 1..12
    int _year;   // z.B. 2010
    mutable bool _isValid; // interner Cache fuer Resultat der Gueltigkeits-
                           // pruefung ( mutable  wegen  isValid() const ).
public:
    MyDate();
    // erzeugt neues ungueltiges Datum-Objekt
    MyDate(int day, int month, int year);
    // erzeugt neues initialisiertes Datum-Objekt
    MyDate (const MyDate & x);
    // Copy-Constructor
    MyDate & operator= (const MyDate & x);
    // Zuweisungsoperator
    bool isLeapYear() const;
    // liefert true falls Schaltjahr
    bool isValid() const;
    // prueft ob akt. Datum gueltig, d.h. plausibel (nicht 31.2.2009)

    MyDate addDays(int n) const;
    // liefert das Datum, das n Tage nach dem aktuellen Datum liegt.
    // n darf auch negativ sein.
    //-------------------------------------------------------------------------
    // Getter-Methoden:
    int getDay() const
    {
        return _day;
    }
    int getMonth() const
    {
        return _month;
    }
    int getYear() const
    {
        return _year;
    }
    //-------------------------------------------------------------------------
```

```
    // **** Uebung 2:

    // Vergleichsoperatoren zum Vergleichen der Datumswerte.
    bool operator==(const MyDate& other) const;
    bool operator!=(const MyDate& other) const;
    bool operator<(const MyDate& other) const;
    bool operator<=(const MyDate& other) const;
    bool operator>(const MyDate& other) const;
    bool operator>=(const MyDate& other) const;

    int daysTil(const MyDate& til) const;
    // liefert Anzahl Tage bis zum angegebenen Datum (kann auch negativ sein)


    //-------------------------------------------------------------------
    // Private Hilfsfunktionen
private:
    MyDate & incDay();
    // inkrementiert das aktuelle Datum um einen Tag und gibt es dann zurueck↩
       .

    MyDate & decDay();
    // dekrementiert das aktuelle Datum um einen Tag und gibt es dann zurueck↩
       .

    int getDaysInMonth() const;
    // liefert die Anzahl Tage des aktuellen Monats im aktuellen Jahr.

    void print() const;


//-------------------------------------------------------------------
};
#endif /* MYDATE_H_ */
```

8.4 MyDate.cpp

```
//===================================================================
// Modul PmSwe, Uebung 2, Aufgaben 1-3:
//   Klasse "MyDate": neu: Vergleichsoperatoren, Differenz zwischen zwei
//   Datumswerten ( daysTil() ),  addDays(x)  auch fuer x < 0.

// File "MyDate.cpp", 4.6.2012, H. Pletscher
//===================================================================
#include <iostream>

#include "MyDate.h"

using namespace std;
//-------------------------------------------------------------------
MyDate::MyDate()
```



```
// erzeugt (ungueltiges) Datum-Objekt
:   _day(0), _month(0), _year(0), _isValid(false)
{
}
//--------------------------------------------------------------------------
MyDate::MyDate(int day, int month, int year)
// erzeugt Datum-Objekt aus int-Werten.
:   _day(day), _month(month), _year(year), _isValid(false)
{
}
//--------------------------------------------------------------------------
MyDate::MyDate(const MyDate & x)
// Copy-Constructor.
:_day(x._day), _month(x._month), _year(x._year), _isValid(x._isValid)
{
}
//--------------------------------------------------------------------------
MyDate & MyDate::operator=(const MyDate & x)
// Zuweisungsoperator.
{
    if (this != &x)
    {
        _day = x._day; _month = x._month; _year = x._year;
        _isValid = x._isValid;
    }
    return *this;
}
//--------------------------------------------------------------------------
bool MyDate::isLeapYear() const
// liefert  true  falls Schaltjahr, sonst  false .
{
    // gewoehnliches Schaltjahr, aber kein Jahrhunderjahr oder wenn, ein Jahr
    // durch 400 teilbar.
    return _year % 4 == 0 && (_year % 100 != 0 || _year % 400 == 0);
}
//--------------------------------------------------------------------------
bool MyDate::isValid() const
// prueft ob akt. Datum gueltig, d.h. plausibel (nicht 31.2.2009).
{
    static int daysInMonth[2][13] =
    {       // Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
        {   0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }, // normal
        {   0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }, // Leap-Year
    };

    if (_isValid) return true;
    if (_day < 1 || _day > 31) return false;
    if (_month < 1 || _month > 12) return false;
//  if (_year <= 1582) return false;  // (Gregorianischer Kalender)
    if (_year <= 0) return false;
```



```
    if ( _day > daysInMonth [!!isLeapYear()] [_month] ) return false;
    _isValid = true;
    return true;
}
//=========================================================================
MyDate & MyDate::incDay()
// inkrementiert das aktuelle Datum um einen Tag und gibt es dann zurueck.
{
    if (_day < getDaysInMonth() ) _day++;
    else
    {
        _day = 1;
        if (_month < 12) _month++;
        else
        {
            _month= 1;
            _year++;
        }
    }
    return *this;
}
//-------------------------------------------------------------------------
MyDate & MyDate::decDay()
// dekrementiert das aktuelle Datum um einen Tag und gibt es dann zurueck.
{
    if (_day > 1) _day--;
    else
    {
        if (_month > 1) _month--;
        else
        {
            _month= 12;
            _year--;
            if (_year <= 0) _isValid = false;
        }
        _day = getDaysInMonth();
    }
    return *this;
}
//-------------------------------------------------------------------------
MyDate MyDate::addDays(int n) const
// liefert das Datum, das n Tage nach dem aktuellen Datum liegt.
// n kann >= 0 oder < 0 sein.
// Hinweis: Implementierung ist noch nicht effizient.

{
    MyDate tmp = *this;
    while (n > 0)
    {
        tmp.incDay(); n--;
    }
```

```
    while (n < 0)
    {
        tmp.decDay(); n++;
    }
    return tmp;
}
//============================================================================
// Vergleichsoperatoren:

bool MyDate::operator== (const MyDate & d) const
// Test auf Gleichheit (beide Werte ungueltig gelten als gleich).
{
    if ( !this->isValid() && !d.isValid() ) return true;
    return _day == d._day && _month == d._month && _year == d._year;
}

bool MyDate::operator!= (const MyDate & d) const
// Test auf Ungleichheit (beide Werte ungueltig, gelten als gleich).
{
    return ! (*this == d);
}

bool MyDate::operator< (const MyDate & other) const
// Test ob kleiner (ein oder beide Werte ungueltig ergeben false).
{
    if ( !this->isValid() || !other.isValid() ) return false;

    if (_year < other._year)   return true;
    if (_year > other._year)   return false;
    if (_month < other._month) return true;
    if (_month > other._month) return false;
    return _day < other._day;
}

bool MyDate::operator<= (const MyDate & other) const
// Test ob kleiner oder gleich.
{
    return  (*this < other) || (*this == other);
}
bool MyDate::operator> (const MyDate & other) const
// Test ob groesser (ein oder beide Werte ungueltig ergeben false).
{
    return  (other < *this);
}
bool MyDate::operator>= (const MyDate & other) const
// Test ob kleiner oder gleich.
{
    return  (other <= *this);
}
//----------------------------------------------------------------------------
int MyDate::getDaysInMonth() const
```



```
// Hilfsfunktion:
// liefert die Anzahl Tage des aktuellen Monats im aktuellen Jahr.
{
    static int daysInMonth[2][13] =
    {       // Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
        {   0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }, // normal
        {   0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }, // Leap-Year
    };

    return  daysInMonth [!!isLeapYear()] [_month] ;

}
//-----------------------------------------------------------------------------
int MyDate::daysTil(const MyDate & til) const
// liefert Anzahl Tage bis zum angegebenen Datum (kann auch negativ sein).
// Hinweis: Implementierung ist noch nicht effizient.
{
    MyDate temp(til);
    int n = 0;

    while (temp > *this)
    {
        n++;
        temp.decDay();
    }
    while (temp < *this)
    {
        n--;
        temp.incDay();
    }
    return n;
}
//-----------------------------------------------------------------------------
void MyDate::print() const
// Hilfsfunktion
{   cout << _day << "." << _month << "." << _year;
}
```

8.5 Doxygen

8.5.1 mainpage.h

```
/**
 * @mainpage
 * @brief Hilfsklassen, die häufig gewünschte Fähigkeiten in ↩
    objektorientierter Form anbieten.
 *
 * Momentan werden erst zwei Klassen angeboten
 *
 * @see Keyboard Hilft beim Einlesen von der Tastatur.
 * @see StopWatch Stoppuhr.
 * @author wb / pl
 */
```

8.5.2 Keyboard.h

```
/**
 * @class Keyboard
 * @brief Verfügbarkeit von Eingaben prüfen und Zeichen lesen
 *
 * Erlaubt es abzufragen, ob ein Zeichen eingegeben wurde
 * (Funktion "keypressed()") und um Einzelzeichen (ohne Eingabe von
 *  Return ) einzulesen.
 *
 * Für Cygwin und Unix-Umgebungen (Linux etc.).
 *
 * Nur mit Cygwin getestet.
 *
 * Entspricht in etwa  conio.h  von Microsoft.
 * @file Keyboard.h
 * @author H. Pletscher
 * @date  5.4.2010
 * @version 1.0
 */

#ifndef KEYBOARD_H_
#define KEYBOARD_H_

class Keyboard
{
private:
    static int _anzObjs;
public:
    /**
    * Pseudo-Constructor: Kapselt die Tastatur, so dass mit Objektmethoden
    * darauf zugegriffen werden kann.
    *
    * Die Tastatur ist eine Ressource, die nicht angelegt werden kann.
    * Der erste Konstruktor-Aufruf initialisiert die Tastatur, alle weiteren
```

```
    * werden nur registriert (Keyboard-Usage-Zähler, für die korrekte ↩
       Funktion
    * des Destruktors).
    */
    Keyboard();

    /**
    * Destructor: Der Keyboard-Usage-Zähler wird reduziert, und wenn die
    * Tastatur nicht mehr gebraucht wird, wird die Tastatur "geschlossen".
    */
    ~Keyboard();

    /**
    * Prüft, ob eine Taste gedruckt wurde, so dass getch() ohne zu blocken
    * aufgerufen werden kann.
    * @return 1 wenn Zeichen verfügbar, 0 wenn keine Taste gedrückt wurde,
    * -1 bei Fehler.
    */
    bool keypressed();

    /**
    * Liest das nächste zeichen.
    * @return Code des Zeichen.
    */
    int getch();
};

#endif /* KEYBOARD_H_ */
```

8.5.3 Keyboard.cpp

```
/**
 * Klasse  Keyboard , um abzufragen, ob ein Zeichen eingegeben wurde
 * (Funktion "keypressed()") und um Einzelzeichen (ohne Eingabe von
 *  Return ) einzulesen.
 * Fuer Cygwin und Unix-Umgebungen (Linux etc.).
 * Nur mit Cygwin getestet.
 * Entspricht in etwa  conio.h  von Microsoft.
 * @file Keyboard.cpp
 * @author H. Pletscher
 * @date  5.4.201
 * @version 1.0
 */
//=====================================================================
#include <assert.h>
#include <stdio.h>
#include <stdlib.h>
#include <termios.h>
#include <unistd.h>
#include <sys/select.h>
```

```
#include "Keyboard.h"

//=====================================================================
static void init_keyboard();
static void close_keyboard();
static int kbhit();
static int readch();
//=====================================================================
int Keyboard::_anzObjs;
//---------------------------------------------------------------------
Keyboard::Keyboard()
{
    if (_anzObjs == 0) init_keyboard();
    _anzObjs++;
}
//---------------------------------------------------------------------
Keyboard::~Keyboard()
{
    assert (_anzObjs > 0);
    _anzObjs--;
    if (_anzObjs == 0) close_keyboard();
}
//---------------------------------------------------------------------
bool Keyboard::keypressed()
{
    return kbhit();
}
//---------------------------------------------------------------------
int Keyboard::getch()
{
    return readch();
}
//=====================================================================
static struct termios initial_settings, new_settings;
//---------------------------------------------------------------------
static void init_keyboard()
{
    tcgetattr(0, &initial_settings);
    new_settings = initial_settings;
    new_settings.c_lflag &= ~ICANON;   // disable canonical mode
    new_settings.c_lflag &= ~ECHO;
    new_settings.c_lflag &= ~ISIG;
    new_settings.c_cc[VMIN] = 1;
    new_settings.c_cc[VTIME] = 0;
    tcsetattr(0, TCSANOW, &new_settings);
}
//---------------------------------------------------------------------
static void close_keyboard()
{
    tcsetattr(0, TCSANOW, &initial_settings);
}
```



```
//-------------------------------------------------------------------
static int kbhit(void)
{
    struct timeval tv;
    fd_set read_fd;

    tv.tv_sec=0; tv.tv_usec=0;
    FD_ZERO(&read_fd);
    FD_SET(0, &read_fd);
    if(select(1, &read_fd,NULL, NULL, &tv) == -1) return 0; // Error

    if(FD_ISSET(0, &read_fd)) return 1;
    return 0;
}
//-------------------------------------------------------------------
static int readch()
{
    char ch;
    read(0, &ch, 1);
    return ch;
}
//-------------------------------------------------------------------
```

8.6 Qt

8.7 Hello World in Qt

```
#include <QWidgets>
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QLabel label ("Hello Qt World!");
    label.setAlignment(Qt::AlignCenter);
    label.resize(250, 150); // w, h
    label.setWindowTitle("My first Qt-Program");
    label.show();
    return app.exec();
}
```

8.7.1 main.cpp

```
// Modul: PmSwE, UW 13, N-Uebung 9 (Qt 4): Aufgabe 1: Temperatur-Umrechnung
// File: main.cpp, 20.5.2010, H. Pletscher
//-----------------------------------------------------------------------
#include <QtGui>

#include "TemperaturWidget.h"
//-----------------------------------------------------------------------
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    TemperaturWidget mainWindow;

    // **** Objekt-Baum zur Memory-Management Kontrolle ausgeben:
    //      (natürlich nicht bei  richtigen  Programmen)
    //mainWindow.dumpObjectTree();

    mainWindow.show();
    return app.exec();
}
```

8.8 Manuelles Layout

```
#include <QWidgets>
int main(int argc, char *argv[])
{
    QApplication app(argc, argv);
    QLabel label ("Hello Qt World!");
    label.setAlignment(Qt::AlignCenter);
    //Manuelles Layout
    void QWidgets::setFixedSize(int width, int height);
    void QWidgets::setGeometry(int x, int y, int w, int h);
```

```
    void QWidgets:resize (int w, int h);
    void  QWidgets::move (int x, int y);

    // Bsp mit resize und dem QLabel Objekt namens Label
    label.resize(250, 150); // w, h

    label.setWindowTitle("My first Qt-Program");
    label.show();
    return app.exec();
}
```

8.9 Manuelles Layout2

```
int main(int argc, char *argv[])
{
QApplication app(argc, argv);
QDialog mainWindow;
mainWindow.resize(230, 75); // w, h
QLabel * label = new QLabel("0000", &mainWindow);
label->setGeometry(20, 20, 100, 35); // x, y, w, h
QPushButton * incButton = new QPushButton ("Inc");
incButton->setParent( &mainWindow );
incButton->setGeometry(150, 10, 55, 25); // x, y, w, h
QPushButton * decButton = new QPushButton ("Dec", &mainWindow);
decButton->move(150, 40); // x, y (statt  setGeometry() )
decButton->resize(55, 25); // w, h (statt  setGeometry() )
// ......
mainWindow.show();
return app.exec();
}
```

8.10 Zeichnen

```
#include <QtWidgets>
int main(int argc, char *argv[])
{
QApplication app(argc, argv);
// Zeichenoberflaeche:
QPixmap pixmap (200, 200);
pixmap.fill( QColor(Qt::white) );
QPainter painter (&pixmap); //  painter  der auf  pixmap  zeichnet
painter.setRenderHint(QPainter::Antialiasing, true); // "schoen" z.
QPen pen (Qt::blue, 2); // Zeichenstift: blau, Breite= 2
painter.setPen(pen);
QBrush brush (Qt::green); // Pinsel: gruen
painter.setBrush(brush);
painter.drawEllipse(10, 10, 180, 180); // Kreis zeichnen
// In einem QLabel anzeigen:
QLabel label;
label.setPixmap( pixmap );
```

```
label.show();
return app.exec();
}
```

8.10.1 TemperaturWidget.h

```
#ifndef TEMPERATURWIDGET_H
#define TEMPERATURWIDGET_H
//-----------------------------------------------------------------------------
// Modul: PmSwE, UW 13, N-Uebung 9 (Qt 4): Aufgabe 1: Temperaturumrechnung
// File: TemperaturWidget.h, 20.5.2010, H. Pletscher
//-----------------------------------------------------------------------------
#include <QtGui>

//-----------------------------------------------------------------------------
class TemperaturWidget : public QWidget
{
private:
    Q_OBJECT

private:    // **** GUI-Stuff:
    QGridLayout * mainLayout;
    QLabel *label1, *label2, *label3;
    QLineEdit *eingabeLineEdit, *resultLineEdit;
    QGroupBox *groupBox;

    QHBoxLayout * boxLayout;
    QRadioButton * tCelsiusButton, * tFahrenheitButton;

    QPushButton * rechneButton;

public:
    TemperaturWidget(QWidget * parent = 0);

private slots:
    void rechne();

};
//-----------------------------------------------------------------------------
#endif // MYDIALOG_H
```

8.10.2 TemperaturWidget.cpp

```
// Modul: PmSwE, UW 13, N-Uebung 9 (Qt 4): Aufgabe 1: Temperatur-Umrechnung
// File: TemperaturWidget.cpp, 20.5.2010, H. Pletscher
//-----------------------------------------------------------------------------
#include <QtGui>

#include "TemperaturWidget.h"
//-----------------------------------------------------------------------------
```

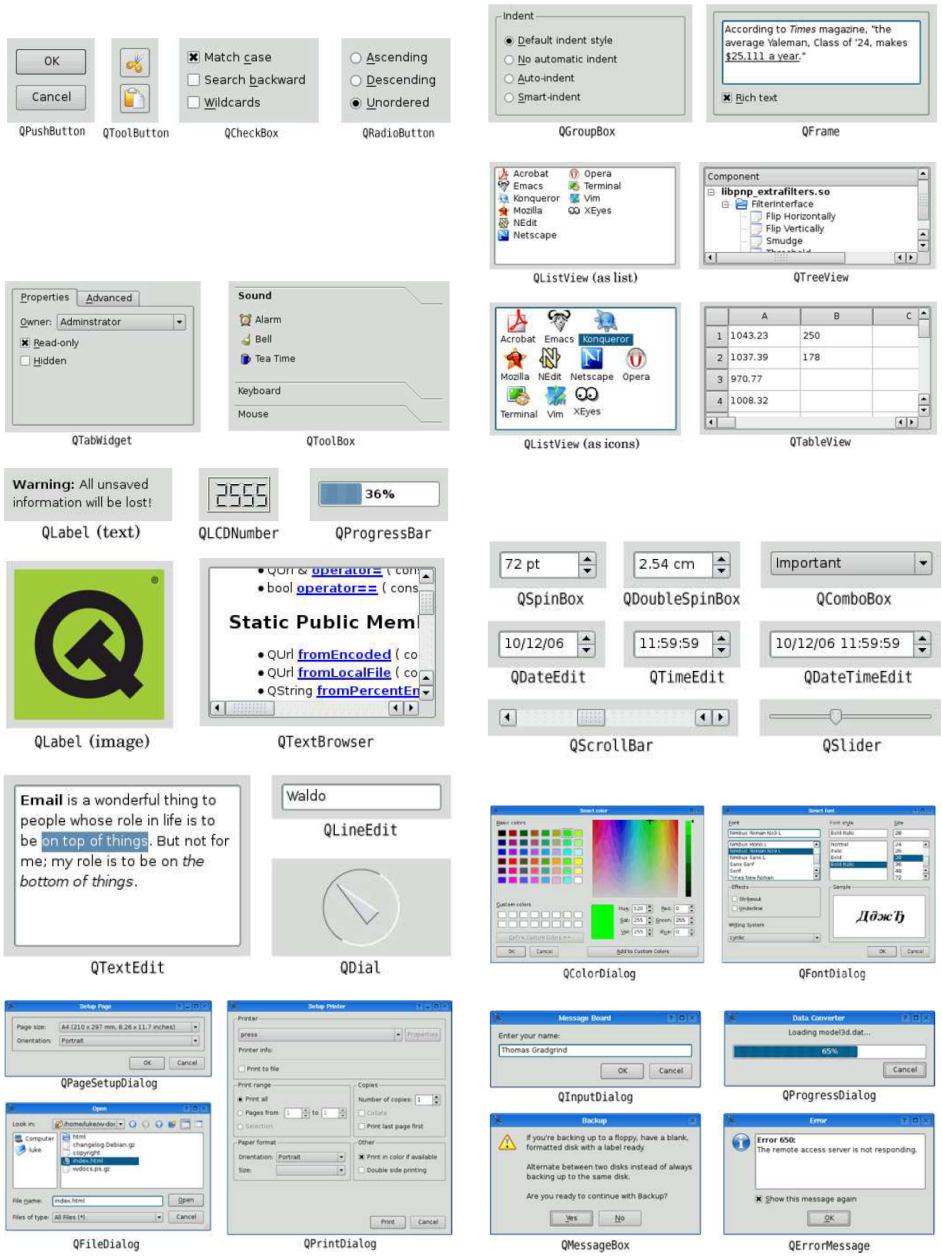
```
TemperaturWidget::TemperaturWidget(QWidget * parent)
    : QWidget(parent)
{
    // **** Darstellung festlegen:
    mainLayout = new QGridLayout;
    this->setLayout(mainLayout);

    // Obere Zeile:
    label1 = new QLabel("<b>Temperatur Umrechnung:</b>");
    mainLayout->addWidget( label1, 0, 0, 1, 3 );

    label1->setAlignment(Qt::AlignLeft);
    label1->setAlignment(Qt::AlignVCenter);
    label1->setAutoFillBackground(true);
    //label1->setStyleSheet("background-color: white;");
    label1->setFont( QFont("Arial", 12) );

    // Mittlere Zeile:
    label2 = new QLabel("Eingabe:");
    mainLayout->addWidget( label2, 1, 0 );
    eingabeLineEdit = new QLineEdit();
    mainLayout->addWidget( eingabeLineEdit, 1, 1 );
    boxLayout = new QHBoxLayout;
    groupBox = new QGroupBox;
    groupBox->setLayout(boxLayout);
    tCelsiusButton = new QRadioButton("°C", groupBox);
    tCelsiusButton->setChecked(true);
    boxLayout->addWidget(tCelsiusButton);
    tFahrenheitButton = new QRadioButton("°F", groupBox);
    boxLayout->addWidget(tFahrenheitButton);
    mainLayout->addWidget(groupBox, 1, 2);

    // Untere Zeile:
    label3 = new QLabel("Ergebnis= ");
    mainLayout->addWidget( label3, 2, 0 );
    resultLineEdit = new QLineEdit();
    resultLineEdit->setReadOnly(true);
    mainLayout->addWidget( resultLineEdit, 2, 1 );

    rechneButton = new QPushButton("Rechne");
    mainLayout->addWidget( rechneButton, 2, 2 );

    // **** Interaktives Verhalten initialisieren:
    QObject::connect(rechneButton, SIGNAL(clicked()),
                     this, SLOT(rechne()) );
    QObject::connect(eingabeLineEdit, SIGNAL(editingFinished()),
                     this, SLOT(rechne()) );
}
//-----------------------------------------------------------------------------
void TemperaturWidget::rechne()
// Slot: wird aufgerufen falls Button  Rechne  betaetigt.
```



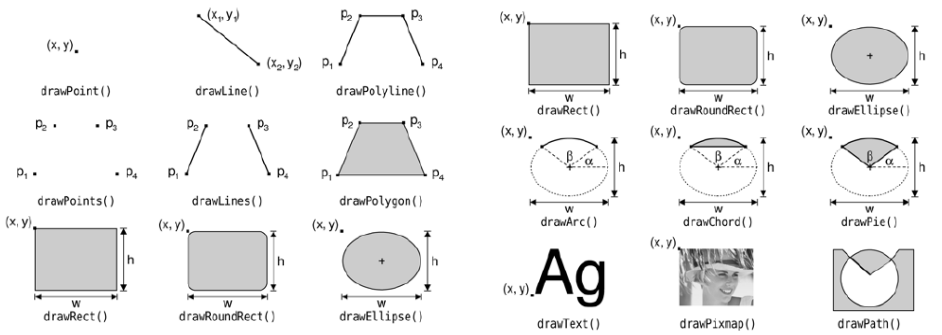
```
{
    QString s = eingabeLineEdit->text();
    bool okay;
    double x = s.toDouble(&okay);
    if (!okay)
    {
        resultLineEdit->setText("Eingabe falsch");
        eingabeLineEdit->setText("0.0");
    }
    else
    {
        double res = 0;
        if (tCelsiusButton->isChecked() )
        {   // Umrechnung Celsius --> Fahrenheit
            res = x * 1.8 +32;
        }
        else
        {   // Umrechnung  Fahrenheit --> Celsius
            res = (x - 32) * 5.0 / 9.0;
        }
        QString r = QString("%1").arg(res, 0,  g , 5 );
        resultLineEdit->setText(r);
    }
}
//-------------------------------------------------------------------------
```

9 QtBibliothek

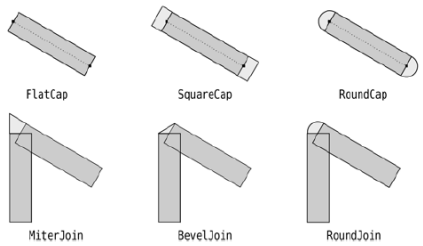
9.1 Widgets

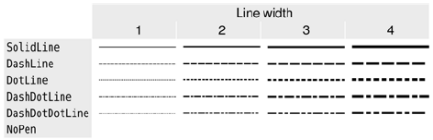


9.2 draw.. Methoden



9.3 QPen Arten





9.4 QBrush