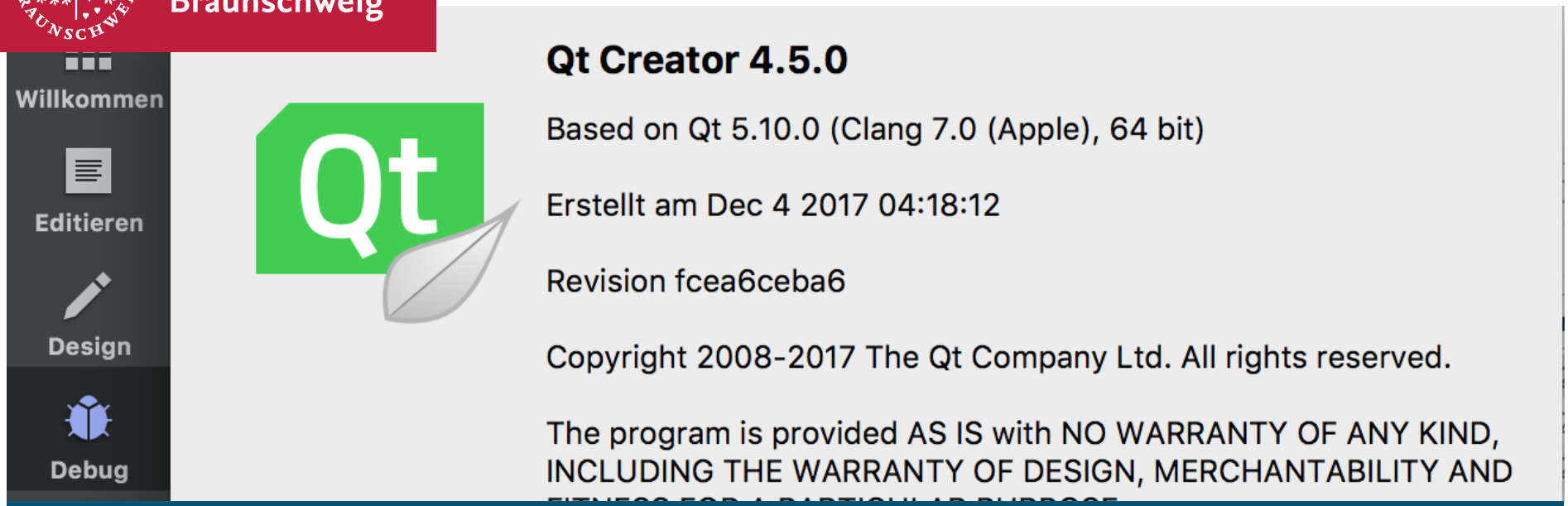
Technische Universität Braunschweig

Institut für Flugführung



Qt und Debugging

Prof. Dr.-Ing. Peter Hecker, Dipl.-Ing. Paul Frost, Andreas Dekiert M. Sc.,
17. Juni 2019

Agenda

09. April	Einführung
16. April	Softwareprojektmanagement
23. April	Entwicklungstools
30. April	GitHub
07. Mai	Software-Dokumentation und Bug-Reporting
14. Mai	Einführung Arduino
21. Mai	Frei
28. Mai	Dateieingabe und -ausgabe
4. & 11. Juni	Tag der Lehre und Exkursionswoche
18. Juni	**Einführung von Qt**
25. Juni	GUI-Erstellung mit Qt
02. Juli	Serielle Kommunikation
09. Juli	API-Anleitung und Projektarbeit
16. Juli	Vorbereitung der Abgabe und Fragen
12. August 10:00	Abgabe

Achtung

Wichtig

Prüfungsanmeldung nicht vergessen!

Lehrziele

Qt und Debugging

Als Teilnehmer soll ich am Ende dieser Übung...

- ☐ wissen, wofür Qt verwendet wird
- ☐ Qt nutzen können
- ☐ einen Debugger zur Fehlerfindung verwenden können

Basics

C++ Objekte

Strukturen

Struktur in C++ `struct`

- Bündelung von Datenfeldern
- Datenorientiert
- Membervariablen sind ohne weitere Definition öffentlich modifizierbar

Beispiel:

```
struct Position {
  float x;
  float y;
  float z;
};
```

Klassen

Klassen in C++ `class`

- Ähnlich wie Strukturen
- Verhaltensorientiert
- Membervariablen sind zunächst privat

Beispiel:

```
class Spieler {
  Position m_Pos;
public:
  void bewegen(Position pos);
  void zurueckSetzen();
};
```

Bevor es losgeht...

Objekte in C++ `class`

Wird eine Struktur oder Klasse zur Deklaration einer Variable als Datentyp verwendet, verweist diese Variable auf EINE Instanz des Objekts.

Metapher:

- Strukturen und Klassen sind quasi Schablonen
- Objekte und Instanzen sind die Ergebnisse der Schablonen

Beispiel:

```
Spieler spielerObjekt;

//dynamisch angelegt:
Spieler *zeigerZumSpielerObjekt = new Spieler;
```


Qt
Qt

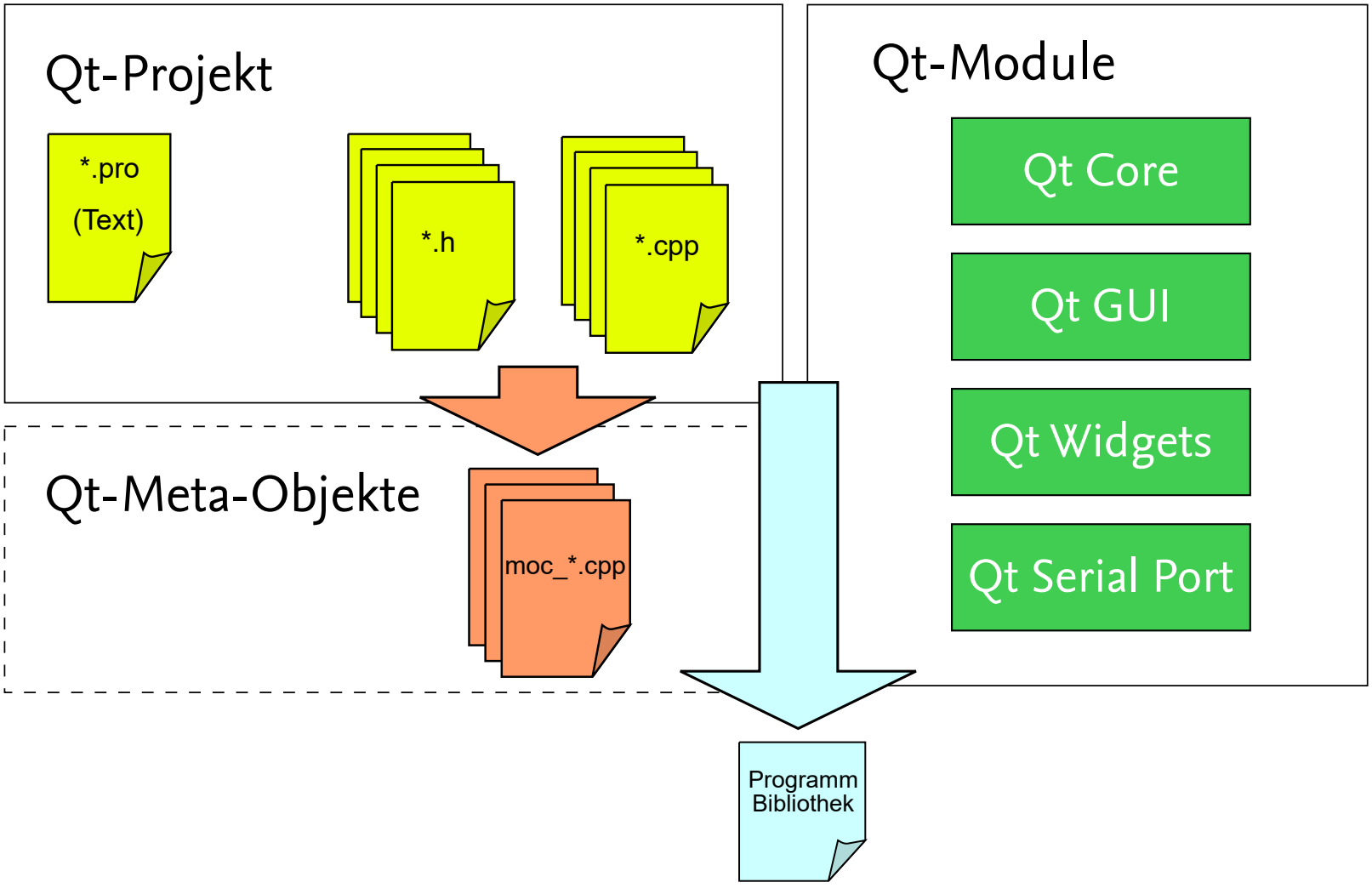

Einführung
Qt Creator
Erstes Projekt

Qt?

Qt ist ein Software-Entwicklungs-Kit, bestehend aus den folgenden Komponenten:

- Fertige Softwarebausteine aus den Bereichen
  - Graphische Benutzeroberflächen (GUI)
  - Netzwerkkommunikation
  - Datenhandling
  - ...
- Entwicklungsumgebung
  - Editor
  - Projektverwaltung
  - Versionskontrolle
  - Designer - für GUIs
  - Linguist - zur Unterstützung von mehreren Sprachen
- Werkzeuge
  - Meta Object Compiler - Erweiterung von C++

Qt-Projekt


Qt-Projekt
*.pro
(Text)
*.h
*.cpp
Qt-Meta-Objekte
moc_*.cpp
Qt-Module
Qt Core
Qt GUI
Qt Widgets
Qt Serial Port
Programm
Bibliothek

Qt Build-System

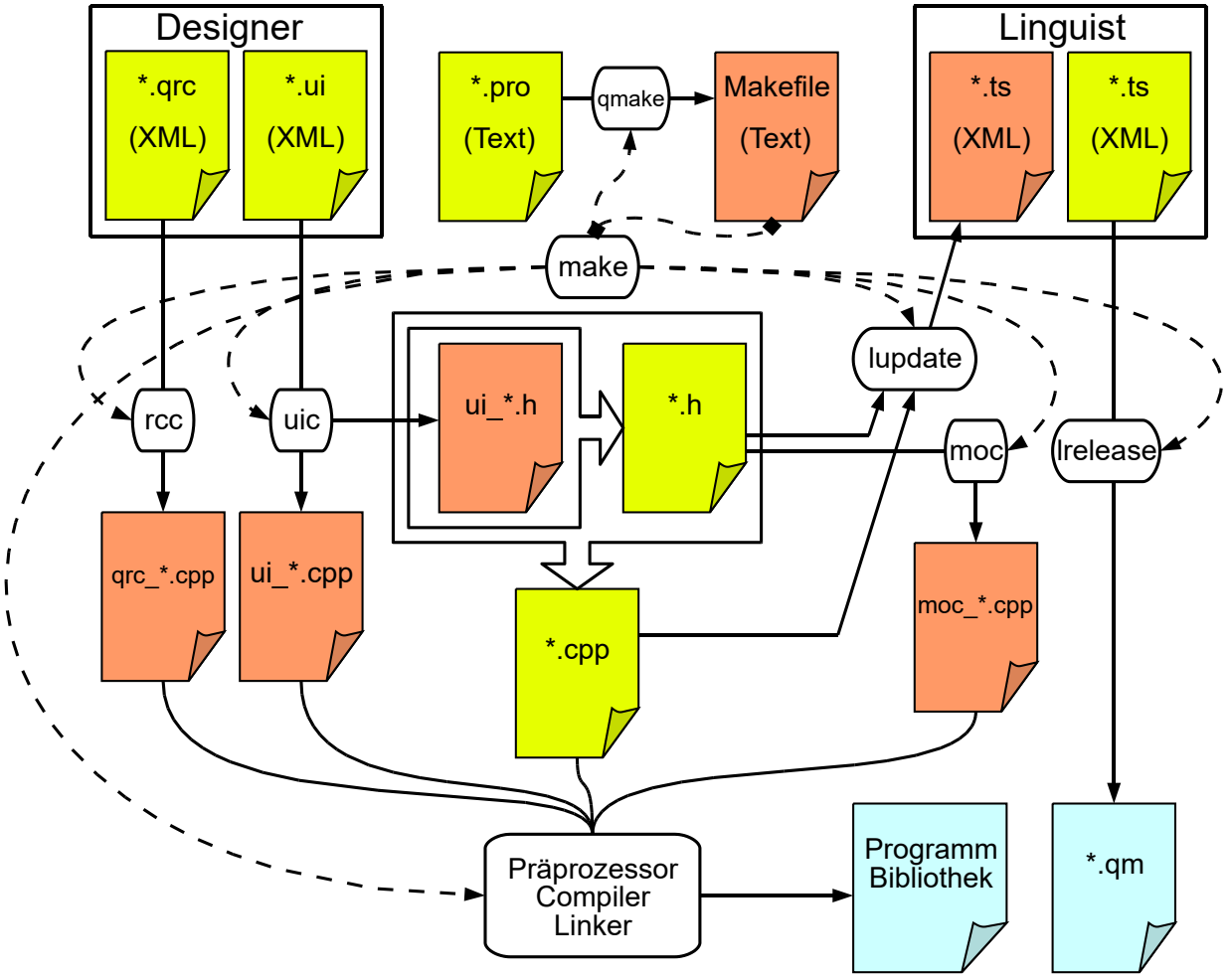


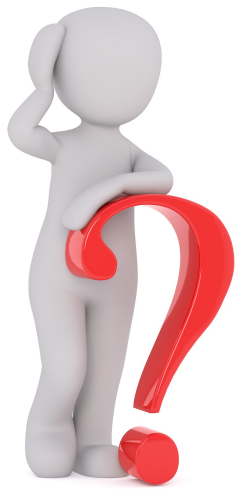
Abbildung 1: Wikimedia Commons - Thornard - Qt-Dokumentation, CC BY-SA 2.0 de, https://commons.wikimedia.org/w/index.php?curid=3349826

Verwendung

Qt wird häufig verwendet für...

- Softwareapplikationen mit graphischen Benutzeroberflächen
- Plattformübergreifende Applikationen
  - Windows
  - Linux
  - Mac
- Eingebettete Systeme
  - Mobilfunkgeräte
  - Automotive-Applikationen
  - Maschinen mit graphischen Benutzerschnittstellen

Und wann genau benötige *ich* Qt?

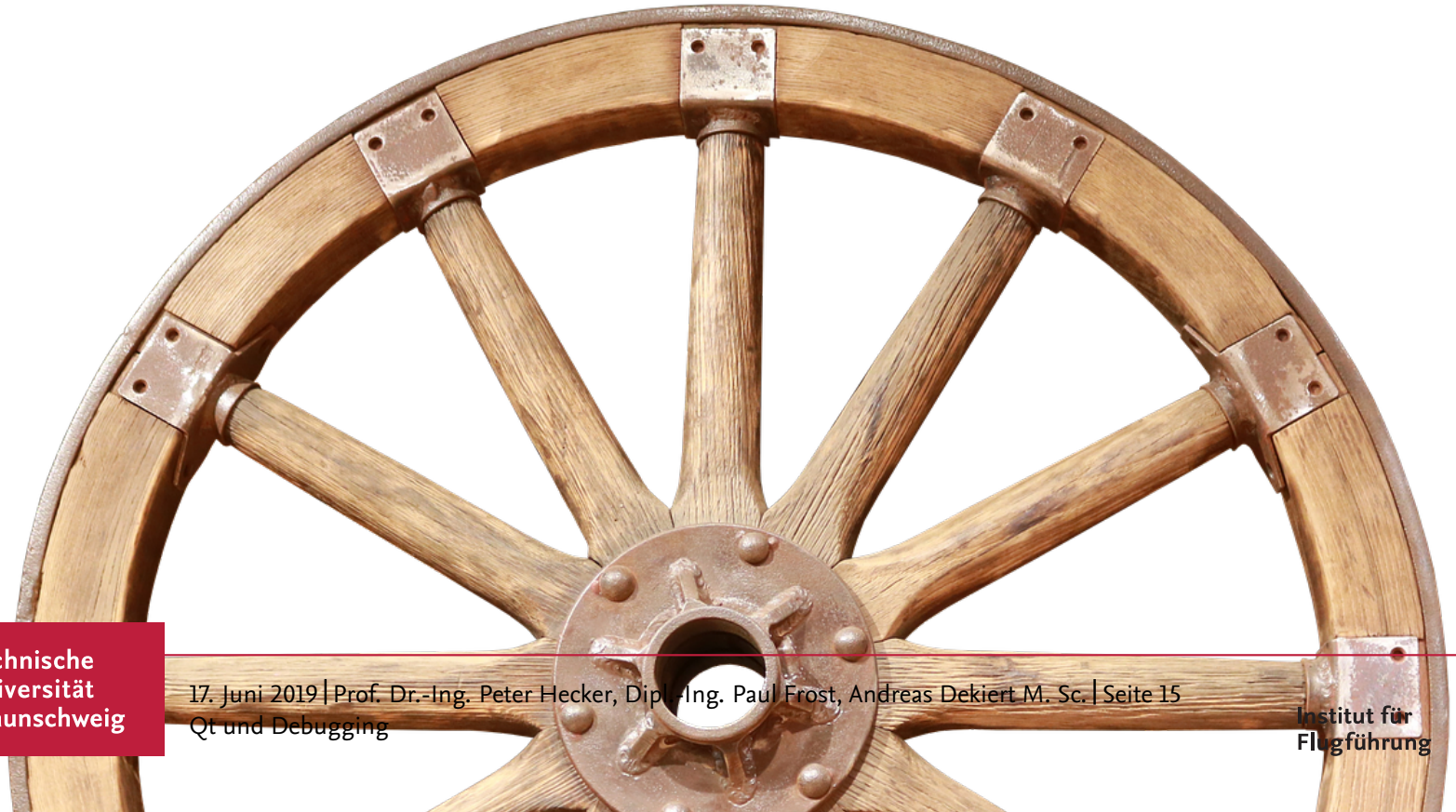


Ich benötige Qt, wenn ich...

- mit wenig Aufwand eine GUI erstellen möchte
- unkompliziert Daten senden und empfangen möchte
- meine Applikation auf unterschiedlichen Systemen betreiben möchte
- Daten aus XML-, json- oder ini-Dateien einlesen möchte

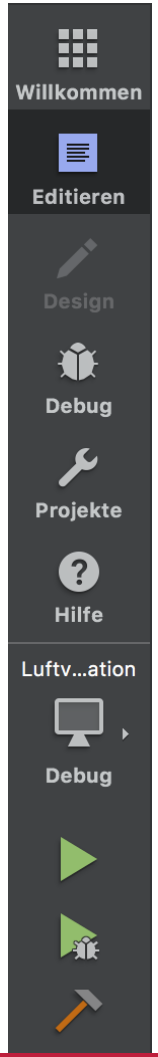
Ich benötige Qt, wenn ich...

- mit wenig Aufwand eine GUI erstellen möchte
- unkompliziert Daten senden und empfangen möchte
- meine Applikation auf unterschiedlichen Systemen betreiben möchte
- Daten aus XML-, json- oder ini-Dateien einlesen möchte
- das Rad nicht neu erfinden möchte



Gibt es Fragen oder Anmerkungen zu dem Unterthema
Einführung Qt?

Die Entwicklungsumgebung



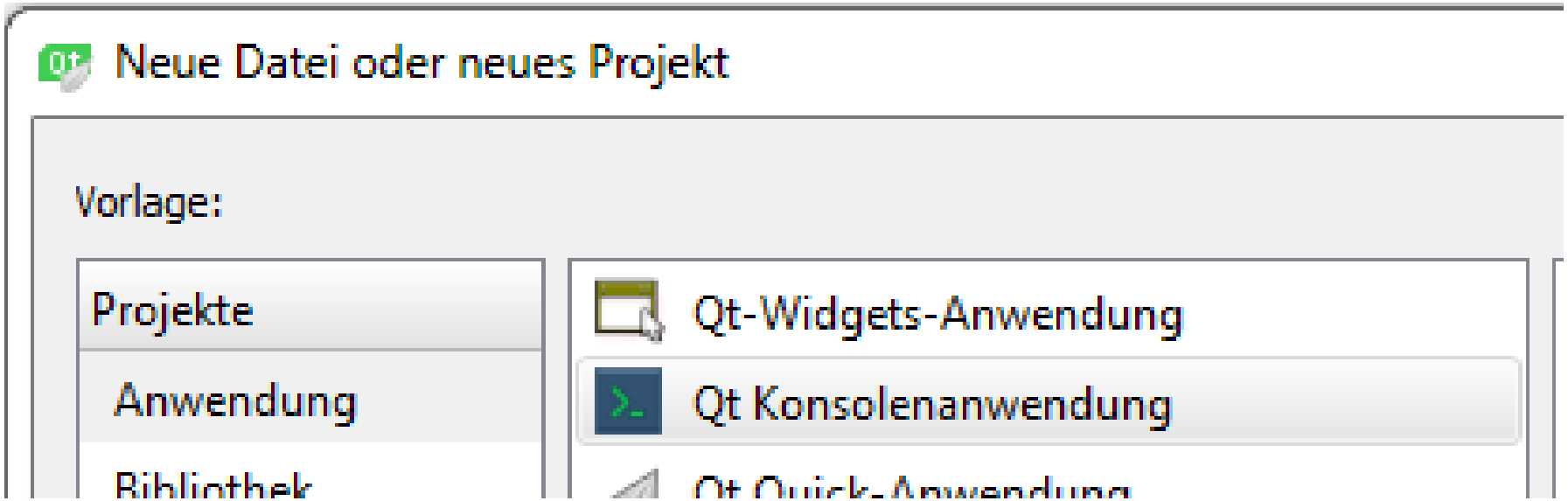
Die Qt Creator IDE umfasst:

- einen **Editor**,
  zur Erstellung von Projekten, Klassen, Methoden, Bibliotheken
- die Anbindung zur **Toolchain**,
  mit welcher die Projekte direkt gebaut werden können
- die Steuerung des **Debuggers**
  zur Fehlersuche
- die **Qt-Dokumentation**
  über die Taste `F1` auch direkt aus dem Quelltext erreichbar
- Einstellung von **Programmierrichtlinien**
  zur einheitlichen Gestaltung des Quelltexts
- ...

Gibt es Fragen oder Anmerkungen zu dem Unterthema
Qt Creator?

Anlegen eines Projekts

1. Neues Projekt
2. Qt Projektvorlage auswählen
3. Projekt einrichten
   - 3.1 Projektname wählen
   - 3.2 Zielverzeichnis auswählen
   - 3.3 Falls erforderlich, Compiler auswählen



Aufbau eines Qt-Projekts

Mit der Erstellung eines Qt-Projekts wird ein Verzeichnis angelegt, das mindestens die folgenden Dateien enthält:

`Projektname.pro` Projektdatei mit:

- Qt-Bibliotheken, die verwendet werden sollen
- Selbst geschriebenen Quelldateien, die kompiliert werden sollen
- Externe Bibliotheken
- Projektvariablen
- Installationsanweisungen (z.B. Kopieren von Dateien)

`main.cpp` Datei mit der Einstiegsfunktion

Die Verwaltung der Projektdatei kann über Qt Creator erfolgen.

Qt-Bibliothek einbinden

- Qt ist modular aus Bibliotheken aufgebaut
- In der Projektdatei (`Projektname.pro`) können Qt-Bibliotheken hinzugefügt werden:

  `QT +=` *Name des Qt-Moduls*
- oder ausgeschlossen werden:

  `QT -=` *Name des Qt-Moduls*

```
QT += core network serialport
QT -= gui
```

Übung

1. Erstelle eine Qt-Kommandozeilen-Applikation
2. Füge über `#include` den Header von QDebug hinzu
3. Füge in der main-Funktion die folgende Zeile hinzu:
   ```
   qDebug() << "API-Uebung";
   ```
4. Baue und starte das Programm

Signale und Slots in Qt

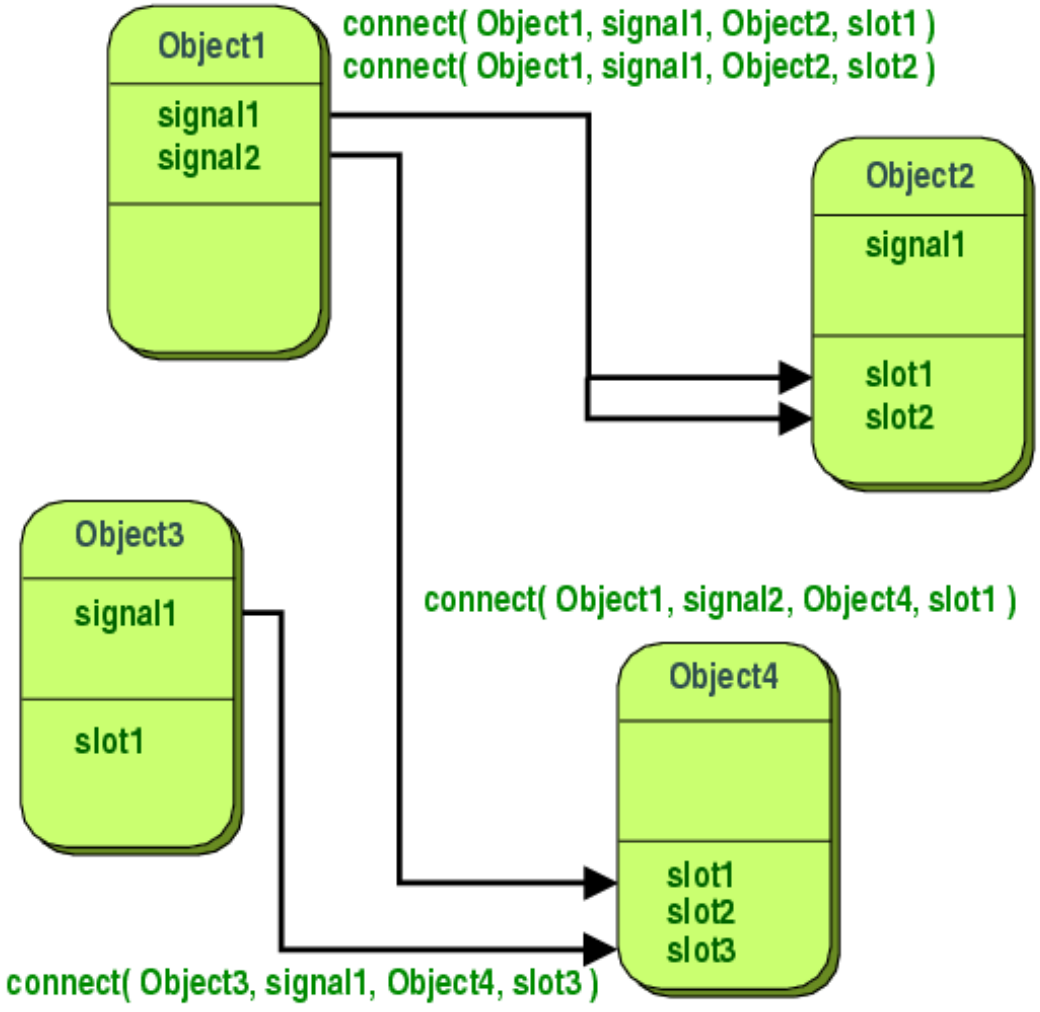
Signale und Slots in Qt

- Signale und Slots können für die Kommunikation zwischen Objekten genutzt werden
- Sehr mächtiges Feature in Qt
- Gut geeignet für die Interaktion mit graphischen Objekten
- Benachrichtigung bei Ereignissen

  Zum Beispiel: Ein Button wurde gedrückt
- Viele Klassen in Qt besitzen bereits Signale und Slots, die miteinander oder mit eigenen Signalen und Slots verbunden werden.

Signale und Slots

Schema



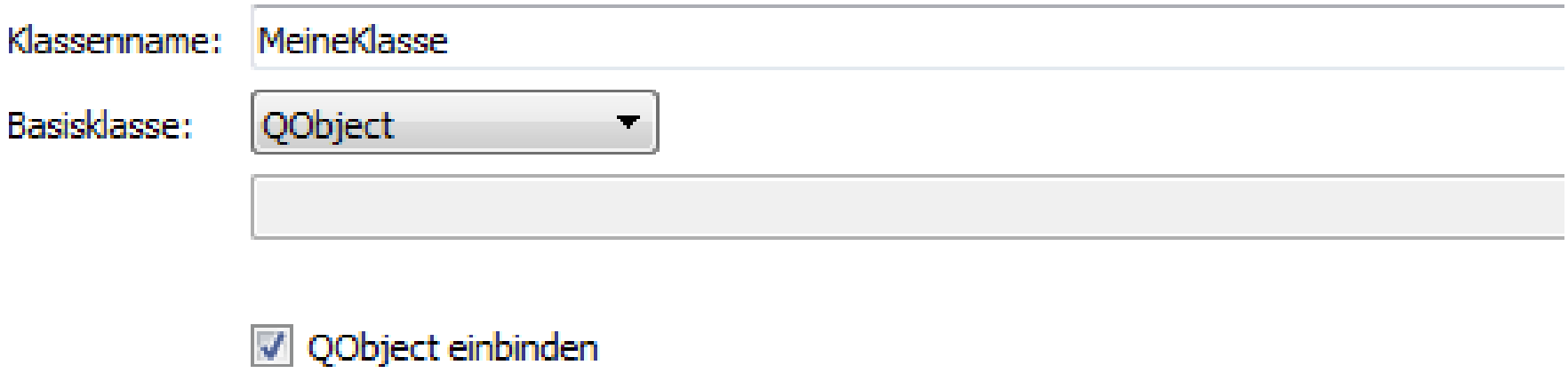
Signale und Slots — Textuelles Beispiel

- Signale zeigen an, dass sich etwas im Objekt verändert hat
  Signal: „ON-Taste gedrückt“
- Slots sind Funktionen bzw. Methoden, die mit Signalen verbunden werden können
  Slot: „Tv einschalten“
- Mit dem Befehl `QObject::connect()` können Signale mit Slots verbunden werden
  Verbinde das Signal: „ON-Taste gedrückt“ mit dem Slot: „Tv einschalten“

→ Wenn das Signal: „ON-Taste gedrückt“ emittiert wird, wird der Slot: „Tv einschalten“ ausgeführt

Eigene Klasse in Qt erstellen

- In Qt Creator können Klassen automatisiert erstellt werden
- Datei→Neu…→C++-Klasse
- Als Basisklasse: QObject auswählen
  Die neue Klasse erweitert die Basisklasse, dabei werden alle Eigenschaften der Basisklasse übernommen.



Aufbau der erstellten Klasse

- Qt Creator erstellt den Konstruktor `MeineKlasse(QObject)`
  → Ein Konstruktor ist die erste Funktion einer Klasse, welche zur Laufzeit ausgeführt wird. Er dient zur Initialisierung der Klasse.
- Der Destruktor `~MeineKlasse()` kann manuell erstellt werden
  → Ein Destruktor wird bevor die Klasse aus dem Speicher gelöscht wird als letztes ausgeführt. Er dient dazu Ressourcen wieder freizugeben.

```
#include <QObject>
class MeineKlasse : public QObject
{
    Q_OBJECT
public:
    explicit MeineKlasse(QObject *parent = 0);
    ~MeineKlasse();
...
};
```

Übung

Ziel

Es soll ein Programm erstellt werden, dass einen Countdown von 10 Sekunden runterzählt.

Die folgenden Qt-Ressourcen werden für die Übung benötigt:

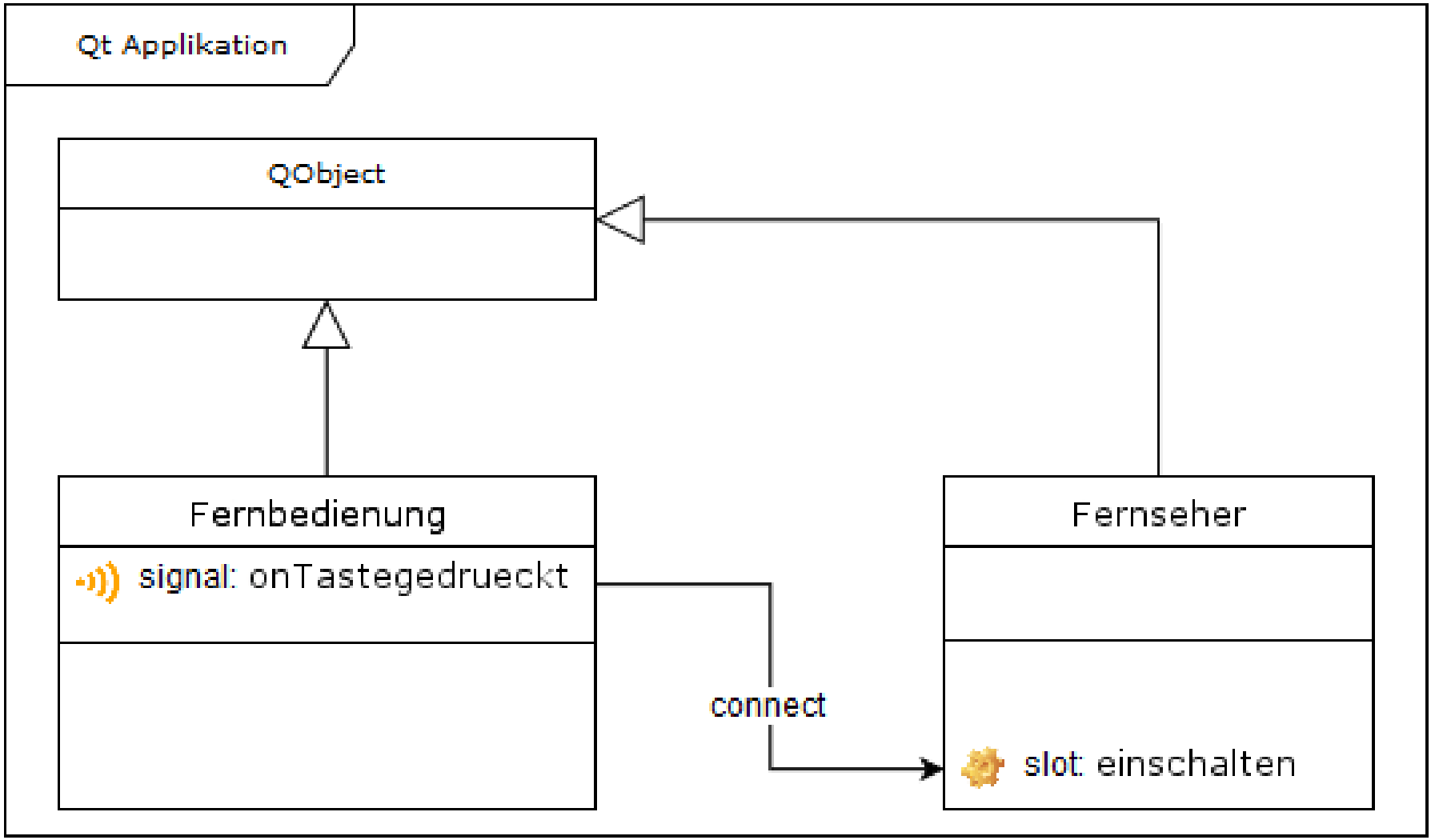
- Verwendete Qt-Objekte:
  - `QObject` als Basisklasse
  - `QTimer` als Trigger
  - `QDebug` zur Ausgabe der übrigen Sekunden
- Benötigtes Qt-Modul:
  - Qt-Core (`QT += core`)

Übung

1. Erstelle als Ableitung der `QObject`-Klasse (Basisklasse) die Klasse Countdown
2. Lege ein Objekt dieser Klasse in der `main`-Funktion an
3. Erweitere die `Countdown`-Klasse um eine private Membervariable `m_remainingTime` des Typs `int`
4. Setze im Konstruktor der `Countdown`-Klasse den Wert der Variablen auf 10

Signale und Slots

Klassendiagramm



Signale und Slots

Klasse mit Slot

Listing 1: Header-Datei einer Klasse mit Slot

```
#include <QObject>

class KlasseMitSlot : public QObject
{
  Q_OBJECT
public:
  explicit KlasseMitSlot(QObject *parent = 0);

public slots:
  void einSlot();
  //Implementierung in klassemitslot.cpp
};
```

Signale und Slots

Klasse mit Slot

Listing 2: Quell-Datei einer Klasse mit Slot

```
#include "klassemitslot.h"
#include <QDebug>

KlasseMitSlot::KlasseMitSlot(QObject *parent) :
    QObject(parent)
{

}

void KlasseMitSlot::einSlot()
{
  qDebug() << "einSlot wurde aufgerufen";
}
```

Übung

1. Erstelle den public slot `timeEllapsed()`
   In dieser Methode soll die Membervariable `m_remainingTime` heruntergezählt und anschließend auf den Startwert gesetzt werden.
2. Füge in der countdown.cpp-Datei über `#include` die Header von `QTimer` und `QDebug` hinzu
3. Suche über die Dokumentation von `QTimer` nach einem Beispiel für die Verwendung des Timers
4. Lege eine Instanz von `QTimer` dynamisch im Konstruktor von Countdown an
5. Verbinde den Timer über die Funktion `connect()` mit dem slot `timeEllapsed()` und starte noch im Konstruktor den Timer mit einem Intervall von 1000 ms

Signale und Slots

Klasse mit Signal

Listing 3: caption

```
#include <QObject>

class KlasseMitSignal : public QObject
{
  Q_OBJECT
public:
  explicit KlasseMitSignal(QObject *parent = 0)
      ;

signals:
  void einSignal();
};
```

Übung

1. Füge in der `countdown.h`-Datei das Signal `countdownWasReset()` hinzu
2. Jedes Mal, wenn der `m_remainingTime` auf den Startwert zurückgesetzt wurde, soll über `emit countdownWasReset();` das Signal ausgelöst werden.
3. Erstelle in der gleichen Klasse einen Slot, der den folgenden Befehl beinhaltet:

   ```
   qDebug() << "Timer finished!"
   ```

4. Verbinde den Slot mit dem zuvor erstellten Signal

Signale und Slots

Regeln

Wichtige Regeln

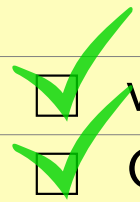
- Die Qt-Applikation muss gestartet sein.
- Objekte, die Signale und Slots verwenden möchten, müssen…
  - …Ableitungen des Objekts `QObject` sein.
  - …das Makro `Q_OBJECT` enthalten.
- Die verbundenen Signale und Slots müssen die gleichen Ein- und Ausgabeparameter besitzen.

Gibt es Fragen oder Anmerkungen zu dem Thema **Qt**?

Abgehakt

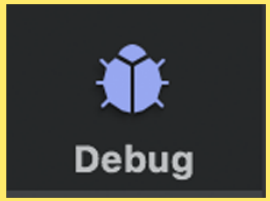
Qt und Debugging

Als Teilnehmer soll ich am Ende dieser Übung...



- [x] wissen, wofür Qt verwendet wird
- [x] Qt nutzen können
- [ ] einen Debugger zur Fehlerfindung verwenden können

Debugging



Qt-Debugger

Praxisbeispiele

Fehlerdiagnose

- Ausgabe von Statusnachrichten oder Variablenwerten
  - Verschlechtert die Lesbarkeit des Codes
  - Bei vielen Ausgaben geht Überblick verloren
  - Unflexibel, welche Variablen ausgegeben werden sollen

```
qDebug() << "Wert:" << irgendeineVariable;
```

Fehlerdiagnose

- Code teilweise auskommentieren
  - Eingrenzung des Fehlers
  - Erleichtert gezieltes Testen einzelner Komponenten
  - Test des gesamten Systems nicht möglich
  - Verschlechterte Lesbarkeit des Codes

```
aufrufFunktionA();
aufrufFunktionB();
aufrufFunktionC();
//aufrufFunktionD();
//aufrufFunktionE();
//aufrufFunktionF();
```

Was ist ein Debugger?

- Werkzeug zum Auffinden von Fehlern in Computerprogrammen
  - Fehlerfindung an fertig kompilierter ausführbarer Datei
  - Keine Änderungen am Quelltext
  - Interna eines (fehlerhaften) Programms untersuchen
- Funktionen eines Debuggers
  - Steuerung der Programmausführung (Pause, schrittweises Ausführen)
  - Inspektion von Variablen
  - Modifikation von Variablen und Arbeitsspeicher-Inhalt

Debugger

Verschiedene Debugger erhältlich, oft in IDEs integriert

- GNU Debugger (GDB): Standard-Debugger unter Linux, Integration z. B. in KDevelop
- Microsoft Visual Studio Debugger: Debugger in Visual Studio
- Spezialisierte Debugger (Verteilte Anwendungen, Spezialhardware, ...)

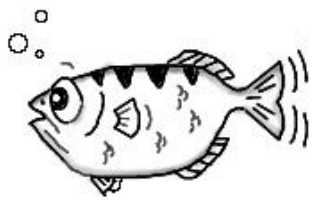


Abbildung 2: GDB-Debugger Logo[1]

[1] https://www.gnu.org/software/gdb/

Debugger – Steuerung des Programmablaufes

- Debugger vollzieht Programmablauf nach
  - An welcher Stelle des Programmablaufs befindet sich das Programm gerade?
  - Aufrufliste (*Callstack*): Welche Funktion hat welche Funktion aufgerufen?
- Nutzer kann mit Debugger Programmausführung anhalten und wiederaufnehmen
- *Breakpoint*: Unterbrechung, beim Erreichen dieses Punktes wird Ablauf pausiert
  - *Conditional Breakpoint*: Unterbrechung bei Erfüllung einer Bedingung
  - *Data Breakpoint*: Unterbrechung bei Änderung eines Speicherbereiches

Debugger Schrittweise Ausführung des Programms

Das Programm kann mit den folgenden Befehlen schrittweise ausgeführt werden:

 *Step Over*:	Nächste Quelltextzeile
 *Step In*:	In Methodenaufruf springen
 *Step Out*:	Aus Methode zum Aufrufer zurück-springen
 *Continue*:	Ausführung fortsetzen

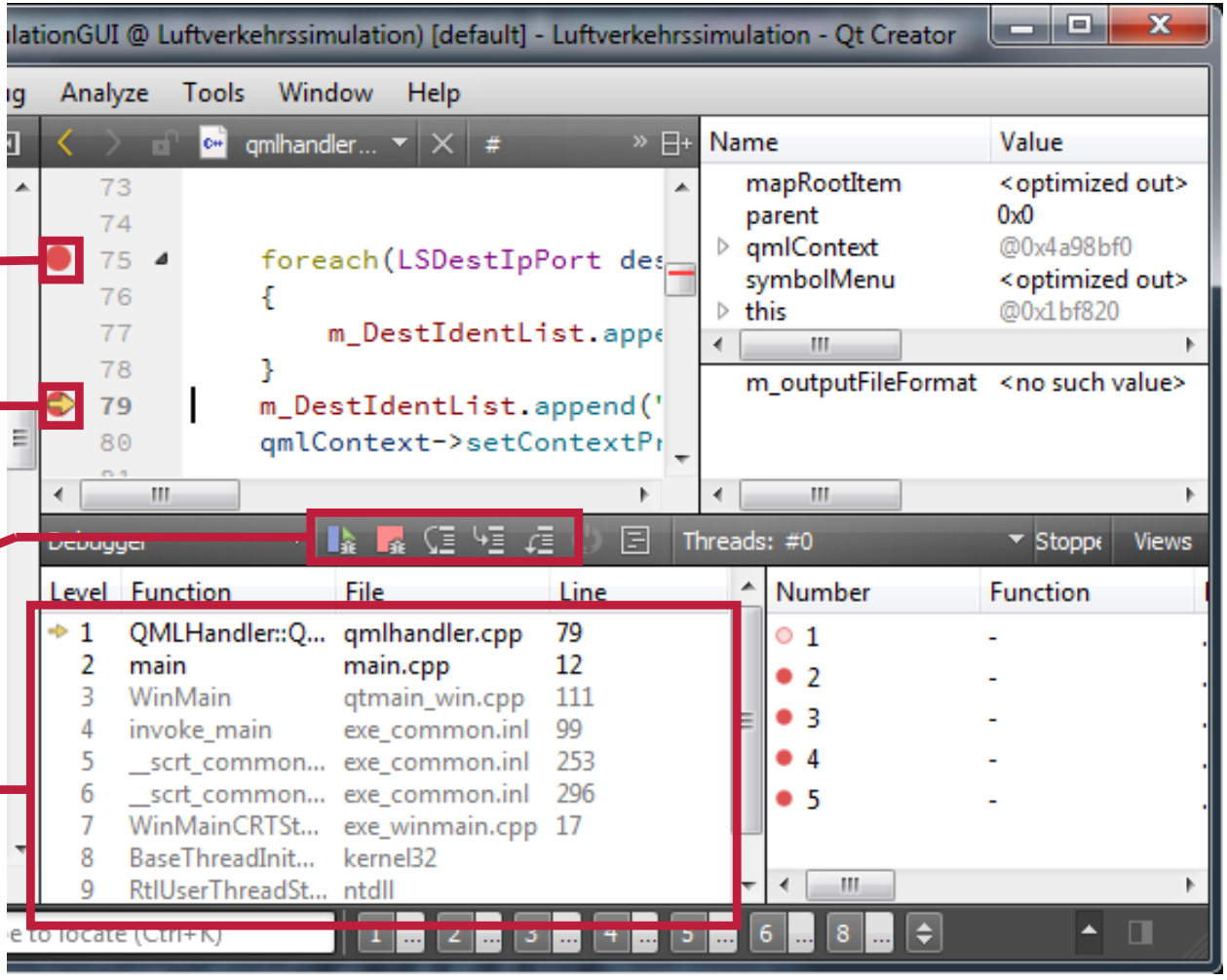
Beispiel: Debugging in Qt Creator

Breakpoint

Programm-zeiger

Ablauf-steuerung

Callstack



Debug- und Release-Versionen

Debug

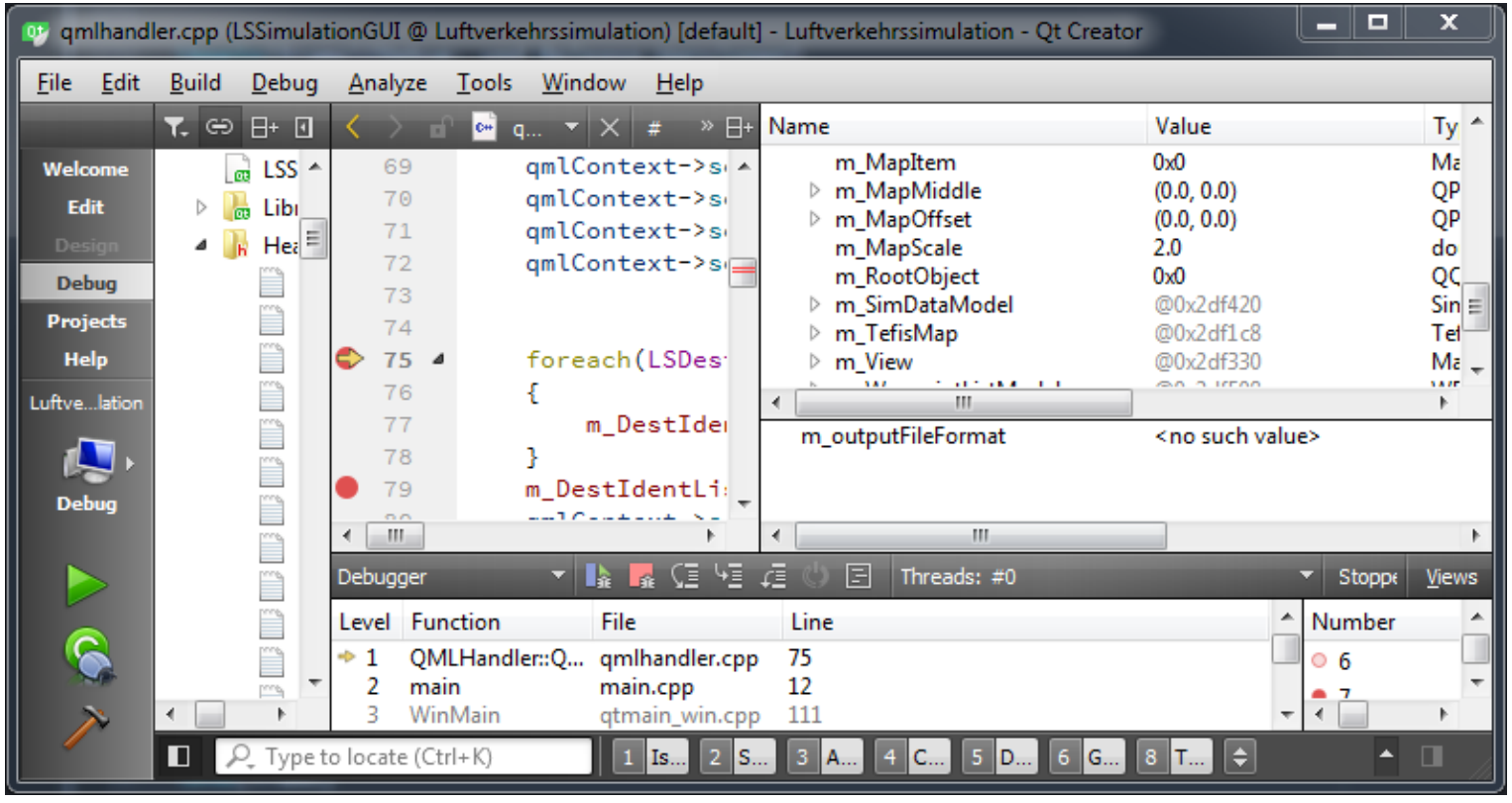
- Zusätzliche Informationen im ausführbaren Programm (Debugsymbole)
  $\Rightarrow$ Mehr Speicherplatz
- Erforderliche Konfiguration für das Debugging

Release

- Optimierungen durch Compiler und Linker
  $\Rightarrow$ kein Rückschluss von Quelltextzeile auf Ablauffortschritt möglich
  $\Rightarrow$ Schnellere Ausführung des Programms
- Empfohlene Konfiguration für Veröffentlichung

Debugger – Umgang mit Variablen

- Auflösen von Variablennamen in Typ und Speicherbereich
- Anzeige des Variableninhalts in der IDE
- Veränderung des Variableninhalts zur Laufzeit durch Debugger



Übung

1. Setze einen Breakpoint in die Zeile, in der die Variable `m_remainingTime` heruntergezählt wird
2. Starte das Programm mit dem Debugger
3. Vollziehe den Ablauf des Programms nach

Bugs vorbeugen

- KISS-Prinzip (Keep it simple and stupid)
  - Komplexe Ausdrücke in mehrere einfache Kommandos zerlegen
  - Eindeutige Bezeichnungen, einfache Struktur
- Immer den Fehlerfall mit einplanen
  - Rückgabewerte von Systemfunktionen überprüfen
  - Fehler richtig behandeln

5 Schritte zur Fehlerbehebung

1. Reproduktionsregel finden
   Oft schwierig, da Regelmäßigkeiten unscheinbar sind, oder Verknüpfungen unintuitiv wirken
2. Hinweise sammeln
   Alle möglichen Zusatzinformationen können helfen, eine Regelmäßigkeit oder die konkrete Codestelle zu finden
3. Fehler bestimmen
   Der Fehler liegt nicht immer im Code
4. Fehler beheben
   Evtl. Anpassung des Designs nötig
5. Testen
   Tritt der Fehler immer noch auf?

Schwierigkeit: Fehler bestimmen

- Manche Probleme sind extrem schwer zu finden
- Oft keine direkte Zuordnung von Fehlverhalten auf Quelltext möglich
- Beispiele für Fehlverhalten
  - Fehler treten nur in Release-, aber nicht in Debug-Version auf
  - Fehler treten nur ab und zu auf
  - Fehler treten nur auf einigen Computern auf
- Gerade bei fremdem Code ist das Erkennen von Bugs sehr aufwändig
- Fehler zu erkennen benötigt eine gewisse Übung

Gibt es Fragen oder Anmerkungen zu dem Thema
Debugging?

Abgehakt

Qt und Debugging

Als Teilnehmer soll ich am Ende dieser Übung...

- ☑ wissen, wofür Qt verwendet wird
- ☑ Qt nutzen können
- ☑ einen Debugger zur Fehlerfindung verwenden können

Literatur

[1] J. Wolf, *Qt 4.6 - GUI-Entwicklung mit C++ : das umfassende Handbuch ; [Grundlagen, Praxis, Referenz ; plattformunabhängige Anwendungen mit Qt 4.6 ; DVD-ROM mit Qt SDK, Openbooks, Zusatzkapiteln und Beispielen ; inkl. Qt creator]*, Ser. Galileo Computing. Bonn: Galileo-Press, 2010, Die DVD-ROM-Beilage enth. 2 Openbooks.

[2] S. Huang, *Qt 5 blueprints : design, build, and deploy cross-platform GUI projects using the amazingly powerful Qt 5 framework*, Ser. Community experience distilled. Birmingham, UK: Packt Publishing, 2015, Includes index.

Ende

Vielen Dank für eure Aufmerksamkeit!