Materialsammlung - generische Programmierung

Prof. Dr. Hans-Jürgen Buhl

Wintersemester 2018/2019

Fachgruppe Mathematik und Informatik
Fakultät 4 — Mathematik und Naturwissenschaften
Bergische Universität Wuppertal

Praktische Informatik
PIBUW - WS 2018/19
Oktober 2018
8. Auflage, 2018

Inhaltsverzeichnis

ConceptsLite (C++20) . . . 3
Default g++-Versionen verschiedener Linux-Distributionen . . . 5
Listen-Initialisierung . . . 6
Compilezeit-Funktionen . . . 7
Compiler-Optionen für C++17f. . . . 8
Physikalisch/technische Maßeinheiten in C++ . . . 10
User defined Literals . . . 11
Structured bindings . . . 15
Generische lambda-Ausdrücke . . . 16
Ergänzende Operationen für std::ratio . . . 17
(Weitere) Template-Metaprogrammierung . . . 18
Typlisten . . . 18
Promote-Type-Metafunktion . . . 19
Template-Template-Parameter . . . 21
Policy based Design . . . 22
Aspect Oriented Programming . . . 23
Literatur . . . 24
Einordnung in die Programmierparadigmen . . . 25
Einleitung . . . 27

1. Generische Programmierung **29**
1.1. Was ist generische Programmierung? . . . 29
1.2. Beispiel einer generischen Funktion mit einem generischen Parameter . . . 30
1.3. weitergehende Tests von `mean2(.,.)` . . . 31
1.4. Einsatzgebiete/Beispielrepositorien für generische Konstrukte: STL, ... 32
1.5. Instanzen generischer Objekte . . . 33
1.5.1. Objekt-Dateien *.o: ldd, nm und c++filt . . . 33
1.5.2. Erstellen und Benutzen von statischen Bibliotheken . . . 45
1.5.3. Erstellen und Benutzen einer „shared object“- Bibliothek . . . 48
1.5.4. Bibliotheksmanagement, dynamically loaded libraries . . . 49
1.6. STL-Templatequellen und -sourcen unter SuSE, zeilenweise Debuggen . . . 50
1.7. Automatisch überprüfte Requirements an Template-Parameter . . . 52
1.7.1. Mit Hilfe des c++11-Modus des g++ . . . 54
1.8. „horrible error messages“ bei STL-Nutzung . . . 56
1.9. Erfragung der Eigenschaften aktueller generischer Parameter . . . 58
1.9.1. C++11 type_traits . . . 59
1.9.2. BOOST type_traits . . . 60

1.9.3. is_arithmetic, true_type and false_type . . . 61
1.9.4. numeric_limits als Typ-Abbildung . . . 61
1.10. Rückblick: typsichere Funktionsbenutzung . . . 63
1.11. Concepts für zielführende knappe Fehlermeldungen: 2020? . . . 63
1.12. Zielgerichtete Fehlermeldungen bei der Standardbibliothek mit Konzepten 67
1.13. Orte, wo statische Zusicherungen benutzt werden . . . 68
1.14. statische Zusicherunge in C++11 . . . 68
1.15. Fehlermeldungen bei uneingeschränkter Generizität (Fortsetzung von 1.8) 69
1.16. Verbesserte Fehlermeldungen bei Nutzung von StaticAssert . . . 72
1.16.1. RandomAccessIterator . . . 72
1.16.2. Nicht instanziierbare Klassen . . . 75
1.16.3. Erzwingung gleicher Typen . . . 76
1.16.4. Funktionen mit „(int/float/...) type promotion“-Returntyp . . . 77
1.16.5. Auf Unterklassen eingeschränkte Generizität . . . 78
1.16.6. g++ type_traits Compiler Extensions . . . 78
1.16.7. Type Traits in D . . . 79
1.16.8. C++ *has_member* fehlt . . . 81
1.16.9. SFINAE . . . 82
1.16.10. C++11: Traits mit decltype statt sizeof()-Tricks . . . 84
1.16.11. Überladene Templatefunktionen/bedingte Template-Spezifikationen 85
1.16.11.1. enable_if-Funktionen . . . 85
1.16.11.2. Konflikt beim enable_if-Funktionsüberladen . . . 87
1.16.11.3. bedingte „template class specializations“ . . . 88
1.17. Template-Deklarationen für eine Sammlung von Template-Instanzen . . . 90
1.18. Wo ist die Template-Instanz? . . . 91
1.19. C++11 extern template . . . 91
1.20. Generic Programming . . . 92
1.21. C++14: Generic lambdas, Lambda capture expressions . . . 93
1.22. C++17: Neues bezüglich generischer Konstrukte? . . . 93
1.23. Workaround: (explizite) Nutzung von Typetraits statt von Concepts . . . 93
1.24. Assoziierte Typen, Tags, Tag-Dispatching . . . 95
1.25. Generic Programming Techniques of the BOOST Libraries . . . 97
1.26. POD-Typen und trait-fallweises Überladen . . . 98
1.27. Eigene Klassen-Tags und Tag-Dispatching oder fallweise Spezialisierung . 99
1.28. Iteratoren . . . 99
1.29. Programmieren mit Konzepten . . . 100

2. Metaprogrammierung **103**
2.1. Metafunktionen . . . 103
2.2. Metafunktionen in /usr/include/c++/4.7/type_traits und Feldlängen . . . 105
2.3. Factorial, Combinations, IF, id, add und die Rekursion statt der Schleife 108
2.4. Rechnende Compiler: . . . 110
2.5. Typfunktionen: längerer Datentyp, IfThenElse-Werte . . . 111
2.6. Template Nontype Parameter . . . 112

2.7. Compilezeit-Fehlermeldungen in constexpr-Metafunktionen 113
2.8. C++11 Metaprogramming Examples 115
2.9. Fortgeschrittene Metaprogrammierung 120
2.9.1. DSL-Extensions: C++11 Compile-time rational arithmetic 120
2.9.2. Unrolled Loops: Durch Rekursion wegoptimierte Schleifen 122
2.9.3. Expression templates 125
2.10. Vor- und Nachteile der Metaprogrammierung 127
2.11. Die BOOST Metaprogramming Library MPL 128
2.12. Metaprogramme für die Manipulation von Typen in C++ 128
2.13. Spracherweiterung (DSL) Maßeinheiten 129
2.13.1. Ein Bug und sein Einfluß auf neue Programmiersprachen 129
2.13.2. DSLs 132
2.13.3. Units and Measure in F# 132
2.13.4. SI-Einheitssystem 133
2.13.5. Boost.Units 133
2.13.6. Erweiterung des C++-Typsystems um Units 143
2.13.7. Nachteile von DSLs 146
2.14. User-Defined Literals for Scientific Quantities, ... 146
2.15. Literaturhinweise zum Metaprogrammieren 146

3. Template template-Parameter, Policy-basiertes Klassendesign 147
3.1. Templates als Template-Parameter 147
3.2. Policies (Strategien, Entscheidungen, Implementierungsvarianten) 148
3.3. Beispiel: polare oder karth. Koordinaten/long double, double oder float . 152
3.4. Entwurfsmuster Strategie 154
3.5. Policies als Template Template-Parameter 155
3.6. Orthogonale Policy-Dimensionen 156
3.7. Policies (Fortsetzung) 156
3.8. Loki 156
3.9. A Policy-Based flex_string Implementation 157

4. AOP in komplexen Unternehmensanwendungen 159

A. Quelloffenes Eclipse mit CDT/UML2.5/OCL2.4-Tools i

B. Ausblick xiii

Abbildungsverzeichnis

1.1. Die Phasen der Compilation 33

Tabellenverzeichnis

0.1. C++ Requirements DefaultConstructible, ... 27

Vorbemerkungen:

ConceptsLite (C++20)

```
//============================================================================
// Name        : C++ConceptsLite.cpp
// Description : Example for Concepts Lite, ab g++ 6 mit -fconcepts
//============================================================================
#include <iostream>
#include <cstdlib>
#include <cassert>
#include <type_traits>
// using namespace std;

/*
template <typename T>
concept constexpr bool CopyConstructible() {
        return requires(T t){
            { T(t) };
            { t.~T() };
        };
}
*/

template <typename T>
concept constexpr bool CopyConstructible =
            std::is_copy_constructible<T>::value &&
            std::is_destructible<T>::value;

template <typename T>
concept constexpr bool NullConstructible(){
        return requires(T t){
            { T(0) };
        };
}

template <typename T, typename U = T>
concept constexpr bool Addable(){
        return requires(T t, U u){
          { t + u } -> decltype(t + u);
        };
}

template <CopyConstructible T>
   requires NullConstructible<T>() && Addable<T>()
T sum(T array[], int n)
```

```
{
        T result {0};
        for (int i = 0; i < n; ++i)
                result = result + array[i];
        return result;
}

int main(int argc, char* args[]) {
        std::cout << "Testrahmenprogramm:" << std::endl;
        int arr1[] {1, 2, 3};
        std::cout << sum(arr1, 3) << std::endl;
        assert(sum(arr1, 3) == 6);
        return EXIT_SUCCESS;
}
```

```
g++ -std=c++1z -O0 -g3 -Wall -c -fmessage-length=0 -Wno-attributes -Wno-attributes -fconcepts -MMD -MP -MF"src/NewC++.d" -MT"
../src/NewC++.cpp: In function 'int main()':
../src/NewC++.cpp:66:26: error: cannot call function 'T sum(T*, int) [with T = MyClass]'
  std::cout << sum(arr1, 3) << std::endl;
                          ^
../src/NewC++.cpp:38:3: note:   constraints not satisfied
 T sum(T array[], int n)
   ^~~
../src/NewC++.cpp:19:24: note: within 'template<class T> concept const bool CopyConstructible<T> [with T = MyClass]'
 concept constexpr bool CopyConstructible =
                        ^~~~~~~~~~~~~~~~~
../src/NewC++.cpp:19:24: note: 'std::is_copy_constructible<MyClass>::value' evaluated to false
../src/NewC++.cpp:24:24: note: within 'template<class T, class U> concept bool Addable() [with T = MyClass; U = MyClass]'
 concept constexpr bool Addable(){
                        ^~~~~~~
../src/NewC++.cpp:24:24: note:     with 'MyClass t'
../src/NewC++.cpp:24:24: note:     with 'MyClass u'
../src/NewC++.cpp:24:24: note: the required expression '(t + u)' would be ill-formed
../src/NewC++.cpp:31:24: note: within 'template<class T> concept bool NullConstructible() [with T = MyClass]'
 concept constexpr bool NullConstructible(){
                        ^~~~~~~~~~~~~~~~
../src/NewC++.cpp:31:24: note:     with 'MyClass t'
../src/NewC++.cpp:31:24: note: the required expression '(T)(0)' would be ill-formed
In file included from /usr/include/c++/7/cassert:44:0,
                 from ../src/NewC++.cpp:12:
```

Constraints and concepts
Named requirements
Concepts library (C++20)
Concepts of experimental::ranges
Concepts-Lite
Concepts Lite: Constraining Templates with Predicates
Concepts definieren
type_traits

Default g++-Versionen verschiedener Linux-Distributionen

g++-Versionen aktueller (September 2018) Linux-Distributionen:

Linux-Distribution	g++-Version
OpenSuse 13.2	4.8.3
Suse Leap 42.2	4.8.5
Suse Leap 42.3	4.8.5
Suse Leap 15.0	7.3.1
Suse Tumbleweed 201807	8.1.1
Ubuntu 16.04	5.5.0
Ubuntu 17.04	7.0.1
Ubuntu 17.10	7.2.0
Ubuntu 18.04	7.3.0
Debian 9	6.3.0
CentOS 7.3	4.8.5
Fedora 26	7.1.1
OpenIndiana Hipster 2017.04	4.8.5

Listen-Initialisierung

```
//============================================================================
// Name        : New3C++.cpp
// Author      : HJB
// Version     :
// Copyright   : PD
// Description : Listen-Initialisierung
//============================================================================

#include <iostream>
#include <cstdlib>
#include <cassert>
#include <type_traits>
#include <list>
// using namespace std;

int main() {
        std::cout << "Testrahmenprogramm:" << std::endl;

        std::list<int> l1 {7, 5, 3};

        /* instead of the sequence:
         l1.push_back(7);
         l1.push_back(5);
         l1.push_back(3);
         */
        assert(none_of(l1.begin(), l1.end(), [](auto v){return v == 4;}));
        // see also: any_of()/all_of();
        return EXIT_SUCCESS;
}
```

std::initializer_list
Constructors and member initializer lists
Why should I prefer to use member initialization list?

list initialization
Initializer list for objects with default constructor

Compilezeit-Funktionen

cPower.cpp

```
2  // Name        : cPower.cpp
8
9  #include <iostream>
10 #include <type_traits>
11
12 template<int n, int m> struct cpower :
13 std::integral_constant<int, n * cpower<n, m-1>::value>
14 {
15   static_assert(m>=2, "negative Potenzen vorerst nicht erlaubt");
16 };
17
18 template<int n> struct cpower<n, 1>:
19 std::integral_constant<int, n>
20 {
21 };
22
23 template<int n> struct cpower<n, 0>:
24 std::integral_constant<int, 1>
25 {
26   static_assert((n!=0), "not defined");
27 };
28
29 int main(){
30
31   std::cout << cpower<3,4>::value << std::endl;
32   std::cout << cpower<0,4>::value << std::endl;
33   std::cout << cpower<5,0>::value << std::endl;
34
35   //std::cout << cpower<0,0>::value << std::endl;
36   // error: static assertion failed: not defined
37
38   //std::cout << cpower<3,-4>::value << std::endl;
39   // error: static assertion failed: negative Potenzen vorerst nicht erlaubt
40
41
42   //double d_array[cpower<3,4>::value];
43   //std::cout << sizeof(d_array)/sizeof(d_array[0]) << std::endl;
44
45 }
46
```

Outl | Mak | Tas

- iostream
- type_traits
- cpower
- cpower<n,1>
- cpower<n,0>
- main() : int

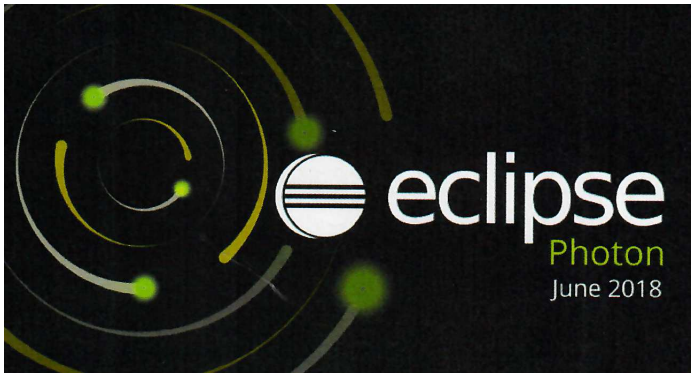
Problems | Tasks | Console | Properties | Valgrind

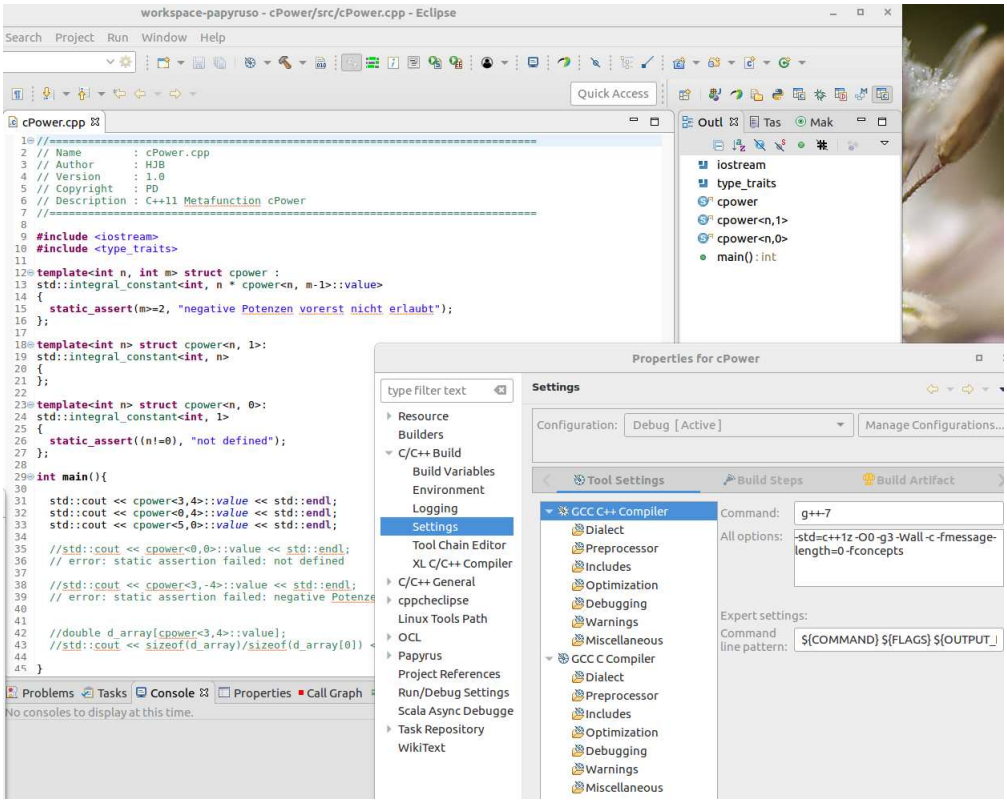
```
CDT Build Console [cPower]
11:38:28 **** Incremental Build of configuration Debug for project cPower ****
make all
Building file: ../src/cPower.cpp
Invoking: GCC C++ Compiler
g++ -std=c++0x -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"src/cPower.d" -MT"src/cPower.d" -o "src/cPower.o" "../src/cPower.cpp
Finished building: ../src/cPower.cpp

Building target: cPower
```

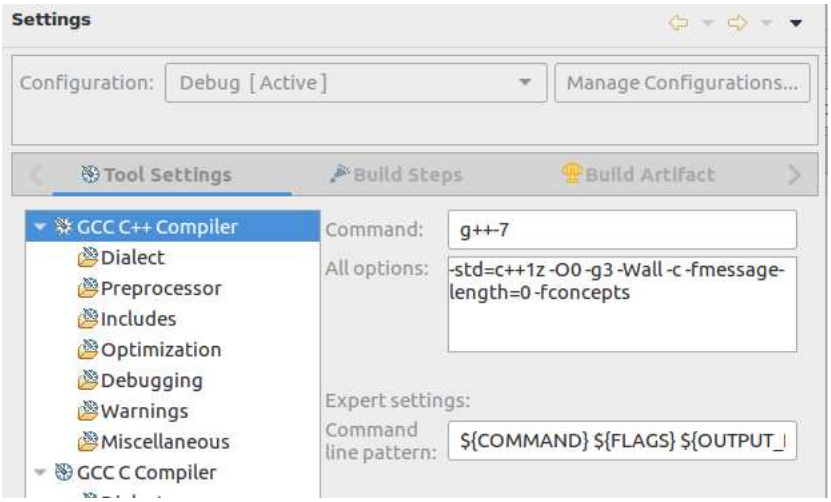
Beispiel zur Metaprogrammierung. Benutzt wird hier die IDE Eclipse, auf den Ausbildungslaborrechnern der Fachgruppe Mathematik und Informatik als `eclipse-papyrusp` aufruf- und benutzbar:

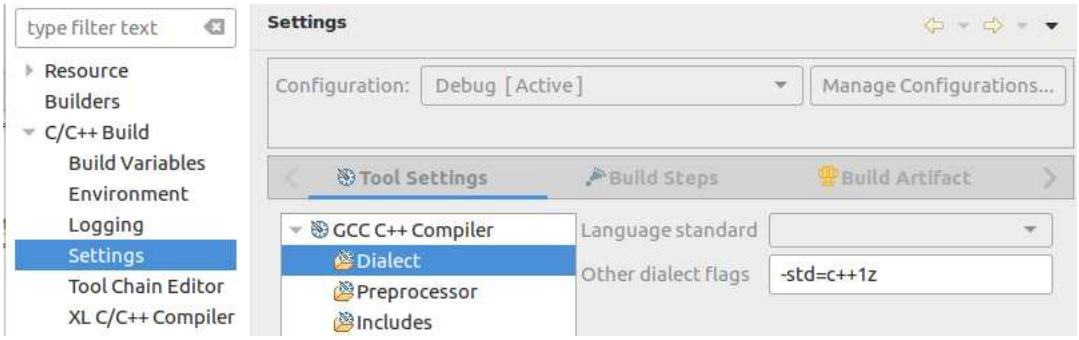
Compiler-Optionen für C++17f.

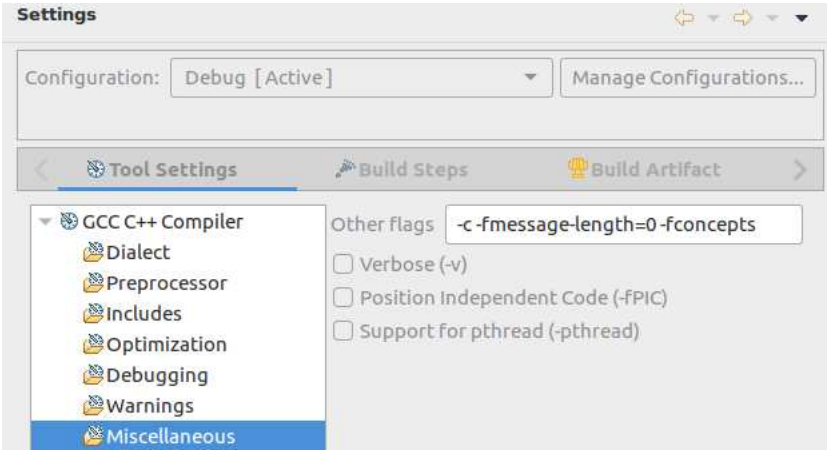




g++-7, `-std=c++1z` und `-fconcepts`







Hinweise zur Installation von Eclipse auf Ihrem eigenen Linux-Rechner finden Sie in Anhang A (Seite i).

Physikalisch/technische Maßeinheiten in C++

Idee aus Jürgen Wolf: C++, Das umfassende Handbuch, Seite 816f.:
`std::chrono::duration<.,.>` zweckentfremdet für allgemeine physikalisch/technische Maßeinheiten:

```
//============================================================
// Name        : MessSkalen.cpp
// Author      : HJB
// Version     : 1.0
// Copyright   : PD
// Description : MessSkalen
//============================================================

#include <iostream>
#include <type_traits>
#include <chrono>

using Meter =  std::chrono::duration<long double, std::ratio<1>>;
using Kilometer = std::chrono::duration<long double, std::kilo>;
using Meilen = std::chrono::duration<long double,
                                     std::ratio<1609344,1000>>;

int main(int argc, char* args[]) {
    std::cout << "Testrahmenprogramm MessSkalen:" << std::endl <<
        std::endl;

    Meter e1{150.0};
    Kilometer e2{e1};
    std::cout << e1.count() <<
        " Meter entspricht " << e2.count() << " Kilometer." << std::endl;
    Meilen e3{e1}; Meilen e4{10.1};
    std::cout << e1.count() << " Meter entspricht " <<
        e3.count() << " Meilen." << std::endl;
    Kilometer e5 = e1 + e4;
    std::cout << e5.count() << " Kilometer, gemischte Arithmetik " <<
        std::endl;
    std::cout << static_cast<Meter>(e5).count() << " Meter, cast" <<
        std::endl;

    return 0;
}
```

User defined Literals

Vollständiger mit überladenem `operator<<` und `operator''''`:

```
//============================================================
// Name        : MessSkalen.cpp
// Author      : HJB
// Version     : 1.0
// Copyright   : PD
// Description : MessSkala-Laenge
//============================================================

#include <iostream>
#include <type_traits>
#include <chrono>

using Laenge =  std::chrono::duration<long double, std::ratio<1>>;
using Meter =  std::chrono::duration<long double, std::ratio<1>>;
using Kilometer = std::chrono::duration<long double, std::kilo>;
using Meilen = std::chrono::duration<long double,
                                     std::ratio<1609344,1000>>;

constexpr Meter operator"" _m(long double d)
{
        return Meter{d};
}
constexpr Kilometer operator"" _km(long double d)
{
        return Kilometer{d};
}

std::ostream& operator<<(std::ostream& os, const Laenge& l)
{
        os << l.count() << " Meter";
        return os;
}
std::ostream& operator<<(std::ostream& os, const Kilometer& l)
{
        os << l.count() << " Kilometer";
        return os;
}
std::ostream& operator<<(std::ostream& os, const Meilen& l)
{
        os << l.count() << " Meilen";
        return os;
}

int main(int argc, char* args[]) {
    std::cout << "Testrahmenprogramm MessSkala-Laenge:" << std::endl <<
        std::endl;

    Laenge e1{150.0_m};
    Kilometer e2{250.0_km};
```

```
    std::cout << e1 << std::endl;
    std::cout << e2 << std::endl;
    Laenge e3 = e1 + e2;
    std::cout << std::chrono::duration_cast<Kilometer>(e3) << std::endl;
    std::cout << e3 << std::endl;
    std::cout << std::endl << std::endl;


    Meilen e4{e1};
    Meilen e5{6.3};
    std::cout << e2 << " entspricht " << e4 << std::endl;
    Kilometer e6 = e2 + e5;
    std::cout << e6 << ", gemischte Arithmetik " << std::endl;
    std::cout << std::chrono::duration_cast<Meter>(e6).count() <<
        " Meter, cast" << std::endl;
    std::cout << std::chrono::duration_cast<Meilen>(e6) <<
        ", cast" << std::endl;

        return 0;
}
```

Aufgabe: Wandeln Sie das Programm für die Umwandlung von Währungskursen ineinander ab.
Aufgabe 2: Vergleichen Sie mit Handling Scientific Quantities (M. Semenov):

```
template <int M, int L, int T>
Quantity<M,L,T> operator+(const Quantity<M,L,T>& lhs, const Quantity<M,L,T
    >& rhs)
{
    return Quantity<M,L,T>(lhs)+=rhs;
}
template <int M, int L, int T>
Quantity<M,L,T> operator-(const Quantity<M,L,T>& lhs, const Quantity<M,L,T
    >& rhs)
{
    return Quantity<M,L,T>(lhs)-=rhs;
}
template <int M1, int L1, int T1, int M2, int L2, int T2>
Quantity<M1+M2,L1+L2,T1+T2> operator*(const Quantity<M1,L1,T1>& lhs, const
     Quantity<M2,L2,T2>& rhs)
{
    return Quantity<M1+M2,L1+L2,T1+T2>(lhs.getValue()*rhs.getValue());
}
template <int M, int L, int T>
Quantity<M,L,T> operator*(const double& lhs, const Quantity<M,L,T>& rhs)
{
    return Quantity<M,L,T>(lhs*rhs.getValue());
}

template <int M1, int L1, int T1, int M2, int L2, int T2>
Quantity<M1-M2,L1-L2,T1-T2> operator/(const Quantity<M1,L1,T1>& lhs,
            const Quantity<M2,L2,T2>& rhs)
```

```
{
    return Quantity<M1-M2,L1-L2,T1-T2>(lhs.getValue()/rhs.getValue());
}

template <int M, int L, int T>
Quantity<-M, -L, -T> operator/(double x, const Quantity<M,L,T>& rhs)
{
    return Quantity<-M,-L,-T>(x/rhs.getValue());
}
// ...
```

A. Quelloffenes Eclipse mit CDT/UML2.5/OCL2.4-Tools

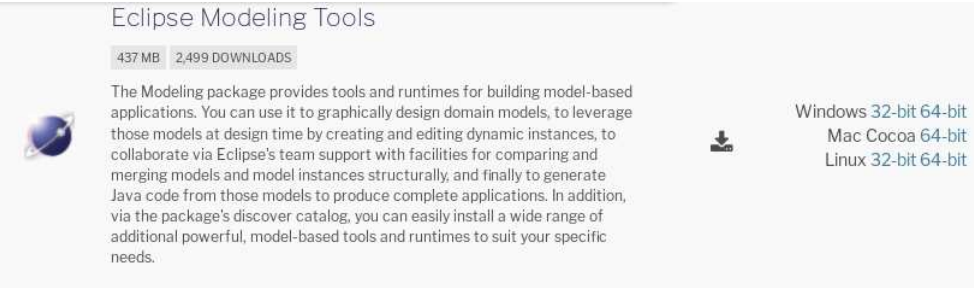
Hilfsmittel (Tools) zur „state of the art“ -Entwicklung von C++-Anwendungen:
Verfügbar (vorinstalliert) ist auf allen Ausbildungsclustern (CIP/IT/PI: l101, ...) der Fachgruppe Mathematik/Informatik der BUW die aktuelle
`eclipse ide 2018-09`

aufrufbar als `eclipse-papyrus1809` mit den Komponenten:

```
Eclipse-Modelling (inkl. Java) mit
   CDT (C/C++ Development Environment inklusive Eclipse standalone Debugger),
   CDT-Linux Tools (für Gcov, Gprof, Perf, Valgrind, ...),
   CUTE (C++ Unit-Tests),
   UML (Papyrus mit Papyrus-Designer (CPP- und Java-Codeerzeugung),
   Eclipse OCL 6.5.0 (Object Constraint Language für Codeverträge/Constraints),
   PyDev (Python),
   D Development Tools (mit gesondert installiertem dmd, dub),
   Scala,
   Kotlin,
   OcaIDE (Ocaml),
   ...,
   cppcheclipse (mit gesondert installiertem cppcheck)
```

Hinweis zur Installation auf dem eigenen Linux-Notebook:

eclipse ide 2018-09 Packages:



Installiere *Eclipse Modeling Tools*, (zur Zeit die Version 2018-09 durch Download der Datei `eclipse-modeling-2018-09-linux-gtk-x86_64.tar.gz` von `http://www.eclipse.org/downloads/packages/`, installiere sie mittels:

```
/Downloads> gunzip eclipse-modeling-2018-09-linux-gtk-x86_64.tar.gz
buhl@rhea3:~/Downloads> ls -al ec*
-rw-r--r-- 1 buhl users 446816013 20. Jun 09:05 eclipse-modeling
    -2018-09-linux-gtk-x86_64.tar
```

wechsle ins Zielverzeichnis für selbstinstallierte Software (etwa `$HOME/sw`) und entpacke `eclipse` dorthin:

```
~/sw> tar xf ~/Downloads/eclipse-modeling-2018-09-linux-gtk-x86_64.
    tar
~/sw> ls -al ecli*
drwxr-xr-x 8 buhl users 4096 20. Jun 14:19 eclipse
~/sw> mv eclipse eclipse-modeling-2018-09-linux-gtk-x86_64
~/sw> cd ~/bin
```

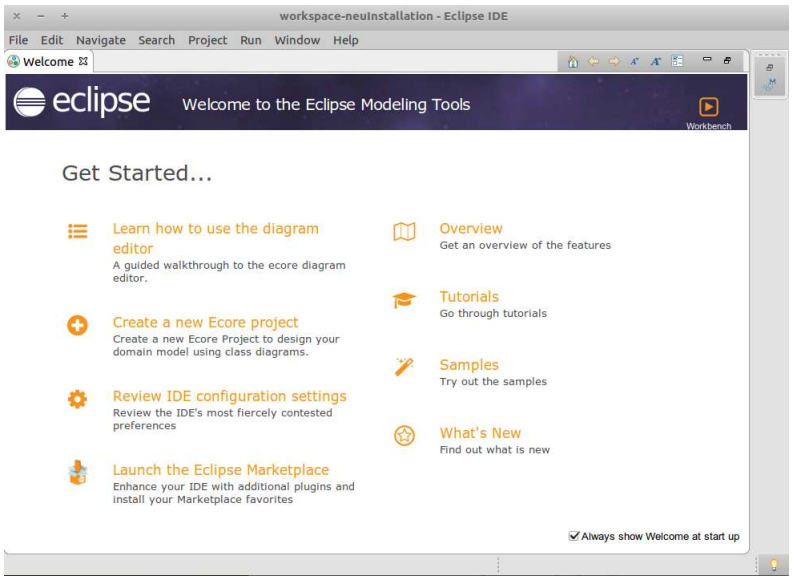
Erzeuge in `$HOME/bin` ein Startskript $HOME/bin/eclipse-papyrus1809 mit dem Inhalt:

```
#!/bin/sh
#
$HOME/sw/eclipse-modeling-2018-09-linux-gtk-x86_64/eclipse $*
```

und gib ihm Ausführbarkeitsrechte:

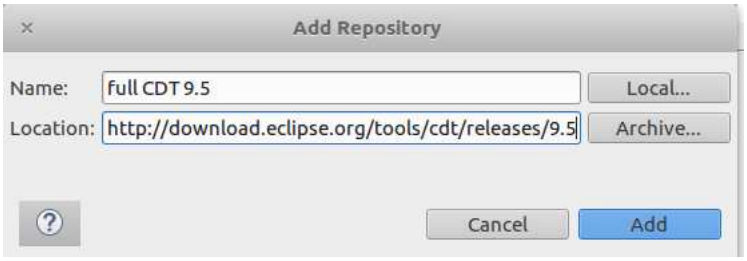
```
~/bin> chmod 755 $HOME/bin/eclipse-papyrus1809
```

Nach Aus- und wieder Einloggen (bzw. Start einer neuen Shell) kann num mittels `exlipse-papyrus1809` die aktuelle Eclipse-IDE gestartet werden:

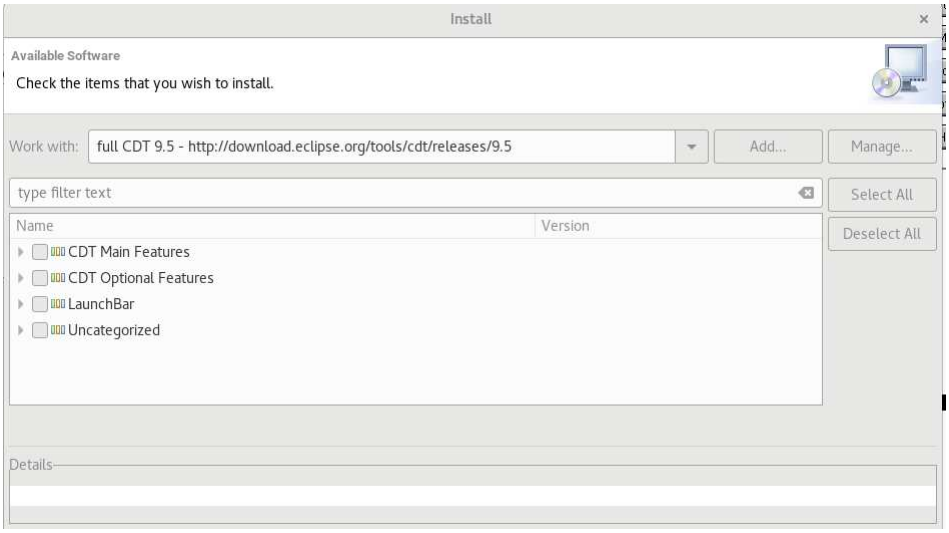


Ergänze dann unter `Help, Install New Software, Add`

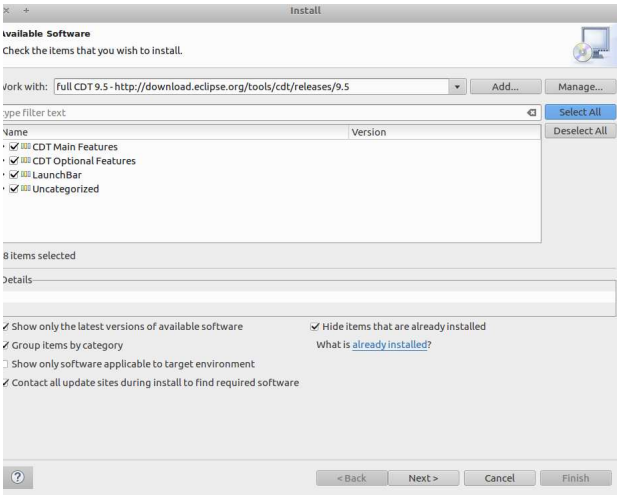
`Name: full CDT 9.5`
`Location: http://download.eclipse.org/toools/cdt/releases/9.5`



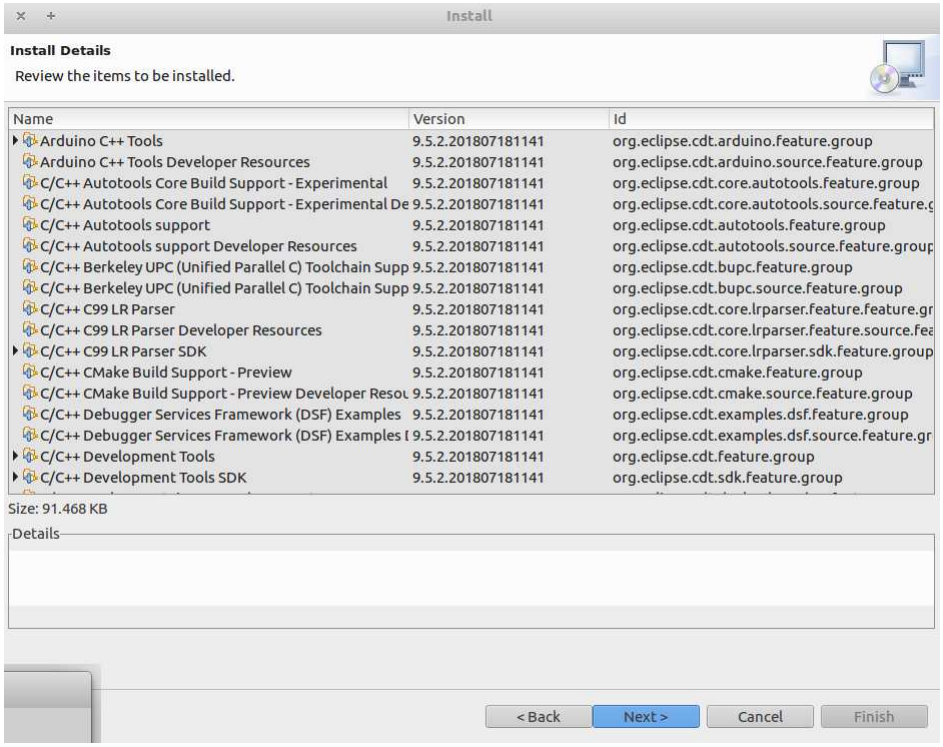
drücke `Add` und im erscheinenden Komponentenüberblick



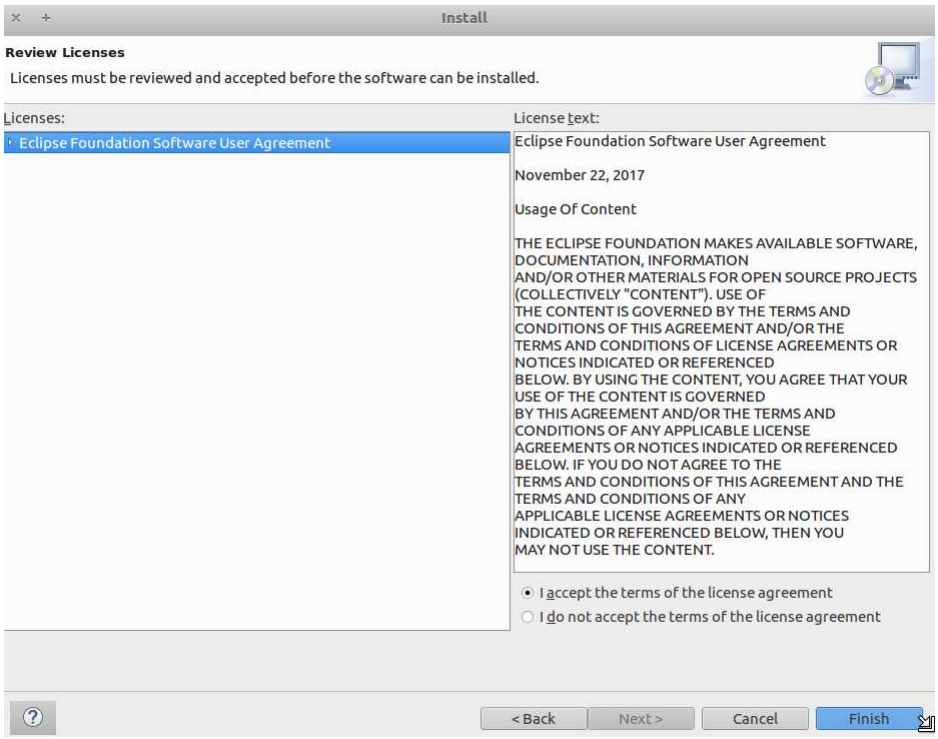
`Select All`:



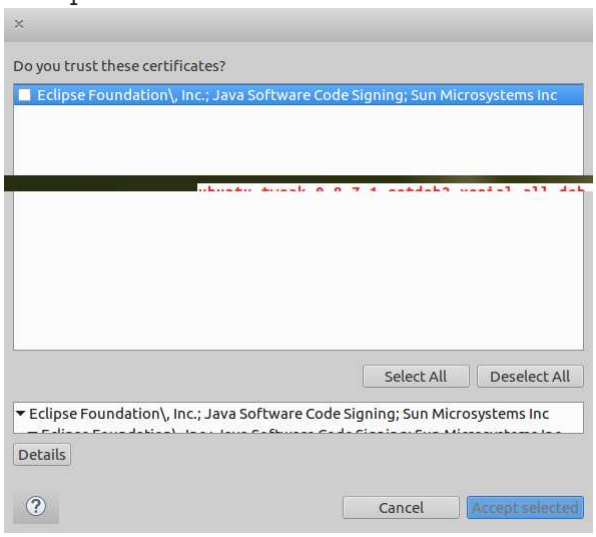
Die Taste `Next>` erzeugt einen Überblick aller zu installierenden Komponenten:



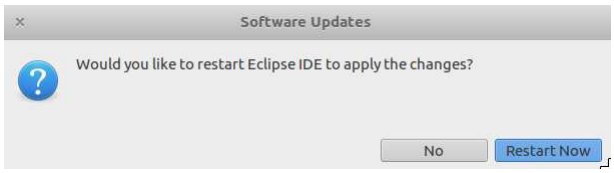
Nach erneuter Betätigung von `Next>`



und Akzeptierung der Lizenzbedingungen (“I accept ...“ gefolgt von der Betätigung der Taste `Finish`) läuft die Installation.
Beantworten Sie dabei ”Do you trust these certificates“ durch `Select All` gefolgt von `Accept Selected`:



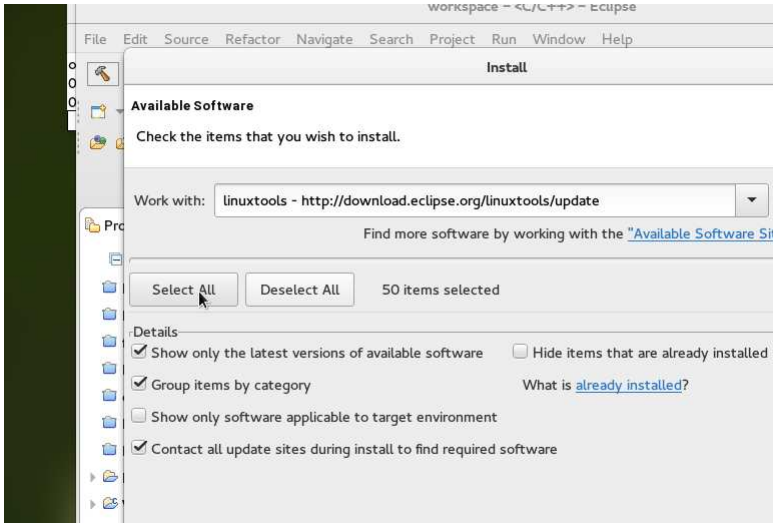
Abschließend (nach Installationsende) sollten Sie in



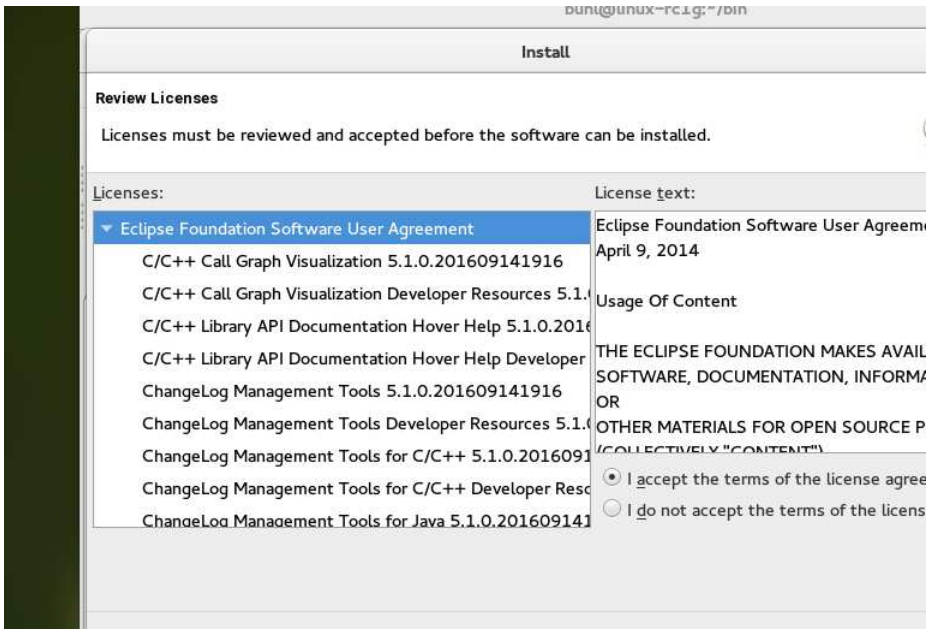
die Taste `Restart Now` betätigen.

Zur analog ablaufenden Installation der Eclipse-Plugins für die Linux-Developertools ergänze unter `Help, Install New Software, Add`

`linuxtools`
`http://download.eclipse.org/linuxtools/update`



das Linuxtools-Repositorium und wähle `Select All` an:

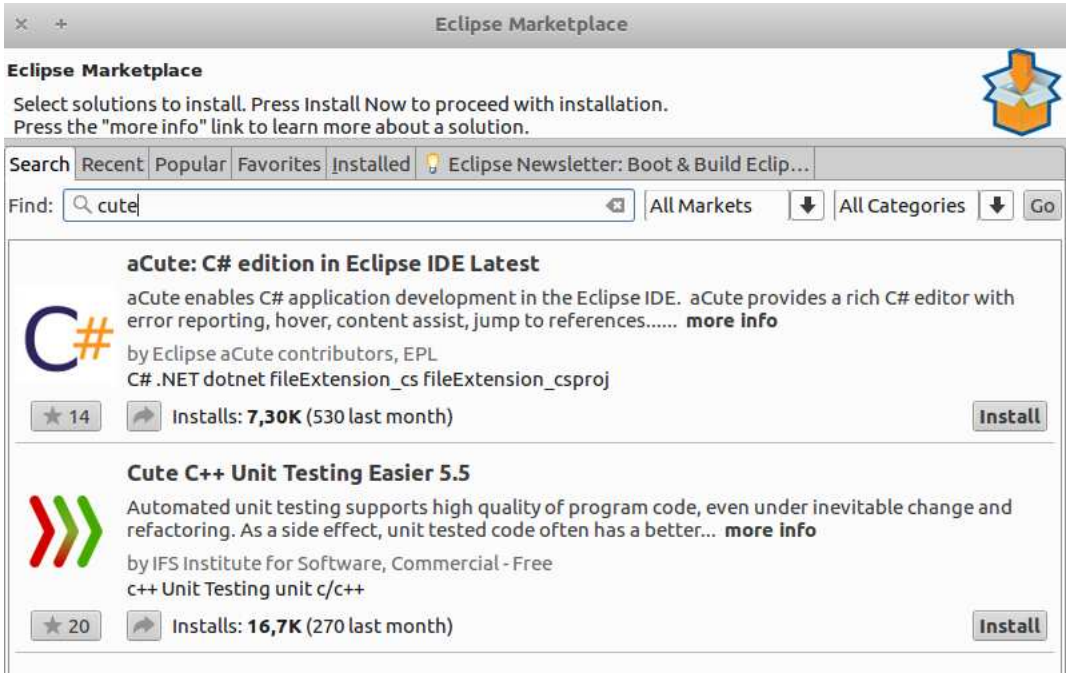


Die Linuxtools bieten Eclipse-Integration qualitätssteigernder Tools für die C++-Entwicklung:

- Callgraph
- ChangeLog
- GProf
- Gcov (oder lcov)
- Libhover
- Man Page
- LTTng
- OProfile
- Perf
- Systemtap
- Valgrind

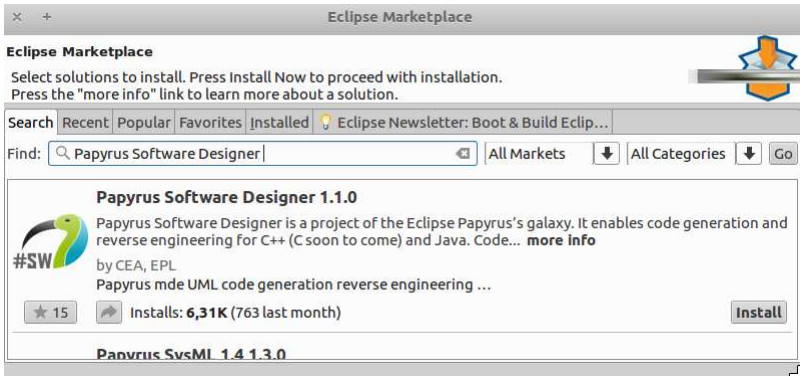
Ähnlich installiert man Cute 5.5, ein C++-Unit-Test-Plugin:

`Help, Eclipse Marketplace`

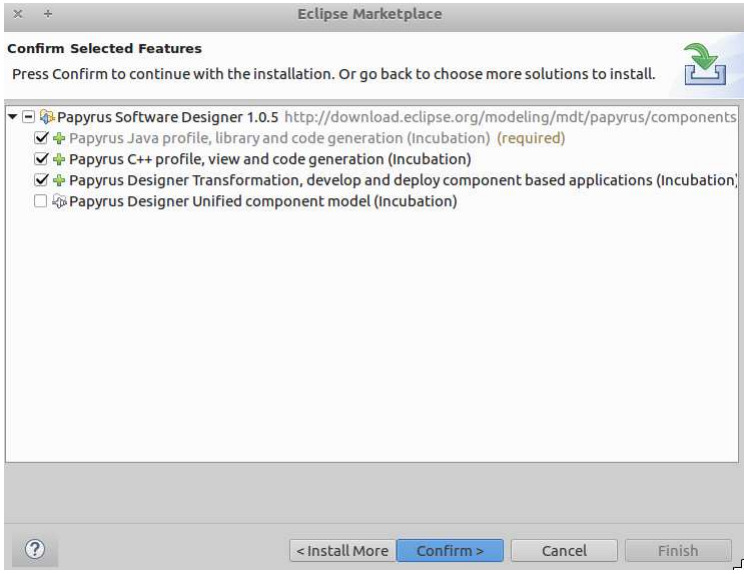


`Install` Cute C++ Unit Testing Easier 5.5, `Confirm`, "I accept ... license...", `Finish`

Ergänze dann unter `Help, Eclipse Marketplace` das UML-Tool `Papyrus Software Designer 1.1.0` (für die Erstellung von UML-Modellen):

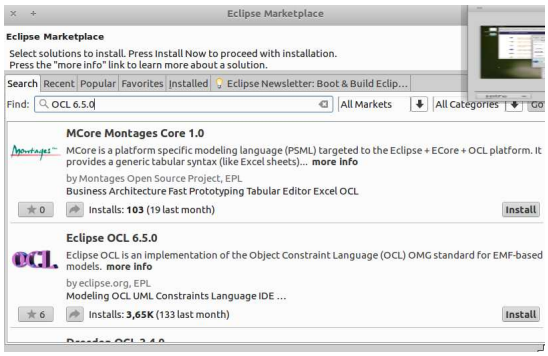


`Install`



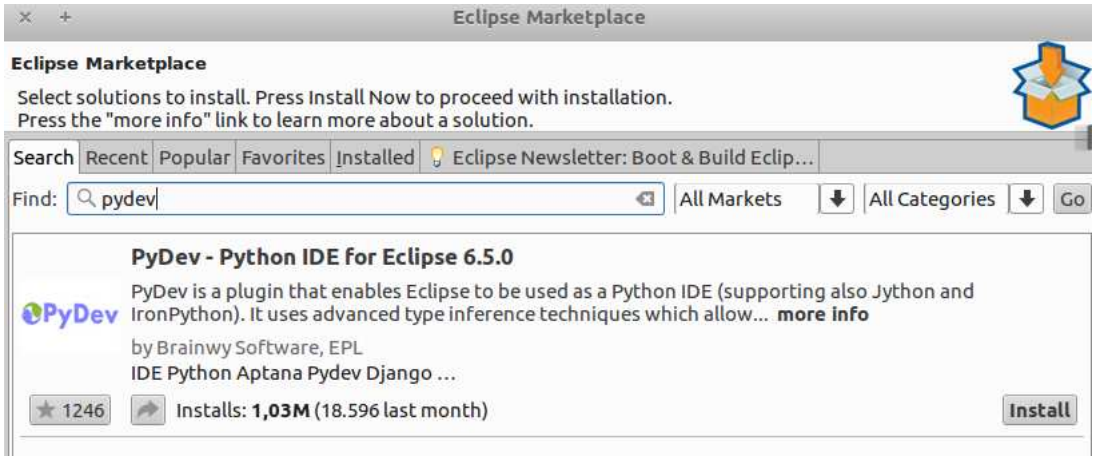
`Confirm`
`Accept Licence, Finish`
`Restart Now`

Ergänze unter `Help, Eclipse Marketplace` dann die `Eclipse OCL 6.5.0` zur Erstellung von formalen Constraints (Codeverträge an die UML-Komponenten):

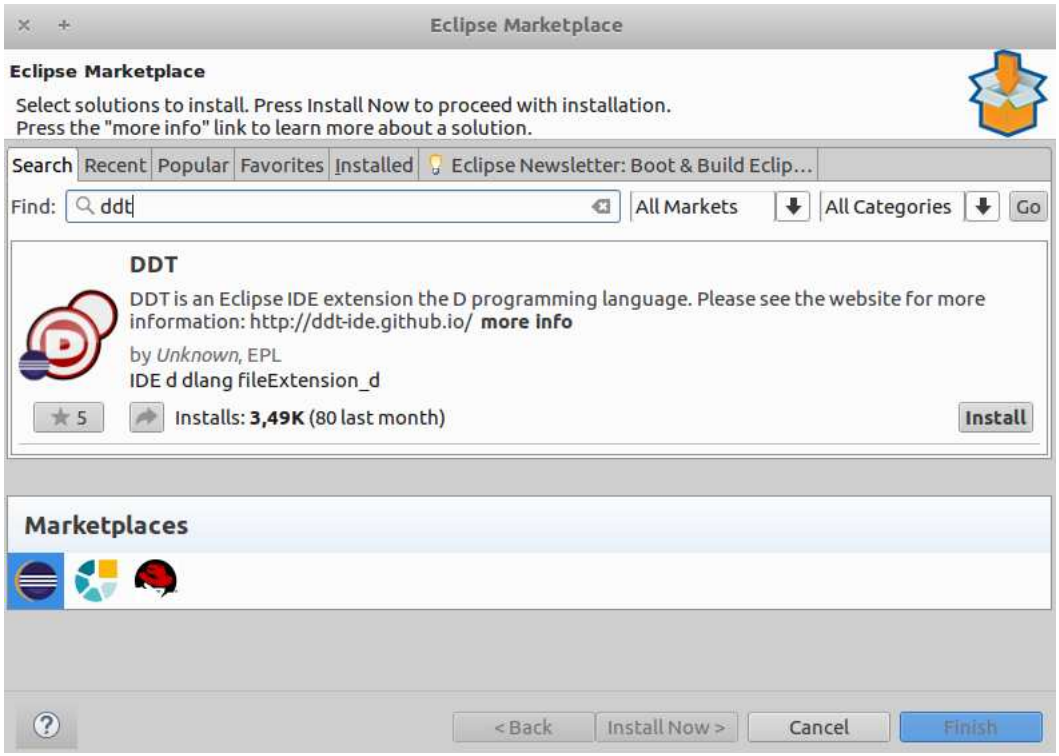


`Install, accept licence, Finish, Restart Now`

Bei Bedarf kann man unter `Help, Eclipse Marketplace` schließlich noch die `Python`-Entwicklungsumgebung `PyDev 6.5.0` installieren



`Install, Confirm, Accept licence, Finish, Restart Now`
und `D-Unterstützung` (ddt 1.0.3, sofern auf Ihrer Maschine `dmd` und `dub` installiert sind)

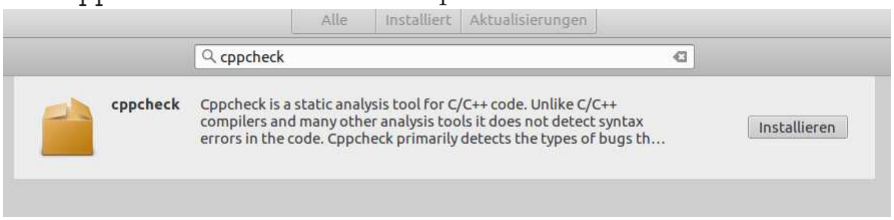


`Install, accept licence, Finish, Select All, Accept selected, Restart Now`

und ... hinzuinstallieren.

(Vorinstalliert auf den Ausbildungsclustern der Fachgruppe als `eclipse-papyrusp` sind zum Beispiel zusätzlich: `Scala IDE 4.7.x, Kotlin 0.8.7, OcaIDE 1.2.21 (für Ocaml)`.)

`cppcheclipse 1.1.0` (`Eclipse-Marketplace` Plugin für cppcheck) ist nach Installation von `cppcheck` ebenfalls sehr empfehlenswert:



Getting started with CDT development
CDT Documentation, Tutorials, ...
Eclipse CDT (C/C++ Development Tooling)
Eclipse für C/C++-Programmierer, dritte Auflage

Hinweis zu verfügbaren Softwareentwicklungssystemen:
GNU g++ für Linux
gcc7 vor den Toren
Compiler: GCC 7.1 kennt die Sprachfeatures von C++17
GNU Compiler Collection 7.3
GCC 8.2
GCC, the GNU Compiler Collection
C++17: Standardbibliotheksänderungen

Cygwin für Windows, Cygwin
mingw-64
Windows 10 Linux-Subsystem
C++17 Features In Visual Studio 2017 Version 15.3 Preview

Microsoft Imagine (früher MSDNAA): VisualStudio 201x für Windows