Materialpaket 09_DSGN – Design

C/C++, Autor: Prof. Dr.-Ing. Carsten Link

Version 1.3.1 March 3, 2019

Contents

1 Kompetenzen und Lernegebnisse

Durch das Bearbeiten dieses Materialpaketes erwerben Sie diese Kompetenzen (Wissen, Fähigkeiten, Fertigkeiten zur Problemlösung):

Sie können ein bestehendes C++-Projekt kritisch analysieren, übernehmen und erweitern.

Die oben genannten Kompetenzen erwerben Sie, indem Sie Lernziele erreichen, welche sich prüfen lassen. Lernegebnisse: Sie können nachweislich[1]:

- ein fremdes C++-Projekt mittlerer Komplexität analysieren bzgl.
  - der Verwendung von Idiomen und Patterns
  - Verständlichkeit und Wartbarkeit
  - Einhaltung von *style guides*

[1] Sie können das Erzielen der einzelnen Lernergebnisse beispielsweise bei einem Testat im Praktikum oder einer Aufgabe in der Modulprüfung nachweisen

- ein fremdes C++-Projekt mittlerer Komplexität erweitern

2 Konzepte

Im Folgenden wird auf Grundsätze zum Programmierstil eingegangen. Des weiteren soll ein bestehendes Projekt mittlerer Komplexität analysiert und modifiziert werden.

2.1 Programmierstil

Während sich die Sprachunabhängigen Design Patterns mit Architektur beschäftigen, sind Idiome Sprachspezifische Muster. Regeln für den Programmierstil setzen noch eine Ebene tiefer an: die Art und Weise, wie Quelltext im Detail formuliert werden soll. Hier habe sich viele Konventionen entwickelt, die es leichter machen, fremden C++-Quellcode zu lesen. Dem Programmierer sollte klar sein, dass Quellcode viel häufiger gelesen als geschrieben wird.

Bjarne Stroustrup und Herb Sutter pflegen die *C++ Core Guidelines*[2]. Dieser Katalog ist sehr umfangreich. Daher wird hier verwiesen auf den Foliensatz von O. Maudal – dort gibt er 34 Richtlinien zur Diskussion[3], von denen die meisten unter den C++-Programmierer akzeptierter Konsens sind (die originale Nummerierung beginnt bei Null):

0. Show that you care. (Or: Do sweat the small stuff)
1. always compile with -Wall and -Werror
2. always use tools to support the building process
3. do not *prefix member variables, use postfix* if you must
4. public stuff should be declared first, then the private stuff
5. single argument constructors should usually be explicit
6. initialize the state of the object properly
7. use a consistent naming convention, camelCase or under_score
8. do not prefix queries and modifiers with get/set
9. do not import namespaces
10. query functions should be declared const
11. non-const functions are modifiers
12. prefer free-standing functions
13. use anonymous namespaces for private free-standing functions
14. do not inline stuff in the class definition
15. by default keep your stuff in the implementation file
16. avoid member functions that both modifies and queries
17. default arguments are depreciated, use delegation if you must

[2] https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md
[3] http://www.pvv.org/~oma/CPPIdiomsByExample_Nov2008.pdf und https://de.slideshare.net/olvemaudal/cpp-idioms-byexamplenov2008

18. the K&R vs BS war is over, use an extra space around & and *
19. by not specifying const you say that something will change
20. reduce scope of variables
21. for-loops in C++ are often not written like this
22. in C++ you do not need to explicitly return from main
23. inject side-effects if you must have them
24. make sure headers compile by itself
25. include your own header file first, standard libraries last
26. prefer forward declarations in header files
27. don't need braces here
28. avoid side-effects if you can, prefer free-standing functions
29. do not open a namespace when implementing free-standing functions
30. operator overloading is sometimes a nice thing
31. namespaces usually corresponds to directories, and vice versa
32. use include guards in header files
33. real professionals indent by four spaces

2.2 CppRobots

Das Projekt CppRobots[4] basiert auf der Idee des alten Programmiererspiels C-Robots[5]. Hierbei spielen nicht Spieler gegeneinander, sondern Programme, die von den Spielern geschrieben wurden. In einem Spiel treten sogenannte *Agents* gegeneinander an. Ziel ist es, möglichst viele andere Agents zu besiegen und selbst nicht besiegt zu werden.

[1]

3 Material zum aktiven Lernen

Da eine Programmiersprache nur durch aktive Verwendung erlernt werden kann, werden im folgenden Aufgaben zum praktischen Üben vorgestellt. Zunächst wird ein Grundgerüst (C++Programm) erstellt, welches dann auf mehrere Arten Modifiziert wird. Insbesondere die Modifikationen ermöglichen es dem Lernenden (und auch dem Lehrenden), die Qualität des Kompetenzerwerbs bzgl. dieses Materialpakets bewerten zu können.

3.1 Aufgabe: Grundgerüst

Laden Sie Quellcode von CppRobots auf ihren Rechner, compilieren und starten Sie ihn.

[4] https://github.com/braak/CppRobots
[5] http://crobots.deepthought.it/home.php

Identifizieren Sie im Quellcode

- Patterns
- Idiome
- Style Guide
- besonders elegante Code-Stellen

3.2 Aufgabe: Modifikationen

Stellen Sie sicher, dass Sie jede einzelne der nachfolgenden Modifikationen innerhalb weniger Minuten (5 - 10) vor Zuschauern (Testatsituation) umsetzen können. Konkret sollen Sie im Testat in der Lage sein, das gegebene Grundgerüst um mindestens eine zufällig ausgewählte Modifikation zu erweitern. Bereiten Sie dazu auf ihrer Arbeitsumgebung ein Verzeichnis vor, welches ausschließlich das Grundgerüst enthält.

Modifikationen:

1. Erweitern Sie das Programm um einen weiteren Agenten
2. Fügen Sie einen weiteren Testfall hinzu

3.3 Verständnisfragen

Nach Bearbeitung des Kapitels "Konzepte", der Erstellung des Grundgerüsts sowie dem Üben der Modifikationen sollten Sie in der Lage sein, die folgenden Fragen zu beantworten.

1. Welche Nachteile können sich ergeben, wenn Sie sich nicht an *always compile with -Wall and -Werror* halten?
2. Welche Nachteile können sich ergeben, wenn Sie sich nicht an *single argument constructors should usually be explicit* halten?
3. Welche Nachteile können sich ergeben, wenn Sie sich nicht an *6. initialize the state of the object properly* halten?
4. Welche Nachteile können sich ergeben, wenn Sie sich nicht an *7. use a consistent naming convention, camelCase or under_score* halten?
5. Welche Nachteile können sich ergeben, wenn Sie sich nicht an *9. do not import namespaces* halten?
6. Welche Nachteile können sich ergeben, wenn Sie sich nicht an *10. query functions should be declared const* halten?
7. Welche Nachteile können sich ergeben, wenn Sie sich nicht an *14. do not inline stuff in the class definition* halten?
8. Welche Nachteile können sich ergeben, wenn Sie sich nicht an *15. by default keep your stuff in the implementation file* halten?
9. Welche Nachteile können sich ergeben, wenn Sie sich nicht an *19. by not specifying const you say that something will change* halten?

10. Welche Nachteile können sich ergeben, wenn Sie sich nicht an *20. reduce scope of variables* halten?
11. Welche Nachteile können sich ergeben, wenn Sie sich nicht an *32. use include guards in header files* halten?

4 Nützliche Links

- Bjarne Stroustrup's C++ Style and Technique FAQ http://www.stroustrup.com/bs_faq2.html

5 Literatur

- [PPP] Stroustrup, Bjarne: Programming - Principles and Practice using C++
- [TCPL] Stroustrup, Bjarne: The C++ Programming Language, Fourth Edition