



Technische
Universität
Braunschweig

Institut für
Flugführung



Entwicklungsumgebungen und Debugging

Werkzeuge für die Erstellung von Quelltext

Prof. Dr.-Ing. Peter Hecker, Dipl.-Ing. Paul Frost, 09. Mai 2017

Agenda

- 04. April Kick-Off
- 11. April Projektmanagement
- 18. April Prozessmodelle
- 25. April Versionsverwaltung
- 02. Mai Einführung Arduino/Funduino
- 09. Mai Entwicklungsumgebungen und Debugging**
- 16. Mai Dokumentation und Testing
- 23. Mai Dateieingabe und -ausgabe
- 30. Mai GUI-Erstellung mit Qt
- 06. Juni Exkursionswoche
- 13. Juni Bibliotheken
- 20. Juni Netzwerke
- 27. Juni Projektarbeit
- 04. Juli Projektarbeit
- 11. Juli Vorbereitung der Abgabe



Teil I

Wiederholung

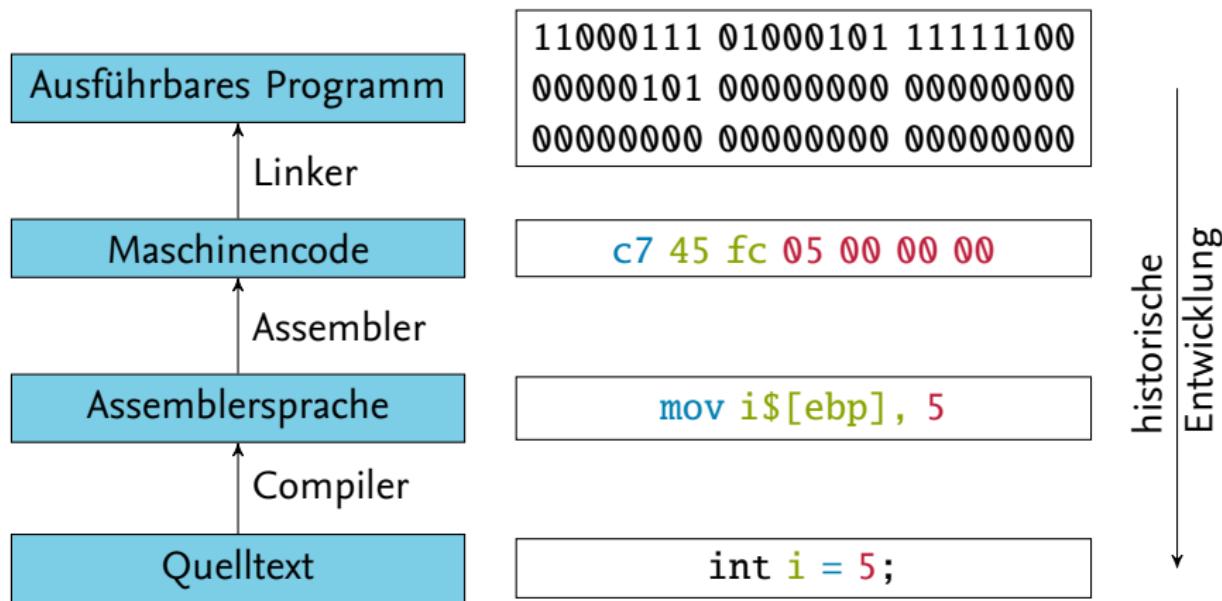
Praxistemonstration



Teil II

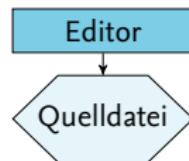
Entwicklungsumgebungen

Vom Quelltext zum ausführbaren Programm



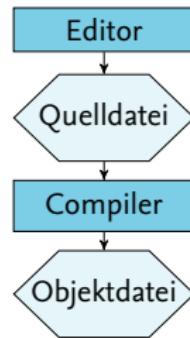
Toolchain

1. Quelltext erstellen(Quelldatei)



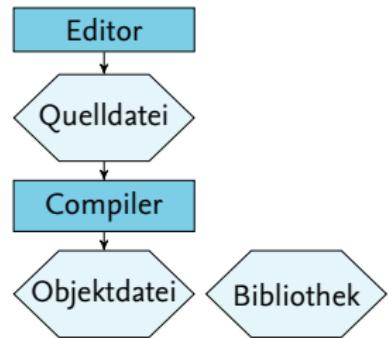
Toolchain

1. Quelltext erstellen(Quelldatei)
2. Umwandlung in Maschinencode:
Kompilieren (Objektdatei)



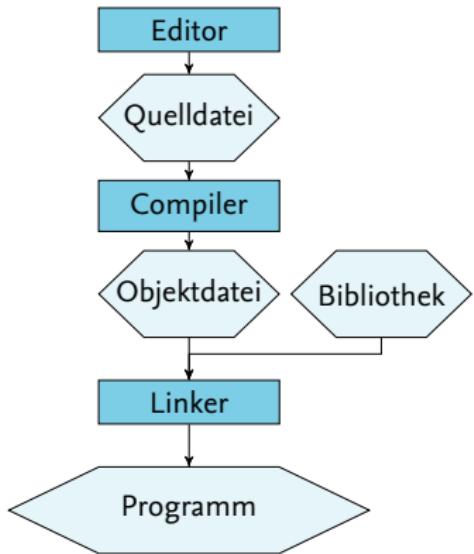
Toolchain

1. Quelltext erstellen(Quelldatei)
2. Umwandlung in Maschinencode:
Kompilieren (Objektdatei)



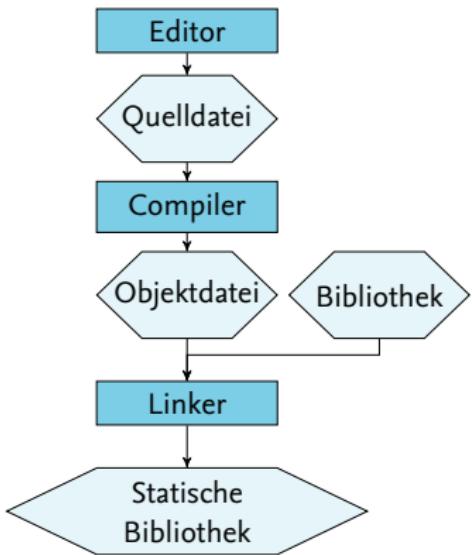
Toolchain

1. Quelltext erstellen(Quelldatei)
2. Umwandlung in Maschinencode:
Kompilieren (Objektdatei)
3. Maschinencode und
Bibliotheken zusammenfassen:
Linken (Lib/Exec)



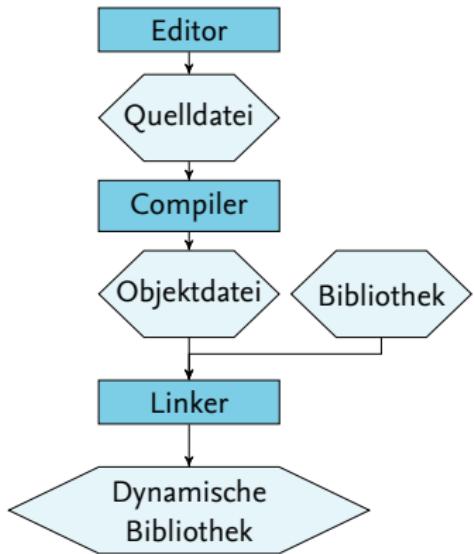
Toolchain

1. Quelltext erstellen(Quelldatei)
2. Umwandlung in Maschinencode:
Kompilieren (Objektdatei)
3. Maschinencode und
Bibliotheken zusammenfassen:
Linken (Lib/Exec)



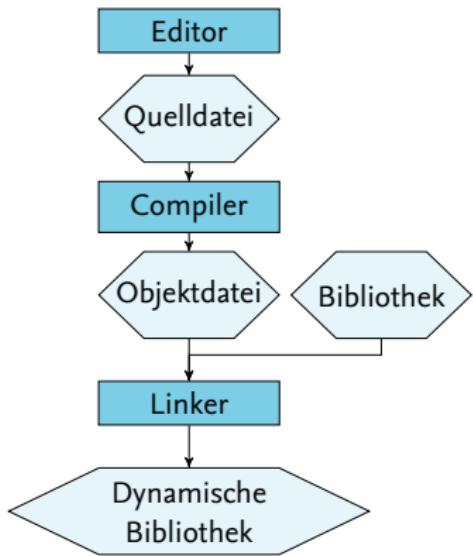
Toolchain

1. Quelltext erstellen(Quelldatei)
2. Umwandlung in Maschinencode:
Kompilieren (Objektdatei)
3. Maschinencode und
Bibliotheken zusammenfassen:
Linken (Lib/Exec)



Toolchain

1. Quelltext erstellen(Quelldatei)
 2. Umwandlung in Maschinencode:
Kompilieren (Objektdatei)
 3. Maschinencode und
Bibliotheken zusammenfassen:
Linken (Lib/Exec)
- Bei Änderungen
 - Quelltext kompilieren
 - Erneutes Linken



Erstellen einer ausführbaren Datei

- Quellcode erstellen:
→ Quelldateien

main.cpp
eins.cpp/eins.h
zwei.cpp/zwei.h



Erstellen einer ausführbaren Datei

- Quellcode erstellen:
→ Quelldateien

main.cpp

eins.cpp/eins.h

zwei.cpp/zwei.h

- Kompilierung:

```
$ g++ -c eins.cpp zwei.cpp main.cpp
```

- Objektdateien

main.o

eins.o

zwei.o



Erstellen einer ausführbaren Datei

- Quellcode erstellen:
→ Quelldateien

main.cpp

eins.cpp/eins.h

zwei.cpp/zwei.h

- Kompilierung:

```
$ g++ -c eins.cpp zwei.cpp main.cpp
```

- Objektdateien

main.o

eins.o

zwei.o

- Linken:

```
$ g++ -o Beispiel eins.o zwei.o main.o
```

- Programm

Beispiel.exe



Plattformübergreifende Softwareentwicklung

- Bisherige Betrachtung:
Maschinencode wird auf der Maschine erzeugt, auf der dieser genutzt werden soll
- Was benötigt man für eine andere Konfiguration (Portierung)?
 - Anderer Prozessor: Neuer Maschinencode nötig
 - Anderes Betriebssystem/Compiler: Evtl. Änderungen im Quelltext nötig
 - Andere Bibliotheken: Andere Funktionen sind verfügbar
 - Evtl. benötigt eine Konfiguration andere Aufrufe für Compiler und Linker



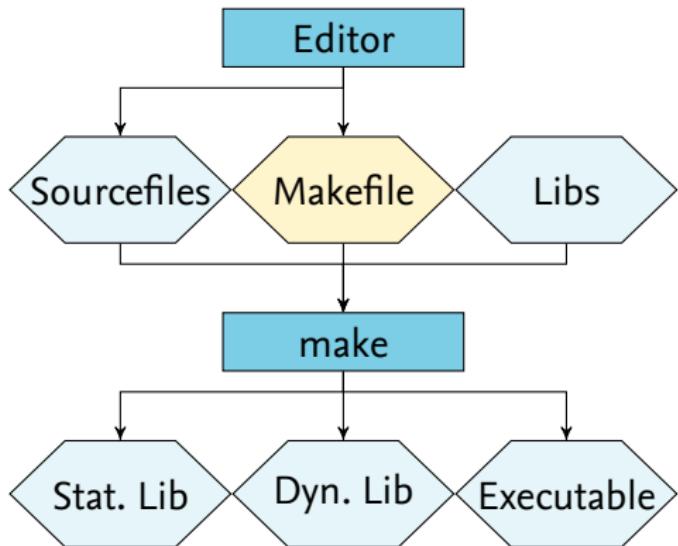
Automatisierung durch Makefiles

Makefiles können die Erstellung von Software automatisieren

- Bauanleitung für ein Projekt
- Pro Konfiguration ein Makefile
- Aufruf von Compiler und Linker mit den passenden Parametern
- Nur geänderte Quellen werden kompiliert
- Erstellen durch Aufruf von make



Makefiles



Vorteile

- Sehr mächtig und flexibel
- Nach der Erstellung komfortabel

Nachteile

- Erstellung nicht intuitiv
- Komplexe Makefiles sind fehleranfällig
- Aufwand bei Erstellung



Beispiel-Makefile für Linux

Listing 1: code/MakefileProjekt/makefile

```
1 OBJS = eins.o main.o zwei.o
2 PROC = Beispiel
3 all: $(PROC)
4 $(PROC): $(OBJS)
5     g++ -o $@ $^
6 clean:
7     rm -f $(OBJS)
8     rm -f $(PROC)
```

- Übersetzen starten: `$ make`
- Aufräumen: `$ make clean`
- Verschiedene Ziele in einem Makefile: `$ make <target>`
- Erstellen von Makefiles aufwändig, Automatisierung wünschenswert



GNU Build System (Autotools)

- Portable Konfiguration von Quellcode auf Unix-Systemen
 - Erstellung eines Konfigurations-Skripts
 - Steuerung durch Templates (Makrosprache M4)
- Aufruf eines Konfigurationsskripts: `$./configure`
 - Test auf Compiler, Linker, Bibliotheken, Features, ...
 - Erstellung eines Makefiles für aktuelle Konfiguration
- Nachteile
 - Posix-kompatibles System wird vorausgesetzt
 - Makrosprache M4 ist kompliziert, Struktur unübersichtlich
- Moderne Alternativen
 - Apache Ant: <http://ant.apache.org/>
 - CMake: <http://www.cmake.org/>



CMake

- Moderner Makefile-Generator
 - Open-Source: BSD-Lizenz
 - Erzeugung von Projektdateien für verschiedene Entwicklungsumgebungen
- Aufbau und Prinzip
 - Konfigurationsdatei `CMakeLists.txt` definiert die benötigten Komponenten
 - Programm `cmake` interpretiert die Konfigurationsdatei
→ **Erstellt Makefiles oder IDE-Projektdaten**
- Vorteile gegenüber Autotools
 - Simples Konfigurationsskript, keine Makrosprache
 - Direkte Erzeugung bekannter Projektformate



Beispiel CMakeLists.txt für CMake

Listing 2: code/CMakeProjekt/CMakeLists.txt

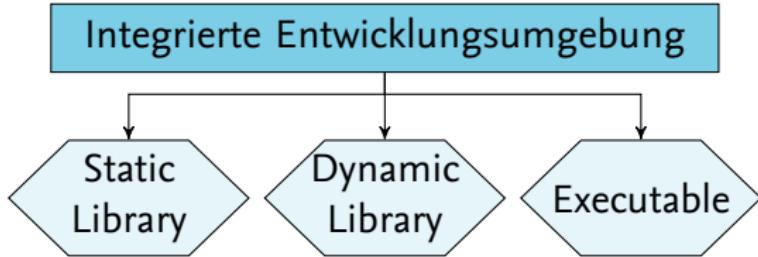
```
1 PROJECT (BeispielProjekt)
2 SET (Sources main.cpp eins.cpp zwei.cpp)
3 SET (Headers eins.h zwei.h)
4
5 ADD_EXECUTABLE(Beispiel ${Sources} ${Headers})
6 TARGET_LINK_LIBRARIES(Beispiel)
```

- Erstellung eines Makefiles mit cmake: `$ cmake CMakeLists.txt`
- Erstellung von Konfigurationsdateien für IDEs möglich



Integrierte Entwicklungsumgebungen

- Integrated Development Environment (IDE)
 - Grafische Bedienoberfläche für die Software-Entwicklung
 - Erweiterter Texteditor (Syntaxhighlighting, Syntaxchecking, Codecompletion)
 - Projektverwaltung (nutzt Makefiles oder eigene Formate als Basis)
 - Assistenten (Klassen, Projekte, GUI, etc.)
 - Integration weiterer Werkzeuge
- Gesamte Toolchain in einem Werkzeug vereint



Microsoft Visual Studio

- IDE von Microsoft
 - Läuft primär unter MS Windows
 - Mehrere Sprachen möglich (C/C++, Visual Basic, C#, ...)
 - Verschiedene Editionen verfügbar
 - Community (kostenlos)
 - Professional
 - Premium
 - Ultimate
 - Erhältlich im Rahmen von DreamSpark über das Gauß-IT-Zentrum

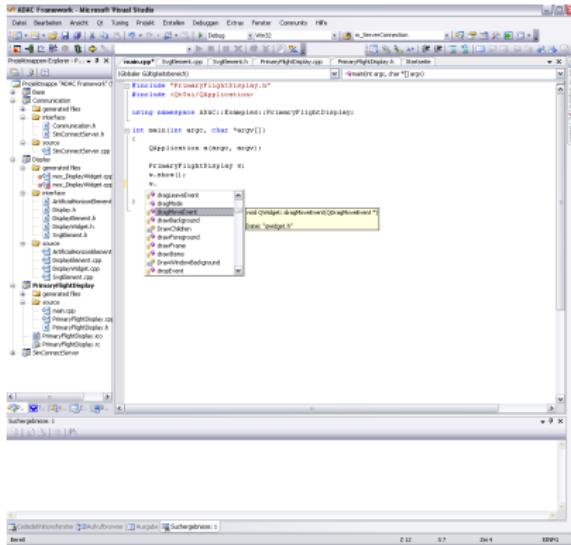


Abbildung 1: Visual Studio

Eclipse

- Anpassbare Entwicklungsumgebung
 - Open Source: Eclipse Public License (EPL)
 - Plattformübergreifend (Java)
 - Basis: IBM Visual Age for Java
- Plug-In Architektur
 - Kern lädt Erweiterung (Plug-In)
 - Viele Plug-Ins verfügbar
- Fokus: Java-Programmierung
 - Weitere Sprachen als Plug-In: C/C++, C#, Perl, PHP, ...
 - Embedded Entwicklung

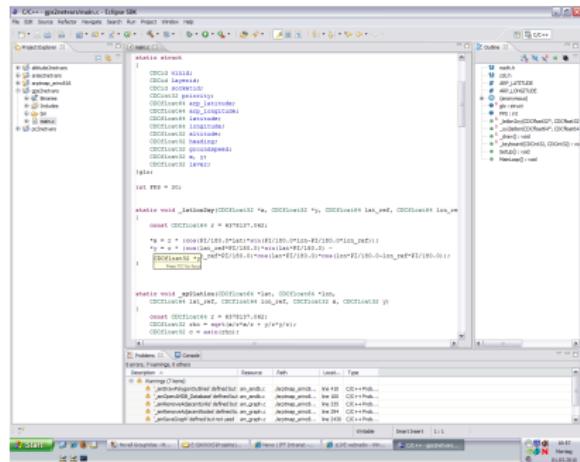


Abbildung 2: Eclipse



Qt Creator

- IDE für das Qt Framework
 - Open Source: GPL
 - Plattformübergreifend
 - Speziell angepasst für die Entwicklung mit Qt
 - Nutzbar mit verschiedenen Compilern
 - GCC (Linux)
 - MinGW (Windows)
 - Visual Studio Compiler
 - Einfache Entwicklung auch für Mobilgeräte

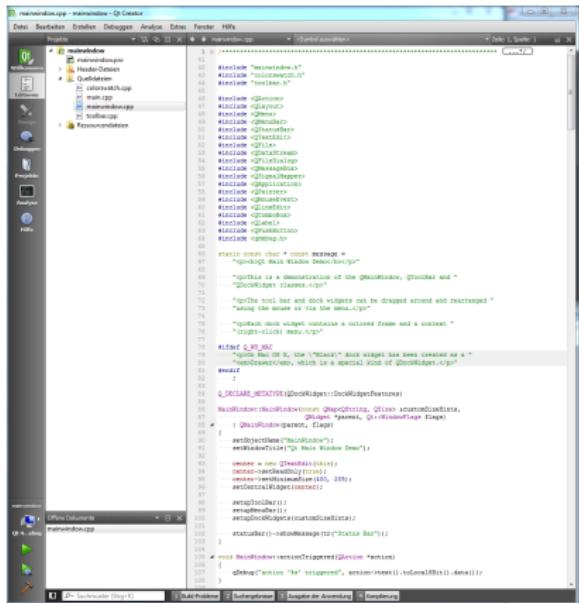
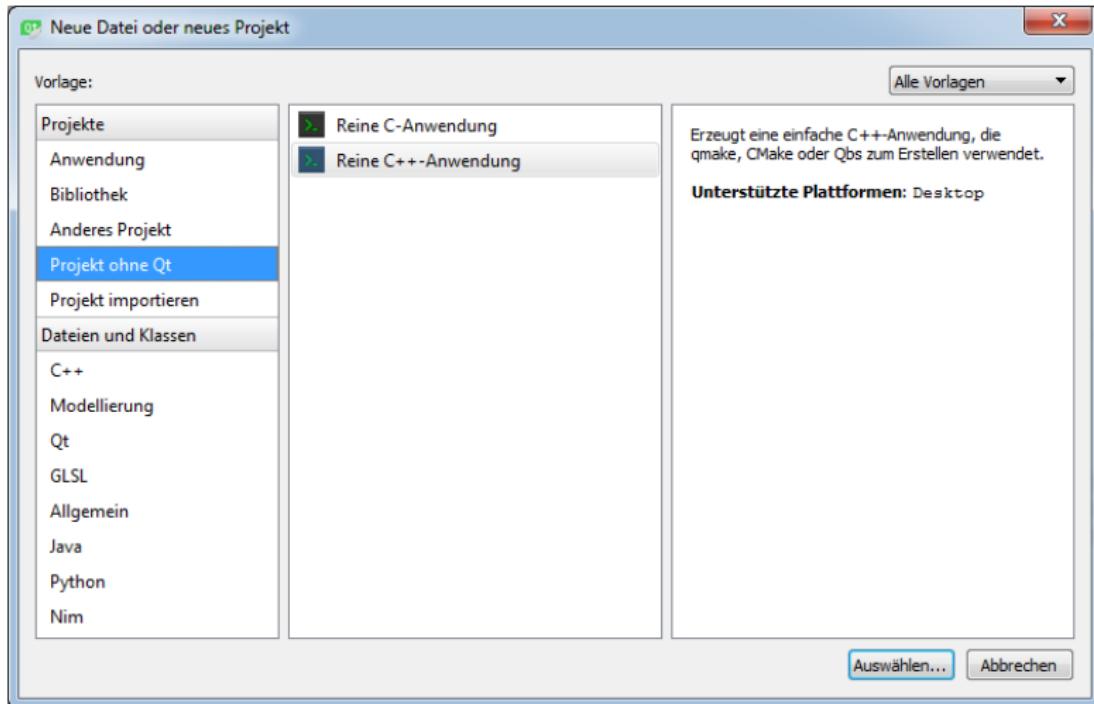


Abbildung 3: Qt Creator

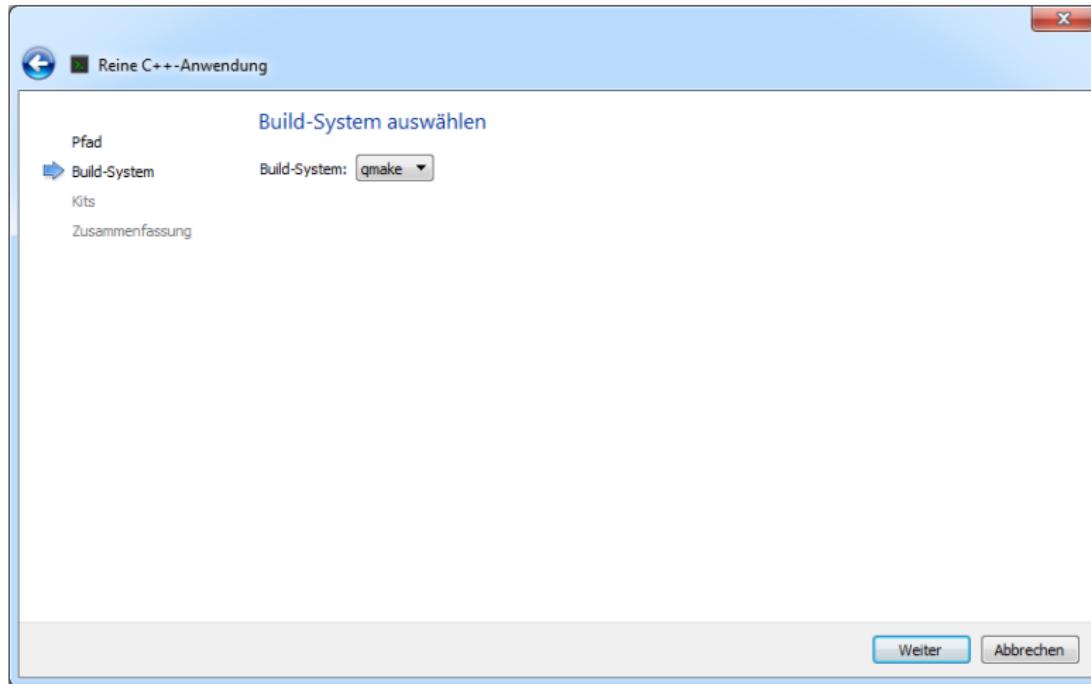
Beispiel

Erstes Qt-Projekt



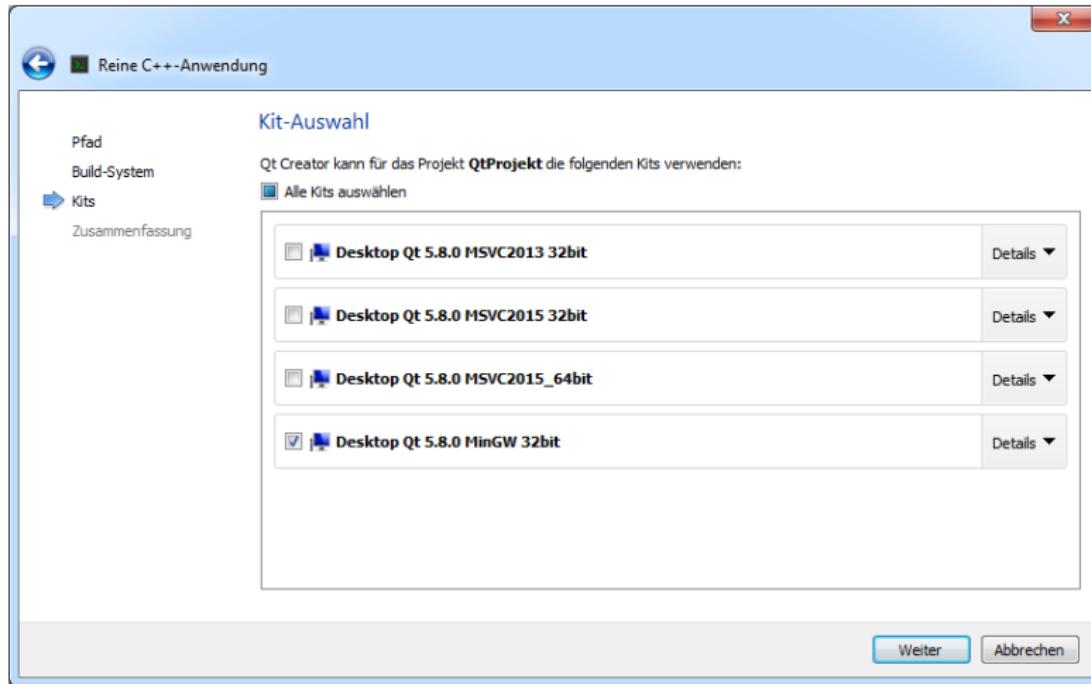
Beispiel

Erstes Qt-Projekt



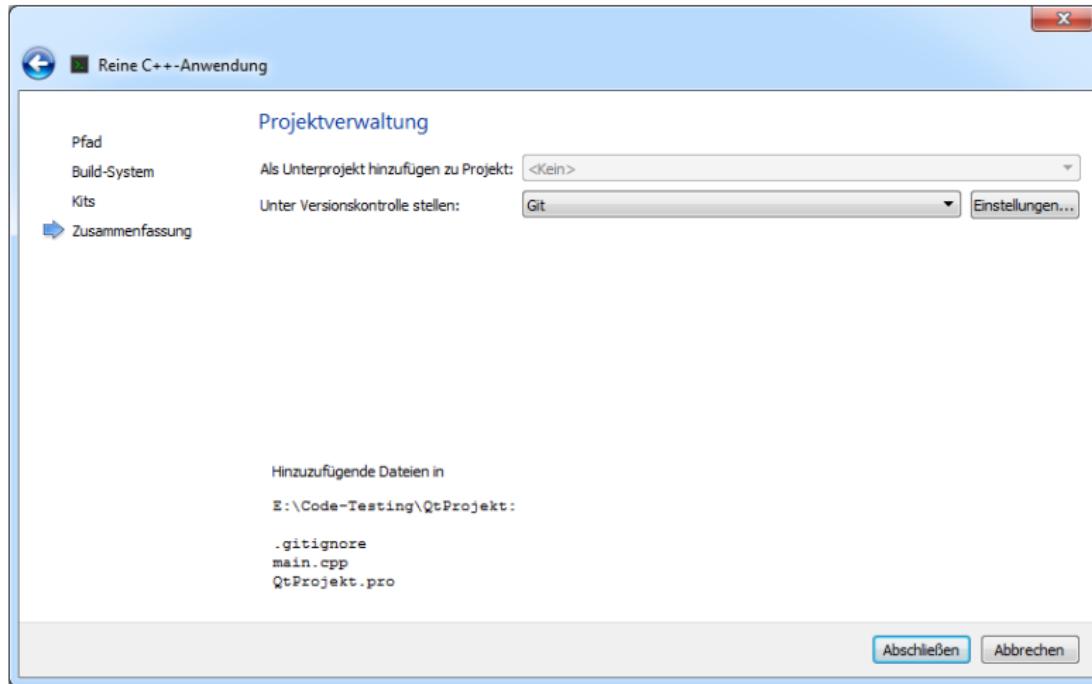
Beispiel

Erstes Qt-Projekt



Beispiel

Erstes Qt-Projekt



Beispiel

Erstes Qt-Projekt

The screenshot shows the Qt Creator interface with a project titled "QtProjekt [master]" in the center-left pane. The project structure includes a "QtProjekt.pro" file and a "Quelldateien" folder containing "main.cpp". The main editor area displays the following CMake-like configuration code:

```
1 TEMPLATE = app
2 CONFIG += console c++11
3 CONFIG -= app_bundle
4 CONFIG -= qt
5
6 SOURCES += main.cpp
7
```

The left sidebar contains toolbars for "Willkommen", "Editieren", "Design", "Debug", "Projekte", "Hilfe", and "QtProjekt". The bottom toolbar includes buttons for "Suchmuster (Strg+K)", "Build-P...", "Sucher...", "Ausga...", "Kompli...", and "Debug...".



Beispiel

Erstes Qt-Projekt

The screenshot shows the Qt Creator IDE interface. On the left is a sidebar with icons for Willkommen, Editieren, Design, Debug, Projekte, and Hilfe. Below that is a toolbar with icons for QtProject, Debug, and Run/Stop. The main window has a menu bar with Datei, Bearbeiten, Erstellen, Debuggen, Analyse, Extras, Fenster, and Hilfe. The central area shows a project tree for 'QtProjekt [master]' containing 'QtProjekt.pro' and 'Quelldateien/main.cpp'. The code editor on the right contains the following 'main.cpp' code:

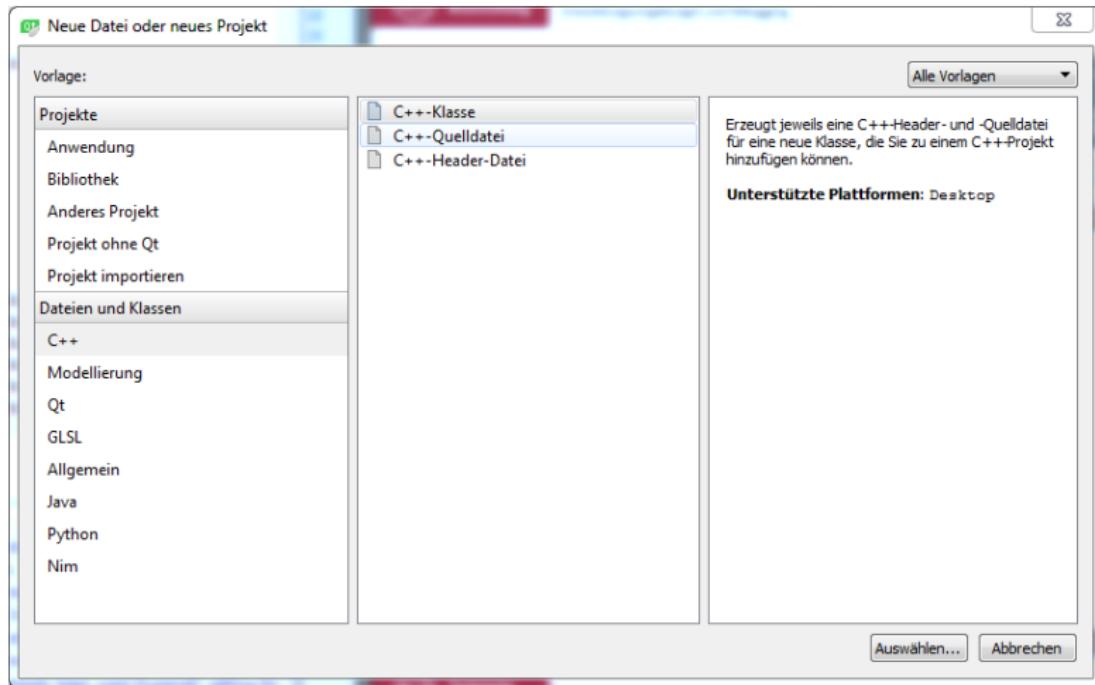
```
#include <iostream>
using namespace std;
int main(int argc, char *argv[])
{
    cout << "Hello World!" << endl;
    return 0;
}
```

At the bottom, there is a search bar labeled 'Suchmuster (Strg+K)' and several status indicators: Build-P..., Sucher..., Ausga..., Kompili..., and Debug....



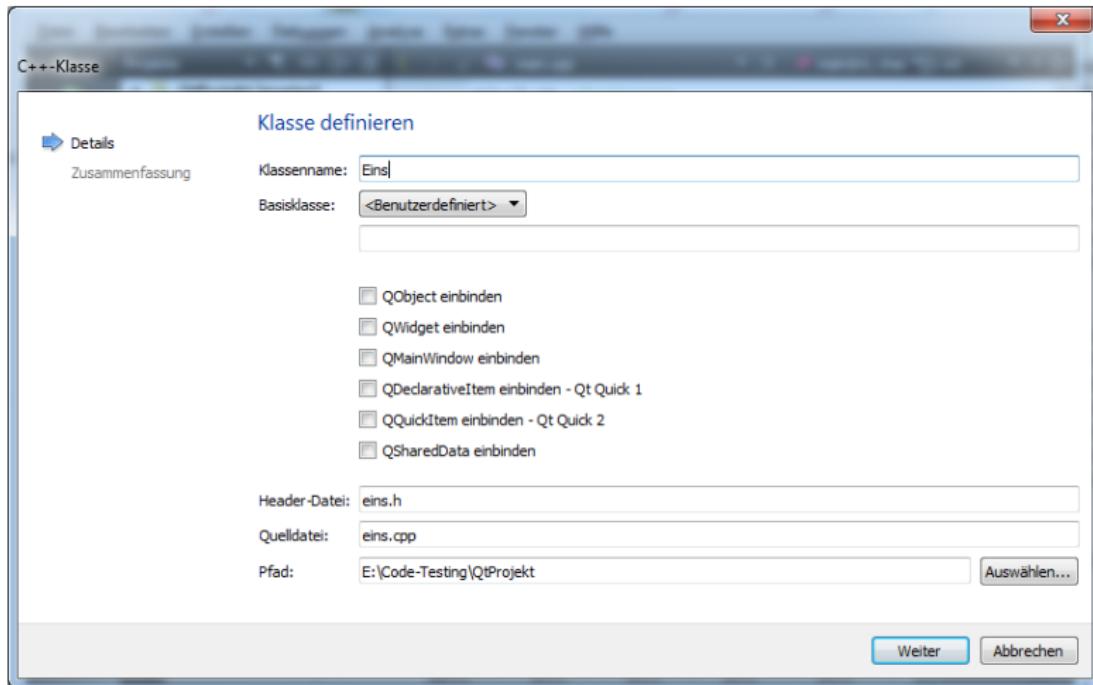
Beispiel

Erstes Qt-Projekt



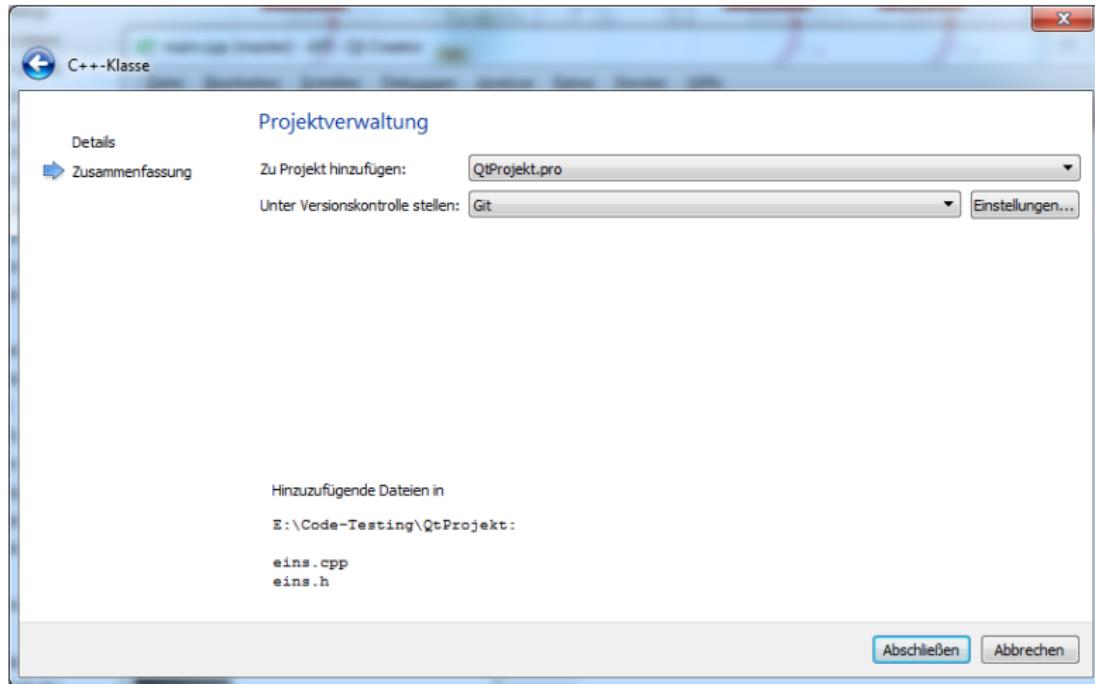
Beispiel

Erstes Qt-Projekt



Beispiel

Erstes Qt-Projekt



Beispiel

Erstes Qt-Projekt

The screenshot shows the Qt Creator interface with the following details:

- Title Bar:** QtProjekt.pro [master] - API - Qt Creator
- Menu Bar:** Datei, Bearbeiten, Erstellen, Debuggen, Analyse, Extras, Fenster, Hilfe
- Toolbars:** Willkommen, Editieren, Design, Debug, Projekte, Hilfe
- Project Tree:** QtProjekt [master] (selected), QtProjekt.pro, Header-Dateien (eins.h), Quelldateien (eins.cpp, main.cpp)
- Code Editor:** QtProjekt.pro (content shown below)
- Code Content:**

```
1 TEMPLATE = app
2 CONFIG += console c++11
3 CONFIG -= app_bundle
4 CONFIG -= qt
5
6 SOURCES += main.cpp \
7     eins.cpp
8
9 HEADERS += \
10     eins.h
11
```
- Bottom Bar:** Suchmuster (Strg+K), Build-P..., Sucher..., Ausga..., Kompili..., Debug...



Beispiel

Erstes Qt-Projekt

The screenshot shows the Qt Creator interface with the following details:

- Title Bar:** eins.h [master] - API - Qt Creator
- Menu Bar:** Datei, Bearbeiten, Erstellen, Debuggen, Analyse, Extras, Fenster, Hilfe
- Toolbars:** Willkommen, Editieren, Design, Debug, Projekte, Hilfe, QtProjekt, Debug, Run, Stop, Build.
- Project Explorer:** Shows the project structure:
 - QtProjekt [master]
 - QtProjekt.pro
 - Header-Dateien
 - eins.h
 - Quelldateien
 - eins.cpp
 - main.cpp
- Code Editor:** Displays the contents of eins.h:

```
1 ifndef EINS_H
2 define EINS_H
3
4 class Eins
5 {
6 public:
7     Eins();
8 };
9
10 endif // EINS_H
```
- Bottom Bar:** Suchmuster (Strg+K), Build-P..., Sucher..., Ausga..., Kompili..., Debug...



Teil III

Debugging

Verwendung des Begriffs Bug



Abbildung 4: Erster echter Bugreport
 Public Domain
<http://commons.wikimedia.org/wiki/File:H96566k.jpg>

- Schon Ende des 19. Jahrhunderts gebräuchlich
- Vorstellung von kleinen Tieren die an Telefonleitungen knabbern
- Vorfall am Mark II Aiken Relay Calculator
- Verbreitung der Bezeichnung durch Grace Hopper

Softwarefehler

Definition

Ein Softwarefehler führt während des Programmablaufs zu einem Abweichen von der Spezifikation.

- Komplexe Software ist in der Regel nie fehlerfrei
- Mathematische Verifizierung ist nur bei einfachen Programmen praktikabel
- Fehlerfreiheit der Software geht von fehlerfreien Spezifikationen aus
⇒ Qualitätsmanagement hilft, Softwarefehler zu reduzieren



Kategorisierung von Softwarefehlern

Zeitpunkt

- Zur Kompilierzeit
- Zur Laufzeit
- Zeitpunkt während der Laufzeit



Kategorisierung von Softwarefehlern

Zeitpunkt

- Zur Kompilierzeit
- Zur Laufzeit
- Zeitpunkt während der Laufzeit

Ursache

- Syntax
- Softwaredesign
- Nutzereingabe
- Netzwerk-synchronisation
- Parallelität



Kategorisierung von Softwarefehlern

Zeitpunkt

- Zur Kompilierzeit
- Zur Laufzeit
- Zeitpunkt während der Laufzeit

Ursache

- Syntax
- Softwaredesign
- Nutzereingabe
- Netzwerk-synchronisation
- Parallelität

Auswirkung

- Bedienung erschwert
- Nicht-Erfüllung der Anforderungen
- Endlosschleife
- Programmabsturz
- Einfrieren



Mars Climate Orbiter

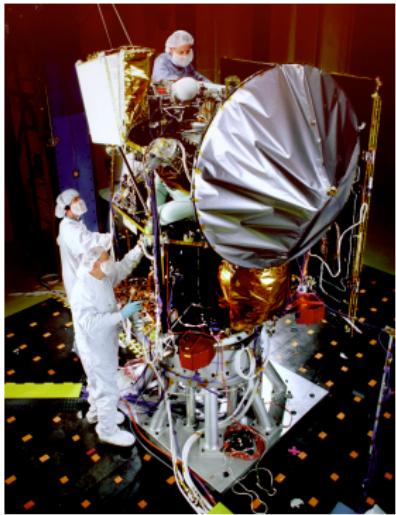


Abbildung 5: Mars Climate Orbiter
Public Domain
[http://commons.wikimedia.org/
wiki/File:Mars_Climate_Orbiter_
during_tests.jpg](http://commons.wikimedia.org/wiki/File:Mars_Climate_Orbiter_during_tests.jpg)

- Marssonde der NASA im Rahmen des Discovery-Programms
- Kurskorrekturen bei Marsannäherung
- NASA-Berechnung in [Newton · Sekunde]
- Lockheed-Martin-Annahme in [Pound-force · Sekunde]

Ergebnis

Totalverlust der Sonde bei Eintritt in die Marsatmosphäre

Ariane V88



- Erstflug der Ariane 5 (ESA-Trägerrakete)
- Komponenten aus der Ariane 4 wurden ungeprüft übernommen
- Speicherüberlauf führte zu Versand von Statusdaten statt Messdaten
- Steuersystem misinterpretiert Statusdaten

Ergebnis

Selbstzerstörung der Rakete

Abbildung 6: Ariane V88

Phrd/de:user:Stahlkocher, CC BY-SA 3.0
[http://commons.wikimedia.org/
wiki/File:Ariane_501_Cluster.svg](http://commons.wikimedia.org/wiki/File:Ariane_501_Cluster.svg)



Technische
Universität
Braunschweig

Heartbleed

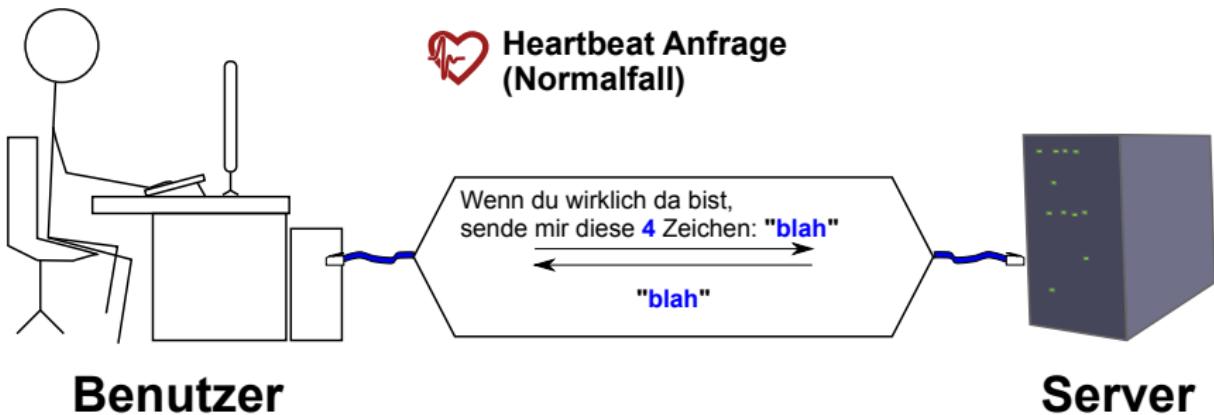


Abbildung 7: vgl. Heartbleed Bug

SomeUser953, Patrick87, CC BY-SA 3.0

http://commons.wikimedia.org/w/index.php?title=File:Heartbleed_bug_explained.svg&lang=de

Heartbleed

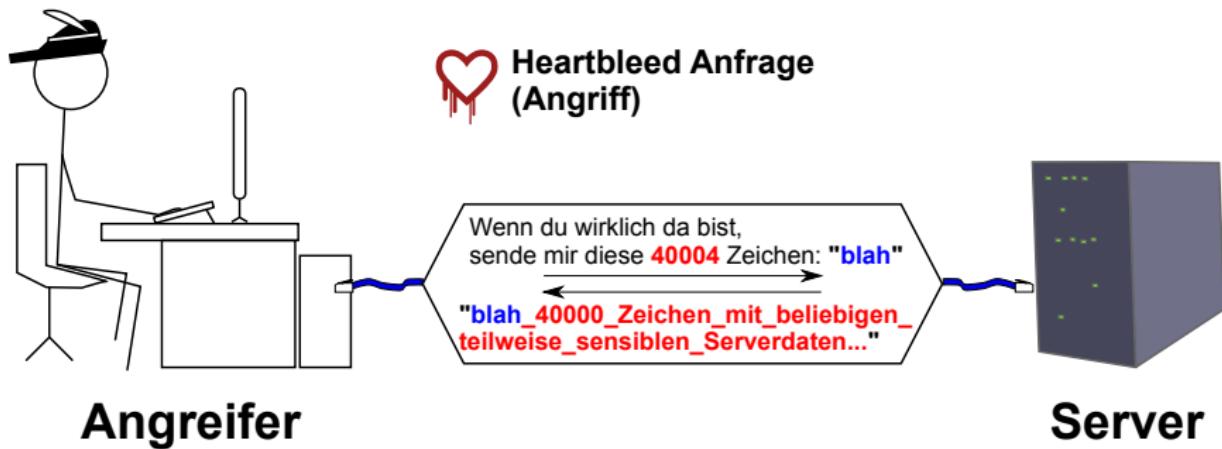


Abbildung 7: vgl. Heartbleed Bug

SomeUser953, Patrick87, CC BY-SA 3.0

http://commons.wikimedia.org/w/index.php?title=File:Heartbleed_bug_explained.svg&lang=de

Fehlerdiagnose

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

Listing 3:

```
1 cout << "Wert der Variable:" << irgendeineVariable << endl;
```



Fehlerdiagnose

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

Listing 4:

```
1 aufrufFunktionA();  
2 aufrufFunktionB();  
3 aufrufFunktionC();  
4 //aufrufFunktionD();  
5 //aufrufFunktionE();  
6 //aufrufFunktionF();
```



Fehlerdiagnose

- Ungewöhnliche Eingaben testen
 - Variablen auf Konstante initialisieren
 - Robustheit gegenüber Grenzwerten

Listing 5:

```
1 int testZahl;  
2  
3 cin >> testZahl;  
4  
5 funktionDieZahlErwartet(testZahl);
```



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
 - Inspektion von Variablen
 - Modifikation von Speicher



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, ...)

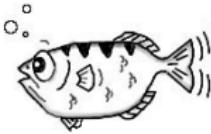


Abbildung 8: GDB-Debugger Logo¹

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 ruft welche Funktion auf?
- Nutzer kann mit Debugger Programmausführung anhalten und wieder aufnehmen
- *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



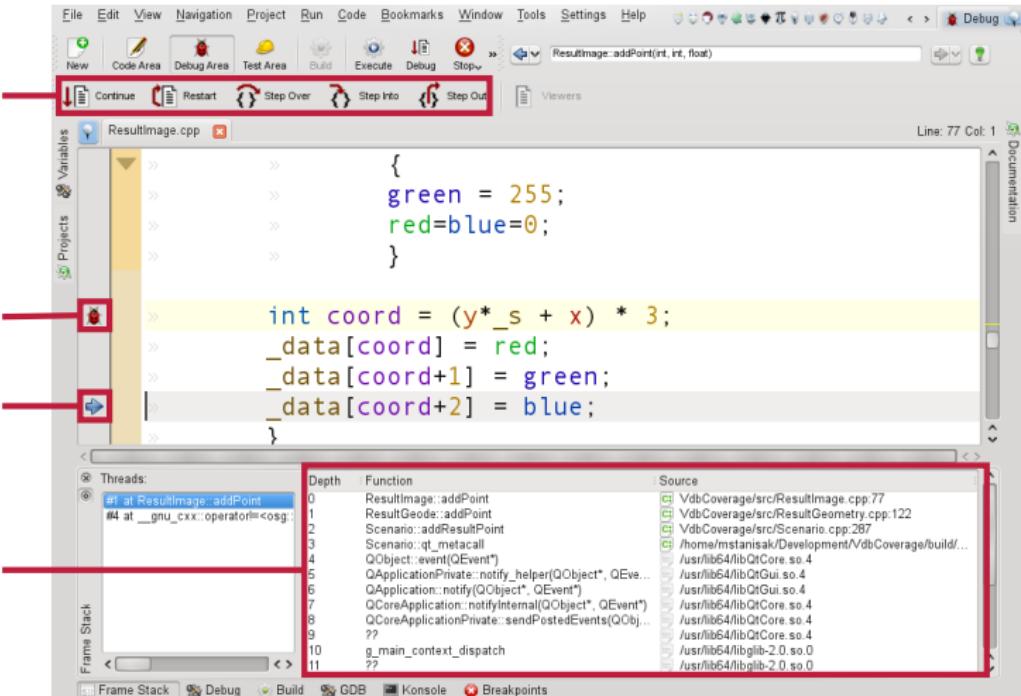
Continue:

Ausführung fortsetzen



Beispiel: Debugging in KDevelop

Ablaufsteuerung



Breakpoint
Programm-
zeiger

Callstack

Beispiel: Debugging in Microsoft Visual Studio

Ablaufsteuerung

Breakpoint

Programmzeiger

Callstack

TriPos (Debugging) - Microsoft Visual C++ 2010 Express

Projektmappe... main.cpp

```
(Globaler Gültigkeitsbereich)
main(int argc, char * argv[])
{
    #ifdef WIN32
        resize_term(30, 100);
    #endif

    running = true;
    while(running)
    {
        update();
    }

    delete smoother; smoother = NULL;
    delete receiver; receiver = NULL;
    delete pool; pool = NULL;
}
```

Aufrüstsliste

Name	Wert	Typ
running	true	volatile
GNSSDiagnosticsD.exe!main(in C++)		
GNSSDiagnosticsD.exe!_tmain(C)		
GNSSDiagnosticsD.exe!mainC(F C)		



Debug- und Release-Versionen

Debug

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

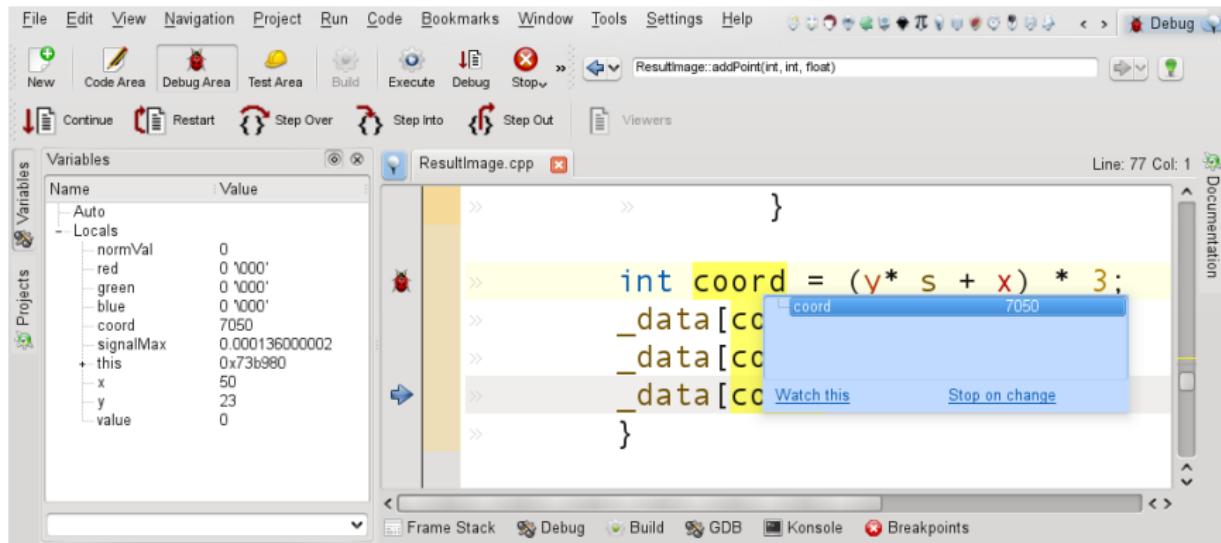
Release

- Optimierungen durch Compiler und Linker
⇒ kein Rückschluss von Quelltextzeile auf Ablauffortschritt möglich
⇒ 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



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
- Höchste Warnstufe des Compilers aktivieren
- Überprüfungen im Debug-Modus durch Zusicherungen (Assertions)
 - Null-Pointer erkennen: `assert(pointer != NULL);`
 - Bei Fehler im Debug-Modus: Meldungsausgabe und Programmende
 - Verhinderung von Null-Pointer-Zugriffsfehler



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 an das Design nötig

5. Testen

Tritt der Fehler immer noch auf?



Schwierigkeit: Fehler bestimmen

- Manche Probleme 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 Programmcode ist Erkennen von Bugs sehr aufwändig
- Fehler zu erkennen benötigt eine gewisse Übung



Beispiel: Unterschied Release-/Debug-Version

- Ein Bug tritt nur im Release-, nicht im Debug-Modus auf
 - Entwickler programmiert in Debug-Version
 - Qualitätssicherung verwendet Release-Version
- Mögliche Ursache: Nicht initialisierte Variable
 - Debug: Initialisierung aller Variablen mit 0
 - Release: Aus Performancegründen keine Vorbelegung
 - Lösung: Variablen bei Deklaration mit Wert vorbelegen
- Mögliche Ursache: Optimierungen
 - Compiler und Linker optimieren den Maschinencode, um Geschwindigkeit zu erhöhen
 - Manchmal führt Optimierung bei komplexem Code zu Fehlverhalten
 - Lösung: Optimierungen reduzieren, Code vereinfachen



Teil IV

Projektarbeit



Teil C API-Spiralmodell fortsetzen

Aufgabe 1

- Setzen Sie die Bearbeitung der Ziele aus Teil A, des von Ihnen definierten Spiralmodells, fort.

⇒ Wichtig für die kommende Woche



Aufgabe 2

- Erstellen Sie zu jeder Anforderung einen Eintrag auf der Wiki-Seite Pflichtenheft
- Ein Eintrag soll die Umsetzung der referenzierten Anforderung beinhalten und kann die folgenden Elemente enthalten:
 - Beschreibung als Text
 - Ablaufdiagramm
 - Klassen- und Objektdiagramm
 - Aktivitätsdiagramm
 - Kollaborationsdiagramm
 - Sequenzdiagramm



Fragen?

Gibt es noch Fragen?

