# Materialsammlung - Softwarequalität

Prof. Dr. Hans-Jürgen Buhl

Wintersemester 2010/11

Bergische Universität Wuppertal
Fachbereich C — Mathematik und Informatik

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Tabellenverzeichnis

# Vorbemerkungen – Softwarequalität heute

**Haftungsausschluß**

Die Überlassung dieser Baupläne erfolgt ohne Gewähr. Der Plan gibt keine Garantie, Gewährleistung oder Zusicherung, daß diese Pläne für einen bestimmten Zweck geeignet sind, daß sie richtig sind oder daß ein Gebäude, das nach diesen Plänen gebaut wird, den Ansprüchen des jeweiligen Erwerbers genügt. Der Planer erklärt sich bereit, Ersatzkopien derjenigen Teile der Pläne zu liefern, die zum Zeitpunkt des Kaufs unleserlich sind. Darüber hinaus wird keinerlei Haftung übernommen. Der Erwerber dieser Pläne sollte beachten, daß in den entscheidenden Phasen des Baus und nach der Fertigstellung geeignete Tests durchzuführen sind und daß die üblichen Vorsichtsmaßnahmen zum Schutz des Lebens der Bauarbeiter zu treffen sind.

(Zitat: Robert L. Baber: Softwarereflexionen, Springer-Verlag)

und in der Praxis:

...
**2. Haftung**
Wir werden immer bemüht sein, ihnen einwandfreie Software zu liefern. Wir können aber keine Gewähr dafür übernehmen, daß die Software unterbrechungs- und fehlerfrei läuft und daß die in der Software enthaltenen Funktionen in allen von Ihnen gewählten Kombinationen ausführbar sind. Für die Erreichung eines bestimmten Verwendungszweckes können wir ebenfalls keine Gewähr übernehmen. Die Haftung für unmittelbare Schäden, mittelbare Schäden, Folgeschäden und Drittschäden ist, soweit gesetzlich zulässig, ausgeschlossen. Die Haftung bei grober Fahrlässigkeit und Vorsatz bleibt hiervon unberührt, in jedem Fall ist jedoch die Haftung beschränkt auf den Kaufpreis.

Hauptgegenstand dieser Veranstaltung ist die konstruktive Qualitätssicherungs- und Spezifikationsmethode *Design by Contract (Spezifikation durch Verträge = SdV)*

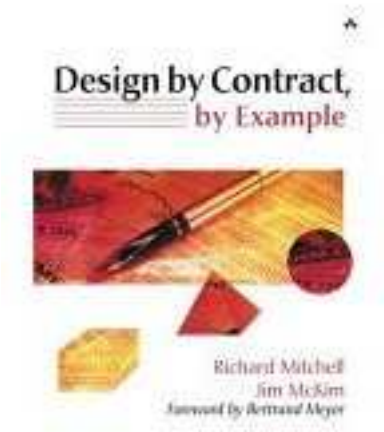



Abbildung 0.1: Design by Contract, by Example von Richard Mitchell und Jim McKim

In den ersten Kapiteln wird Grundwissen zur Softwarequalität und -qualitätssicherung besprochen.

Die Methodik DbC wurde zuerst in der Programmiersprache `Eiffel` thematisiert, ist jedoch heute in (fast) allen neuen Programmiersprachen nutzbar.

http://en.wikipedia.org/wiki/Eiffel_(programming_language)

http://en.wikipedia.org/wiki/Design_by_contract

(ISE-Eiffel 4.5 Quellcode zu den Beispielen oben genannten Buchs
Ein Eiffel Tutorial

Beispiel-Quellen für ISE-Eiffel 6.6:
http://www.math.uni-wuppertal.de/~buhl/teach/exercises/PbC09/source_code.tar.gz)

**Fehlende Softwaregüte: Softwarekatastrophen**

Virginia state govt computer outage
Another near-disaster due to vehicle automation
Absturz: T-mobile ohne Netz
Deutsche Sipgate-VoIP-Nummern nicht erreichbar
    Hintergrund zum Ausfall vom Freitag, den 08.10.10
    zentrale Steuereinheit, die ... als sog. “Cold-Spare” vorliegt

**C++ ohne DbC-Constructs**

Proposal to add Contract Programming to C++ (revision 4)

**Workaround: nana und Eclipse**

GNU nana DbC for C and C++
Eclipse Helios SR1 with CDT...
Eclipse für C/C++-Programmierer

```
eclipse-helios, doxygen, cxxtest, ddd, lcov, valgrind, binutils, ...
```

## Beispiele für Softwaredisfunktionalitäten

### Ein sahniger Brocken

(aus: Die Zeit vom 15.09.2005)

Begleitet von großem Werberummel hat die NASA den Kometen Tempel1 beschossen. Nun zeigen die Daten: Getroffen hat sie gut, gelernt hat sie wenig.

Auch wenn in den offiziellen Mitteilungen der NASA keine Rede davon ist - unter den versammelten Astronomen hat sich längst herumgesprochen, dass der Erfolg von *Deep Impact* nicht nur von aufgewirbeltem Feinstaub verdunkelt wurde. Ein Softwarefehler hat dazu geführt, dass die ersten - und besten - Bilder des Zusammenpralls im Datenspeicher des Begleitsateliten von späteren Aufnahmen überschrieben wurden.

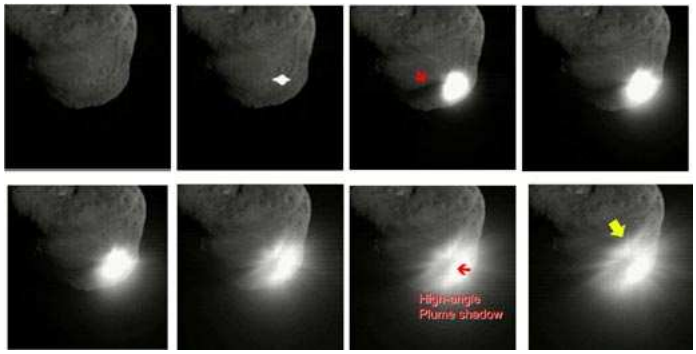



Abbildung 0.2: Bilder von Deep Impact

Der vollständige Artikel: http://www.zeit.de/2005/38/komet

### USV-Software legt Server lahm

APC, Hersteller von unterbrechungsfreien Stromversorgungssystemen (USV), rät in einem Knowledgebase-Artikel dazu, alte Versionen der **PowerChute Business Edition-Software 6.X** umgehend durch die Version 7.X zu ersetzen.

Die Software zur Steuerung unterbrechungsfreier Stromversorgungen und zum sicheren Server-Shutdown hat Probleme mit einem auslaufenden Java-Runtime-Zertifikat. Dies führt dazu, dass die Windows-Server, auf denen die alte Version läuft, zum Teil mehrere Stunden für eine Ab- beziehungsweise Anmeldung benötigen. Die Dienste des Servers wie zum Beispiel Netzwerkfreigaben funktionieren allerdings trotz der Anmeldeprobleme weiterhin.

(aus http://www.heise.de/newsticker/meldung/62344)

**Chaos an Hannovers Geldautomaten (05.10.2003 13:00 Uhr)**

Computerprobleme haben am Samstag alle 240 Geldautomaten der Sparkasse in der Stadt und Region Hannover lahm gelegt. Die Fusion der Stadt- und Kreissparkasse sollte am Wochenende auch technisch umgesetzt werden, sagte der Sprecher des Geldinstituts, Stefan Becker. Beim Hochfahren eines Server habe sich ein Fehler eingeschlichen, so dass die Geldautomaten nicht mehr funktionierten. Die Sparkasse öffnete stadtdessen fünf Filialen, damit Kunden etwa in Einkaufszonen Bargeld abheben können.

(aus: http://www.heise.de/newsticker/meldung/40834)

**THERAC 25**

Selten sind solch schädliche Vorfälle so gut dokumentiert worden wie im Fall des „THERAC 25“, eines computergestützten Bestrahlungsgerätes. Dabei handelt es sich um ein Bestrahlungsgerät, welches in zwei „Modi“ arbeitet: im „X-Modus“ wird ein Elektronenstrahl von 25 Millionen Elektronen-Volt durch Beschuß einer Wolframscheibe in Röntgenstrahlen verwandelt; im „E-Modus“ werden die Elektronen selbst, allerdings „weicher“ mit erheblich reduzierter Energie als Korpuskelstrahlung erzeugt. Je nach therapeutischer Indikation wird die geeignete Strahlungsart eingestellt; in beiden Fällen kann der Bestrahlungsverlauf, nach Modus, Intensität und Bewegungskurve der Strahlungsquelle, mit einem Bildschirm-„Menü“ eingegeben werden.

Als mehrere Patienten berichteten, sie hätten bei Behandlungsbeginn das Gefühl gehabt, „ein heißer Strahl“ durchdringe sie, wurde dies vom Hersteller als „unmöglich“ zurückgewiesen. Erst nach dem Tod zweier Patienten sowie massiven Verbrennungen bei weiteren Personen kam heraus, daß neben dem X- sowie E-Modus mit niedriger Elektronenintensität infolge Programmierfehler ein unzulässiger dritter Zustand auftrat, nämlich direkt wirkende, 25 Millionen Elektronen-Volt „heiße“ Elektronen.

Dies geschah immer dann, wenn ein vorgegebenes „Behandlungsmenü“ mittels Curser-Taste modifiziert wurde. Um aufwendige Umprogrammierung zu vermeiden, wollte der kanadische Hersteller die Benutzung der Curser-Taste verbieten bzw. diese ausbauen und die Tastenlücke mit Klebeband abdichten lassen! Es ist zu befürchten, daß der Fall „THERAC 25“ kein Einzelfall ist. Zumeist ist es mangels entsprechender Vorsorge in computergesteuerten Medizingeräten schwerlich möglich, schädliches Systemverhalten später aufzuklären.

### Berliner Magnetbahn

Computer spielen in allen gesellschaftlichen Bereichen eine immer größere Rolle. Angesichts der von fehlerhafter Software ausgehenden Gefahr wird versucht, die Sicherheit von computergesteuerten Systemen so weit wie möglich zu garantieren.

**Softwarefehler: Kleine Ursache, große Wirkung**

Fünf - Null, tippt der Operator in die Tastatur und erwartet, daß die Magnetschwebebahn auf 50 Stundenkilometer beschleunigen würde. Doch nichts geschah. Wieder tippt er fünf - null und vergaß diesmal nicht die „Enter"-Taste zu betätigen, mit der die Daten erst in den Rechner abgeschickt werden. Die insgesammt eingegebene Tastenfolge „fünf - null - fünf - null" interpretiert der Rechner als Anweisung, auf unsinnige 5050 Stundenkilometer zu beschleunigen. Dies konnte die Bahn zwar nicht, aber immerhin wurde sie so schnell, daß sie nicht mehr rechzeitig vor der Station gebremst werden konnte. Es kam zum Crasch mit Personenschaden – so geschehen vor zwei Jahren bei einer Probefahrt der Berliner M-Bahn.

Vernünftigerweise hätte die den Computer steuernde Software die Fehlerhaftigkeit der Eingabe „5050" erkennen müssen. Schon dieses Beispiel mangelnder Software zeigt, von welcher Bedeutung das richtige Verhalten von Computerprogrammen sein kann. Nicht nur bei Astronauten, die mit softwaregesteuerten Raumfähren ins All starten, hängt heute Leben und Gesundheit von Software ab. Computerprogramme erfüllen mittlerweile in vielen Bereichen sicherheitsrelevante Aufgaben.

### Elektronik-Fehler führt zu Überhitzung bei Volvo-PKW

Kaum ein KFZ-Hersteller, der nicht mit Elektronik, Software und Hightech-Ausstattung das Autofahren komfortabler und die Wartung in der Werkstatt einfacher machen will. Doch die Tücken der Technik lassen für manchen Kunden den PKW zum IT-Sicherheitsrisiko werden. Nachdem vor kurzem erst Softwarefehler bei Mercedes-Dieseln für Aufsehen sorgten, können nun Defekte in der elektronischen Steuerung der Motorkühlung bei Volvo-Personenwagen zur Überhitzung führen.

Der Fehler tritt bei den Modellen S60, S80, V70 und XC70 aus den Baujahren 2000 und 2001 auf, erklärte Volvo, einzelne Modelle aus dem Jahr 1999 seien ebenfalls betroffen. Die fehlerhaft arbeitende Elektronik hat Bosch an Volvo geliefert – wer für den Fehler, der vor allem bei langsamer Fahrt bei hohen Außentemperaturen zur Überhitzung führen kann, verantwortlich ist, steht laut Volvo noch nicht fest. Insgesamt 460.000 Fahrzeuge weltweit ruft der schwedische Hersteller daher in die Werkstätten zurück. Laut dpa erhalten in Deutschland rund 40.000 Besitzer eines Volvo-PKW eine Aufforderung zum Werkstattbesuch – der für die Halter zumindest kostenlos bleibt.

(aus: http://www.heise.de/newsticker/meldung/51019)

**The Patriot Missile**

The Patriot missile defense battery uses a 24 bit arithmetic which causes the representation of real time and velocities to incur roundoff errors; these errors became substantial when the patriot battery ran for 8 or more consecutive hours.

As part of the search and targeting procedure, the Patriot radar system computes a "Range Gate" that is used to track and attack the target. As the calculations of real time and velocities incur roundoff errors, the range gate shifts by substantial margins, especially after 8 or more hours of continous run.

The following data on the effect of extended run time on patriot operations from Appendix II of the report would be of interest to numerical analysists anywhere.

| Hours | Real Time (seconds) | Calculated Time (seconds) | Inaccuracy (seconds) | Approximate Shift In Range Gate (meters) |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 3600 | 3599.9966 | .0034 | 7 |
| 8 | 28800 | 28799.9725 | .0275 | 55 |
| 20a | 72000 | 71999.9313 | .0687 | 137 |
| 48 | 172800 | 172799.8352 | .1648 | 330 |
| 72 | 259200 | 259199.7528 | .2472 | 494 |
| 100b | 360000 | 359999.6667 | .3333* | 687 |

Tabelle 0.1: Divergence in the Range Gate of a PATRIOT MISSILE

a: continuous operation exceeding 20 hours-target outside range gate
b: Alpha battery [at Dhahran] ran continuously for about 100 hours
* corrected value [GAO report lists .3433]

On Februrary 21, 1991 the Partiot Project Office send a message to all patriot sites stating that very long run times "could cause a shift in the range gate, resulting in the target being offset". However the message did not specify "what constitutes very long run times". According to the Army officials, they presumed that the users would not run the batteries for such extended periods of time that the Patriot would fail to track targets. "Therefore, they did not think that more detailed guidance was required".

The air fields and seaports of Dhahran were protected by six Patriot batteries. Alpha battery was to protect the Dhahran air base.
On February 25, 1991, Alpha battery had been in operation for over 100 consecutive hours. That was the day an incomming Scud struck an Army barracks and killed 28 American soldiers.

On February 26, the next day, the modified software, which compensated for the inaccurated time calculation, arrived in Dhahran.

### Kontenabrufverfahren startet wegen Softwareproblemen als Provisorium

Das automatische Kontenabrufverfahren nach dem „Gesetz zur Förderung der Steuerehrlichkeit“, das ab dem 1. April die Abfrage der Kontostammdaten für einige Behörden möglich macht, startet mit Anlaufproblemen. Sie liegen vor allem darin begründet, dass die entsprechende Abfragesoftware der Stammdaten, die ab November 2003 zum Zwecke der Terroristenfahndung entwickelt wurde, nicht richtig skaliert. Diese Software wurde auf ca. 2000 Abfragen pro Tag durch die Polizeifahnder ausgelegt. Mit mehr als täglichen 50.000 Abfragen, die von Finanzämtern, Bafög- oder Sozialämtern ab dem 1. April erwartet werden, ist die Software hoffnungslos überfordert. Für die 18 bis 20 Millionen Konten, die jährlich nach dem Willen des Gesetzgebers gesucht werden sollen, wird derzeit eine völlig neue Schnittstellenspezifikation entwickelt und ein komplett neues Programm geschrieben. Bis dieses Programm für die automatische Abfrage durch die Sachbearbeiter fertig ist, muss die Abfrage wie bisher manuell erfolgen.

Bei dieser manuellen Abfrage reichen Polizeibehörden und Strafverfolger ihre Anfragen auf Papier oder per Fax oder E-Mail bei der Bundesanstalt für Finanzdienstleistungsaufsicht (BaFin) ein und bekommen die gewünschten Kontodaten auf demselben Wege zurück. Dieses Verfahren soll durch eine Suchmaske ersetzt werden, die jede Behörde aufrufen kann – wenn die dahinter liegende Abfragesoftware die Datenmengen bewältigen kann.

(aus: http://www.heise.de/newsticker/meldung/58096)

### Buffer Overflow im Linux-Kernel

Paul Starzetz von isec hat Details zu einer neuen Lücke im Linux-Kernel veröffentlicht, mit der ein Angreifer Programme mit Root-Rechten ausführen kann. Anders als bei vergangenen Veröffentlichungen von Starzetz, wurden die Hersteller aber offenbar nicht vorab informiert, etwa über die geschlossene Mailing-Liste VendorSec. Nach seinen Angaben würde die Linux-Community Veröffentlichungen ohne Embargos von Distributoren bevorzugen. Um aber die Regeln der so genannten Responsible Disclosure einzuhalten, veröffentlicht er diesmal keinen Exploit-Code.

Der Fehler findet sich wieder einmal im Linux ELF-Binary-Loader, in dem Starzetz in der Vergangenheit bereits mehrere Lücken aufdeckte. Diesmal ist ein Buffer Overflow in der Funktion elf_core_dump schuld, der beim Aufruf einer

weiteren Funktion (copy_from_user) mit einer negativen Längenangabe auftritt. Starzetz hat nach eigenen Angaben die Lücke bereits durch ein präpariertes ELF-Binary demonstrieren können, das mit Kernel-Privilegien lief. Ein Proof-of-Concept-Programm ist seinem Advisory beigefügt, das aber nur den Kern des Problems demonstriert.

(aus:http://www.heise.de/newsticker/meldung/59498)

**Auch Superhirne können irren - das Risiko Computer**

Lenkwaffen, Flugsteuerungen, Diagnosegeräte, Verkehrsleitsysteme, Dateien, Produktions-Steuerung – überall hat der Computer das Kommando übernommen. Doch nicht überall gibt er die richtigen Befehle. Mancher Irrtum schon hatte tödliche Folgen. Das Vertrauen in das elektronische Superhirn ist angeschlagen.

Sollten US-Kriegsschiffe, die mit dem computergestützten Waffensystem „Aegis“ ausgerüstet sind, in Zukunft wieder in Spannungsgebieten kreuzen, werden die verantwortlichen Offiziere dort mit der Angst leben, daß sich die Ereignisse des 3. Juli 1988 wiederholen könnten: Damals folgte der Kapitän des Kreuzers „Vincennes“, von elektronischen Befehlen unter Entscheidungsdruck gesetzt, der Logik des Computers, dessen Abtastsystem ein Verkehrsflugzeug mit einer Kampfmaschine verwechselte. Er gab den verhängnisvollen Befehl zum Abfeuern der Raketen. Alle 290 Insassen des iranischen Airbus kamen dabei ums Leben. ...

Aus anderer Quelle:
Auch der erste KI-Unfall, bei dem das „künstlich intelligente“ AEGIS-System des US-Kreuzers „Vincennes“ im Sommer 1988 einen zivilen Airbus mit einem MIG-Militärjet verwechselte, durfte bei heutigem Kenntnisstand durch einen Konzeptfehler mitverursacht worden sein. Aus der „Sicht“ des einzelnen AEGIS-Systems werden alle Signale, die auf einem Richtstrahl innerhalb einer 300 Meilen umfassenden Überwachungszone entdeckt werden, einem einzelnen Objekt zugeordnet. So können ein Militar- und ein Zivil-Jet nur durch ein räumlich getrenntes System unterschieden werden. Offenbar hat das AEGIS-System aber weder Inkonsistenzen der Daten (militärische und zivile Transponder-Kennung) noch die unvollständige räumliche Auflösung dem verantwortlichen Kommandeur übermittelt, der im Vertrauen auf die Datenqualität den Befehl zum Abschuß von fast 300 Zivilisten gab. Offensichtlich ist in Streßsituationen eine menschliche Plausibilitätskontrolle nicht nur bei derart komplexen Systemen erschwert. Aus einem bis dahin fehlerfreien Funktionieren wird induktiv auf korrektes Verhalten im Ernstfall geschlossen. Daher sind besondere Hinweise auf inkonsistente und

unvollständige „Datenlagen“ und gegebenenfalls Sperren gegen automatische Prozeduren zwingend erforderlich.

### Explosion der Ariane 5

http://www.ima.umn.edu/ arnold/disasters/ariane5rep.html

### Neueste Risikoinformationen/Softwareprobleme

... findet man unter: http://catless.ncl.ac.uk/Risks:

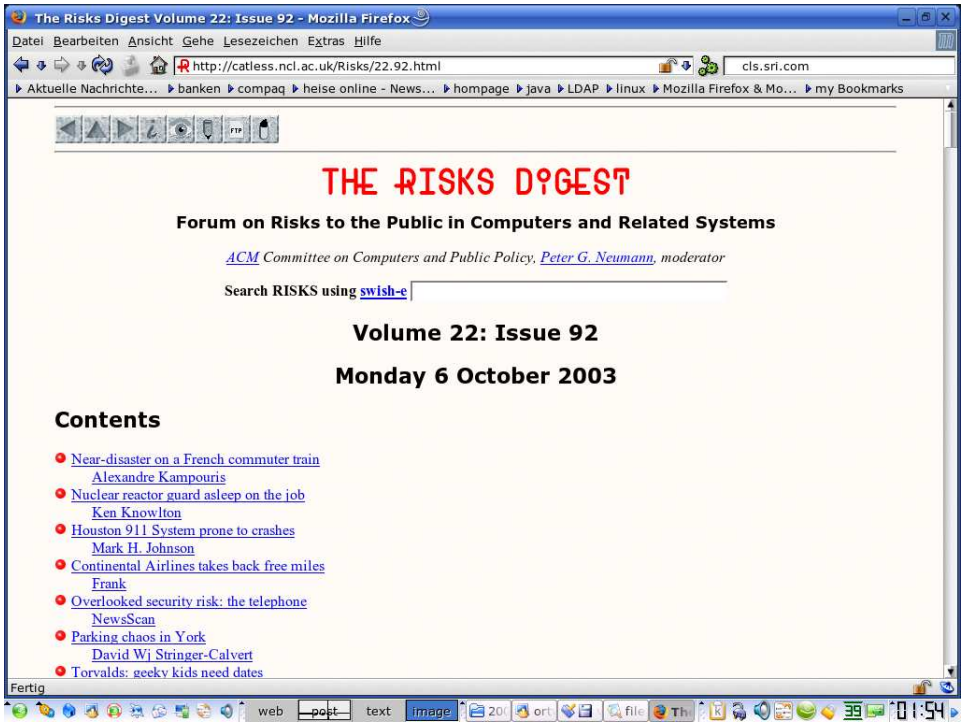



Abbildung 0.3: http://catless.ncl.ac.uk/Risks/22.92.html

# 1 Qualitätsanforderungen an Softwareprodukte

**A. Produktorientiert:**

1. funktionale Korrektheit (benötigt Spezifikation)
2. funktionale Vollständigkeit
3. Robustheit gegenüber dem Benutzer
4. Benutzerfreundlichkeit
5. Effizienz in Laufzeit
6. Effizienz in Arbeitsspeicherbedarf
7. Effizienz in Plattenspeicherbedarf
8. Integrität (gegenüber unauthorisierten Änderungen)
9. Kompatibilität, Integrationsfähigkeit, Standards

**B. Projektorientiert:**

1. Verständlichkeit (des GUI, der Dokumentation, ...)
2. Überprüfbarkeit
3. Wartbarkeit
4. Änderbarkeit, Erweiterbarkeit
5. Portierbarkeit
6. Wiederverwertbarkeit

Siehe auch ISO 9126 und Qualitätsmodelle.

## 1.1 Software Quality Attributes confirming ISO 9126-1

- Funktionalität
  - Angemessenheit
  - Richtigkeit/Sorgfalt
  - Interoperationalität/Kompatibilität
  - Regeltreue
- Ausfallsicherheit
  - Ausgereiftheit
  - Fehlertolleranz
  - Wiederherstellbarkeit
- Bedienbarkeit
  - Verständlichkeit
  - Erlernbarkeit
  - Funktionsfähigkeit
- Effizienz
  - zeitliche Effizienz
  - Ressourcenverbrauch
- Wartungsfreundlichkeit
  - Analysierbarkeit
  - Änderbarkeit
  - Stabilität
  - Testbarkeit
- Portabilität
  - Anpassbarkeit
  - Installierbarkeit
  - Konformität
  - Ersetzbarkeit

## 1.2 Spezifikation wie sie einmal in C++ aussehen könnte

Vergleiche http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2006/n1962.html

```
double sqrt( double r )
    precondition
    {
        r >= 0.;
    }
    postcondition( result )
    {
        equal_within_precision( result * result, r );
    }



int factorial( int n )
precondition
{
    0 <= n && n <= 12;
}
postcondition( result )
{
    result >= 1;
}
{
    if ( n < 2 )
        return 1;
    else
        return n * factorial( n - 1 );
}

template< class T >
class vector
{
    invariant
    {
        ( size() == 0 ) == empty();
        size() == std::distance( begin(), end() );
        size() == std::distance( rbegin(), rend() );
        size() <= capacity();
        capacity() <= max_size();
```

```
    }

                void resize( size_type newsize )
                  postcondition
                  {
                      size() == newsize;
                      if( newsize > oldof size() )
                          all_equals( begin() + oldof size(), end(), T() );
                  }

              void clear();
                  postcondition { empty(); }

              void swap( vector& right )
                  postcondition
                  {
                      oldof *this == right;
                      oldof right == *this;
                  }
              // ...

}; // class 'vector'
```

## 1.3 Prinzipien der ordnungsgemäßen Programmerstellung

1. Konstruktive Voraussicht und methodische Restriktion
2. Strukturierung
3. Modularisierung
4. Lokalität
5. Integrierte Dokumentation
6. Standardisierung
7. Funktionale und informelle Bindung
8. Schmale Datenkopplung
9. Vollständige Schnittstellenspezifikation
10. Lineare Kontrollstrukturen
11. Verbalisierung

## 1.4 Spezifikation einer abstrakten Datenkapsel

### 1.4.1 Axiomatische Spezifikation

TYPES
STACK[X]

FUNCTIONS

| | | |
|---|---|---|
| empty: | STACK[X] | $\nrightarrow$ BOOLEAN |
| new: | | $\rightarrow$ STACK[X] |
| push: | X x STACK[X] | $\nrightarrow$ STACK[X] |
| pop: | STACK[X] | $\nrightarrow$ STACK[X] |
| top: | STACK[X] | $\nrightarrow$ X |

PRECONDITIONS
pre pop (s: STACK[X]) = (not empty(s))
pre top (s: STACK[X]) = (not empty(s))

AXIOMS

| | |
|---|---|
| for all x:X, S : STACK[X]: | empty(new()) |
| | not empty (push(x,S)) |
| | top (push(x,S))=x |
| | pop (push(x,S))=S |

Vollständigkeit + Widerspruchsfreiheit (+ Unabhängigkeit)

### 1.4.2 Beschreibende (denotationale) Spezifikation

$Queue = Qelem^*$
$q_0 = [\ ]$

ENQUEUE $(e : Qelem)$
**ext wr** $q : Queue$
**post** $q = \overleftarrow{q} \frown [e]$

DEQUEUE() $e : Qelem$
**ext wr** $q : Queue$
**pre** $q \neq [\ ]$
**post** $\overleftarrow{q} = [e] \frown q$

ISEMPTY() $r : \mathbb{B}$
**ext rd** $q : Queue$
**post** $r \Leftrightarrow (len\ q = 0)$

„mathematische“ Modellierung mit Hilfe von Folgen, Mengen, ...
vergleiche VDM

Heute wird alternativ als geeignete denotationelle Spezifikationssprache die OCL immer beliebter.

### 1.4.3 Spezifikation durch Verträge

siehe SdV (Übungsblatt 3) und Rest dieser Veranstaltung

## 1.5 Prinzipien der Moduarisierung:

1. Module sollten **syntaktischen Einheiten** der Programmiersprache entsprechen.
2. Module sollten **mit möglichst wenigen anderen Modulen „kommunizieren“.**
3. „Kommunizierende“ Module sollten so **wenig** wie möglich **Informationen (Daten) austauschen**.
4. Jeder **Datentausch** zweier „kommunizierender“ Module muß **offensichtlich** in der Modulspezifikation (und nicht indirekt) kenntlich gemacht werden.
5. Alle **Daten** eines Moduls sollten **nur diesem bekannt** sein (außer im Falle einer gezielten Exportierung an möglichst wenige Nachbarmodule).
6. Ein Modul sollte **abgeschlossen und offen** sein.

## 1.6 Typen der Modularisierung

1. modulare **Zerlegbarkeit** (z.B. Top-Down-Design)
2. modulare **Zusammenfügbarkeit** (z.B. UNIX-Filter)
3. modulare **Verständlichkeit** (d.h. jede Modulbeschreibung selbsterklärend)
4. modulare **„Stetigkeit“**

   Kleine Spezifikationsänderungen wirken sich nur in **wenigen** Modulen aus. (Z.B. dyn. Felder, symbolische Konstanten, ...)
5. modularer **„Schutz“**

   Fehler/Ausnahmebedingungen bleiben in ihrer Auswirkung auf nur **wenige** Module beschränkt. (Z.B. direkte Konsistenzüberprüfung von Tastatureingaben, ...)

## 1.7 Spezifikation durch Verträge: `REQUIRE()`, `ENSURE()` und `invariant()`

- genaue Spezifikation der Methoden des Moduls:
  - **Vorbedingungen** (preconditions) einer Methode sind Bedingungen, die vor dem Aufruf einer Methode erfüllt sein müssen, damit sie ausführbar ist. Vorbedingungen sind boolsche Ausdrücke über den Abfragen des Moduls und den Parametern der Methode.
  - **Nachbedingungen**(postconditions) einer Methode sind Bedingungen, die nach dem Aufruf einer Methode erfüllt sind; sie beschreiben, welches Ergebnis ein Methodenaufruf liefert oder welchen Effekt er erzielt. Nachbedingungen sind boolsche Ausdrücke über den Abfragen des Moduls und den Parametern der Methode, erweitert um ein Gedächniskonstrukt, das die Werte von Ausdrücken vor dem Methodenaufruf liefert. Im Einzelnen:
    * Spezifikation des Funktionsergebnisses
    * genaue Spezifikation der Werte der Referenz- und der dereferenzierten Pointer-Paramter nach Beendigung der Methode
    * Spezifikation der Werte aller Attribute des Moduls nach Beendigung der Methode (häufig werden hier nicht einzeln genannte Attribute als nicht verändert angenommen)

- Definition der erlaubten Stati (Werte aller Attribute) des Moduls zu jedem (beobachtbaren) Zeitpunkt zur Laufzeit des Moduls.

  Sie werden durch Invarianten beschrieben. **Invarianten** eines Moduls sind allgemeine unveränderliche Konsistenzbedingungen an den Zustand des Moduls, die vor und nach dem Aufruf jeder (öffentlichen) Methode gelten. Formal sind Invarianten boolsche Ausdrücke über den Abfragen des Moduls; inhaltlich können sie z.B. Geschäftsregeln (business rules) ausdrücken.

(*vergleiche:* http://userserv.fh-reutlingen.de/∼hug/artikel/ForumWI01%20SdV.pdf)

**Ein Beispiel in C++ mit Hilfe von Nana:**

```
#define EIFFEL_CHECK CHECK_ALL
#include <set>
#include <vector>
#include <eiffel.h>
#include <nana.h>
...
void quicksort(double v[], int l, int h)
{
  REQUIRE(l <= h+1);
  ...
  ENSURE(A(int k=l, k<h, k++, v[k]<=v[k+1]));
```

```
void quicksort(double v[], int n)
{
  REQUIRE(n>=1);
  ID(multiset<double> v_old_contents(&v[0],&v[n]););
  ...
  ENSURE(A(int k=0, k<n-1, k++, v[k]<=v[k+1]));
  ID(multiset<double> v_contents(&v[0],&v[n]));
  ENSURE(v_old_contents == v_contents);
};

class name_list{
     ...
void name_list::put(const string& a_name)    // Push a_name into list
DO
  REQUIRE(/* name not in list */    !has(a_name));
  ID(set<string> contents_old(begin(),end()));
  ID(int count_old = get_count());
  ID(bool not_in_list = !has(a_name));
  ...
  ENSURE(has(a_name));
  ENSURE( (!not_in_list) || (get_count() == count_old + 1));
  ID(set<string> contents(begin(),end()));
  ENSURE( (!not_in_list) || (contents == contents_old + a_name));
END;
...
}
```

## 1.8 Klassifikation der Klassenmethoden gemäß PbC

- const-Methoden (Abfragen/Queries/Observatoren/Getter) teilt man in wesentliche und abgeleitete solche ein.
- Die wesentlichen Observatoren erlauben eine vollständige Spezifizierung des Zustands eines Klassenexemplars.
- Sie (und nur sie) werden nicht durch Nachbedingungen spezifiziert. Sie dienen vielmehr dazu, abgeleitete Observatoren und Modifikatoren (das sind nicht-const-Methoden) in ihren Nachbedingungen näher zu bestimmen.
- Dazu werden die abgeleiteten Observatoren durch eine Nachbedingung unter Benutzung einer oder mehrerer wesentlicher Observatoren spezifiziert.

- Modifikatoren werden durch eine Nachbedingung unter Benutzung aller wesentlicher Observatoren spezifiziert, um den exakten Zustand des Exemplars am Ende des Modifikatoraufrufs anzugeben.

- Verzichte (evtl.) in Nachbedingungen von Modifikatoren darauf, explizit zu spezifizieren, was sich nicht ändert (in der Annahme, dass alles nicht explizit genannte als *ungeändert* zu gelten hat). Leider ist nicht immer klar, was *ungeändert* zu bedeuten hat: Mindestens dann sollten Frameregeln (Rahmenbedingungen) explizit spezifizieren, was nach Aufruf des Modifikators *gleich* ist wie vorher.

- Explizite Spezifikation aller Rahmenbedingungen können bei programminterner Überprüfung der Nachbedingungen fehlerhafte Implementierungen aufdecken!

- Schreibe für jede Methode eine Vorbedingung mit Hilfe von
  - Abfragen und
  - Bedingungen an Methodenparameter.

  Hier (bei den Vorbedingungen) dürfen auch abgeleitete Abfragen, die eventuell effizienter sein können als eine sonst nötige Kombination mehrerer wesentlicher Abfragen, benutzt werden.

- Sorge dafür, dass bei Erfülltsein der Vorbedingungen auf jeden Fall die Nachbedingungen ebenfalls erfüllt sind (oder — in Ausnahmefällen — eine Exception ausgelöst wird).

- Sorge dafür, dass die Abfragen in Vorbedingungen effizient berechnet werden (evtl. durch Hinzufügen weiterer effizienter abgeleiteter Abfragen). Vergesse nicht, die evtl. hinzugefügten neuen abgeleiteten Abfragen durch Nachbedingungen (und Vorbedingungen) zu spezifizieren.

- Nutze Invarianten um die Abhängigkeit von Abfragen zu spezifizieren (Konsistenzbeziehungen).

- Untersuche alle Abfragen paarweise auf Redundanzen und formuliere solche explizit als Invarianten.

- Wann immer Abfrage-Ergebnisse oder Methoden-Parameter eingeschränkte Wertebereiche besitzen, formuliere dies explizit in Form von
  - Vorbedingungen,
  - Nachbedingungen

  oder
  - Invarianten.

- Schreibe die Nachbedingungen von virtuellen (also überschreibbaren) Methoden immer in der Form

  ```
  Vorbedingung implies Nachbedingung
  ```

(`Ensure((!Vorbedingung) || Nachbedingung)`), um die Redefinition in Kindklassen konfliktfrei zu ermöglichen.

## 1.9 Wiederverwendbarkeit

**Vermeide es, das Rad immer wieder neu zu erfinden!**

1. **Algorithmen (Programme)** lösen i. allg. eine Klasse von Problemen, die durch Eingabewerte parametrisiert sind.
   Wiktionary: Algorithmus
   Eigenschaften eines Algorithmus
   nondeterminism in Prolog

2. **Unterprogramme** (Funktionen, Prozeduren, Operatoren) lösen eine Klasse von Problemen: Gemäß dem Prinzip der methodischen Restriktion sind dabei die einzelnen Parameter jeweils Werte des Wertebereiches eines festen Typs.
   Prototype
   Deklaration
   C functions without prototypes
   Descriptions of function semantics (Page 420)
   exceptions specification
   exception class
   C++ std exception hierarchy

3. **Unterprogramme mit konformen Feldparametern** (in Pascal bzw. open-array-Parameter in Modula2) erlauben es Parametern, einer Klasse von Feldern anzugehören (variable Dimension);

   ```
   PROCEDURE EuklNorm (v:ARRAY OF REAL): REAL;
   ```

4. **Dynamische Felder / Teilfeld-Selektoren/Array slicing** erlauben einen in der Dimension noch nicht festgelegten Feldtyp bzw. Projektion auf ein Teilfeld:

   ```
   TYPE vector = ARRAY[*] OF REAL;
   a := t[*,2];
   ... t[min, k:l] ...
   ```

5. **Polymorphismen** (d.h. Überladen) **von Funktionen/Operatoren** erlauben die Benutzung einer mit demselben Namen versehenen Klasse von Funktionen, in denen jeder Parameter aus einer (disjunkten) Vereinigung von Typen stammen darf:

$writeln(x : real);$ $\quad\quad$ $k := i * j;$
$writeln(i : integer);$ $\quad\quad$ $z := x * y;$
$\ldots$

6. **Unterprogramme als Parameter** anderer Unterprogramme erlauben Algorithmen für eine Klasse von Unterprogrammen gleicher Signatur:

```
function Bisection (function f(x : real) : real;
                    xLeft, xRight        : real;
                    success              : boolean
                    ) : real;
```

7. **Generizität** ermöglicht Parametrisierung nach Typen:

```
generic
      type T is private;
procedure swap (x, y : in out T) is
      t : T
begin
      t := x; x := y; y := t
end swap
      ⋮
procedure int_swap is new swap (INTEGER);
```

Ada generic procedure swap, see page 86
Ada generic model

**Eingeschränkte Generizität** schränkt die aktuellen Typ-Parameter ein:

```
generic
    type T is private;
    with function "<=" (a, b : T) return BOOLEAN is <>;
function minimum (x, y : T) return T is
begin
   if x <= y then return x;
   else           return y
   end if
end minimum
```

(Ähnliches kann durch Textproprozessoren oder die typunsichere Benutzung des typungebundenen Zeigers `ADDRESS` erreicht werden oder in Java mittels constrained generic classes:

```
class ListObject<T extends Comparable<T>>{ ... }
```

Difference between a Java interface and a Java abstract class)

Irreführende unspezifische Fehlermeldungen bei der Nutzung von (uneingeschränkten) C++-Templates:

```
In file included from /usr/include/c++/4.5/algorithm:63:0,
                 from bad_error_eg.cpp:3:
/usr/include/c++/4.5/bits/stl_algo.h: In Funktion ≫void std::__insertion_sort(_RandomAccessIterator, _RandomAccessIterator) [with _RandomAccessIterator = _
/usr/include/c++/4.5/bits/stl_algo.h:3358:4:   instantiated from ≫void std::__inplace_stable_sort(_RandomAccessIterator, _RandomAccessIterator) [with _Rand
/usr/include/c++/4.5/bits/stl_algo.h:5415:2:   instantiated from ≫void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41:   instantiated from here
/usr/include/c++/4.5/bits/stl_algo.h:2103:4: Fehler: no match for ≫operator<≪ in ≫__i.__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator* [wit
/usr/include/c++/4.5/bits/stl_algo.h: In Funktion ≫void std::__merge_without_buffer(_BidirectionalIterator, _BidirectionalIterator, _BidirectionalIterator,
/usr/include/c++/4.5/bits/stl_algo.h:3364:7:   instantiated from ≫void std::__inplace_stable_sort(_RandomAccessIterator, _RandomAccessIterator) [with _Rand
/usr/include/c++/4.5/bits/stl_algo.h:5415:2:   instantiated from ≫void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41:   instantiated from here
/usr/include/c++/4.5/bits/stl_algo.h:2963:4: Fehler: no match for ≫operator<≪ in ≫__middle.__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator*
/usr/include/c++/4.5/bits/stl_algo.h: In Funktion ≫void std::__unguarded_linear_insert(_RandomAccessIterator) [with _RandomAccessIterator = __gnu_cxx::__no
/usr/include/c++/4.5/bits/stl_algo.h:2111:6:   instantiated from ≫void std::__insertion_sort(_RandomAccessIterator, _RandomAccessIterator) [with _RandomAcc
/usr/include/c++/4.5/bits/stl_algo.h:3358:4:   instantiated from ≫void std::__inplace_stable_sort(_RandomAccessIterator, _RandomAccessIterator) [with _Rand
/usr/include/c++/4.5/bits/stl_algo.h:5415:2:   instantiated from ≫void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41:   instantiated from here
/usr/include/c++/4.5/bits/stl_algo.h:2064:7: Fehler: no match for ≫operator<≪ in ≫__val < __next.__gnu_cxx::__normal_iterator<_Iterator, _Container>::ope
In file included from /usr/include/c++/4.5/vector:61:0,
                 from bad_error_eg.cpp:1:
/usr/include/c++/4.5/bits/stl_algobase.h: In Funktion ≫_ForwardIterator std::lower_bound(_ForwardIterator, _ForwardIterator, const _Tp&) [with _ForwardIter
/usr/include/c++/4.5/bits/stl_algo.h:2975:4:   instantiated from ≫void std::__merge_without_buffer(_BidirectionalIterator, _BidirectionalIterator, _Bidirec
/usr/include/c++/4.5/bits/stl_algo.h:3364:7:   instantiated from ≫void std::__inplace_stable_sort(_RandomAccessIterator, _RandomAccessIterator) [with _Rand
/usr/include/c++/4.5/bits/stl_algo.h:5415:2:   instantiated from ≫void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41:   instantiated from here
/usr/include/c++/4.5/bits/stl_algobase.h:976:4: Fehler: no match for ≫operator<≪ in ≫__middle.__gnu_cxx::__normal_iterator<_Iterator, _Container>::operat
In file included from /usr/include/c++/4.5/algorithm:63:0,
                 from bad_error_eg.cpp:3:
/usr/include/c++/4.5/bits/stl_algo.h: In Funktion ≫_FIter std::upper_bound(_FIter, _FIter, const _Tp&) [with _FIter = __gnu_cxx::__normal_iterator<std::comp
/usr/include/c++/4.5/bits/stl_algo.h:2982:4:   instantiated from ≫void std::__merge_without_buffer(_BidirectionalIterator, _BidirectionalIterator, _Bidirec
/usr/include/c++/4.5/bits/stl_algo.h:3364:7:   instantiated from ≫void std::__inplace_stable_sort(_RandomAccessIterator, _RandomAccessIterator) [with _Rand
/usr/include/c++/4.5/bits/stl_algo.h:5415:2:   instantiated from ≫void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41:   instantiated from here
/usr/include/c++/4.5/bits/stl_algo.h:2461:4: Fehler: no match for ≫operator<≪ in ≫__val < __middle.__gnu_cxx::__normal_iterator<_Iterator, _Container>::o
/usr/include/c++/4.5/bits/stl_algo.h: In Funktion ≫_OIter std::merge(_IIter1, _IIter1, _IIter2, _IIter2, _OIter) [with _IIter1 = std::complex<float>*, _IIt
/usr/include/c++/4.5/bits/stl_algo.h:2838:4:   instantiated from ≫void std::__merge_adaptive(_BidirectionalIterator, _BidirectionalIterator, _BidirectionalI
/usr/include/c++/4.5/bits/stl_algo.h:3315:7:   instantiated from ≫void std::__stable_sort_adaptive(_RandomAccessIterator, _RandomAccessIterator, _Pointer,
/usr/include/c++/4.5/bits/stl_algo.h:5417:2:   instantiated from ≫void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41:   instantiated from here
/usr/include/c++/4.5/bits/stl_algo.h:5299:4: Fehler: no match for ≫operator<≪ in ≫__first2.__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator*
/usr/include/c++/4.5/bits/stl_algo.h: In Funktion ≫_BidirectionalIterator3 std::__merge_backward(_BidirectionalIterator1, _BidirectionalIterator1, _Bidirec
/usr/include/c++/4.5/bits/stl_algo.h:2847:4:   instantiated from ≫void std::__merge_adaptive(_BidirectionalIterator, _BidirectionalIterator, _BidirectionalI
/usr/include/c++/4.5/bits/stl_algo.h:3315:7:   instantiated from ≫void std::__stable_sort_adaptive(_RandomAccessIterator, _RandomAccessIterator, _Pointer,
/usr/include/c++/4.5/bits/stl_algo.h:5417:2:   instantiated from ≫void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41:   instantiated from here
/usr/include/c++/4.5/bits/stl_algo.h:2740:4: Fehler: no match for ≫operator<≪ in ≫* __last2 < __last1.__gnu_cxx::__normal_iterator<_Iterator, _Container>
/usr/include/c++/4.5/bits/stl_algo.h: In Funktion ≫_OIter std::merge(_IIter1, _IIter1, _IIter2, _IIter2, _OIter) [with _IIter1 = __gnu_cxx::__normal_iterat
/usr/include/c++/4.5/bits/stl_algo.h:3163:4:   instantiated from ≫void std::__merge_sort_loop(_RandomAccessIterator1, _RandomAccessIterator1, _RandomAccess
/usr/include/c++/4.5/bits/stl_algo.h:3261:4:   instantiated from ≫void std::__merge_sort_with_buffer(_RandomAccessIterator, _RandomAccessIterator, _Pointer
/usr/include/c++/4.5/bits/stl_algo.h:3312:4:   instantiated from ≫void std::__stable_sort_adaptive(_RandomAccessIterator, _RandomAccessIterator, _Pointer,
/usr/include/c++/4.5/bits/stl_algo.h:5417:2:   instantiated from ≫void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41:   instantiated from here
/usr/include/c++/4.5/bits/stl_algo.h:5299:4: Fehler: no match for ≫operator<≪ in ≫__first2.__gnu_cxx::__normal_iterator<_Iterator, _Container>::operator*
/usr/include/c++/4.5/bits/stl_algo.h: In Funktion ≫_OIter std::merge(_IIter1, _IIter1, _IIter2, _IIter2, _OIter) [with _IIter1 = std::complex<float>*, _IIt
/usr/include/c++/4.5/bits/stl_algo.h:3163:4:   instantiated from ≫void std::__merge_sort_loop(_RandomAccessIterator1, _RandomAccessIterator1, _RandomAccess
/usr/include/c++/4.5/bits/stl_algo.h:3263:4:   instantiated from ≫void std::__merge_sort_with_buffer(_RandomAccessIterator, _RandomAccessIterator, _Pointer
/usr/include/c++/4.5/bits/stl_algo.h:3312:4:   instantiated from ≫void std::__stable_sort_adaptive(_RandomAccessIterator, _RandomAccessIterator, _Pointer,
/usr/include/c++/4.5/bits/stl_algo.h:5417:2:   instantiated from ≫void std::stable_sort(_RAIter, _RAIter) [with _RAIter = __gnu_cxx::__normal_iterator<std:
bad_error_eg.cpp:8:41:   instantiated from here
/usr/include/c++/4.5/bits/stl_algo.h:5299:4: Fehler: no match for ≫operator<≪ in ≫* __first2 < * __first1≪
```

zum Beispiel in

```
#include <vector>
#include <complex>
#include <algorithm>

int main()
 {
     std::vector<std::complex<float> > v;
     std::stable_sort(v.begin(), v.end());
 }
```

obwohl der Algorithmus `stable_sort()` schon unzählige Male zuvor problemlos benutzt wurde.
Natürlich sollte `stable_sort()` nur benutzt werden, wenn der `value_type` des zu sortierenden Containers einen `operator <=` besitzt, aber wie kann man das in der C++-Quelle der STL maschinell überprüfbar codieren?

Ansätze zur eingeschränkten Generizität in C++:

**Pseudo-Signaturen von Klassen**:

```
template<std::CopyConstructible T>
  requires Addable<T>
  T sum(T array[], int n)
  {
    T result = 0;
    for (int i = 0; i < n; ++i)
      result = result + array[i];
    return result;
  }
```

mit:

```
auto concept CopyConstructible<typename T> {
  T::T(T const&);
  T::~T();
};

auto concept Addable<typename T, typename U = T> {
  typename result_type;
  result_type operator+(T, U);
};
```

C++ Concepts: a Postmortem

**Eingeschränkte Generizität in C++0x mit Hilfe von des Makros BOOST_STATIC_ASSERT():**

```
#include <limits>
#include <boost/static_assert.hpp>

template <class UnsignedInt>
class myclass
{
private:
   BOOST_STATIC_ASSERT((std::numeric_limits<UnsignedInt>::
      digits >= 16)
                       && std::numeric_limits<UnsignedInt>::
                          is_specialized
                       && std::numeric_limits<UnsignedInt>::
                          is_integer
                       && !std::numeric_limits<UnsignedInt>::
                          is_signed);
public:
   /* details here */
};
// ...
#include <iterator>
#include <boost/static_assert.hpp>
#include <boost/type_traits.hpp>

template <class RandomAccessIterator>
RandomAccessIterator foo(RandomAccessIterator from,
                         RandomAccessIterator to)
{
   // this template can only be used with
   // random access iterators...
   typedef typename std::iterator_traits<
         RandomAccessIterator>::iterator_category cat;
   BOOST_STATIC_ASSERT(
      (boost::is_convertible<
         cat,
         const std::random_access_iterator_tag&>::value));
   //
   // detail goes here...
   return from;
}
// ...
```

**Boost Concept Check Library und Usage-Pattern:**

```
template<typename RanIter>
BOOST_CONCEPT_REQUIRES(
      ((Mutable_RandomAccessIterator<RanIter>))
      ((LessThanComparable<typename Mutable_RandomAccessIterator<
         RanIter>::value_type>)),
      (void)) // return type
      stable_sort(RanIter,RanIter);
```

mit für die STL vordefinierten Konzepten der Art:

```
template <class X>
struct InputIterator
  : Assignable<X>, EqualityComparable<X>
{
 private:
    typedef std::iterator_traits<X> t;
 public:
    typedef typename t::value_type value_type;
    typedef typename t::difference_type difference_type;
    typedef typename t::reference reference;
    typedef typename t::pointer pointer;
    typedef typename t::iterator_category iterator_category;

    BOOST_CONCEPT_ASSERT((SignedInteger<difference_type>));
    BOOST_CONCEPT_ASSERT((Convertible<iterator_category, std::
       input_iterator_tag>));

    BOOST_CONCEPT_USAGE(InputIterator)
    {
        X j(i);              // require copy construction
        same_type(*i++,v);   // require postincrement-
           dereference returning value_type
        X& x = ++j;          // require preincrement returning X
           &
    }
  private:
    X i;
    value_type v;

    // Type deduction will fail unless the arguments have the
       same type.
    template <typename T>
    void same_type(T const&, T const&);
};
```

**Nachteile der BCCL gegenüber der Pseudo-Signaturen:**

What's the difference between C++0x concepts and The Boost Concept Check Library

- Compiler brauchen Templates bis zur entgültigen Instantiierung nicht zu übersetzen, also auch nicht syntaktisch zu analysieren. Das Auftreten möglicher Fehlermeldungen ist deshalb bis zur Instantiierung aufgeschoben. Um eine vollständige Testabdeckung zu erreichen, benötigt man ein „Urmuster" des Gebrauchs aller nach Konzept vorgeschriebenen Operationen, die testcompiliert vorhandene fehlende Operationsdefinitionen aufdecken würde: einen **Archetyp** des Konzepts.

  Die ursprünglich in den C++0x geplanten Konzepte hätten Archtypen automatisch erzeugt und benutzt, somit die Templatedefinition automatisch vollständig typgechecked. Das bisherige C++-template-Handling läßt mögliche in der Dokumentation einer Template-Bibliothek unerwähne Requirements eines generischen Objekts lange unentdeckt und frustriert zu unvorhergesehenen Zeiten dessen Benutzer mit einer bis dahin nie aufgetretenen Fehlermeldung(skaskade): Motivierendes BCCL-Beispiel.

  Bei Benutzung der BCCL hat man eigene Konzepte selbst mit Archetypen auszustatten und diese testzucompilieren (siehe Abschmitt 1.21.3). Die in der BCCL vordefinierten für die STL nötigen Konzepte sind in der Datei `boost/concept_archetype.hpp` mit Archetypen ausgestattet.

  Zum Beispiel das Konzept `InputIterator` mit den geforderten Operationen `++i, (void)i++, *i++,*i; Defaultkonstruktor, operator=, operator->` `(TrivialIterator); Kopierkonstruktor, swap() (Assignable);` `operator==, operator!= (EqualtityComparable); Defaultkonstruktor` `(DefaultConstructible)` (vgl. STL InputIterator) und dem folgenden Archetpyp dafür:

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

// Iterator Archetype Classes

template <class T, int I = 0>
class input_iterator_archetype
{
private:
  typedef input_iterator_archetype self;
public:
  typedef std::input_iterator_tag iterator_category;
  typedef T value_type;
  struct reference {
```

```
      operator const value_type&() const { return
          static_object<T>::get(); }
    };
    typedef const T* pointer;
    typedef std::ptrdiff_t difference_type;
    self& operator=(const self&) { return *this;  }
    bool operator==(const self&) const { return true; }
    bool operator!=(const self&) const { return true; }
    reference operator*() const { return reference(); }
    self& operator++() { return *this; }
    self operator++(int) { return *this; }
  };
```

- BCCL unterstützt die Überprüfung von semantischen Requirements wie z.B. Benutzbarkeit in Multipass-Algorithmen ... **nicht**!
- BCCL unterstützt das Syntaxremappping (temporäres renaming) **nicht**.
- BCCL unterstützt Kontext-basiertes Überladen **nicht**.

**Händische Konstruktion von Archetypen zur vollständigen Testabdeckung:**

```
template <class T, int I = 0>
class input_iterator_archetype
{
private:
  typedef input_iterator_archetype self;
public:
  typedef std::input_iterator_tag iterator_category;
  typedef T value_type;
  struct reference {
    operator const value_type&() const { return static_object
      <T>::get(); }
  };
  typedef const T* pointer;
  typedef std::ptrdiff_t difference_type;
  self& operator=(const self&) { return *this;  }
  bool operator==(const self&) const { return true; }
  bool operator!=(const self&) const { return true; }
  reference operator*() const { return reference(); }
  self& operator++() { return *this; }
  self operator++(int) { return *this; }
};
```

**Fallweises Überladen in C++:**

```
template<typename T>
typename std::enable_if<std::is_pod<T>::value,void>::type
copy(T const* source,T* dest,unsigned count)
{
    memcpy(dest,source,count*sizeof(T));
}

template<typename T>
typename std::enable_if<!std::is_pod<T>::value,void>::type
copy(T const* source,T* dest,unsigned count)
{
    for(unsigned i=0;i<count;++i)
    {
        *dest++=*source++;
    }
}
```

Zur Vertiefung:

ConceptC++ Publications
Usage Pattern vs. pseudo-signature
Generic Programming Techniques

## 1.10 Begriffshierarchien

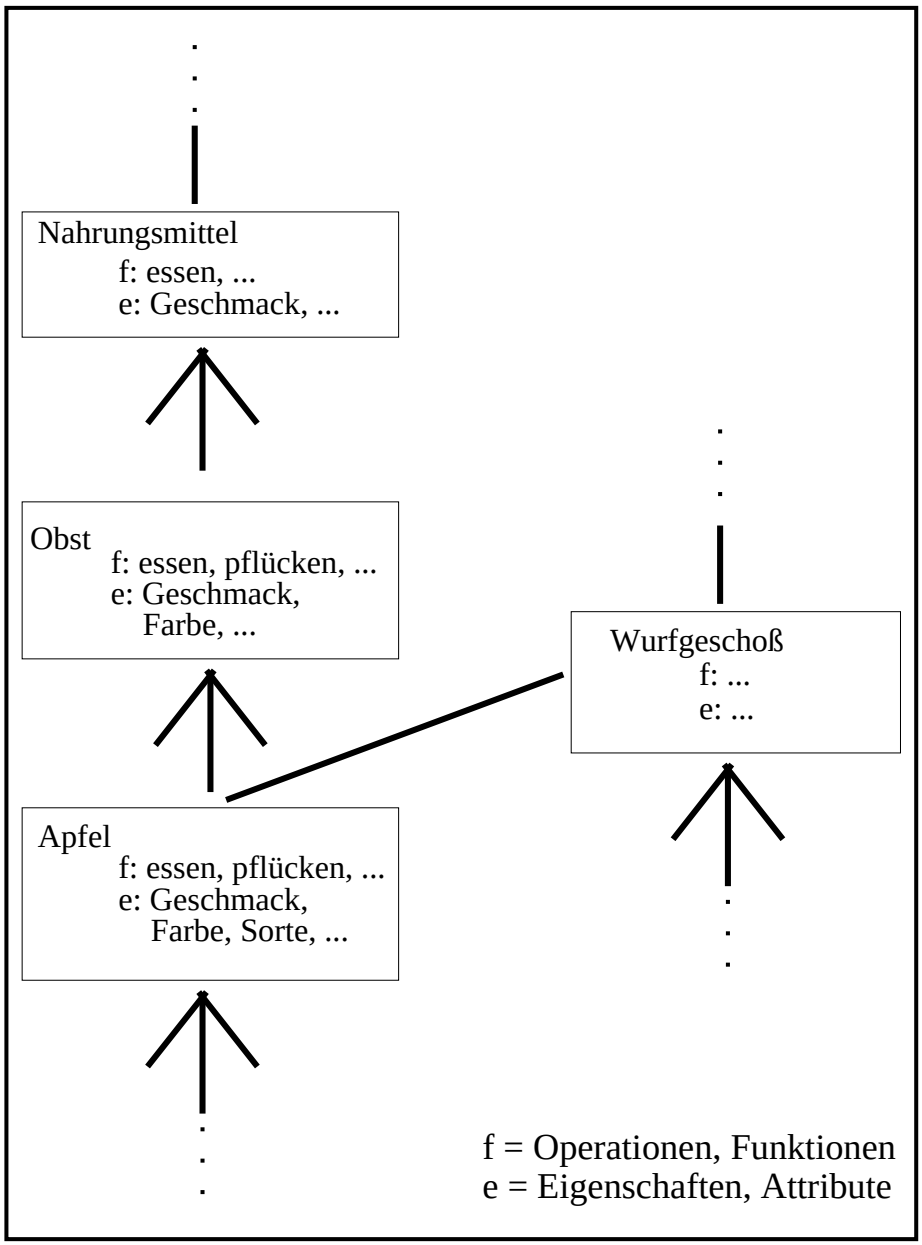



Abbildung 1.1: Begriffshierarchien

## 1.11 Umgangssprachliche Spezifikation?

„Informelle Beschreibung“: Auf einem Parkplatz stehen PKW's und Motoräder. Zusammen seien es $n$ Fahrzeuge mit insgesamt $m$ Rädern. Bestimme die Anzahl $P$ der PKW's.
**„Lösung“:** Sei

$P :=$ Anzahl der PKW's
$M :=$ Anzahl der Motoräder

$$\left\{ \begin{array}{r} P + M = n \\ 4P + 2M = m \end{array} \right\} \Leftrightarrow \left\{ \begin{array}{c} M = n - P \\ P = \frac{m-2n}{2} \end{array} \right\} \Leftrightarrow \left\{ \begin{array}{c} M = \frac{4n-m}{2} \\ P = \frac{m-2n}{2} \end{array} \right\}$$

**„Algorithmus“:**

```
M := (4 * n - m) / 2;
P := (m - 2 * n) / 2;
write (M,P);
```

**Problem:** ****Null-Euro-Rechnung, Null-Euro-Mahnung,...

$(m, n) = (9, 3) \Rightarrow P = 1\frac{1}{2}$
$(m, n) = (2, 5) \Rightarrow P = -4$

Vor der Entwicklung eines Algorithmus ist zunächst für das Problem eine *Spezifikation* bestehend aus

1. Definitionsbereich,
2. Wertebereich *und*
3. für die Lösung wichtigen Eigenschaften (insbesondere funktionaler Zusammenhang zwischen Eingabe- und Ausgabegrößen)

anzufertigen.
Besser ist also:

**Eingabe:** $m \in \{0, 1, ..., INT\_MAX\}, n \in \{0, 1, ..., INT\_MAX/2\}$

**Vorbedingungen:** $m$ gerade, $2n \leq m \leq 4n$

**Ausgabe:** $P \in \{0, 1, ..., INT\_MAX\}$, falls die Nachbedingung erfüllt ist (sonst „keine Lösung“)

**Nachbedingung:** Ein $(P, M) \in \{0, 1, ..., INT\_MAX\}$ mit

$$\begin{array}{c} P + M = n \\ 4P + 2M = m \end{array}$$

## 1.12 Erste einfache Contracts in Eiffel: Vorbedingungen und Klassen-Invarianten

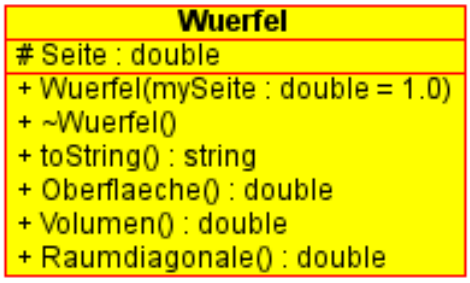
```
class
        WUERFEL

inherit
        DOUBLE_MATH

create
        make

feature -- Access

        seite : REAL_64

        make( s : REAL_64 )  is

                        require
                                argument_nonnegative: s >= 0.0
                        do
                                seite := s
                        end
feature -- Status report

        Oberflaeche : REAL_64 is
                        -- berechnet die Oberfkaeche
                        do
                                Result := (6 * seite ^ 2)
                        end

        Volumen : REAL_64 is
                        -- berechnet das Volumen
                        do
                                Result := seite ^ 3
                        end


invariant
        seite_nonnegative: seite >= 0.0

end
```

## 1.13 IDE für C++-Programmierung: Eclipse Helios

Eclipse Helios mit Linux-Tools
http://www.eclipse.org/cdt/
CDT Documentation, Tutorials, ...
C/C++ Development User Guide
Eclipse für C/C++-Programmierer
„S. Bauer: Eclipse für C/C++-Programmierer“ in der Bibliothek der BUW

## 1.14 Ein erstes C++-Projekt mit Umbrello und Doxygen

Umbrello
Umbrello Handbuch




Umbrello automatische Quelltexterzeugung liefert:

```
#ifndef WUERFEL_H
#define WUERFEL_H

#include <string>

/**
  * class Wuerfel
  *
  */

class Wuerfel
{
public:

  // Constructors/Destructors
  //


  /**
   * Empty Constructor
   */
  Wuerfel ( );

  /**
   * Empty Destructor
   */
  virtual ~Wuerfel ( );

  // Static Public attributes
```

```
  //

  // Public attributes
  //


  // Public attribute accessor methods
  //


  // Public attribute accessor methods
  //



  /**
   * @param  mySeite
   */
   Wuerfel (double mySeite = 1.0 )
  {
  }


  /**
   * @return string
   */
  string toString ( ) const
  {
  }


  /**
   * @return double
   */
  double Oberflaeche ( ) const
  {
  }


  /**
   * @return double
   */
  double Volumen ( ) const
  {
  }


  /**
   * @return double
   */
  double Raumdiagonale ( ) const
  {
  }

protected:

  // Static Protected attributes
  //

  // Protected attributes
  //

  double Seite;
public:
```

```
  // Protected attribute accessor methods
  //

protected:

public:


  // Protected attribute accessor methods
  //


  /**
   * Set the value of Seite
   * @param new_var the new value of Seite
   */
  void setSeite ( double new_var )   {
      Seite = new_var;
  }

  /**
   * Get the value of Seite
   * @return the value of Seite
   */
  double getSeite ( )   {
    return Seite;
  }
protected:


private:

  // Static Private attributes
  //

  // Private attributes
  //

public:


  // Private attribute accessor methods
  //

private:

public:


  // Private attribute accessor methods
  //

private:


  void initAttributes ( ) ;

};

#endif // WUERFEL_H
```

Die zugehörige Implementierungsdatei:

```
#include "Wuerfel.h"

// Constructors/Destructors
//
```

```
Wuerfel::Wuerfel ( ) {
initAttributes();
}

Wuerfel::~Wuerfel ( ) { }

//
// Methods
//


// Accessor methods
//


// Other methods
//

void Wuerfel::initAttributes ( ) {
}
```

Beachte die speziellen dokumentationserzeugenden Kommentare gemäß Doxygen-Syntax.

Ein Aufruf von `doxygen` erzeugt aud dem Quelltext automatisch Dokumentation im pdf oder html-Format. Vergleiche dazu:
Doxywizard usage
Doxygen: Generate documentation from source code
Doxygen Manual

**Constructor & Destructor Documentation**

**Wuerfel::Wuerfel ( )**

Empty Constructor

Definition at line 6 of file Wuerfel.cpp.

**Wuerfel::~Wuerfel ( ) [virtual]**

Empty Destructor

Definition at line 10 of file Wuerfel.cpp.

**Wuerfel::Wuerfel ( double mySeite = 1.0 ) [inline]**

**Parameters:**
*mySeite*

Definition at line 49 of file Wuerfel.h.

**Member Function Documentation**

**double Wuerfel::getSeite ( ) [inline]**

Get the value of Seite

**Returns:**
the value of Seite

Hinweis zum Erzeugen der pdf-Dokumentation:

```
cd documentation/latex
make pdf
evince refman.pdf
```

## 1.15 Ein erstes Eclipse-C++-Projekt

Erzeuge in `$HOME/bin` eine Datei `eclipse` mit dem Inhalt

```
#!/bin/sh
/usr/local/sw/eclipse/eclipse $*
```

sofern Eclipse-Helios im Pfad `/usr/local/sw/eclipse/` installiert wurde. Erzeuge dann eine Desktop-Starterdatei ähnlich:

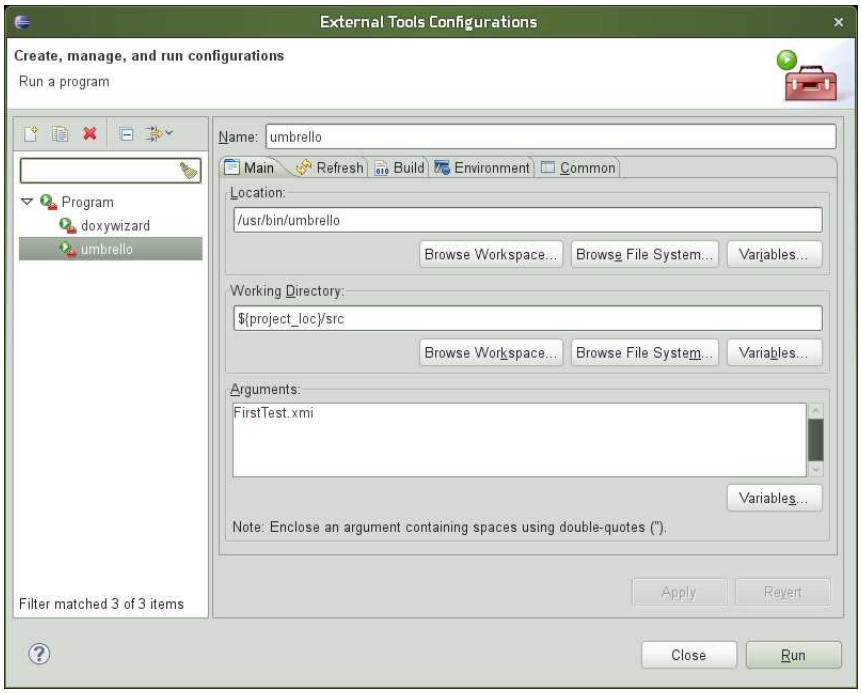
```
#!/usr/bin/env xdg-open

[Desktop Entry]
Version=1.0
Type=Application
Terminal=false
Icon[de_DE]=eclipse
Name[de_DE]=helios
Exec=/home/username/bin/eclipse
Name=helios
Icon=eclipse
```

Jetzt steht auf dem Desktop ein Icon zum Start von Eclipse-Helios bereit:

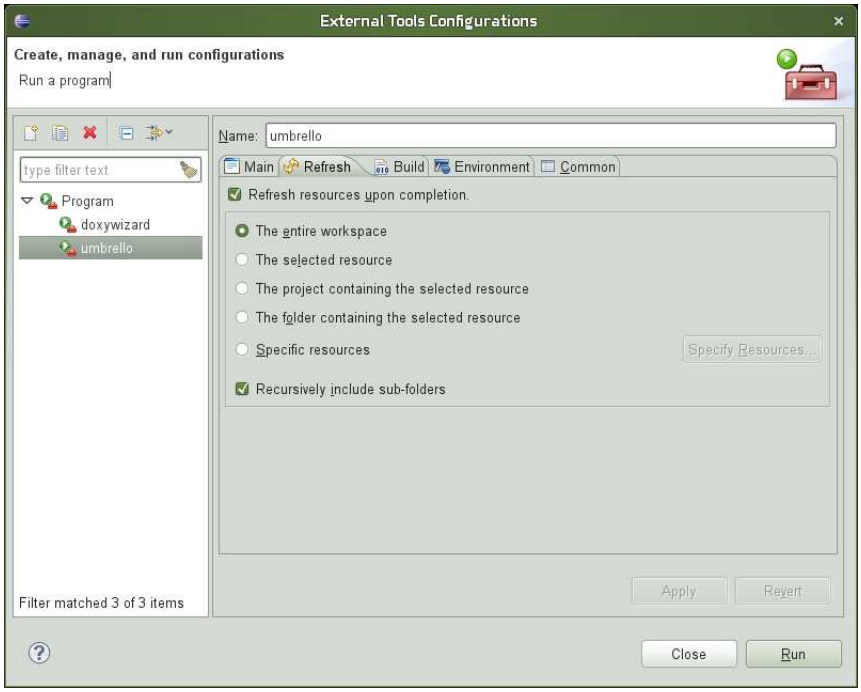

Ein erstes Eclipse C++-Projekt:
http://max.berger.name/howto/cdt/ar01s04.jsp#helloworld

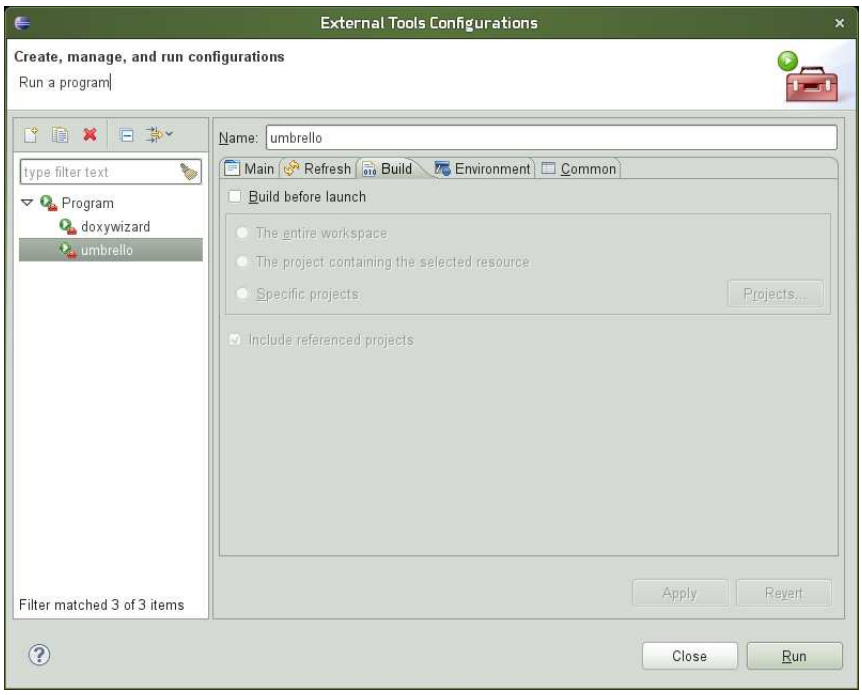
### Umbrello in eclipse starten:

```
External Tools
   External Tools Configuration
      New launch configuration
```

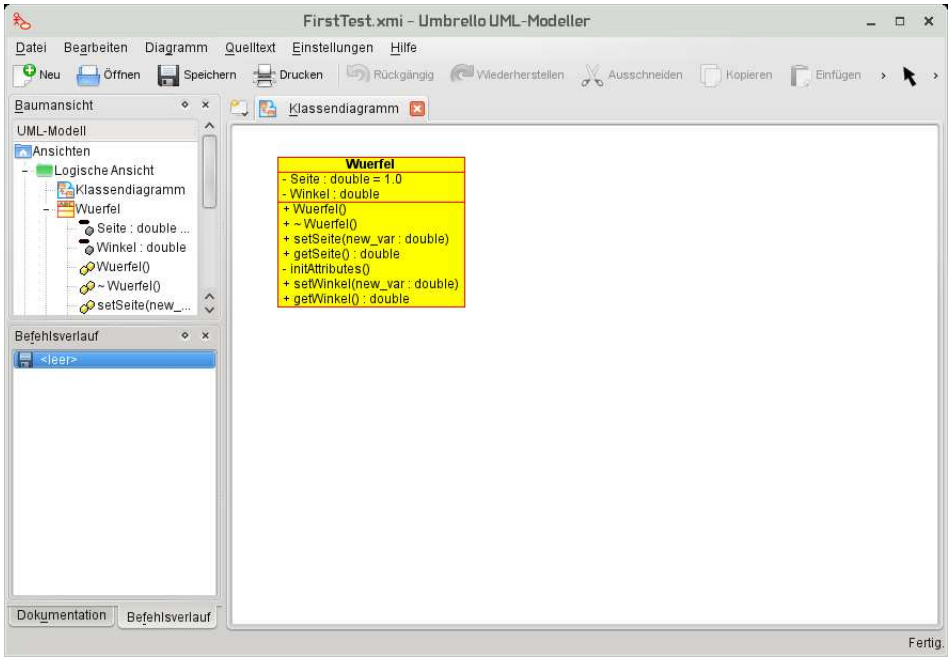



External tools variables (Warum `${project_loc}` statt `${workspace_loc}`?)



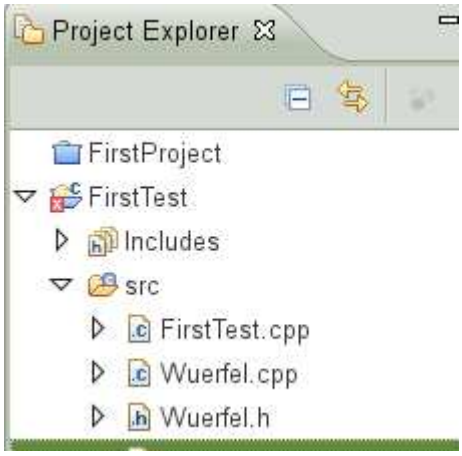



Nach Betätigung des `Run`-Knopfes können Sie ein UML-Modell konzipieren



und mittels des Umbrello-Assistenten für Quelltext-Generierung nach Angabe des Ordners für die Quelldateien (das Projekt-Quellverzeichniß händisch ohne Benutung der eclipse-Variablen eintragen)

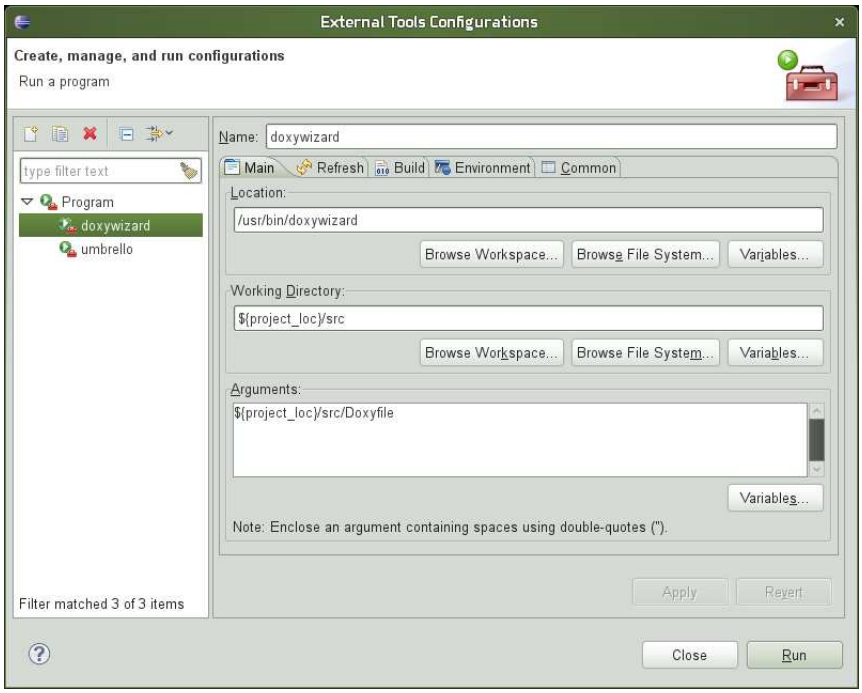

die Quelldateien in Eclipse unmittelbar benutzen:



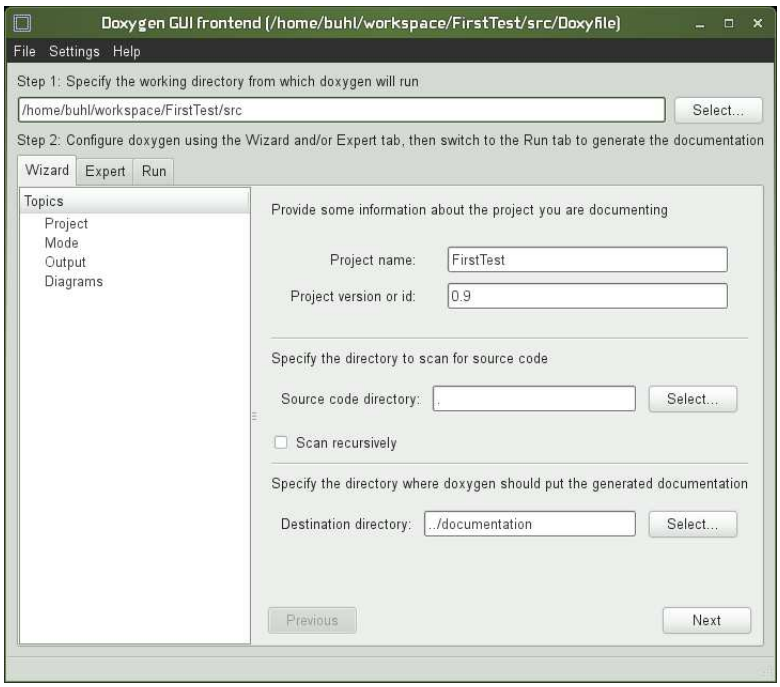

Selbst das Quelltext-Klassen(re)importieren von in Eclipse geänderten Klassen zurück ins Umbrello-UML-Modell funktioniert einwandfrei.

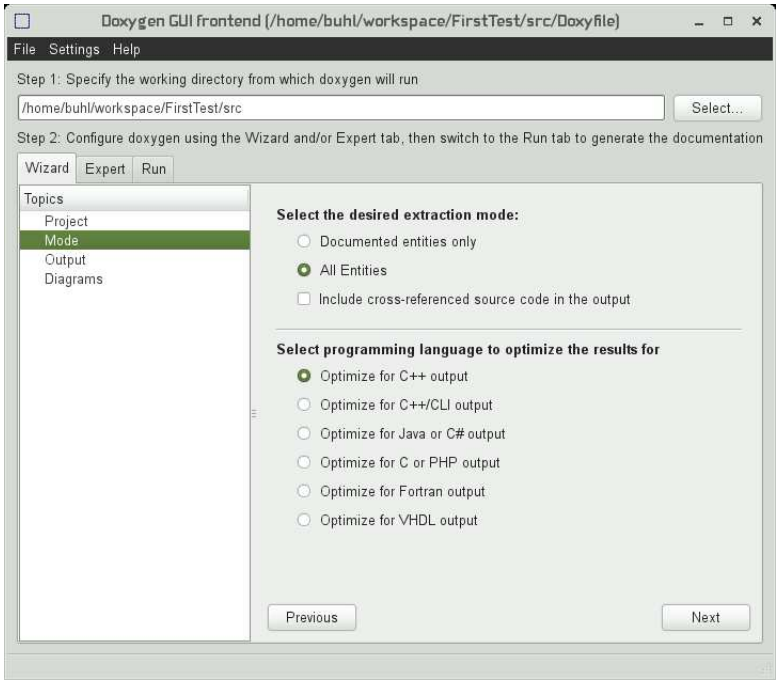
### Doxygen-Integration in eclipse:

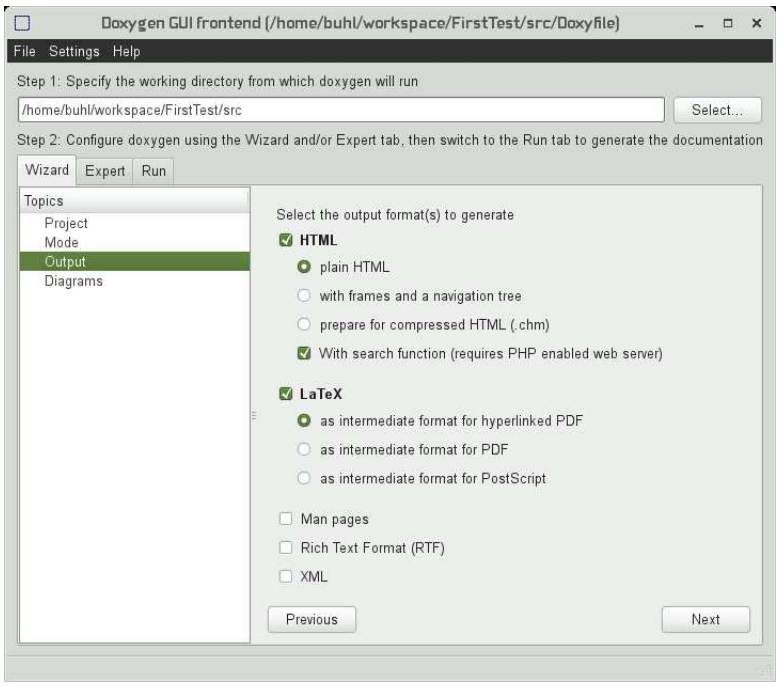
```
External Tools
   External Tools Configuration
      New launch configuration
```

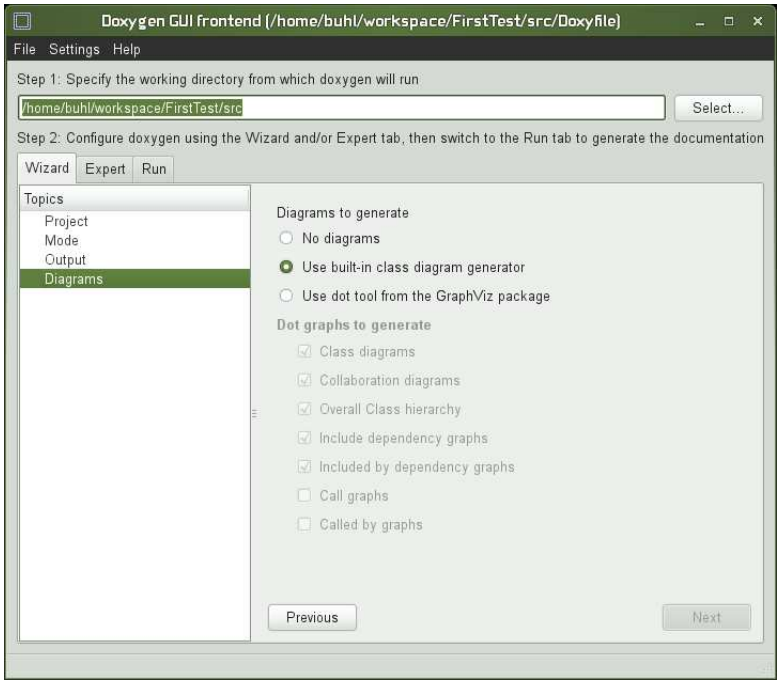



`Refresh` und `Build`-Einstellungen analog wie bei `umbrello`. Nach Betätigung des `Run`-Knopfes können Sie nun die `Doxywizard`-Konfiguration anlegen:

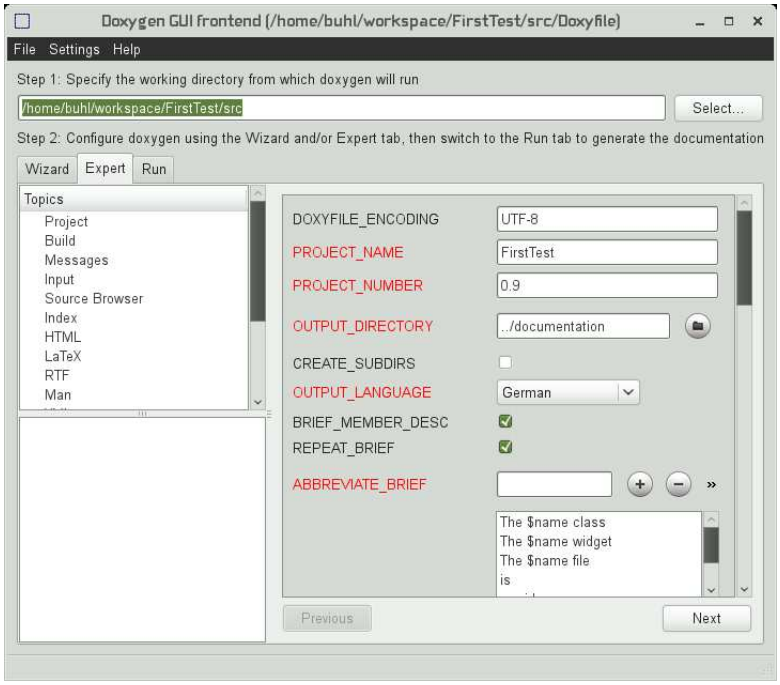



Doxygen GUI frontend (/home/buhl/workspace/FirstTest/src/Doxyfile)
File Settings Help
Step 1: Specify the working directory from which doxygen will run
/home/buhl/workspace/FirstTest/src
Select...
Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation
Wizard
Expert
Run
Topics
Project
Mode
Output
Diagrams
Select the desired extraction mode:
Documented entities only
All Entities
Include cross-referenced source code in the output
Select programming language to optimize the results for
Optimize for C++ output
Optimize for C++/CLI output
Optimize for Java or C# output
Optimize for C or PHP output
Optimize for Fortran output
Optimize for VHDL output
Previous
Next


Doxygen GUI frontend (/home/buhl/workspace/FirstTest/src/Doxyfile)
File Settings Help
Step 1: Specify the working directory from which doxygen will run
/home/buhl/workspace/FirstTest/src
Select...
Step 2: Configure doxygen using the Wizard and/or Expert tab, then switch to the Run tab to generate the documentation
Wizard
Expert
Run
Topics
Project
Mode
Output
Diagrams
Select the output format(s) to generate
HTML
plain HTML
with frames and a navigation tree
prepare for compressed HTML (.chm)
With search function (requires PHP enabled web server)
LaTeX
as intermediate format for hyperlinked PDF
as intermediate format for PDF
as intermediate format for PostScript
Man pages
Rich Text Format (RTF)
XML
Previous
Next

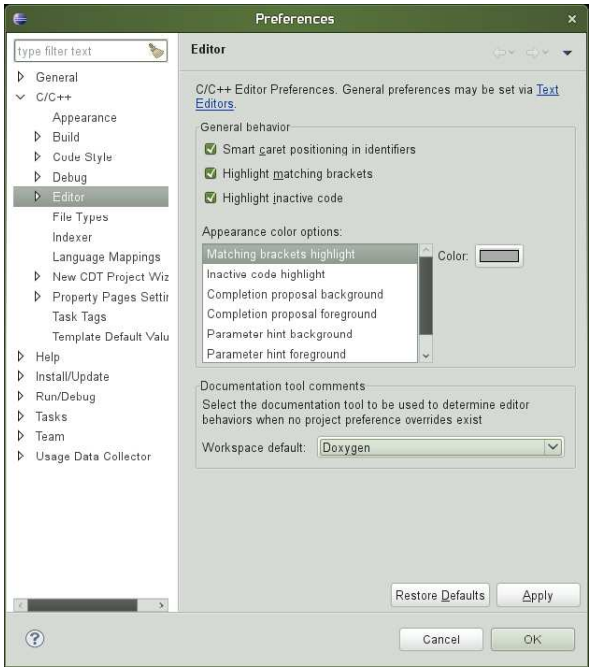



Vergessen Sie nicht, über den `Expert`-Reiter die `OUTPUT_LANGUAGE` auf `German` umzustellen:

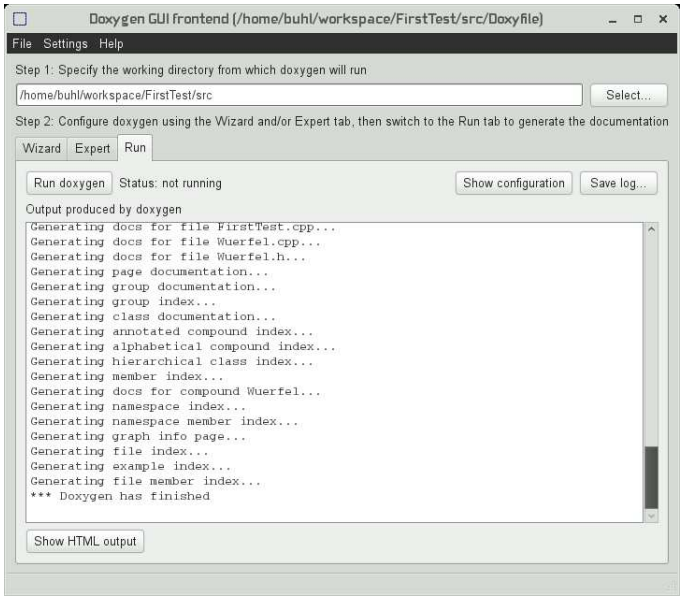


Schließlich müssen Sie noch die eclipse-Editor-Unterstützung für `doxygen`-Spezialkommentare aktivieren:

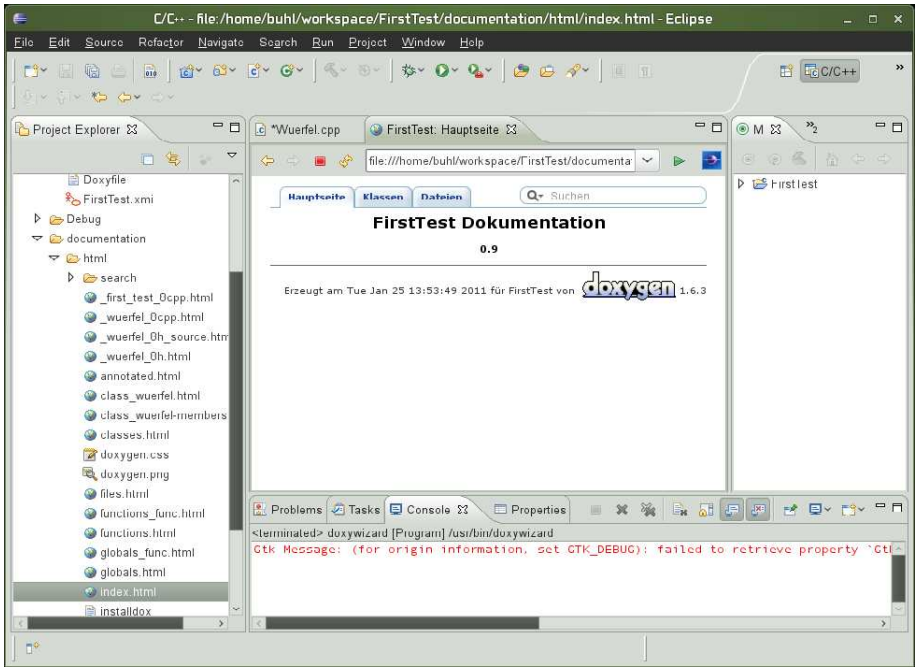
`Windows`
`Preferences`
`Editors`




Jetzt können Sie über das externe Tool `doxywizard` jederzeit eine aktuelle Dokumentationsversion erzeugen



und mittels des `Show HTML output`-Knopfes in einem externen Browser betrachten oder Sie doppelklicken die Datei `index.html` im `documentation`-Unterordner des eclipse `Project Explorers`:




doxygen-Spezialkommandos: http://www.stack.nl/~dimitri/doxygen/docblocks.html
Beachten Sie insbesondere `@mainpage`

```
/*!
 * @mainpage Wuerfel, ...
 *
 * @version 0.99
 * @author HJB
 * @date 25. AprilJanuar 2011
 */
```

und:

```
/*! Testrahmenprogramm
  *
  * erzeugt ein Auto und einen LKW,
  * druckt deren Attribute
  *
  * @param[in] argc Anzahl der Kommandozeilen-Parameter
  * @param[in] argv Feld mit den Kommandozeilen-Parametern
  * @return EXIT_SUCCESS
  */
```

```
...
/*!
 * lese (Kfz-)Kennzeichen
 * @return Kennzeichen
 *
 * @warning eventuell die Zeichenkette "unbekannt"
 */
...
```

sowie die Contract-Dokumentation in `doxygen`:

`@pre { description of the precondition }`
Starts a paragraph where the precondition of an entity can be described. The paragraph will be indented. The text of the paragraph has no special internal structure. All visual enhancement commands may be used inside the paragraph. Multiple adjacent `@pre` commands will be joined into a single paragraph. Each precondition will start on a new line. Alternatively, one `@pre` command may mention several preconditions. The `@pre` command ends when a blank line or some other sectioning command is encountered.

`@post { description of the postcondition }` Starts a paragraph where the postcondition of an entity can be described. The paragraph will be indented. The text of the paragraph has no special internal structure. All visual enhancement commands may be used inside the paragraph. Multiple adjacent `@post` commands will be joined into a single paragraph. Each postcondition will start on a new line. Alternatively, one `@post` command may mention several postconditions. The `@post` command ends when a blank line or some other sectioning command is encountered.

`@invariant { description of invariant }` Starts a paragraph where the invariant of an entity can be described. The paragraph will be indented. The text of the paragraph has no special internal structure. All visual enhancement commands may be used inside the paragraph. Multiple adjacent `@invariant` commands will be joined into a single paragraph. Each invariant description will start on a new line. Alternatively, one `@invariant` command may mention several invariants. The `@invariant` command ends when a blank line or some other sectioning command is encountered.

Ein Beispiel für die Contract-Befehle in `doxygen`

```
/*!
 * Konstruktor
 * @param[in] mySeite
 * @pre{ mySeite >= 0 }
 */
...
```

```
/*!
 * Oberflaeche
 * @return Oberflaeche
 * @post{ approximatelyEqualTo(result, 6.0 * Seite * Seite, 1.0) }
...
/*!
 * 3D-Wuerfel
 *
 * @invariant{ Seite >= 0.0 }
 */
class Wuerfel{
...
}
 */
```

und die Ergenisse:

**Wuerfel Klassenreferenz**

`#include <Wuerfel.h>`

Aufstellung aller Elemente

**Öffentliche Methoden**

| | |
|---|---|
| | **Wuerfel** (double mySeite=1.0) |
| virtual | **~Wuerfel** () |
| virtual bool | **invariant** () |
| std::string | **toString** () |
| double | **Oberflaeche** () = 6.0 * pow(**Seite**, 2) |
| double | **Volumen** () = pow(**Seite**,3) |
| double | **Raumdiagonale** () = **Seite** * sqrt(3.0) |

**Geschützte Attribute**

| | |
|---|---|
| double | **Seite** |

---

**Ausführliche Beschreibung**

3D-Wuerfel

**Invariant:**
{ Seite >= 0.0 }

Definiert in Zeile **33** der Datei **Wuerfel.h**.

---

**Beschreibung der Konstruktoren und Destruktoren**

**Wuerfel::Wuerfel ( double *mySeite* = *1.0* )**

Konstruktor

**Parameter:**
[in] *mySeite*

**Vorbedingung:**
{ mySeite >= 0 }

Definiert in Zeile **20** der Datei **Wuerfel.cpp**.

**Dokumentation der Funktionen**

**static double myown_sqrt ( double *x* ) [static]**

Quadratwurzel nach Newton

**Parameter:**
[in] *x*

**Rückgabe:**
Wurzel

**Vorbedingung:**
{ x >= 0.0 }

**Nachbedingung:**
{approximatelyEqualTo(result*result, x, 1.0)}

Definiert in Zeile **37** der Datei **double_math.h**.

Benutzt **approximatelyEqualTo()**, **ENSURE()** und **result**.

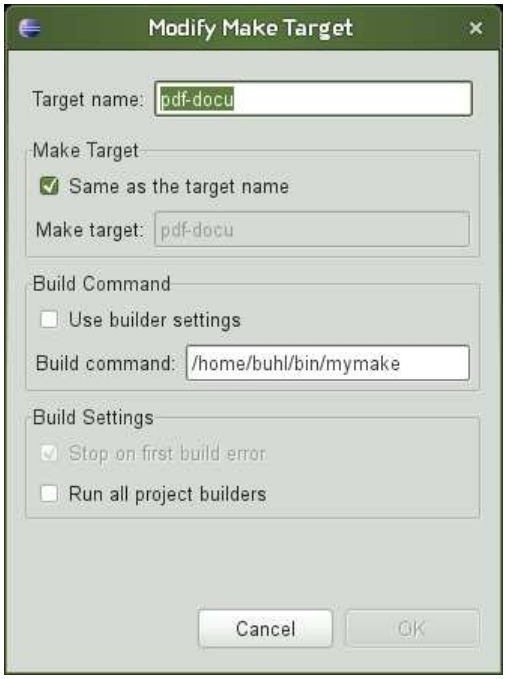
Wird benutzt von **main()**.

Beispieldokumentationen:
http://gcc.gnu.org/onlinedocs/libstdc++/latest-doxygen/modules.html (STL-Docu)
http://www.stack.nl/~dimitri/doxygen/results.html
http://www.stack.nl/~dimitri/doxygen/projects.html

doxygen-Handbuch
Quick Reference

**Ein Make-Target zur Erzeugung der pdf-doygen-Dokumentation:**
Bei angeklickter Projekt-Quelldatei

```
Project
   Make Target    >
      Create ...
```

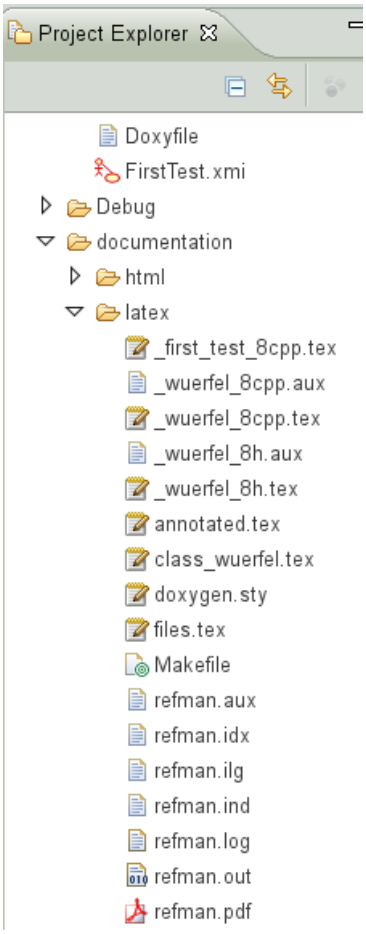


```
> cd $HOME/bin

> emacs mymake
#!/bin/sh
cd ../documentation/latex
make

>chmod 755 mymake
```

Im Kontext einer Quelldatei kann nun mittels

```
Project
   Make Target    >
      Build ...
         pdf-docu    Build
```

die Datei `refman.pdf` erzeugt




und benutzt werden.

## 1.16 Nachbedingungen mit Gleitkommawerten: absolute oder relative Abweichung

**Contracts in C++ mit nana**
nana
savannah
Download- und Installationsbeschreibung

Konfiguration von `eclipse` zur `nana`-Nutzung: Projekt anklicken, dann:

```
File
  Properties
    C/C++ Build
      Settings
        GCC C++ Compiler
          Includes (-I)
            /home/username/include

        GCC C++ Linker
          Libraries
            Library search path (-L)
              /home/username/lib
            Libraries (-l)
              nana
```

### 1.16.1 Klasse Wuerfel

```
/*
 * Wuerfel.h
 *
 *  Created on: 22.04.2009
 *      Author: buhl
 */
#ifndef WUERFEL_H_
#define WUERFEL_H_
#define EIFFEL_DOEND
#ifndef EIFFEL_CHECK
#define EIFFEL_CHECK  CHECK_ALL
//                                          Makros CHECK() und folgende
//                    CHECK_LOOP            Makros INVARIANT() und folgende
//                     CHECK_INVARIANT      Methode invariant() und folgende
//                      CHECK_ENSURE        Nachbedingungen und folgende
//                       CHECK_REQUIRE      Vorgedingungen
//                        CHECK_NO
```

```
#endif
#include <eiffel.h>
#include <nana.h>
#include <string>
class Wuerfel{
public:
        Wuerfel(double mySeite = 1.0);
        virtual ~Wuerfel();
        virtual bool invariant();

        std::string  toString();

        double Oberflaeche();

        double Volumen();

        double Raumdiagonale();
protected:
        double Seite;
};
#endif /* WUERFEL_H_ */
```

### 1.16.2 Contract der Klasse Wuerfel

```
/*
 * Wuerfel.cpp
 *
 *  Created on: 22.04.2009
 *      Author: buhl
 */
#include <sstream>
#include <limits>
#include <cmath>
#include "Wuerfel.h"
#include "double_adds.h"
#include "double_math.h"

Wuerfel::Wuerfel(double mySeite): Seite(mySeite)
{
        REQUIRE(mySeite >= 0);
        ENSURE(invariant());
}

Wuerfel::~Wuerfel() {
```

```
        // TODO Auto-generated destructor stub
}

bool Wuerfel::invariant()
{
        return Seite >= 0.0;
}

std::string Wuerfel::toString()
{
        std::ostringstream help;
        help << Seite << ";" << Oberflaeche() << ";" << Volumen() << ";"
             << Raumdiagonale();
        return help.str();
}
double Wuerfel::Oberflaeche()
DO
        double result = 6.0 * pow(Seite, 2);
        ENSURE(result == 6.0 * Seite * Seite);
        ENSURE(invariant());
        return result;
}

double Wuerfel::Volumen()
DO
        double result = pow(Seite,3);
        ENSURE(withinEpsilonOf(result, Seite*Seite*Seite, 1.0E-6));
        ENSURE(invariant());
        return result;
}

double Wuerfel::Raumdiagonale()
DO
        double result = Seite * sqrt(3.0);
        ENSURE(approximatelyEqualTo(result, Seite * myown_sqrt(3.0), 1.0));
        ENSURE(invariant());
        return(result);
}
```

Naiver Wertevergleich, absolute Abweichung und relative Abweichung: Diskutieren Sie den sinnvollen Einsatz!

http://msdn.microsoft.com/en-us/library/6x7575x3(VS.80).aspx

http://www.ib.cnea.gov.ar/~oop/biblio/libstdc++/structstd_1_1numeric_limits_3_01double_01_4-members.html

http://www2.roguewave.com/support/docs/leif/sourcepro/html/stdlibref/numeric-limits.html

http://docs-pdf.sun.com/800-7895/800-7895.pdf

http://www.cplusplus.com/reference/clibrary/cfloat/

http://realtimecollisiondetection.net/blog/?p=89

### 1.16.3 double_adds.h

```
/*
 * double_adds.h
 *
 *  Created on: 13.06.2009
 *      Author: buhl
 */

#ifndef DOUBLE_ADDS_H_
#define DOUBLE_ADDS_H_

#define EIFFEL_DOEND
#ifndef EIFFEL_CHECK
#define EIFFEL_CHECK  CHECK_ALL
//                                          Makros CHECK() und folgende
//                    CHECK_LOOP            Makros INVARIANT() und folgende
//                     CHECK_INVARIANT      Methode invariant() und folgende
//                      CHECK_ENSURE        Nachbedingungen und folgende
//                       CHECK_REQUIRE      Vorgedingungen
//                        CHECK_NO
#endif
#include <eiffel.h>
#include <nana.h>

#include <limits>
#include <cmath>
#include <algorithm>

static bool withinEpsilonOf(double left, double right, double delta)
{
        return fabs(left - right) <= delta;
}

static bool approximatelyEqualTo(double left, double right, double factor)
{
        return fabs(left - right) <=  std::numeric_limits<double>::epsilon( ) * f
```

```
                                                std::max(fabs(left), fabs(right));
}

#endif /* DOUBLE_ADDS_H_ */
```

### 1.16.4 double_math.h

Alternativ-Implementierungen für Nachbedingungen:

```
/*
 * double_math.h
 *
 *  Created on: 13.06.2009
 *      Author: buhl
 */

#ifndef DOUBLE_MATH_H_
#define DOUBLE_MATH_H_

#define EIFFEL_DOEND
#ifndef EIFFEL_CHECK
#define EIFFEL_CHECK  CHECK_ALL
//                                          Makros CHECK() und folgende
//                    CHECK_LOOP            Makros INVARIANT() und folgende
//                     CHECK_INVARIANT      Methode invariant() und folgende
//                      CHECK_ENSURE        Nachbedingungen und folgende
//                       CHECK_REQUIRE      Vorgedingungen
//                        CHECK_NO
#endif
#include <eiffel.h>
#include <nana.h>

#include "double_adds.h"

#include <limits>
#include <cmath>
#include <algorithm>

static double myown_sqrt(double x)
{
        double xold, xi, result;
        REQUIRE(x >= 0.0);
        if (x == 0.0) {
           result = 0.0;
```

```
        } else {
                xi = x / 2.0;
                do{
                        xold = xi;
                        xi = 0.5 * (xi + x/xi);
                } while (xi != xold);
                result = xi;
        };
        ENSURE(approximatelyEqualTo(result*result, x, 1.0));
        return result;
}

#endif /* DOUBLE_MATH_H_ */
```

## 1.17 Sprachliche Konstrukte einer beispielhaften objektorientierten Programmiersprache: Eiffel

*Objektorientierte* Programmiersprachen ermöglichen die **Datenkapselung** und eine **evolutionäre** Programmerstellung:

> Nutze vorhandene Objektklassen (Typen) oder erzeuge neue Objektklassen, wobei **bei Teilstrukturgleichheit** möglichst viel durch **Vererbung** existierender Klassen realisiert wird.

### 1.17.1 Vererbung und Erweiterung/Namensmodifikation: rename

Namensänderung

Neues *feature*

```
class Multiindex inherit
  ARRAY[CARDINAL] rename
                    count as Dimension,
                    clear_all as Null
                  end;
feature
    abs: CARDINAL is
      require not empty
      do ...
      ensure
            - - abs = For all i:lower..upper:
                      SUM item(i)
      end - - abs
end - - class Multiindex
```

### 1.17.2 Vererbung und Abänderung: redefine, undefine

Vergleiche: http://docs.eiffel.com/book/platform-specifics/inheritance#Undefine

alternative Implementierung ⟶

alternative Implementierung ⟶

auch die Vorbedingung wird geerbt ⟶

zusätzliche Nachbedingung ⟶

alte Invariante wird geerbt ⟶

```
class Multiindex inherit
   ARRAY[CARDINAL]rename
                        count as Dimension,
                        clear_all as Null
                    redefine
                        abs,
                        put,
                        make
                    undefine
                        has
                    end;
feature
   abs: CARDINAL;
   put(v:like item; i:INTEGER)
       - - replace i-th entry, if in index interval, by v
       ⋮
   ensure then
              abs = old abs - old item(i) + v
   end - - put
      ⋮
invariant
       - - abs = For all i:lower..upper:Sum item(i)
end - - class Multiindex
```

Die `undefine has`-Klausel macht das Feature `has` ab hier wieder rein virtuell (=0).

In C++ ist ein „undefine“-Analogon wie folgt realisierbar:

```
class Auto {
public:
    virtual void setZulGesamtgewicht(double zulGesamtgewicht)
    {
        this->zulGesamtgewicht = zulGesamtgewicht;
    }
    ...
}
```

```
class LKW : public Auto{
public:
    virtual void setZulGesamtgewicht(double zulGesamtgewicht)
    {
        if (zulGesamtgewicht >= 3500.0 && zulGesamtgewicht < 44000.0)
          this->zulGesamtgewicht = zulGesamtgewicht;
    }
    ...
}

class SchwererLKW  : public LKW{
public:
     virtual void setZulGesamtgewicht(double zulGesamtgewicht)=0;
     ...
}
```

Wenn aus Effektivitätsgründen **redundante Daten** oder **Methoden** angelegt werden, so sollten diese Redundanzen explizit spezifiziert werden!

Es gelten folgende Regeln bei der Vererbung (von is-a-Methoden):

a) Vorbedingungen können in einer Kindklasse abgeschwächt werden.

b) Nachbedingungen in einer Kindklasse müssen stärker sein als diejenigen der Elterklasse.

c) Invarianten in der Kindklasse müssen ebenfalls stärker als in der Elterklasse sein.

Dann ist ein echtes *Subcontracting* realisiert.

Bemerkung: Es reicht die Kindnachbedingung im Falle des Eintreffens der Eltervorbedingung stärker als die Elternachbedingung zu realisieren. Im Falle „Kindvorbedingung **and not** Eltervorbedingung“ darf die Kindnachbedingung frei gewählt werden.

Ein Beispiel mit Contracts in `nana`:

```
class name_list{
...
public:

  /////////// basic queries:

  unsigned int get_count() const;   // number of items in stack

  bool has(const string& a_name) const;
...
```

```
////////////// (pure) modificators:

  virtual void put(const string& a_name); // Push a_name into list
}

void name_list::put(const string& a_name)    // Push a_name into list
DO
   REQUIRE(/* name not in list */    !has(a_name));
   ID(set<string> contents_old(begin(),end()));
   ID(int count_old = get_count());
   ID(bool not_in_list = !has(a_name));
...
   ENSURE(has(a_name));
   ENSURE( (!not_in_list) || (get_count() == count_old + 1));
   ID(set<string> contents(begin(),end()));
   ENSURE( (!not_in_list) || (contents == contents_old + a_name));
END
...
///////////////// child class relaxed_name_list ///////////
////////////////   (more user friendly)   ////////////////

class relaxed_name_list : public name_list{
    ///////// (pure) modificators: (redefined)

    virtual void put(const string& a_name);   // Push a_name into list
...
}
void relaxed_name_list::put(const string& a_name)    // Push a_name into list
DO
   REQUIRE(/* nothing */  true);       // usable without conditions
   ID(set<string> contents_old(begin(),end()));
   ID(int count_old = get_count());
   ID(bool not_in_list = !has(a_name));
...
   ENSURE(has(a_name));
   ENSURE((!not_in_list) || (get_count() == count_old + 1));  // &&
   ENSURE(  not_in_list  || (get_count() == count_old));
   ID(set<string> contents(begin(),end()));
   ENSURE(  not_in_list  || (contents == contents_old));
   ENSURE((!not_in_list) || (contents == contents_old + a_name));
END
```

### 1.17.3 Generizität

```
class STACK[T]
feature
    ⋮
end - - class STACK[T]
```

### 1.17.4 Eingeschränkte Generizität

```
class VECTOR[T –> ADDABLE]
feature
    ⋮
end - - class VECTOR
```

### 1.17.5 Polymorphie und „late binding“

```
class Rectangle inherit
      POLYGON redefine perimeter
                  end
  feature {NONE}
      side1: REAL;
      side2: REAL;
  feature {ANY}
      perimeter: REAL is
      do
         Result := 2 ∗ (side1 + side2)
      end - - perimeter
      ⋮
end - - class Rectangle
```

Wegen des Zusammenhangs Rectangle $\subset$ POLYGON und MethodenVon(Rectangle) $\supset$ MethodenVon(POLYGON) gilt in der Anwendung:

*perimeter* für die Menge aller Erben von POLYGON verfügbar unter gleichem Namen.

```
⋮
p : POLYGON;
r : Rectangle;
⋮
!!p; !!r;
⋮
print (p.perimeter);          - - perimeter aus
⋮                             - - POLYGON
p := r
print (p.perimeter);          - - perimeter aus
⋮                             - - Rectangle
```

### 1.17.6 Aufgeschobene Feature-Implementierungen

```
deferred class Stack[T]
feature
        nb_elements : INTEGER is
          defered
        end - - nb_elements
        empty : BOOLEAN is
        do
           Result := (nb_elements = 0)
        ensure Result = (nb_elements = 0)
        end - - empty
      ⋮
end - - class STACK[T]
```

... dienen der partiellen Implementierung einer Gruppe möglicher Implementierungen (Schablone). Sie stehen somit in Konkurrenz und ergänzen generische Klassen.

**Mehrfachvererbung** ist ein wichtiges Werkzeug zur Wiederverwendung von Programmcode. Java Interfaces mit rein abstrakte Typen (nur Deklarationen, keine Definitionen) zwingen leider zur Codeduplizierung. Deshalb wird in in neueren Programmiersprachen von Mehrfachvererbung von Methoden-Implementierungen, wenn auch nicht unbedingt von Attributen Gebrauch gemacht: „Traits are like interface but can have method bodies (i.e., not just method declarations)“

- JavaFX: http://java.sun.com/javafx/1/tutorials/core/classes/index.html#mixins
- Scala: http://blogs.sun.com/sundararajan/entry/scala_for_java_programmers

- Fortress: http://www.cs.hmc.edu/~stone/FOOL/FOOLWOOD07/Allen-slides.pdf (Seite 4)
- ...

Insgesamt:

– "Objektorientiertes" Programmieren (als Alternative zum funktionalen Top-Down-Entwurf und zum datengesteuerten Entwurf nach Jackson) ist die Softwarekonstruktion mit Hilfe der **Adaption** von Sammlungen abstrakter Datentyp-Implementierungen.

– Unterklassen können sich von ihren Basisklassen unterscheiden durch:

1) mehr Operationen
2) mehr Daten (Attribute)
3) eingeschränkte Wertebereiche der Daten
4) alternative Implementierungen
5) schwächere Methodenvoraussetzungen

## Resümee

1. Objektorientiertes Programmieren setzt eine gute Kenntnis der vorhandenen Klassenhierarchien voraus! Diese sind heute jedoch häufig nicht ausreichend dokumentiert (fehlende Spezifikation, fehlende Fixierung der Design-Ideen, ...). Häufig steht nur ein Browser zur Betrachtung der Quellen der Klassen zur Verfügung, und der Programmierer muß sich selbst den Durchblick durch die Konzeption der Klassenbibliotheken erkämpfen.

2. Einige **objektorientierte** Sprachen bieten gar keine mitgelieferten Klassenbibliotheken an. Andere haben sprachspezifisch bzw. sogar herstellerspezifisch eigene — zwar häufig an Smalltalk angelehnte, aber dennoch in wichtigen Details abweichende — Klassenhierarchien. Für viele Gebiete in der Informatik/Mathematik/Anwendungswissenschaft fehlen geeignete Klassenbibliotheken gänzlich.

3. Geordnete **evolutionäre** objektorientierte Entwicklung im Team erfordert ein richtiges Management (open-close-Phasen, Versions-Management, ...)

4. Objektorientierte Programmiersprachen sollten syntaktische Sprachmittel für **Zusicherungen** besitzen (mindestens Aussagenlogik, besser **Prädikatenlogik**). Diese sollten **in** den geforderten **Klassenhierarchien** (zumindest in Kommentarform) **intensiv genutzt** werden. Eine etwa VDM ähnliche Syntax wäre gewiß interessant.

Wir müssen anspruchsvoller werden im Hinblick auf die Verläßlichkeit und die Qualität unserer Software. Die Benutzer müssen kritischer werden und weniger bereit, Softwareerzeugnisse geringer Qualität zu akzieptieren.

(Zitat: Robert L. Baber: Softwarereflexionen, Springer-Verlag)

## Forschungsministerium fördert Standard für IT-Sicherheit

Trotz des flächendeckenden Einsatzes von Computersystemen in sicherheitsrelevanten Bereichen fehlt bislang eine standardisierte Methode, die das fehlerfreie Funktionieren solcher Systeme garantiert. Das Bundesministerium für Bildung und Forschung (BMBF) will nun Arbeiten fördern, bei denen mit Methoden der Verifikation der so genannte geschlossene integrierte Korrektheitsbeweis erbracht werden kann. Damit sollen sich Fehler bereits im Entwurf von autonomen oder integrierten Computersystemen erkennen und korrigieren lassen – eine sorgfältige Spezifikation vorausgesetzt. Alle möglichen Fehlersituationen könnten aber nur dann abgefangen werden, wenn bereits in der Planung die entsprechenden Einsatzszenarien definiert wurden, betonte Projektleiter Prof. Dr. Wolfgang Paul gegenüber heise Security.

Für die erste zweijährige Forschungsphase werde das BMBF 7,2 Millionen Euro zur Verfügung stellen, teilte das Ministerium am heutigen Mittwoch in Berlin mit. An dem Projekt beteiligen sich neben der Universität Saarland unter anderen auch die TUs Darmstadt, Karlsruhe, München sowie Infineon, T-Systems und BMW.

Die Entwicklung eines integrierten Korrektheitsbeweises gilt zurzeit als eine der größten Herausforderungen der Informatik. Er soll die Funktionen bei der Entwicklung von Hard- und Systemsoftware bis zur Netzwerk- und Anwendungsebene laufend überprüfen. Zunächst sollen die mathematischen Grundlagen entwickelt, vollständig formalisiert und für Informatikanwendungen in den Bereichen Embedded Systems, Kommunikation und Anwendungssoftware erschlossen werden. Darauf aufbauend sollen die Projektpartner Demonstratoren entwickeln und mit ihnen Computersysteme für Chipkarten, Telekommunikation und Automobilelektronik von der Hardware bis zur Anwendungssoftware überprüfen. Im Rahmen des Projektes werden auch Softwaretools entwickelt, die den Verifikationsprozess unterstützen. (dab/c't)

**Link:** http://www.heise.de/newsticker/data/dab-01.10.03-002/

Siehe auch (Thema Produkthaftung):
http://www.heise.de/newsticker/result.xhtml?url=/newsticker/meldung/86839

## Jedes zehnte Unternehmen hat IT-Sicherheitsprobleme

06.12.2010 12:24

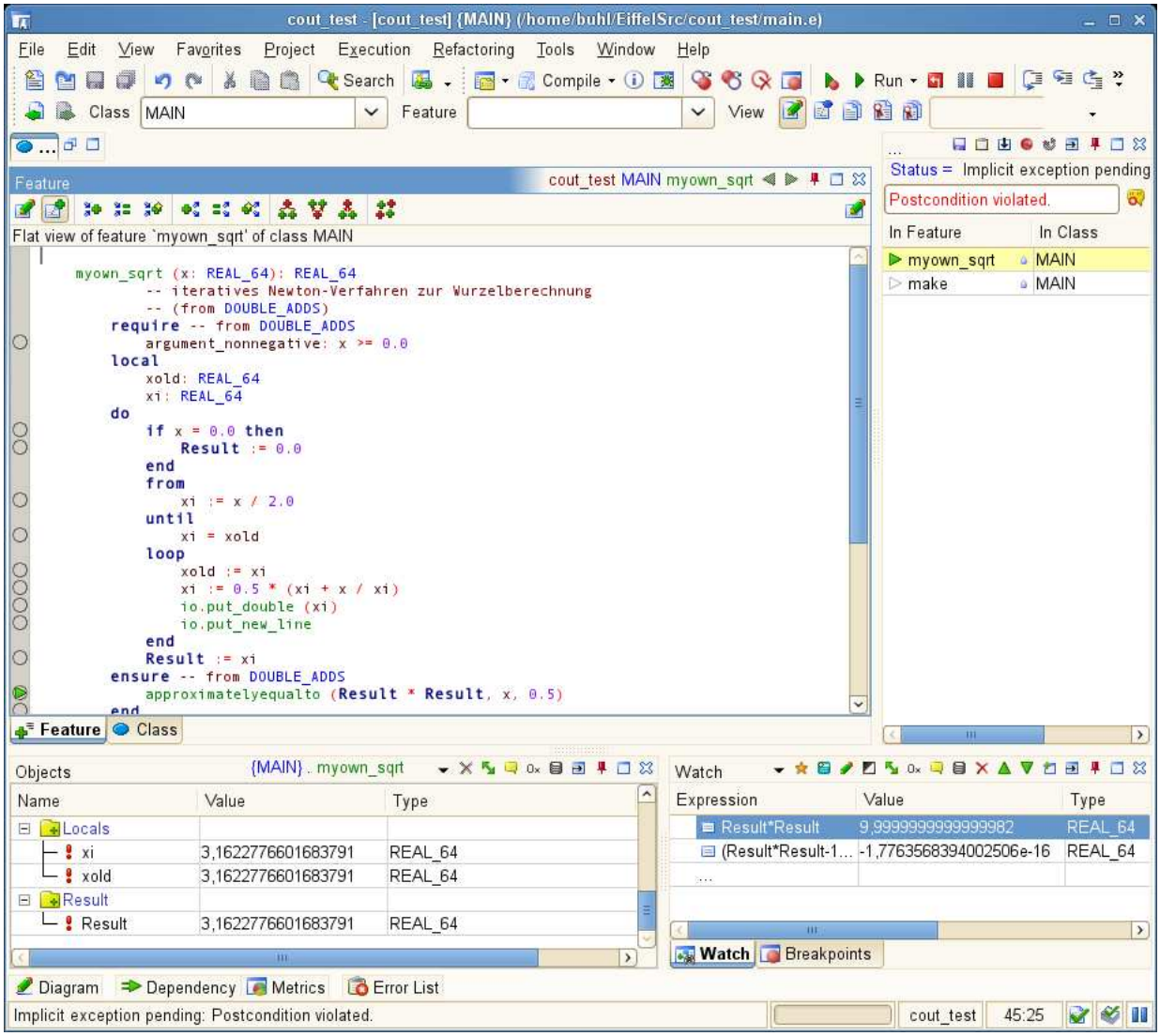
Mehr als jedes zehnte Unternehmen in Deutschland hat Probleme mit der Sicherheit seiner Informationssysteme. Das geht aus einer Mitteilung[1] des Statistischen Bundesamtes anlässlich des 5. Nationalen IT-Gipfels[2] am morgigen Dienstag in Dresden hervor. Danach gaben 74 Prozent der betroffenen Unternehmen an, dass bei ihnen 2009 aufgrund von Hardware- oder Softwarefehlern Daten zerstört oder verändert wurden und bestimmte Dienste ihrer Informations- und Kommunikationssysteme nicht verfügbar waren.

28 Prozent der Unternehmen hatten Probleme, weil Schadsoftware oder nicht autorisierte Zugriffe zur Veränderung beziehungsweise Zerstörung von Daten führten. Phishing-Angriffe störten bei 3 Prozent der betroffenen Firmen die Systeme. Stärker fiel der unbedachte Umgang der Belegschaft mit vertraulichen Daten ins Gewicht: In 11 Prozent der Firmen legten Mitarbeiter vertrauliche Daten offen. Deshalb führen laut Erhebung inzwischen 25 Prozent der Unternehmen mit zehn und mehr Beschäftigten obligatorische Schulungen der Mitarbeiter zum Thema IT-Sicherheit durch.

Die Zahlen hat das Statistische Bundesamt im Rahmen seiner jährlichen Erhebung zur Nutzung von Informations- und Kommunikationstechnik in Unternehmen ermittelt.

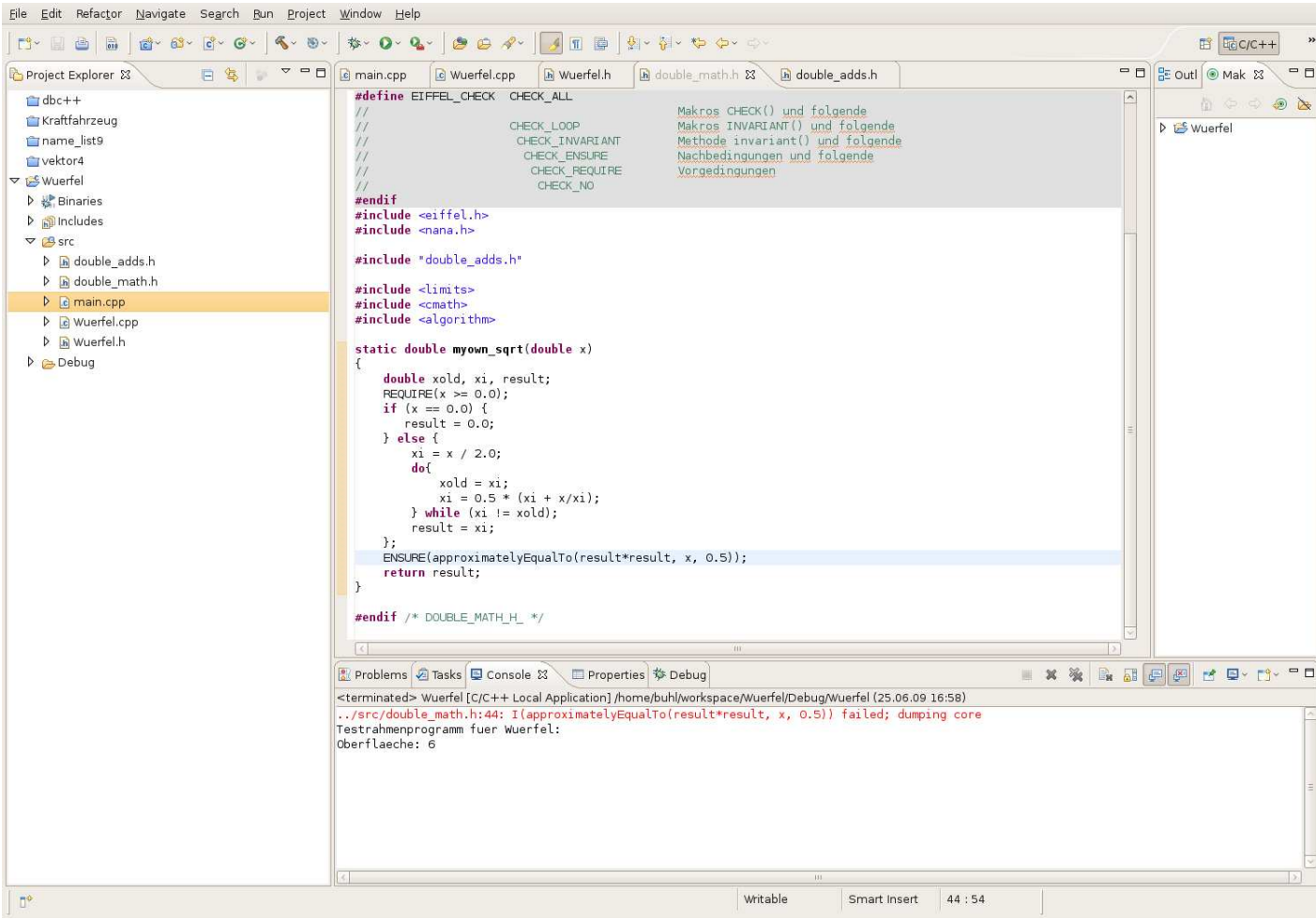
## 1.18 Vertragsverletzungen zur Laufzeit

**Vertragsverletzungen eines Eiffel-Programms in EiffelStudio:**



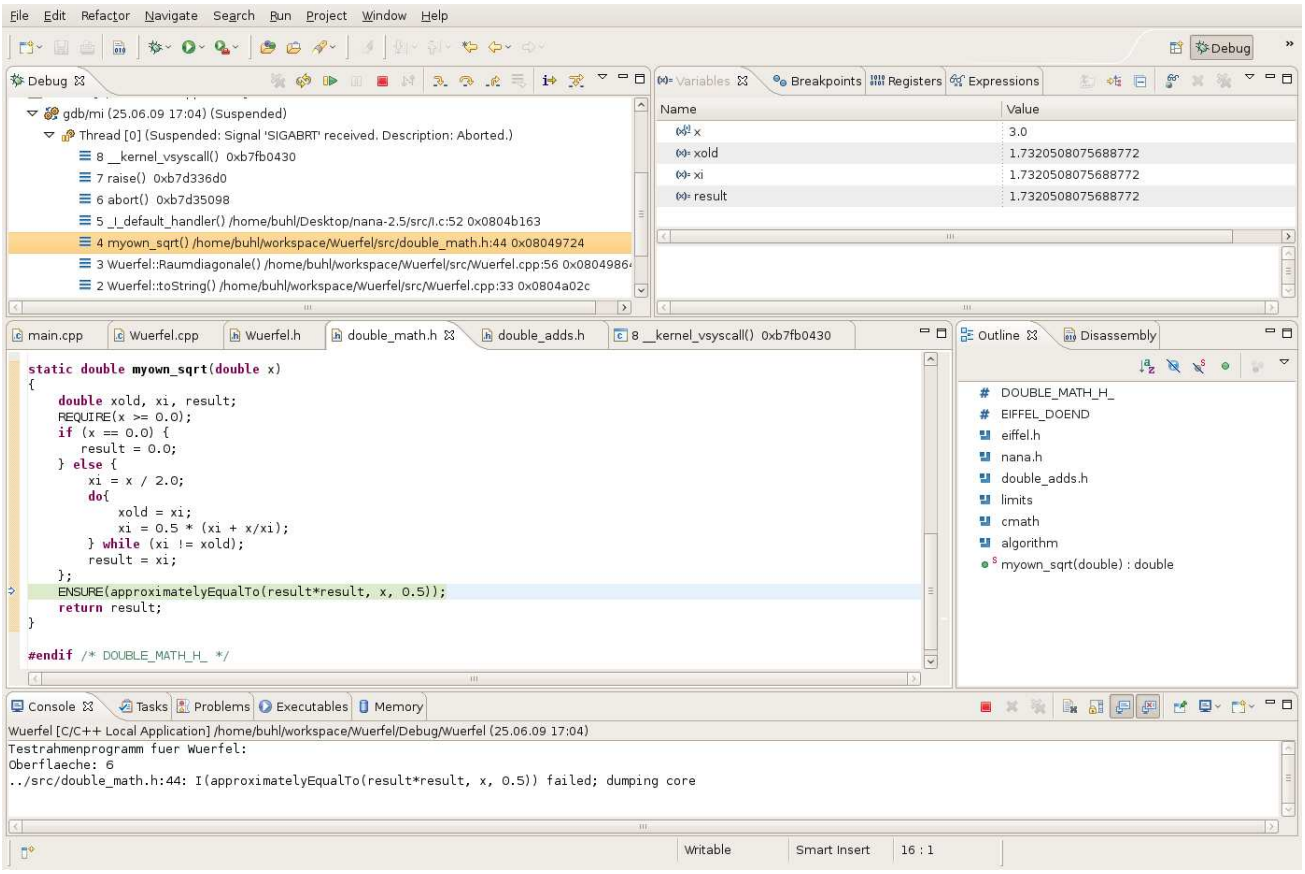

Anzeige der Art der Exception: Postcondition violated
Stackframe zum Zeitpunkt der Vertragsverletzung
Attributwerte zum Zeitpunkt der Vertragsverletzung
Berechnung von arithmetischen Ausdrücken unter Benutzung der Attributwerte möglich im `Watch`-Fenster

**Vertragsverletzungen von Nana/C++-Programmen in Eclipse:**



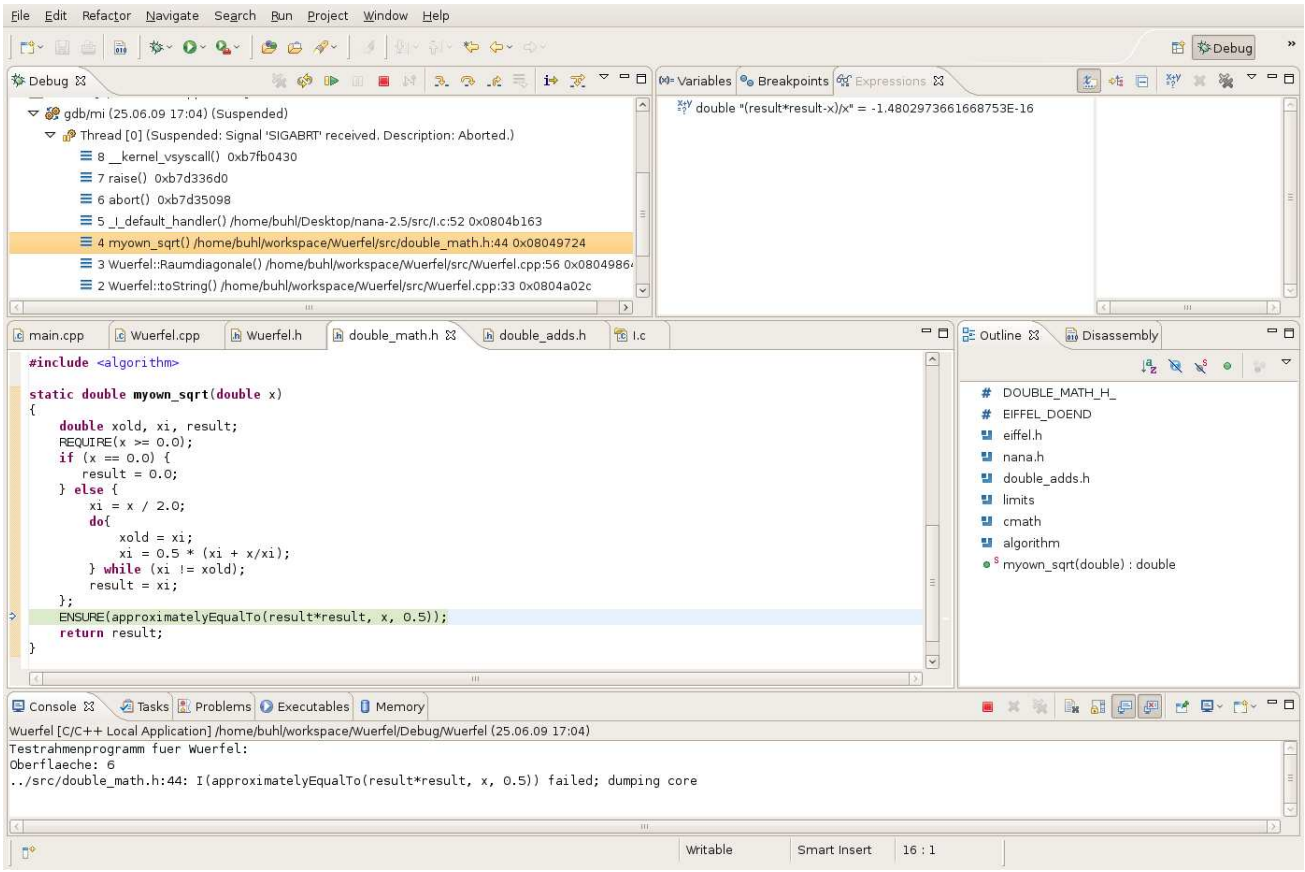

einfacher Programm-Abbruch (abort()) mit Informationen zu:
verletzte Zusicherung und Hinweis zum erfolgslosen `core`-Dump,

**Programm-Abbruch im Eclipse-Debugger** mit den Möglichkeiten der Programmuntersuchung: Abbruchstelle, Attributinhalte der aktiven Instanzen, ...

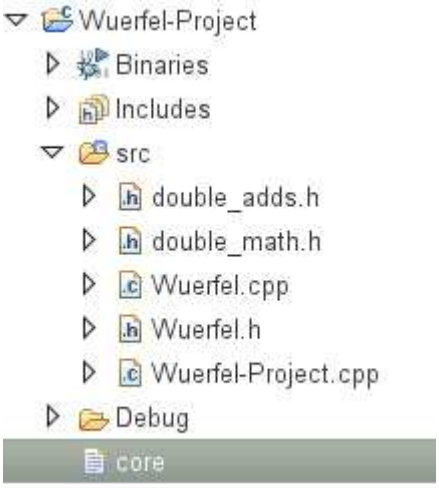



(klicke im Debug-Fenster die Routine unmittelbar unter `_I_default_handler()` an)

Berechnung von Ausdrücken unter Benutzung dieser Attributwerte (nach Anwahl von `Watch` im `Variables`-Fenster):

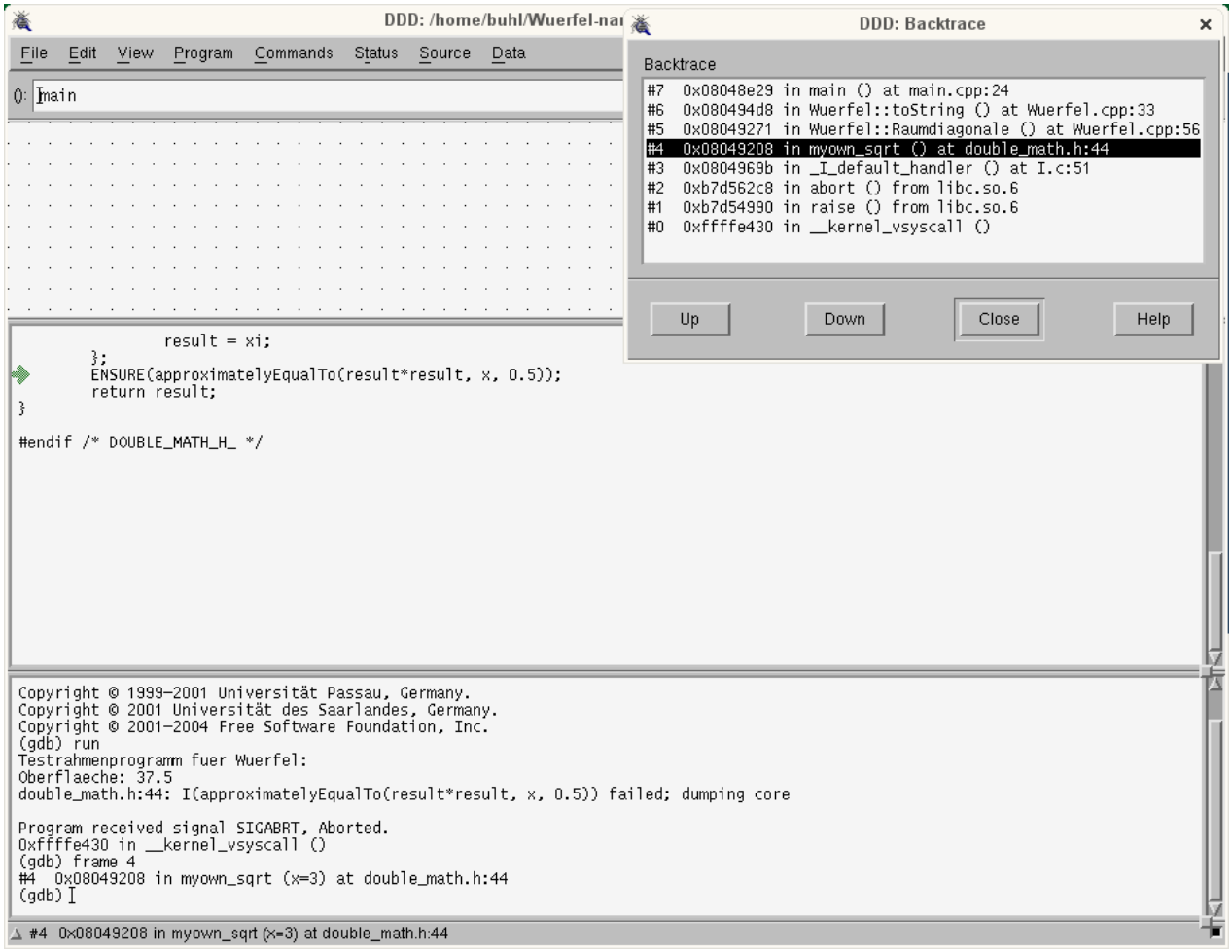



Entstehende `core`-Dateien werden bei Terminierung des debugten Executables in das Projektverzeichnis geschrieben:




Im obigen Beispiel ist die Forderung, eine Genauigkeit von einer halben GPU-Maschinengenauigkeit zur erreichnen natürlich unsinnig!

**Vertragsverletzungen von Nana/C++-Programmen bei Debug-Lauf in ddd:**

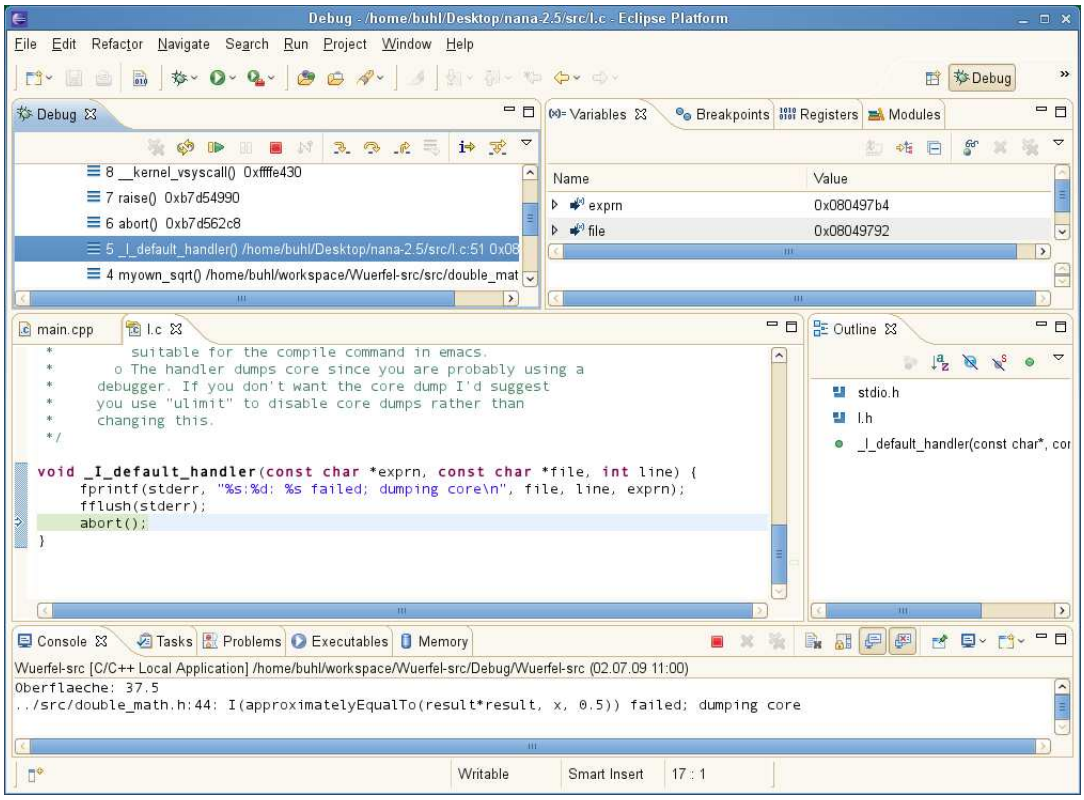


### Vertragsverletzungen bei Ausführung von Nana/C++-Programme in der Kommandozeile: core-Datei

Will man nicht den Debug-Modus von `eclipse` beziehungsweise die direkte Ausführung eines mit `nana`-Contracts ausgestatteten C++-Programms in einem Debugger wie dem `ddd` benutzen, sondern solche Programme direkt von der Kommandozeile aus starten, so wird man nach einer Vertragsverletzung und der Fehlermeldung vergebens nach der Datei `core`

```
double_math.h:44: I(approximatelyEqualTo(result*result, x, 0.5))
failed; dumping core
Abgebrochen
```

suchen. Auf modernen Linux-Systemen ist aus Sicherheitsgründen nämlich die Hauptspeicher-Dump-Funktionalität der stdlib-Funktion `abort()`, die von `nana` bei Vertragsverletzungen zum Programmabbruch genutzt wird,




abgeschaltet (damit `core`-Dateien nicht unnötig die Platte füllen).

Nach Eintrag einer Zeile

```
*    soft    core    unlimited
```

in der Datei `/etc/security/limits.conf` durch den Administrator eines Linux-Systems kann jedoch jeder Nutzer vermittels des `bash`-Kommandos `ulimit -c unlimited` eine Erzeugung von solchen Dumps wieder aktivieren:

```
ulimit -a
core file size          (blocks, -c) 0
...

ulimit -c unlimited
ulimit -a
core file size          (blocks, -c) unlimited
...
```

Jetzt werden `core`-Dateien erzeugt:

```
./main
double_math.h:44: I(approximatelyEqualTo(result*result, x, 0.5)) failed;
dumping core
Abgebrochen (core dumped)

ls -al core
-rw------- 1 buhl users 237568  2. Jul 11:21 core
```

und können im Debugger analysiert werden:

```
gdb ./main core
GNU gdb (GDB; openSUSE 11.1) 6.8.50.20081120-cvs
...
Core was generated by `./main'.
Program terminated with signal 6, Aborted.
#0  0xffffe430 in __kernel_vsyscall ()

(gdb) bt
#0  0xffffe430 in __kernel_vsyscall ()
#1  0xb7d50990 in raise () from /lib/libc.so.6
#2  0xb7d522c8 in abort () from /lib/libc.so.6
#3  0x0804969b in _I_default_handler (
    exprn=0x8049868 "I(approximatelyEqualTo(result*result, x, 0.5))",
    file=0x804984b "double_math.h", line=44) at I.c:51
#4  0x08049208 in myown_sqrt (x=3) at double_math.h:44
#5  0x08049271 in Wuerfel::Raumdiagonale (this=0xbfefa918) at Wuerfel.cpp:56
#6  0x080494d8 in Wuerfel::toString (this=0xbfefa918) at Wuerfel.cpp:33
#7  0x08048e29 in main (argc=<value optimized out>, argv=<value optimized out>)
    at main.cpp:24
(gdb) ...
```

Vergleiche auch http://www.akadia.com/services/ora_enable_core.html.

**Automatisch erzeugte ausführlichere Meldung einer Vertragsverletzung bei Programmstart in der Kommandozeile: backtrace**

Eine Modifikation des Programms `main()` zum Abfangen des Signals `SIGABRT` kann dessen Verhalten bei Vertragsverletzung beeinflussen. Prinzipielles Vorgehen dazu:

```
...
#include <csignal>
...
void ABRT_signalhandler(int signum)
{
  std::cerr << "modifizierter SIGABRT-Handler" << std::endl;
  signal(SIGABRT,SIG_DFL);
  abort();
}

int main(int argc, char* argv[]){
        signal(SIGABRT, ABRT_signalhandler);
...
```

Benutzt wird nun `execinfo.h` und die Funktion `backtrace_symbols` (vgl. http://www.gnu.org/software/libc/manual/html_node/Backtraces.html), um bei Vertragsverletzung automatisch den `backtrace` auf `cout` zu schreiben, ohne einen Debugger bemühen zu müssen:

```
/*  main.cpp               */
 *  Created on: 22.04.2009 */

#include <iostream>
#include <cstdlib>
#include <string>
#include <cassert>
#include <csignal>
#include <execinfo.h>

#include "Wuerfel.h"
#include "double_math.h"

class ExceptionTracer
 {
 public:
     ExceptionTracer()
     {
         void * array[25];
```

```
        int nSize = backtrace(array, 25);
        char ** symbols = backtrace_symbols(array, nSize);

        for (int i = 0; i < nSize; i++)
        {
            std::cout << symbols[i] << std::endl;
        }

        free(symbols);
    }
};

void ABRT_signalhandler(int signum)
{
  std::cout << "SIGABRT: backtrace is " << std::endl;
  ExceptionTracer::ExceptionTracer();
  std::cout << "resuming abort " << std::endl << std::endl;
  signal(SIGABRT,SIG_DFL);
  abort();
}

int main(int argc, char* argv[]){
        signal(SIGABRT, ABRT_signalhandler);

        std::cout << "Testrahmenprogramm fuer Wuerfel:" << std::endl;
        Wuerfel w(2.5);
        // assert( w.toString() == "1;6;1;1.73205" );

        std::cout << "Oberflaeche: " << w.Oberflaeche() << std::endl;
        std::cout << "w.toString()= " << w.toString() << std::endl;
        std::cout << "sqrt(3.0) = " << myown_sqrt(3.0) << std::endl;

        // ...

        return 0;
}
```

Vergleiche zu `ExceptionTracer`: http://www.ibm.com/developerworks/linux/library/l-cppexcep.html

Jetzt wird nach der `nana`-Fehlermeldung automatisch der `backtrace` angezeigt:

```
./main
...
double_math.h:44: I(approximatelyEqualTo(result*result, x, 0.5)) failed;
dumping core
SIGABRT: backtrace is
./main [0x401447]
./main [0x401132]
/lib64/libc.so.6 [0x7f3d880256e0]
/lib64/libc.so.6(gsignal+0x35) [0x7f3d88025645]
/lib64/libc.so.6(abort+0x183) [0x7f3d88026c33]
./main [0x401c80]
./main [0x401776]
./main [0x4017f6]
./main [0x401abb]
./main [0x4012fb]
/lib64/libc.so.6(__libc_start_main+0xe6) [0x7f3d88011586]
./main [0x400f19]
resuming abort
Abgebrochen
```

Leider endhält das `main()`-Binary nur genügend Symbolinformationen, wenn Sie `g++` mit der Option `-rdynamic` übersetzt haben:

```
./main
...
double_math.h:44: I(approximatelyEqualTo(result*result, x, 0.5)) failed;
dumping core
SIGABRT: backtrace is
./main(_ZN15ExceptionTracerC1Ev+0x23) [0x401af7]
./main(_Z18ABRT_signalhandleri+0x30) [0x4017e2]
/lib64/libc.so.6 [0x7f4ca97926e0]
/lib64/libc.so.6(gsignal+0x35) [0x7f4ca9792645]
/lib64/libc.so.6(abort+0x183) [0x7f4ca9793c33]
./main(__libc_csu_fini+0) [0x402330]
./main [0x401e26]
./main(_ZN7Wuerfel13RaumdiagonaleEv+0x70) [0x401ea6]
./main(_ZN7Wuerfel8toStringEv+0x39) [0x40216b]
./main(main+0x92) [0x4019ab]
/lib64/libc.so.6(__libc_start_main+0xe6) [0x7f4ca977e586]
./main [0x4015c9]
resuming abort
Abgebrochen
```

Filtern Sie schließlich die Ausgaben von `main()` durch den Demangler `c++filt`, so wird eine akzeptable Fehleranzeige erreicht:

```
./main | c++filt
...
double_math.h:44: I(approximatelyEqualTo(result*result, x, 0.5)) failed;
dumping core
...
./main(ExceptionTracer::ExceptionTracer()+0x23) [0x401af7]
./main(ABRT_signalhandler(int)+0x30) [0x4017e2]
/lib64/libc.so.6 [0x7ff33cbcf6e0]
/lib64/libc.so.6(gsignal+0x35) [0x7ff33cbcf645]
/lib64/libc.so.6(abort+0x183) [0x7ff33cbd0c33]
./main(__libc_csu_fini+0) [0x402330]
./main [0x401e26]
./main(Wuerfel::Raumdiagonale()+0x70) [0x401ea6]
./main(Wuerfel::toString()+0x39) [0x40216b]
./main(main+0x92) [0x4019ab]
/lib64/libc.so.6(__libc_start_main+0xe6) [0x7ff33cbbb586]
./main [0x4015c9]
```

(http://en.wikipedia.org/wiki/Name_mangling#Standardised_name_mangling_in_C.2B.2B)

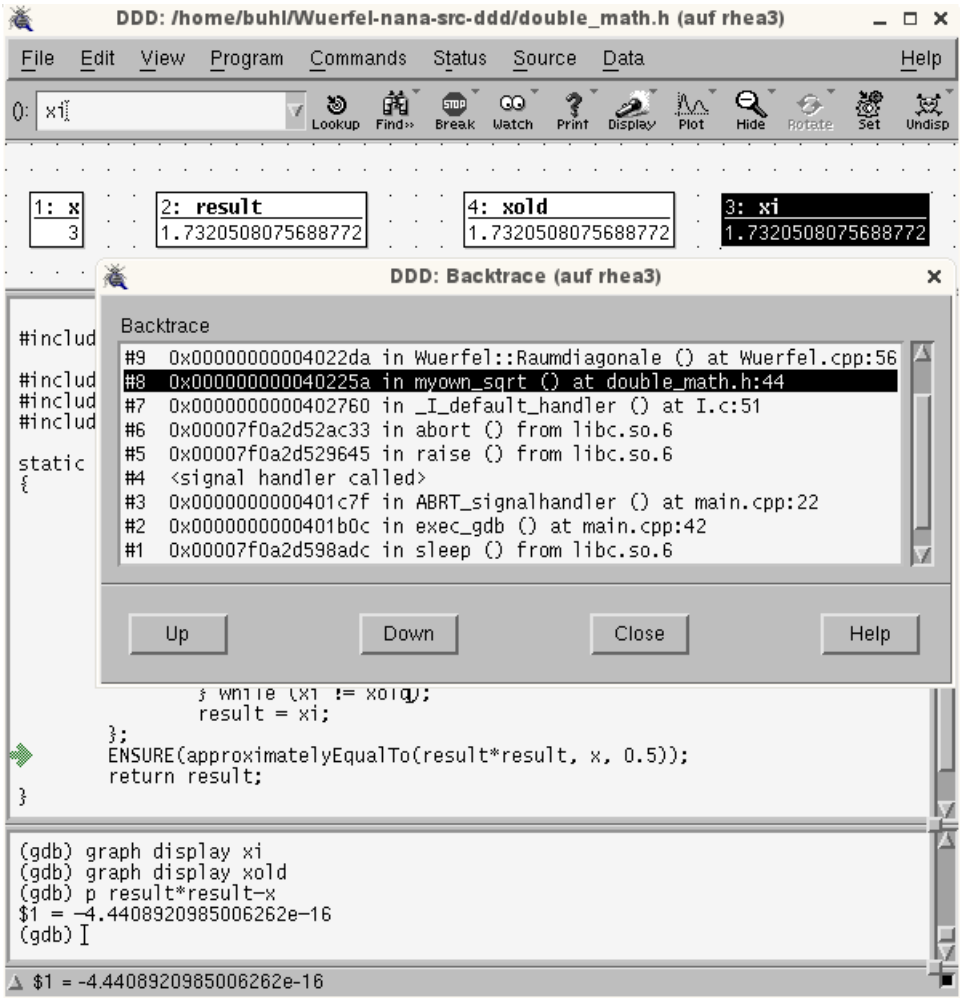
**Automatischer Start von `ddd` bei einer Vertragsverletzung bei Programmstart in der Kommandozeile**

Um bei einer Vertragsverletzung auch ohne `core`-Dump automatisch direkt in den Debugger `ddd` zu wechseln, um dort die Abbruchstelle zu untersuchen (Variableninhalte, Ausdrücke in Variableninhalten zum Abbruchzeitpunkt, ...) modifizieret man den ABRT-Signalhandler folgendermaßen:

```
#include <csignal>
...
void exec_ddd();
void ABRT_signalhandler(int signum)
{
  std::cerr << "SIGABRT: starting ddd... " << std::endl;
  exec_ddd();
  std::cerr << "resuming abort " << std::endl << std::endl;
  signal(SIGABRT,SIG_DFL);
  abort();
}

static int ddd_process_pid = 0;
void exec_ddd()
{
    // Create child for running ddd
    int pid = fork();

    if (pid < 0) /* error */
    {
        abort();
    }
    else if (pid) /* parent */
    {
        // C++ nana-application
        ddd_process_pid = pid; // save debugger pid
        sleep(10);             // give ddd time to attach
                               // Continue C++ nana-application
    }
    else /* child */
    {
        // ddd process
        std::stringstream args;
        args << "--pid=" << getppid();
        execl("/usr/bin/ddd",
              "ddd", "--debugger", "gdb",
```

```
                args.str().c_str(), (char *) 0);

        std::cerr << "\nFailed to exec ddd\n" << std::endl;
    }
}


int main(int argc, char* argv[]){
        signal(SIGABRT, ABRT_signalhandler);

        std::cout << "Testrahmenprogramm fuer Wuerfel:" << std::endl;
        ...
}
```

Vergleiche: http://www.codeproject.com/KB/debug/java_cpp_debugging.aspx?display=Print




Bemerkung: Diese Methode benötigt nicht das Schreiben einer `core`-Datei auf die Festplatte, funktioniert also auch ohne Änderung der Datei `/etc/security/limits.conf`.

## 1.19 Nichtänderungs-Verträge für Attribute: Framebedingungen

```
/*
 * Wuerfel.cpp
 */


#include <sstream>
#include <limits>
#include <cmath>

#include "Wuerfel.h"
#include "double_adds.h"
#include "double_math.h"

Wuerfel::Wuerfel(double mySeite): Seite(mySeite)
{
        REQUIRE(mySeite >= 0);
        ENSURE(invariant());
}

Wuerfel::~Wuerfel() {
        // TODO Auto-generated destructor stub
}

bool Wuerfel::invariant()
{
        return Seite >= 0.0;
}

std::string Wuerfel::toString()
{
        std::ostringstream help;
        help << Seite << ";" << Oberflaeche() << ";" << Volumen() << ";"
             << Raumdiagonale();
        return help.str();
}



double Wuerfel::Oberflaeche()
DO
```

```
        ID(double Seite_old = Seite);
        double result = 6.0 * pow(Seite, 2);
        ENSURE(approximatelyEqualTo(result, 6.0 * Seite * Seite, 1.0));
        ENSURE(Seite == Seite_old);
        ENSURE(invariant());
        return result;
}

double Wuerfel::Volumen()
DO
        ID(double Seite_old = Seite);
        double result = pow(Seite,3);
        ENSURE(approximatelyEqualTo(result, Seite*Seite*Seite, 1.0));
        ENSURE(Seite == Seite_old);
        ENSURE(invariant());
        return result;
}

double Wuerfel::Raumdiagonale()
DO
        ID(double Seite_old = Seite);
        double result = Seite * sqrt(3.0);
        ENSURE(approximatelyEqualTo(result, Seite * myown_sqrt(3.0), 1.0));
        ENSURE(Seite == Seite_old);
        ENSURE(invariant());
        return(result);
}
```

## 1.20 Ultimative Nichtänderungsverträge: const-Methoden

```
/*
 * Wuerfel.h
 *
 *  Created on: 22.04.2009
 *      Author: buhl
 */
#ifndef WUERFEL_H_
#define WUERFEL_H_
#define EIFFEL_DOEND
#ifndef EIFFEL_CHECK
#define EIFFEL_CHECK  CHECK_ALL
//                                          Makros CHECK() und folgende
//                    CHECK_LOOP            Makros INVARIANT() und folgende
//                     CHECK_INVARIANT      Methode invariant() und folgende
//                      CHECK_ENSURE        Nachbedingungen und folgende
//                       CHECK_REQUIRE      Vorgedingungen
//                        CHECK_NO
#endif
#include <eiffel.h>
#include <nana.h>
#include <string>

class Wuerfel{
public:
        Wuerfel(double mySeite = 1.0);
        virtual ~Wuerfel();

        virtual bool invariant() const;

        std::string  toString() const;

        double Oberflaeche() const;

        double Volumen() const;

        double Raumdiagonale() const;
protected:
        double Seite;
};
#endif /* WUERFEL_H_ */
```

```
/*
 * Wuerfel.cpp
 *
 *  Created on: 22.04.2009
 *      Author: buhl
 */

#include <sstream>
#include <limits>
#include <cmath>
#include "Wuerfel.h"
#include "double_adds.h"
#include "double_math.h"

Wuerfel::Wuerfel(double mySeite): Seite(mySeite)
{
        REQUIRE(mySeite >= 0);
        ENSURE(invariant());
}

Wuerfel::~Wuerfel() {
        // TODO Auto-generated destructor stub
}

bool Wuerfel::invariant() const
{
        return Seite >= 0.0;
}

std::string Wuerfel::toString() const
DO
        std::ostringstream help;
        help << Seite << ";" << Oberflaeche() << ";" << Volumen() << ";"
             << Raumdiagonale();
        return help.str();
}

double Wuerfel::Oberflaeche() const
DO
        double result = 6.0 * pow(Seite, 2);
        ENSURE(approximatelyEqualTo(result, 6.0 * Seite * Seite, 1.0));
        return result;
}
```

```
double Wuerfel::Volumen() const
DO
        double result = pow(Seite,3);
        ENSURE(approximatelyEqualTo(result, Seite*Seite*Seite, 1.0));
        return result;
}

double Wuerfel::Raumdiagonale() const
DO
        double result = Seite * sqrt(3.0);
        ENSURE(approximatelyEqualTo(result, Seite * myown_sqrt(3.0), 1.0));
        return(result);
}
```

http://www.linuxjournal.com/article/7629
http://www.cprogramming.com/tutorial/const_correctness.html
http://en.wikipedia.org/wiki/Const_correctness
http://www.parashift.com/c++-faq-lite/const-correctness.html

Bei const-Methoden ist es in der Regel nicht notwendig, die Klassen-Invariante bei Methodenende zu überprüfen, wohl aber bei Klassenbeginn (eine private-Methode könnte vor Methodenaktivierung die Invariante ungültig gemacht haben!).

-

# 2 Programming by Contract

## 2.1 Spezifikation durch Verträge

(**SdV**, *Design by Contract* [1], *Programming by Contract*) ist eine Methode zur Spezifikation der dynamischen Semantik von Softwarekomponenten mit Hilfe von Verträgen aus erweiterten boolschen Ausdrücken. **SdV** basiert auf der Theorie der abstrakten Datentypen und formalen Spezifikationsmethoden. Spezifizierte Komponenten können Module, Klassen oder Komponenten im Sinne von Komponententechnologien (wie Microsofts COM, .NET oder Suns EJB) sein. Verträge ergänzen das Kunden-Lieferanten-Modell:

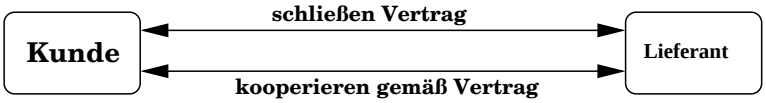



Abbildung 2.1: Kunden-Lieferanten-Modell

Grundlegend für die Vertragsmethode ist das **Prinzip der Trennung von Diensten in Abfragen und Aktionen** (*command-query separation)*:

- **Abfragen** geben Auskunft über den Zustand einer Komponente, verändern ihn aber nicht. Sie liefern als Ergebnis einen Wert. Die Abfragen einer Komponente beschreiben ihren abstrakten Zustand.
- **Aktionen** verändern den Zustand einer Komponente, liefern aber kein Ergebnis. Die Aktionen einer Komponente bewirken ihre Zustandsveränderungen.

Diesem Prinzip folgend sind seiteneffektbehaftete Funktionen als Dienste zu vermeiden[2].

[1] „Design by Contrakt“ ist ein Warenzeichen von Interactive Software Engeneering.
[2] In bestimmten Fällen, z.B. bei Fabrikfunktionen, können Seiteneffekte sinnvoll sein. Solche Funktionen sind nicht als Spezifikatoren verwendbar und sollten entsprechend gekennzeichnet sein.

Ein Grund dafür ist, dass Abfragen als **Spezifikatoren** dienen, d.h. als Elemente von Verträgen. **Verträge** setzen sich aus Bedingungen folgender Art zusammen:

- **Invarianten** einer Komponente sind allgemeine unveränderliche Konsistenzbedingungen an den Zustand einer Komponente, die vor und nach jedem Aufruf eines Dienstes gelten. Formal sind Invarianten boolsche Ausdrücke über den Abfragen der Komponente; inhaltlich können sie z.B. Geschäftsregeln (busisness rules) ausdrücken.

- **Vorbedingungen** (preconditions) eines Dienstes sind Bedingungen, die vor dem Aufruf eines Dienstes erfüllt sein müssen, damit er ausführbar ist. Invarianten sind boolsche Ausdrücke über den Abfragen der Komponente und den Parametern des Dienstes.

- **Nachbedingungen** (postconditions) eines Dienstes sind Bedingungen, die nach dem Aufruf eines Dienstes erfüllt sind; sie beschreiben, welches Ergebnis ein Dienstaufruf liefert oder welchen Effekt er erzielt. Nachbedingungen sind boolsche Ausdrücke über den Abfragen der Komponente und den Parametern des Dienstes, erweitert um ein Gedächniskonstruckt, das die Werte von Ausdrücken vor dem Dienstaufruf liefert.

Verträge legen Pflichten und Nutzen für Kunden und Lieferanten fest. Die Verantwortlichkeiten sind klar verteilt:
Der Lieferant garantiert die Nachbedingung jedes Dienstes, den der Kunde aufruft, falls der Kunde die Vorbedingung erfüllt. Eine verletzte Vorbedingung ist ein Fehler des Kunden, eine verletzte Nachbedingung oder Invariante (bei erfüllter Vorbedingung) ist ein Fehler des Lieferanten.

| | Kunde | Lieferant |
|---|---|---|
| Pflicht | Die Vorbedingung einhalten. | Die Nachbedingung herstellen und die Invariante erfüllen. |
| Nutzen | Ergebnisse/Wirkungen nicht prüfen, da sie durch die Nachbedingungen garantiert sind. Anweisungen ausführen, die die Nachbedingungen herstellen und die Invarianten erhalten | Aufrufe, die die Vorbedingung verletzen, ignorieren. (Die Vorbedingungen nicht prüfen.) |

Tabelle 2.1: Pflichten - Nutzen von Kunden und Lieferanten

Schwache Vorbedingungen erleichtern den Kunden die Arbeit, starke Vorbedingungen dem Lieferanten. Je schwächer die Nachbedingungen sind, umso freier ist der Lieferant und umso ungewisser sind die Kunden über das Ergebnis/den Effekt. Je stärker die Nachbedingungen sind, umso mehr muß der Lieferant leisten.

Siehe auch:
Spezifikation durch Vertrag — eine Basistechnologie für eBusiness

Einige Beispielverträge für eine Klasse `vektor`:

- friend-Funktion `Norm()` (abgeleitete Abfrage)

```
double Norm(const vektor& v)
{
  REQUIRE(v.invariant());
  ...
  ENSURE(approximatelyEqualTo(qsum, S(int k=v.lo(), k<=v.hi(),k++,
                                      v(k)*v(k)), 2.0));
  ENSURE(approximatelyEqualTo(result*result,qsum,2.0));
  ...
}
```

- Methode `normalize()` (Modifikator)

```
void vektor::normalize()
DO
        REQUIRE(Norm(*this)!=0.0);
        ID(vektor value_old(*this));
        ...
        ENSURE(approximatelyEqualVekTo(result*n, value_old, 2.0));
        ENSURE(approximatelyEqualTo(Norm(result), 1.0, 2.0));
        ...
END
```

- i-ter Einheitsvktor (statische Klassenmethode)

```
vektor vektor::ei(int n, int i)
{
  REQUIRE((n>=1) && (1<=i) && (i<=n));
  ...
  ENSURE(result.lo()==1);
  ENSURE(result.hi()==n);
  ENSURE(E1(int k=result.lo(), k<=result.hi(), k++,
            result(k)!=0.0));
  ENSURE(result(i)==1.0);
  ENSURE(result.invariant());
  ...
}
```

- Konstruktor

```
vektor::vektor(const double x[], int n) : low(1), high(n)
{
  REQUIRE((n>=1) && (x!=0));
  REQUIRE("x[] hat mindestens n Komponenten");
  ...
  ENSURE(lo()==1 && hi()==n);
  ENSURE(A(int k=lo(), k<=hi(), k++, (*this)(k)==x[k-lo()]));
END
```

- Modifikator

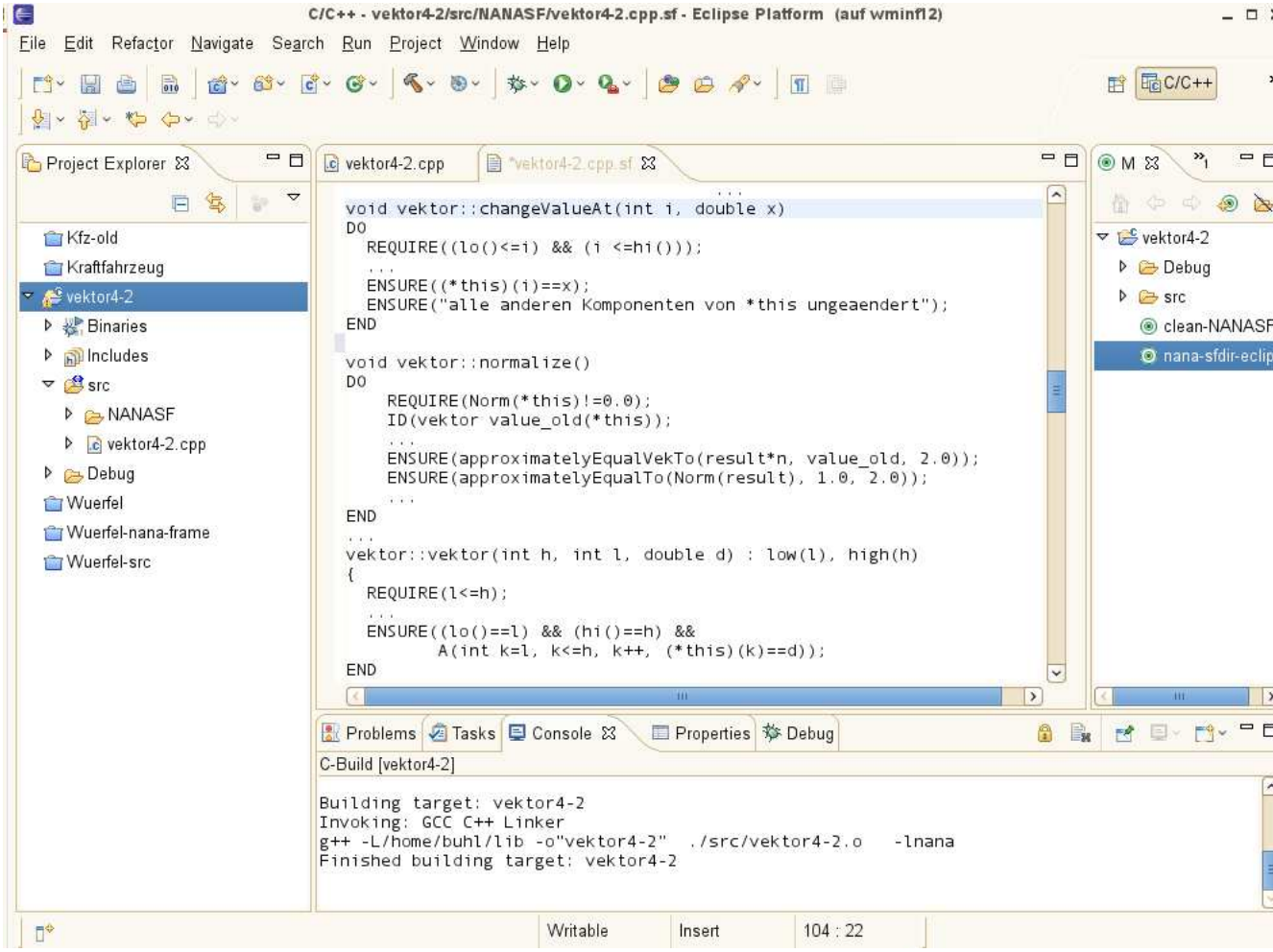
```
void vektor::changeValueAt(int i, double x)
DO
  REQUIRE((lo()<=i) && (i <=hi()));
  ...
  ENSURE((*this)(i)==x);
  ENSURE("alle anderen Komponenten von *this ungeaendert");
END
```

  Überlegen Sie sich einen expliziten Nichtänderungsvertrag für „alle anderen Komponenten“ von `*this` (Frame-Bedingung).

- `operator!=` (abgeleitete Abfrage)

```
bool vektor::operator!=(const vektor& w) const
DO
  REQUIRE(w.invariant());
  ...
  ENSURE(result == ((hi()-lo())!=(w.hi()-w.lo())) ||
     E(int k=lo(), k<=hi(), k++, (*this)(k)!=w(k-lo()+w.lo())));
  ...
}
```

## 2.2 Alle Verträge der Klasse Klasse vektor




### 2.2.1 Klassendeklaration/ Interface

```
#define EIFFEL_DOEND
#define EIFFEL_CHECK CHECK_ALL

#include   <iostream>
#include   <limits>
#include   <cmath>
#include   <algorithm>

#include   <eiffel.h>
#include   "nana.h"
```

```
class vektor
{
private:
  int low;              // v(low..high)
  int high;

  double* vp;           // Startadresse fuer dyn. verwaltetes Exemplar

public:
  virtual bool invariant() const;

  // Grundlegende Abfragen:
  int lo() const;
  int hi() const;

  double  operator()(int i) const;

  // Modifikatoren:
  double& operator()(int i);
  void changeValueAt(int i, double x);
  void normalize();

  // Konstruktoren:
  vektor(int h, int l = 1, double d = 0.0);
  //  v(l..h) = d

  vektor(int h, double d);
  //  v(1..h) = d

  vektor(const double x[], int n);
  // v(1..n) = x[0..n-1]

  // Kopierkonstruktor, Destuktor, Wertzuweisung
  vektor(const vektor& w);
  virtual ~vektor();
  vektor& operator=(const vektor& w);

  // abgeleitete Abfragen:
  bool operator==(const vektor& w) const;
  bool operator!=(const vektor& w) const;

  friend ostream& operator<<(ostream& os, const vektor& v);

  // Operationen:
```

```
  vektor operator+(const vektor& w) const;
  friend vektor operator+(const vektor& w, double a);
  friend vektor operator+(double a, const vektor& w);
  friend vektor operator*(const vektor& w, double a);
  friend vektor operator*(double a, const vektor& w);

  friend double Skalarprodukt(const vektor& v, const vektor& w);
  friend double Norm(const vektor& v);
  friend bool approximatelyEqualVekTo(const vektor& left,
                                      const vektor& right,
                                      double factor);

  static vektor ei(int n, int i);
};
```

Die `private`-Deklarationen sollten eigendlich dem Klassenbenutzer nicht kenntlich sein, also von `nana-sfg` unterdrückt werden!

#### 2.2.1.1 Grundlegende Abfragen

```
int vektor::lo() const
DO
        ...
}
int vektor::hi() const
DO
        ...
}
double  vektor::operator()(int i) const
DO
  REQUIRE((lo()<=i) && (i<=hi()));
  ...
  ENSURE(result == vp[i-low]);    // Nachbedingung nur fuer
                                  // das Implementierungsteam
                                  // damit automatische Ueberpruefung
                                  // stattfinden kann
}
```

#### 2.2.1.2 Klassen-Invariante

```
bool vektor::invariant() const    // Vorsicht! Kein DO
{
  ...                             // nicht lo(), hi(), operator()
                                  // benutzen, da sonst Endlosrekursion
}
```

Die Klasseninvariante ist primär für die automatische Überprüfung der Gültigkeit von `*this`, also als Unterstützung des Implementierungsteams konzipiert. Der Klassenbenutzer braucht nicht direkt zu wissen, was sie überprüft, da er durch `public`-Methoden und Konstruktoren nur gültige Klassenexemplare erzeugen kann.

#### 2.2.1.3 Konstruktoren

```
vektor::vektor(int h, int l, double d) : low(l), high(h)
{
  REQUIRE(l<=h);
  ...
  ENSURE((lo()==l) && (hi()==h) &&
         A(int k=l, k<=h, k++, (*this)(k)==d));
END
...
vektor::vektor(int h, double d) : low(1), high(h)
{
  REQUIRE(h>=1);
  ...
  ENSURE((lo()==1) && (hi()==h) &&
         A(int k=lo(), k<=hi(), k++, (*this)(k)==d));
END
...
vektor::vektor(const double x[], int n) : low(1), high(n)
{
  REQUIRE((n>=1) && (x!=0));
  REQUIRE("x[] hat mindestens n Komponenten");
  ...
  ENSURE(lo()==1 && hi()==n);
  ENSURE(A(int k=lo(), k<=hi(), k++, (*this)(k)==x[k-lo()]));
END
...
vektor::vektor(const vektor& w) : low(w.low), high(w.high)
{
  REQUIRE(w.invariant());
  ...
  ENSURE((hi()-lo())==(w.hi()-w.lo()) &&
         A(int k=lo(), k<=hi(), k++, (*this)(k)==w(k-lo()+w.lo())));
END
```

#### 2.2.1.4 Destruktor

```
vektor::~vektor()
DO
        ...
}
```

#### 2.2.1.5 abgeleitete Abfragen/Operationen auf vektor-Exemplaren

```
ostream& operator<<(ostream& os, const vektor& w)
{
  ...
  ENSURE("Drucke alle vektor-Komponenten auf Stream os");
  ...
};

bool vektor::operator!=(const vektor& w) const
DO
  REQUIRE(w.invariant());
  ...
  ENSURE(result == ((hi()-lo())!=(w.hi()-w.lo())) ||
                 E(int k=lo(), k<=hi(), k++, (*this)(k)!=w(k-lo()+w.lo())));
  ...
}

bool vektor::operator==(const vektor& w) const
DO
  REQUIRE(w.invariant());
  ...
  ENSURE(result == !((*this) != w));
  ...
}

double Skalarprodukt(const vektor& v, const vektor& w)
{
  REQUIRE((v.hi()-v.lo()) == (w.hi()-w.lo()));
  REQUIRE(v.invariant());
  REQUIRE(w.invariant());
  ...
  ENSURE(approximatelyEqualTo(result, S(int k=v.lo(), k<=v.hi(),k++,
                                        v(k)*w(k-v.lo()+w.lo())), 2.0));
  ...
}
```

```
double Norm(const vektor& v)
{
  REQUIRE(v.invariant());
  ...
  ENSURE(approximatelyEqualTo(qsum, S(int k=v.lo(), k<=v.hi(),k++,
                                                  v(k)*v(k)), 2.0));
  ENSURE(approximatelyEqualTo(result*result,qsum,2.0));
  ...
}

bool approximatelyEqualVekTo(const vektor& left, const vektor& right,
                             double factor)
{
        ...
        ENSURE("relativer Abstand zwischen vektor left und right ist klein");
        ...
}
```

#### 2.2.1.6 Modifikatoren

```
double& vektor::operator()(int i)  // ermoeglicht externe Modifikationen
                                   // deshalb besser durch folgende
DO                                 // Methode ersetzen!
  REQUIRE((lo()<=i) && (i<=hi()));
  ...
  ENSURE(invariant());
  ...
}
                                                                        ...
void vektor::changeValueAt(int i, double x)
DO
  REQUIRE((lo()<=i) && (i <=hi()));
  ...
  ENSURE((*this)(i)==x);
  ENSURE("alle anderen Komponenten von *this ungeaendert");
END

vektor& vektor::operator=(const vektor& w)
DO
  ENSURE(w.invariant());
  ...
  ENSURE(*this == w);
  ...
}
```

```
void vektor::normalize()
DO
  REQUIRE(Norm(*this)!=0.0);
  ID(vektor value_old(*this));
  ...
  ENSURE(approximatelyEqualVekTo(result*n, value_old, 2.0));
  ENSURE(approximatelyEqualTo(Norm(result), 1.0, 2.0));
  ...
END
```

#### 2.2.1.7 Operationen, die vektor-Exemplare erzeugen

```
vektor vektor::operator+(const vektor& w) const
DO
  REQUIRE(w.invariant());
  REQUIRE((hi()-lo())==(w.hi()-w.lo()));
  ...
  ENSURE(result.lo()==lo());
  ENSURE(result.hi()==hi());
  ENSURE(A(int k=lo(),k<=hi(),k++,
                 approximatelyEqualTo(result(k),
                                      (*this)(k)+w(k-lo()+w.lo()),2.0)));
  ENSURE(result.invariant());
}

vektor operator+(const vektor& w, double a)
{
  REQUIRE(w.invariant());
  ...
  ENSURE(result.lo()==w.lo());
  ENSURE(result.hi()==w.hi());
  ENSURE(A(int k=result.lo(),k<=result.hi(),k++,
                 approximatelyEqualTo(result(k),w(k)+a,2.0)));
  ENSURE(result.invariant());
  ...
}

vektor operator+(double a, const vektor& w)
{
  REQUIRE(w.invariant());
  ...
  ENSURE(result.lo()==w.lo());
  ENSURE(result.hi()==w.hi());
```

```
  ENSURE(A(int k=result.lo(),k<=result.hi(),k++,
                  approximatelyEqualTo(result(k),a+w(k),2.0)));
  ENSURE(result.invariant());
  ...
}

vektor operator*(const vektor& w, double a)
{
  REQUIRE(w.invariant());
  ...
  ENSURE(result.lo()==w.lo());
  ENSURE(result.hi()==w.hi());
  ENSURE(A(int k=result.lo(),k<=result.hi(),k++,
                  approximatelyEqualTo(result(k),w(k)*a,2.0)));
  ENSURE(result.invariant());
  ...
}

vektor operator*(double a, const vektor& w)
{
  REQUIRE(w.invariant());
  ...
  ENSURE(result.lo()==w.lo());
  ENSURE(result.hi()==w.hi());
  ENSURE(A(int k=result.lo(),k<=result.hi(),k++,
                  approximatelyEqualTo(result(k),a*w(k),2.0)));
  ENSURE(result.invariant());
  ...
}

vektor vektor::ei(int n, int i)           // static Klassenmethode
{
  REQUIRE((n>=1) && (1<=i) && (i<=n));
  ...
  ENSURE(result.lo()==1);
  ENSURE(result.hi()==n);
  ENSURE(E1(int k=result.lo(), k<=result.hi(), k++, result(k)!=0.0));
  ENSURE(result(i)==1.0);
  ENSURE(result.invariant());
  ...
}
```

#### 2.2.1.8 benutzte klassenexterne Hilfsfunktionen/-Methoden

```
bool approximatelyEqualTo(double left, double right, double factor)
{
  ENSURE("relativer Abstand zwischen left und right ist klein");
  ...
}
```

### 2.2.2 Quellcode

http://www.math.uni-wuppertal.de/~buhl/teach/exercises/PbC09/vektor4-2.cpp

## 2.3 Ein Vertrag zur Klasse rationalNumber

Die Klasse

```
#define EIFFEL_DOEND
#define EIFFEL_CHECK  CHECK_ALL
#include <eiffel.h>
#include "nana.h"
#include <iostream>
#include <climits>
#include <cstdlib>
...
class rationalNumber
{
  long Z, Nenner;
public:
  void kuerze();              // sollte privat sein
public:
  rationalNumber(long z = 0, long n = 1.0);
  long getZ() const;
  long getN() const;
  virtual bool invariant() const;
};

// Definitionen:

bool MulOk( long a, long b );
bool DivOk( long a, long b );
bool AddOk( long a, long b );

long ggT(long a, long b);
long kgV(long a, long b);

rationalNumber operator+ (const rationalNumber &r1, const rationalNumber &r2);
rationalNumber operator- (const rationalNumber &r1, const rationalNumber &r2);
rationalNumber operator* (const rationalNumber &r1, const rationalNumber &r2);
rationalNumber operator/ (const rationalNumber &r1, const rationalNumber &r2);

bool operator==(const rationalNumber left, const rationalNumber right);
ostream &operator<< (ostream &os, const rationalNumber &r);
```

und ihre Verträge:

```
// grundlegende Abfragen:

inline long rationalNumber::getZ() const { return Z; };
inline long rationalNumber::getN() const { return Nenner; };

// abgeleitete Abfragen:

bool operator==(const rationalNumber left, const rationalNumber right)
{
   REQUIRE(left.invariant());
   REQUIRE(right.invariant());
   ...
   ENSURE("left repraesentiert denselben Bruch wie right");
   return(left.getN()==right.getN()) &&
         (left.getZ()==right.getZ());
}

// Die Klassen-Invariante:

bool rationalNumber::invariant() const{
   return (getN() > 0) &&
          (ggT(getN(), getZ())==1) &&
          ((getZ()!=0) || (getN()==1));

}

// Konstruktor:

inline rationalNumber::rationalNumber(long z, long n) : Z(z), Nenner(n)
{
   REQUIRE(n != 0);
   ...
   ENSURE((getZ()==0) || (z % getZ() == 0));
   ENSURE((getN()==0) || (n % getN() == 0));
   ENSURE(z * getN() == n * getZ());       // (z / getZ()) == (n / getN())
END;

// Hilfsfunktionen zur Vermeidung von Integer-Overflows:
bool MulOk( long a, long b )
{
 ...
}
```

```
bool DivOk( long a, long b )
{
 ...
}
bool AddOk( long a, long b )
{
  ...
}

// Funktionen zum Kuerzen, um unnötige Overflows
// solang wie moeglich aufzuschieben UND eindeutige
// Repraesentanten zu ermoeglichen
long ggT(long a, long b)
{
  ID(long a_old(a));
  ID(long b_old(b));
  ...
  ENSURE(result >= 0);
  ENSURE((result == 0) || labs(a_old) % result == 0);
  ENSURE((result == 0) || labs(b_old) % result == 0);
  ENSURE("result ist groesster solcher Teiler von a_old und b_old, ausser wenn:");
  ENSURE(!((a_old==0)&&(b_old==0)) || (result==0));
  ...
}
long kgV(long a, long b)
{
  ...
  ENSURE(result % a == 0);
  ENSURE(result % b == 0);
  ENSURE("result ist kleinstes solches gemeinsames Vielfaches von a und b");
  // ENSURE(ggT(a,b) == labs(a*b)/result);  // Konsistenz zu ggT()
  ...
}
...
void rationalNumber::kuerze()
{                               // Invariante nicht erfüllt(!),
  ...
  ID(long Z_old(getZ()));
  ID(long N_old(getN()));
  ...
  ENSURE(ggT(getZ(),getN())==1);
  ID(long f = N_old/getN());
  ENSURE(f * getZ()==Z_old);
END
```

```
// Operationen mit rationalNumber's:

rationalNumber operator+ (const rationalNumber &r1, const rationalNumber &r2)
{
  REQUIRE(r1.invariant());
  REQUIRE(r2.invariant());
  ...
  ENSURE(r.invariant());
  ENSURE("r ist Summe von r1 und r2");
}
rationalNumber operator- (const rationalNumber &r1, const rationalNumber &r2)
{
  REQUIRE(r1.invariant());
  REQUIRE(r2.invariant());
  ...
  ENSURE(r.invariant());
  ENSURE(r+r2 == r1);   // Konsistenz zu operator+
}
rationalNumber operator* (const rationalNumber &r1, const rationalNumber &r2)
{
  REQUIRE(r1.invariant());
  REQUIRE(r2.invariant());
  ...
  ENSURE(r.invariant());
  ENSURE("r ist Produkt von r1 und r2");
}
rationalNumber operator/ (const rationalNumber &r1, const rationalNumber &r2)
{
  REQUIRE(r1.invariant());
  REQUIRE(r2.invariant());
  ...
  ENSURE(r.invariant());
  ENSURE(r*r2==r1);         // Konsistenz zu operator*
}
ostream &operator<< (ostream &os, const rationalNumber &r)
{
  ...
  ENSURE("Druckt rationalNumber in der Form (./.) auf os");
  ...
}
```

## 2.4 Ein Vertrag mit Queries, Invariants und Actions

Das folgende Beispiel (in der **Contract Specification Language CLEO** formuliert) modelliert eine (mathematische) Menge als `Set`:

```
INTERFCE Set[Element]

    QUERIES
       Count:INTEGER
          - - Number of elements in the set.

       Has(IN x:Element):BOOLEAN
          - - Does the set contain x?
          POST
             result IMPLIES(Count>0)

       IsEmpty:BOOLEAN
          - - Does the set contain no element?

    INVARIANTS
       Count>=0
       IsEmpty=(Count=0)
    ACTIONS
       Put(IN x:Element)
          - - include x into the set.
          POST
            Has(x)
            OLD (Has(x)) IMPLIES (Count=OLD(Count))
            NOT OLD (Has(x)) IMPLIES (Count=OLD(Count)+1)

      Remove(IN x:Element)
          - - Exclude x from the set.
          POST
            NOT Has(x)
            OLD (Has(x)) IMPLIES (Count=OLD(Count)-1)
            NOT OLD (Has(x)) IMPLIES (Count=OLD(Count))
          WipeOut
            - - Exclude all elements from the set.
            POST
              Count=0
    END Set
```

Um das Verhalten der Aktionen **Put** und **Remove**, des Hinzufügens eines Elementes zur Menge und des Entfernens eines Elementes aus der Menge, zu spezifizieren,

ist der Zustand der Menge vor einem Aktionsaufruf mit ihrem Zustand nach dem Aktionsaufruf zu vergleichen. Mit OLD (...) geklammerte Ausdrücke liefern den Wert des geklammerten Ausdrucks vor einem Dienstaufruf. OLD-Ausdrücke dürfen nur in Nachbedingungen auftreten. der Ausdruck

Count=OLD(Count)+1

bedeutet, das sich der Wert von Count durch die Ausführung des Dienstes um 1 erhöht. Dabei muss es sich um eine Aktion handeln, denn da eine Abfrage q den Zustand ihrer Komponente unverändert lässt, gelten für sie Nachbedingungen der Art

q=OLD(q)

mit beliebigem Attribut q.

Vergleiche auch: http://userserv.fh-reutlingen.de/ hug/artikel/ForumWI01%20SdV.pdf

## 2.5 Contracting

Erinnerung an Kapitel 2.5.2/3.1:

| PbC | Verpflichtungen | Vorteile |
|---|---|---|
| Benutzer der Klasse | delegiert nur bei erfüllter Vorbedingung | kommt in den Genuß der garantierten Nachbedingung und Invarianten |
| Anbieter der Klasse | (nur bei gültiger Vorbedingung:) muß die Nachbedingung erfüllen | braucht Vorbedingung nicht überprüfen |

Tabelle 2.2: Verpflichtungen/Vorteile von Verträgen zwischen Komponentenanbieter und -benutzer

**Klassen-Invarianten** (Gültige Attributwertkombinationen)

- schränken Werte von Attributen ein, trennen gültige von ungültigen Exemplaren einer Klasse
- spezifizieren Redundanzen (vgl. Day/Month/Year, Count/IsEmpty, ...)

**Methoden-Vorbedingungen** (an Attribute und Parameter)

- schränken den Bereich ein, in dem die Methode erfolgreich sein muß

**Methoden-Nachbedingungen** (an Attribute und Parameter)

- spezifizieren (formal) das Ergebnis der Methode (das **was**, nicht das **wie**)

Was vor und nach jeder Methode gelten muß:

Konstruktor:
$\{Vorbed\_an\_Parameter\}\ Konstruktor\ \{\text{Inv}\}$

Jede andere Methode M:

$\{Vorbed_M \wedge Inv\}\ M\ \{Nachbed_M \wedge Inv\}$

### 2.5.1 Subcontracting (is-a-Vererbung)

Ein erstes Beispiel:

**Ursprüngliche Definition:**
invert(epsilon:REAL) is - - Invert martix with precision epsilon
    require epsilon>= 10^(−6)
    ...
    ensure abs ((Current * inverse) - Identy) <= epsilon
end

**Redefinition:**
invert(epsilon:REAL)is - - Invert matrix with precision epsilon
    require else epsilon>= 10^(−20)
...
    ensure abs ((Current * inverse) - Identy) <= (epsilon/2)
end

aus: http://www.cse.yorku.ca/course_archive/2004-05/F/3311/sectionA/22-InheritDBCgen.pdf

**Unternehmer und Subunternehmer:**

Ein Vertrag zwischen Kunde und Unternehmer laute:

```
                    Interface Directory
.
.
.
ACTIONS
   Put(IN k:Keys, IN v:Values)
      PRE
         NOT Has(k)
      POST
         Has(k)
         ValueFor(k)= v
         Count = OLD(Count)+1
.
.
.
```

Kann er durch den Unternehmer allein nicht zeitgerecht erfüllt werden, so kann sich dieser eventuell folgendermaßen aus seiner Notlage befreien: Ein anderer Unternehmer biete den folgenden Vertrag an:

```
                    Interface DirectoryB
.
.
.
ACTIONS
   Put(IN k:Keys, IN v:Values)
      PRE
         TRUE
      POST
         Has(k)
         ValueFor(k)= v
         NOT OLD(Has(k)) IMPLIES Count = OLD(Count)+1
         OLD(Has(k))     IMPLIES Cout = OLD(Count)
.
.
.
```

Er kann als Subunternehmer des Unternehmers in die Pflicht genommen werden, da DirectoryB den Vertrag Directory vollständig erfüllt. Der Kunde kann, da an die Vorbedingungen von Directory gebunden, keinen Unterschied zwischen der Benutzung von Directory und DirectoryB feststellen. Lediglich wenn er sich nicht an die Vorbedingungen des Vertrags Directory hielte, wäre das Verhalten seiner Software anders, aber das darf er ja nicht!
Subcontracting (is-a-Vererbung) verlangt also folgende Beziehungen zwischen den Verträgen der public Methoden einer Kindklasse und denjenigen ihrer Elterklasse:

- Invarianten des Kindes dürfen nicht schwächer sein als diejenigen der Elterklasse,
- Vorbedingungen einer Kindmethode dürfen nicht stärker sein als die Vorbedingungen der Eltermethode,
- Nachbedingungen einer Kindmethode dürfen beim Eintreten der Vorbedingung der Eltermethode nicht schwächer sein als die Nachbedingung der Eltermethode, andernfalls dürfen sie beliebig sein. (Diese Beliebigkeit wird natürlich im allgemeinen vom Subunternehmer dazu genutzt, einen möglichst großen Marktwert für seine Mehrfunktionalität — es gibt mehr Fälle, in denen die Vorbedingung der Kindmethode erfüllt ist, als die Fälle in denen lediglich die Vorbedingung der Eltermethode erfüllt sind — zu erreichen.)

Kurz:

$$Invariante_{Kindklasse} = Invarinte_{Elterklasse} \wedge ...$$
$$Vorbedingung_{Kindmethode} = Vorbedingung_{Eltermethode} \vee ...$$
$$Nachbedingung_{Kindmethode} = \begin{cases} Nachbedingung_{Eltermethode} \wedge ... & \text{, falls } Vorbedingung_{Eltermethode} \\ \text{beliebig} & \text{, sonst} \end{cases}$$

Diese formalen Subcontracting-Regeln können in der Anwendung häufig logisch zusammengefaßt werden. Im obigen Beispiel also:

```
-------------------------------- Directory
Pre-Put_Directory(k,v)  = NOT Has(k)

Post-Put-Directory(k,v) = Has(k) AND ValueFor(k) = v AND
                          Count = OLD(Count)+1
```

liefert nach den obigen Regeln:

```
-------------------------------- DirectoryB
Pre-Put_DirectoryB(k,v)  = TRUE

Post-Put_DirectoryB(k,v) =
  (Pre-Put_Directory(k,v) IMPLIES Post-Put-Directory(k,v))
 AND
  ((NOT Pre-Put_Directory(k,v) AND Pre-Put_DirectoryB(k,v)) IMPLIES "Beliebiges"

  = (NOT OLD(Has(k))) IMPLIES (Has(k) AND ValueFor(k) = v AND
                                         Count = OLD(Count)+1))
    AND ((OLD(Has(k)) AND TRUE) IMPLIES "Beliebiges")
```

Nach dem Namen der Methode `Put()` ist "Beliebiges" natürlich einzig sinnvoll durch

```
Has(k) AND Count = OLD(Count) AND ValueFor(k) = ?
```

zu ersetzen. Dabei hat man für den letzten Anteil (Wert beim Schlüssel `k`) noch eine gewisse Entscheidungsfreiheit. Mögliche, sinnvoll erscheinende Spezifikationen:

- Der alte Wert bleibt erhalten.
- Der alte Wert wird überschrieben.
- Der Wert wird als „unbestimmt“ gekennzeichnet, da er in der Historie mit unterschiedlichen Wertbindungen versehen werden sollte.

Entscheidet man sich für die vorraussichtlich marktrelevanteste Spezifikation (Wertüberschreibung), so erhält man nach ein paar Zusammenfassungen:

```
-------------------------------- DirectoryB
Pre-Put_DirectoryB(k,v)  = TRUE
Post-Put_DirectoryB(k,v) =
 (NOT OLD(Has(k))) IMPLIES (Has(k) AND ValueFor(k) = v AND
                                      Count = OLD(Count)+1)) AND
 ((OLD(Has(k)) AND TRUE) IMPLIES (Has(k) AND Count = OLD(Count) AND
                                                ValueFor(k) = v))
```

```
  = Has(k) AND ValueFor(k) = v AND
    (NOT OLD(Has(k)) IMPLIES Count = OLD(Count)+1) AND
    ((OLD(Has(k)) IMPLIES Count = OLD(Count))
```

Ein weiteres Subcontracting-Beispiel:
**„Löse ein LGS“**

```
------------------------------ LoeseLGS-Elter
ACTIONS
   LoeseLGS( IN A : Matrix,
             IN b : Vektor,
            OUT x : Vektor )
      PRE
         NOT Det(A) = 0
      POST
         || A * x - b || <= EPSILON
```

sowie ein Subcontract:

```
------------------------------ LoeseLGS-Kind
ACTIONS
   LoeseLGS( IN A : Matrix,
             IN b : Vektor,
            OUT x : Vektor )
      PRE
         TRUE
      POST
         NOT Det(A) = 0 IMPLIES || A * x - b || <= EPSILON
         Det(A) = 0     IMPLIES "x ist eine Minimalstelle von || A * x - b ||"
```

**Beipiel: Bruecke**

```
------------------------------ Fussgaengerbruecke
   QUERIES
      MaxLast : REAL
      AktLast : REAL
   INVARIANTS
      MaxLast >= 7500
      AktLast <= MaxLast
   ACTIONS
      ueberquereBruecke( IN gew : REAL,
                         OUT Guthaben : INTEGER )
         PRE
            gew  + AktLast <= MaxLast
            gew <= 200
            Guthaben >= 2
         POST
            AktLast = OLD(AktLast) + gew
            Guthaben = OLD(Guthaben) - 2
      verlasseBruecke( IN gew : REAL )
         ...
```

sowie ein Subcontract:

```
------------------------------ Autobruecke
   QUERIES
      MaxLast : REAL
      AktLast : REAL
   INVARIANTS
      MaxLast >= 800000
      AktLast <= MaxLast
   ACTIONS
      ueberquereBruecke( IN gew : REAL,
                         OUT Guthaben : INTEGER )
         PRE
            gew  + AktLast <= MaxLast
            gew <= 20000
            Guthaben >= 20
         POST
            AktLast = OLD(AktLast) + gew
            OLD(gew) <= 200     IMPLIES Guthaben = OLD(Guthaben) - 2
            NOT OLD(gew) <= 200 IMPLIES Guthaben = OLD(Guthaben) - 20
      verlasseBruecke( IN gew : REAL )
         ...
```

**Aufgabe:**

Überlege Contracts und Subcontracts im Umfeld:
— Kunde/Stammkunde
— Firmenkonto/Privatkundenkonto
— Vereinsmitglied /Vorstandsmitglied
— ...

Siehe auch Kapitel 5.6.

# 3 Eclipse mit ...

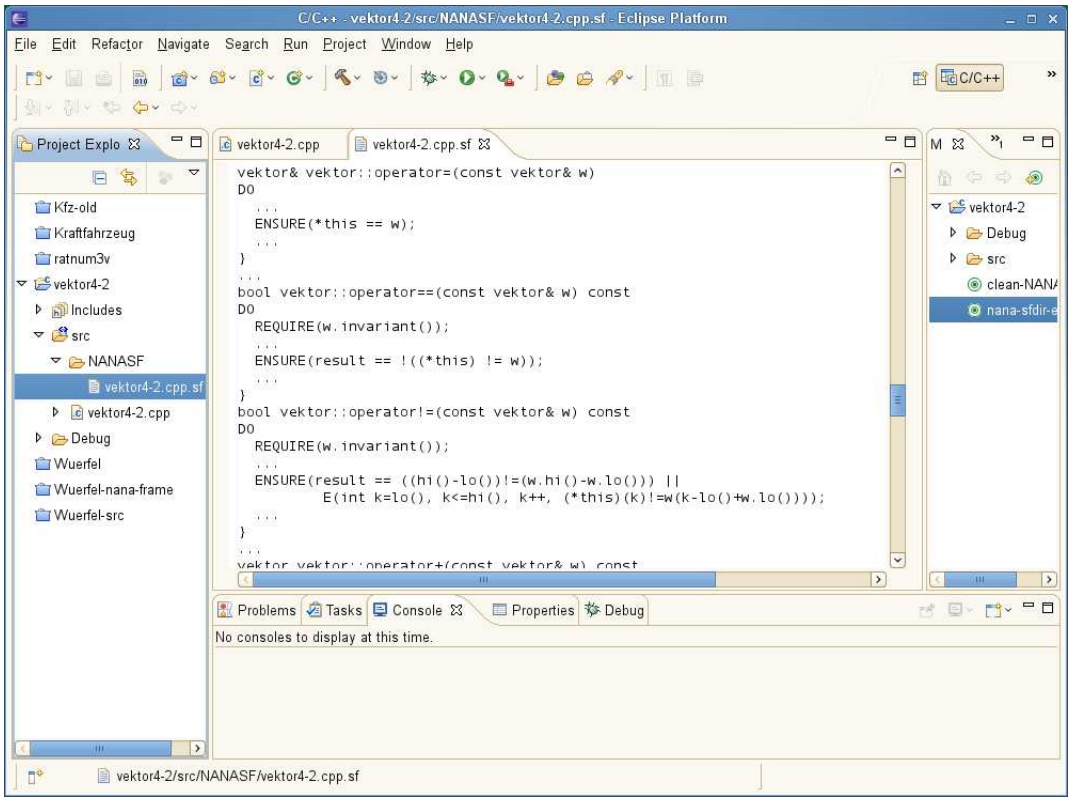
## 3.1 Make-Target für NANASF

Nach Anlegen einer Kopie `nana-sfdir-eclipse` von `nana-sfdir` und zwei kleinen Änderungen

```
...
cd ../src
...
     ${NANABIN-${exec_prefix}/bin}/nana-sfg $f >$TARGET/$f.sf
...
```

kann man mittels zweier kleiner Make-Targets `make-nana-shortform` und `clean-nana-shortform`

auf einfache Art die Short-Form-Vertragsübersichten in `eclipse` generieren beziehungsweise löschen:



## 3.2 Unit-Tests: cxxtest

Sinnvollere Komponententests als rudimentäre `main()`-Testrahmenprogramme (mit Ausdruck des Exemplarstatus unter Zuhilfenahme einer Serialisierungs-Klassenmethode

```
string Auto::toString() const{
ostringstream help;
help << getKennzeichen() << "; " <<
               getAnzahlSitzplaetze() << "; " <<
               getZulGesamtgewicht();
return help.str();
}
```

in der Form

```
int main(int argc, char* argv[]) {

  std::cout << "Testrahmenprogramm für Klasse Auto!" <<
                   std::endl << std::endl;

  Fahrzeuge::Auto a;
  std::cout<< a.toString() << std::endl;

  Fahrzeuge::LKW b;
  std::cout<< b.toString() << std::endl;

  b.setAnzahlSitzplaetze(1);
  assert(b.toString() == "unbekannt; 1; 3500");
  ...
```

kann man zum Beispiel mit Hilfe von Unittest-Tools konstruieren. Wegen diverser Vorteile (unter anderem:

- Doesn't require RTTI
- Doesn't require member template functions
- Doesn't require exception handling
- Doesn't require any external libraries (including memory management, file/console I/O, graphics libraries)
- Is distributed entirely as a set of header files (and a python script).
- Doesn't require the user to manually register tests and test suites

) wollen wir `cxxtest` einsetzen.
Nach dem Download von cxxtest.sourceforge.net können wir das Testrahmenprogramm `main()` durch eine Datei `FahrzeugeTestSuite.h`

```
/*!
 * FahrzeugeTestSuite.h
 *
 *  Created on: 17.04.2009
 *      Author: HJB
 */

#include <cxxtest/TestSuite.h>
#include "Auto.h"
#include "LKW.h"

class FahrzeugeTestSuite: public CxxTest::TestSuite{
public:
  void testAutoDefaultConstructor(void)
  {  Fahrzeuge::Auto a;
     TS_ASSERT_EQUALS(a.getKennzeichen(), "unbekannt");
     TS_ASSERT_EQUALS(a.getAnzahlSitzplaetze(), 4);
     TS_ASSERT_EQUALS(a.getZulGesamtgewicht(), 1000.0);
  }
  void testLKWDefaultConstructor(void)
  {  Fahrzeuge::LKW a;
     TS_ASSERT_EQUALS(a.getKennzeichen(), "unbekannt");
     TS_ASSERT_EQUALS(a.getAnzahlSitzplaetze(), 2);
     TS_ASSERT_EQUALS(a.getZulGesamtgewicht(), 3500.0);
  }
  void testLKWSetAnzahlSitze(void)
  {  Fahrzeuge::LKW a;
     TS_ASSERT_EQUALS(a.getKennzeichen(), "unbekannt");
     TS_ASSERT_EQUALS(a.getAnzahlSitzplaetze(), 2);
     TS_ASSERT_EQUALS(a.getZulGesamtgewicht(), 3500.0);
     a.setAnzahlSitzplaetze(1);
     TS_ASSERT_EQUALS(a.getKennzeichen(), "unbekannt");
     TS_ASSERT_EQUALS(a.getAnzahlSitzplaetze(), 1);
     TS_ASSERT_EQUALS(a.getZulGesamtgewicht(), 3500.0);
  }
  // ...
};
```

ersetzen, mittels
`cxxtestgen.pl --error-printer -o runner.cpp FahrzeugeTestSuite.h`

in ein Programm `main()` (in der Datei `runner.cpp`) umwandeln lassen und nach Compilation den Test durchführen:

```
./runner
Running 3 tests...OK!
```

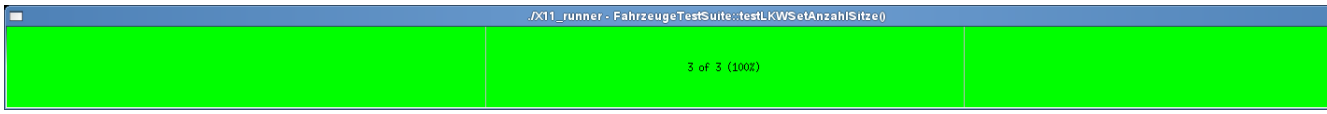
beziehungsweise bem Auftreten von Abweichungen des Testlaufs vom Testorakel:

```
./runner
Running 3 tests..
In FahrzeugeTestSuite::testLKWSetAnzahlSitze:
FahrzeugeTestSuite.h:33: Error: Expected (a.getAnzahlSitzplaetze() == 3), found (
Failed 1 of 3 tests
Success rate: 66%
```

Mittels

```
cxxtestgen.pl --error-printer --gui=X11Gui -o X11_runner.cpp FahrzeugeTestSuite.h
...
g++ -c X11_runner.cpp -I/usr/X11R6/include -L/usr/X11R6/lib -lX11
g++ -c Auto.cpp
g++ -c LKW.cpp
g++ -o X11_runner X11_runner.o Auto.o LKW.o -I/usr/X11R6/include -L/usr/X11R6/lib
```

kann man in größeren Projekten einen GUI-Runner nutzen:

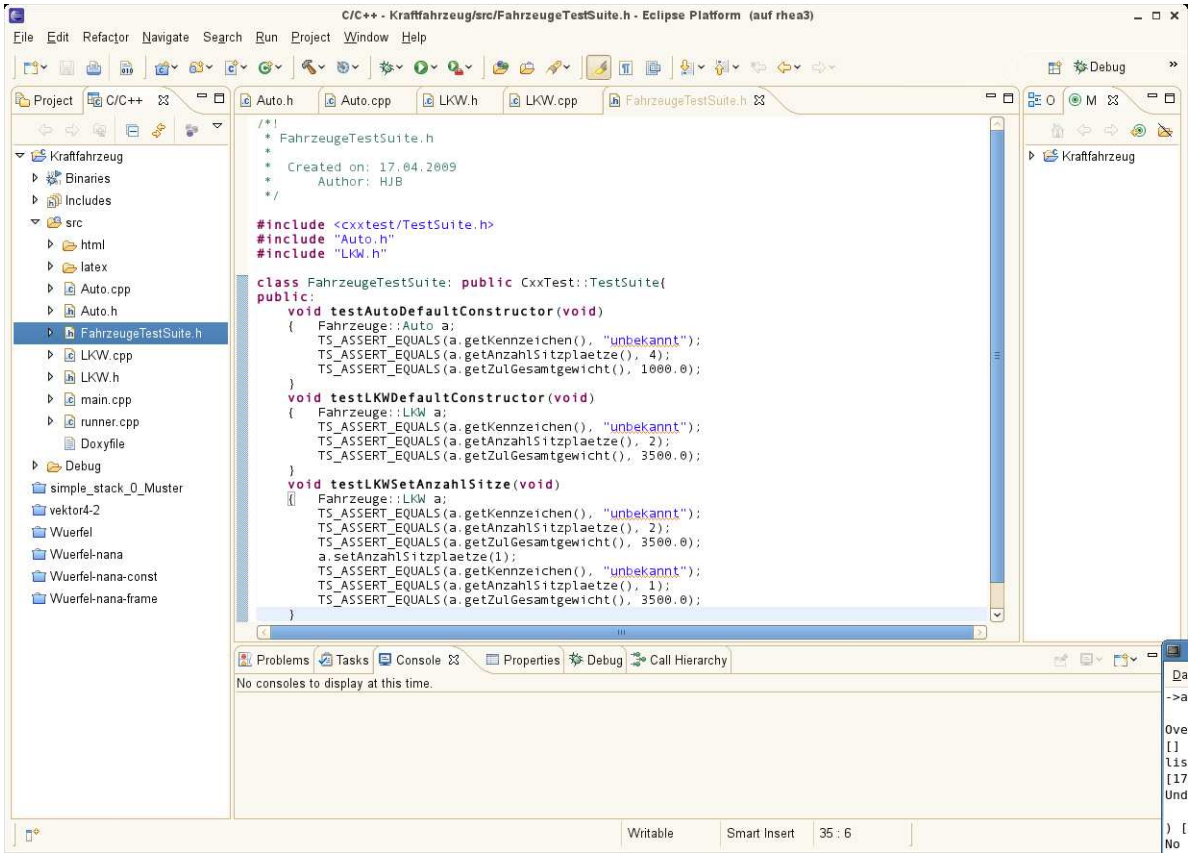



Andere Test-Macros von `cxxtest`:

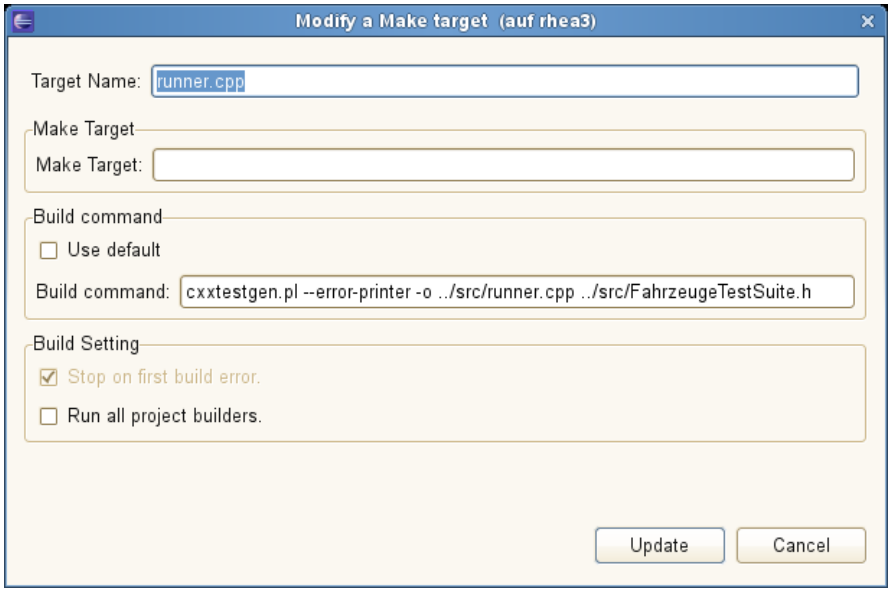
```
TS_FAIL(message)                Fail unconditionally: TS_FAIL("Test not implement
TS_ASSERT(expr)                 Verify (expr) is true
TS_ASSERT_EQUALS(x, y)          Verify (x==y)
TS_ASSERT_SAME_DATA(x, y, size) Verify two buffers are
TS_ASSERT_DELTA(x, y, d)        Verify (x==y) up to d
TS_ASSERT_DIFFERS(x, y)         Verify !(x==y)
TS_ASSERT_LESS_THAN(x, y)       Verify (x<y)
TS_ASSERT_LESS_THAN_EQUALS(x,y) Verify (x<=y)
TS_ASSERT_PREDICATE(R, x)       Verify P(x)
TS_ASSERT_RELATION(R, x, y)     Verify x R y
TS_ASSERT_THROWS(expr, type)    Verify that (expr) throws
                                a specific type of exception
...
```

`cxxtest`-Dokumentation: http://cxxtest.sourceforge.net/guide.pdf

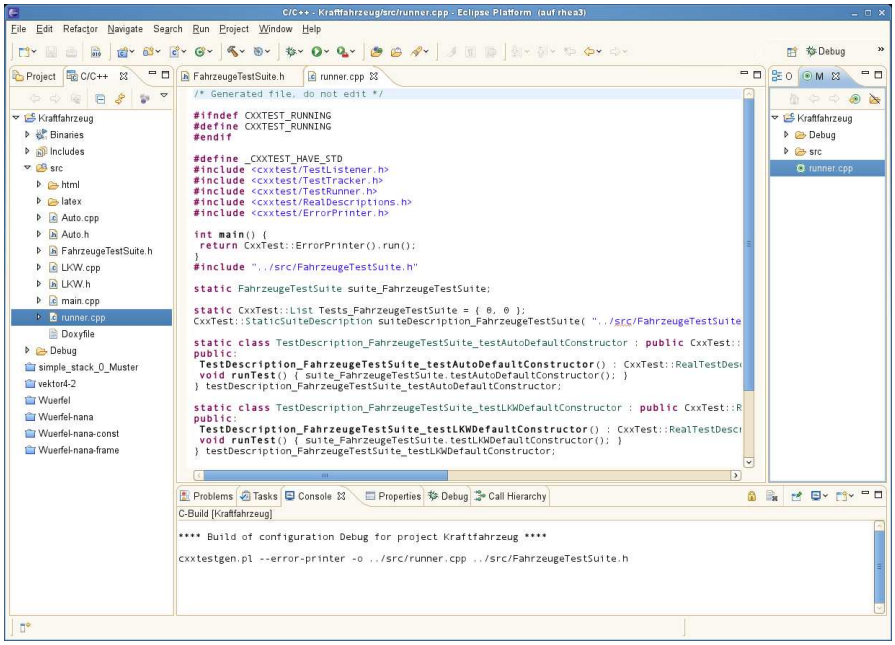
Doch nun zur Integration in `eclipse`: Entfernen Sie Ihre Funktion `main()` oder nennen Sie sie um. Ergänzen Sie Ihr Projekt um eine Header-Datei `NNTestSuite.h` (z.B. `FahrzeugeTestSuite.h`der Form:



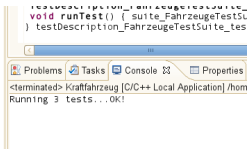
Erzeugen Sie ein MakeTarget (Project-Menü von `eclipse`) `runner.cpp`:




Doppelklick auf dieses Make-Ziel erzeugt immer die aktuelle Version der Datei `runner.cpp` aus `FahrzeugeTestSuite.h`:

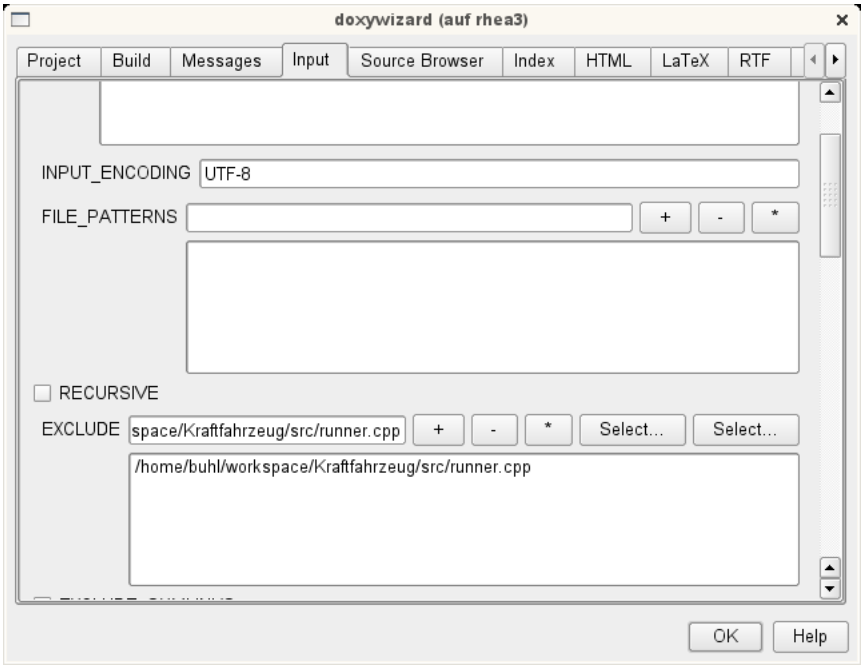



Die `eclipse`-Befehle `Build` und `Run` führen dann immer alle Unit-Tests aus:




Siehe auch:
Unit Testing & CxxTest
CxxTest im Unit-Testing-Jungle
http://cxxtest.com/index.php?title=Articles
http://cxxtest.com/index.php?title=Main_Page
http://de.wikipedia.org/wiki/Testgetriebene_Entwicklung

In `Doxyfile` sollte `runner.cpp` aus der Dokumentation herausgenommen werden:



## 3.3 Testabdeckungsüberprüfung

lcov zur Testabdeckungsüberprüfung:
Installation von `lcov-1.7-1.noarch.rpm` durch den Systemadministrator.
Anwendungsbeispiel:

```
> ls
main.cpp  makefile  Wuerfel.cpp  Wuerfel.h

> g++ -c Wuerfel.cpp -g --coverage
> g++ -o main main.cpp Wuerfel.o -g --coverage

> ls
main      main.gcno  Wuerfel.cpp   Wuerfel.h
main.cpp  makefile   Wuerfel.gcno  Wuerfel.o
```

Nach der Durchführung mindestens eines Testlaufs

```
> ./main
Testrahmenprogramm fuer Wuerfel:
Seite:       1
Oberflaeche: 6
```

existieren Testabdeckungsdaten in `*.gcda`-Dateien

```
ls
main      main.gcda  makefile     Wuerfel.gcda  Wuerfel.h
main.cpp  main.gcno  Wuerfel.cpp  Wuerfel.gcno  Wuerfel.o
```

die mittels lcov ausgewertet werden können:

```
> lcov -d ‘pwd‘ -c -o gcov.info
Capturing coverage data from /home/buhl/Wuerfel
Found gcov version: 4.3.2
Scanning /home/buhl/Wuerfel for .gcda files ...
Found 2 data files in /home/buhl/Wuerfel
Processing /home/buhl/Wuerfel/Wuerfel.gcda
Processing /home/buhl/Wuerfel/main.gcda
Finished .info-file creation
```

Die erzeugte Datei `gcov.info` kann schließlich in eine html-Übersicht gewandelt werden:

```
> genhtml -o html gcov.info
Reading data file gcov.info
Found 20 entries.
```

```
Found common filename prefix "/usr/include/c++/4.3"
Writing .css and .png files.
Generating output.
Processing file /home/buhl/Wuerfel/Wuerfel.cpp
Processing file /home/buhl/Wuerfel/main.cpp
Processing file iostream
Processing file sstream
Processing file ostream
Processing file iosfwd
Processing file streambuf
Processing file bits/basic_string.tcc
Processing file bits/basic_ios.h
Processing file bits/char_traits.h
Processing file bits/stl_iterator_base_funcs.h
Processing file bits/basic_string.h
Processing file bits/allocator.h
Processing file bits/stl_iterator_base_types.h
Processing file bits/locale_facets.h
Processing file bits/ios_base.h
Processing file ext/atomicity.h
Processing file ext/type_traits.h
Processing file ext/new_allocator.h
Processing file i586-suse-linux/bits/gthr-default.h
Writing directory view page.
Overall coverage rate:
  lines......: 9.3% (18 of 194 lines)
  functions..: 8.6% (7 of 81 functions)
```

die mittels

```
> konqueror html/index.html
```

studiert werden kann:

# LCOV - code coverage report

| | | | Found | Hit | Coverage |
|---|---|---|---|---|---|
| Current view: | directory | Lines: | 194 | 18 | 9.3 % |
| Test: | gcov.info | Functions: | 81 | 7 | 8.6 % |
| Date: | 2009-04-30 | | | | |

| Directory | Line Coverage | | | Functions | | |
|---|---|---|---|---|---|---|
| /home/buhl/Wuerfel | | 89.5 % | 17 / 19 | 63.6 % | 7 / 11 |
| /usr/include/c++/4.3 | | 2.3 % | 1 / 44 | 0.0 % | 0 / 18 |
| bits | | 0.0 % | 0 / 116 | 0.0 % | 0 / 45 |
| ext | | 0.0 % | 0 / 13 | 0.0 % | 0 / 6 |
| i586-suse-linux/bits | | 0.0 % | 0 / 2 | 0.0 % | 0 / 1 |

Generated by: LCOV version 1.7

# LCOV - code coverage report

| | | | Found | Hit | Coverage |
|---|---|---|---|---|---|
| Current view: | directory - /home/buhl/Wuerfel | Lines: | 19 | 17 | 89.5 % |
| Test: | gcov.info | Functions: | 11 | 7 | 63.6 % |
| Date: | 2009-04-30 | | | | |

| Filename | Line Coverage | | | Functions | | |
|---|---|---|---|---|---|---|
| Wuerfel.cpp | | 83.3 % | 10 / 12 | 50.0 % | 4 / 8 |
| main.cpp | | 100.0 % | 7 / 7 | 100.0 % | 3 / 3 |

Generated by: LCOV version 1.7

# LCOV - code coverage report

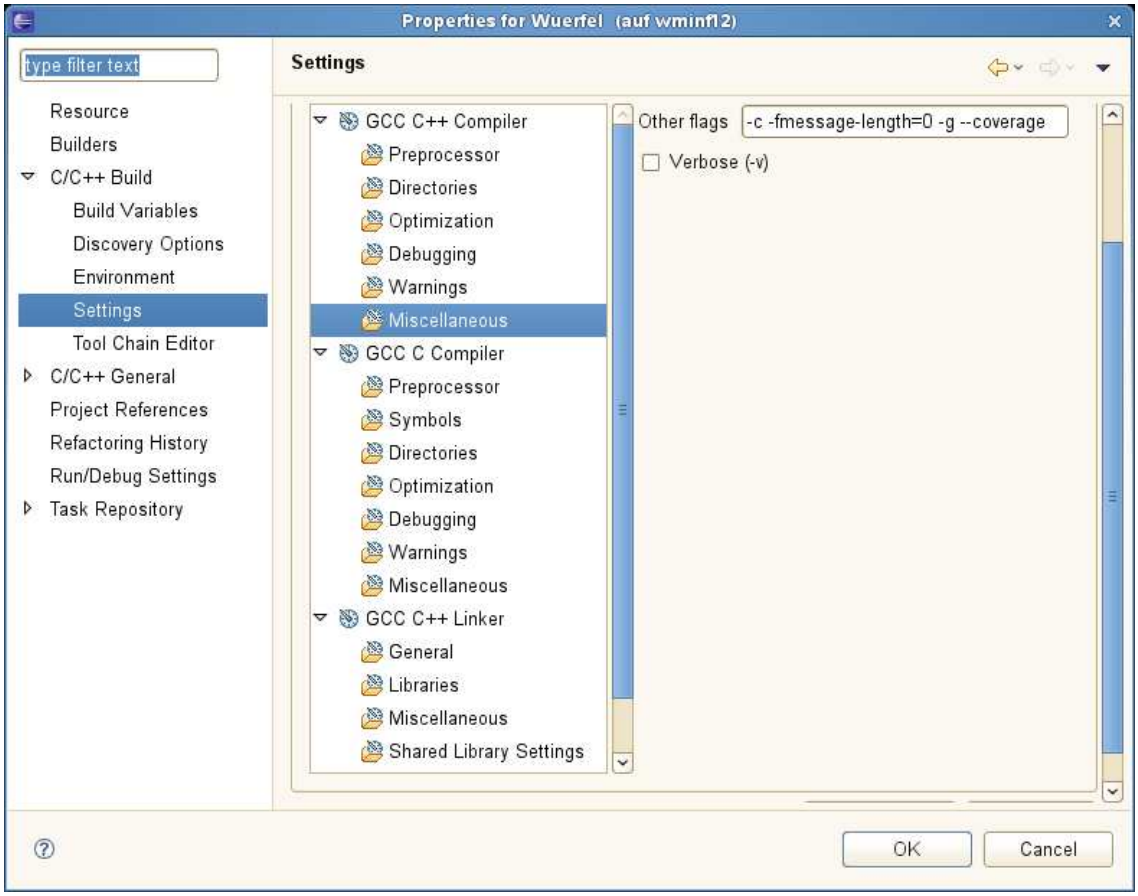
| Current view: | directory - home/buhl/Wuerfel - Wuerfel.cpp (source / functions) | | Found | Hit | Coverage |
|---|---|---|---|---|---|
| Test: | gcov.info | Lines: | 12 | 10 | 83.3 % |
| Date: | 2009-04-30 | Functions: | 8 | 4 | 50.0 % |

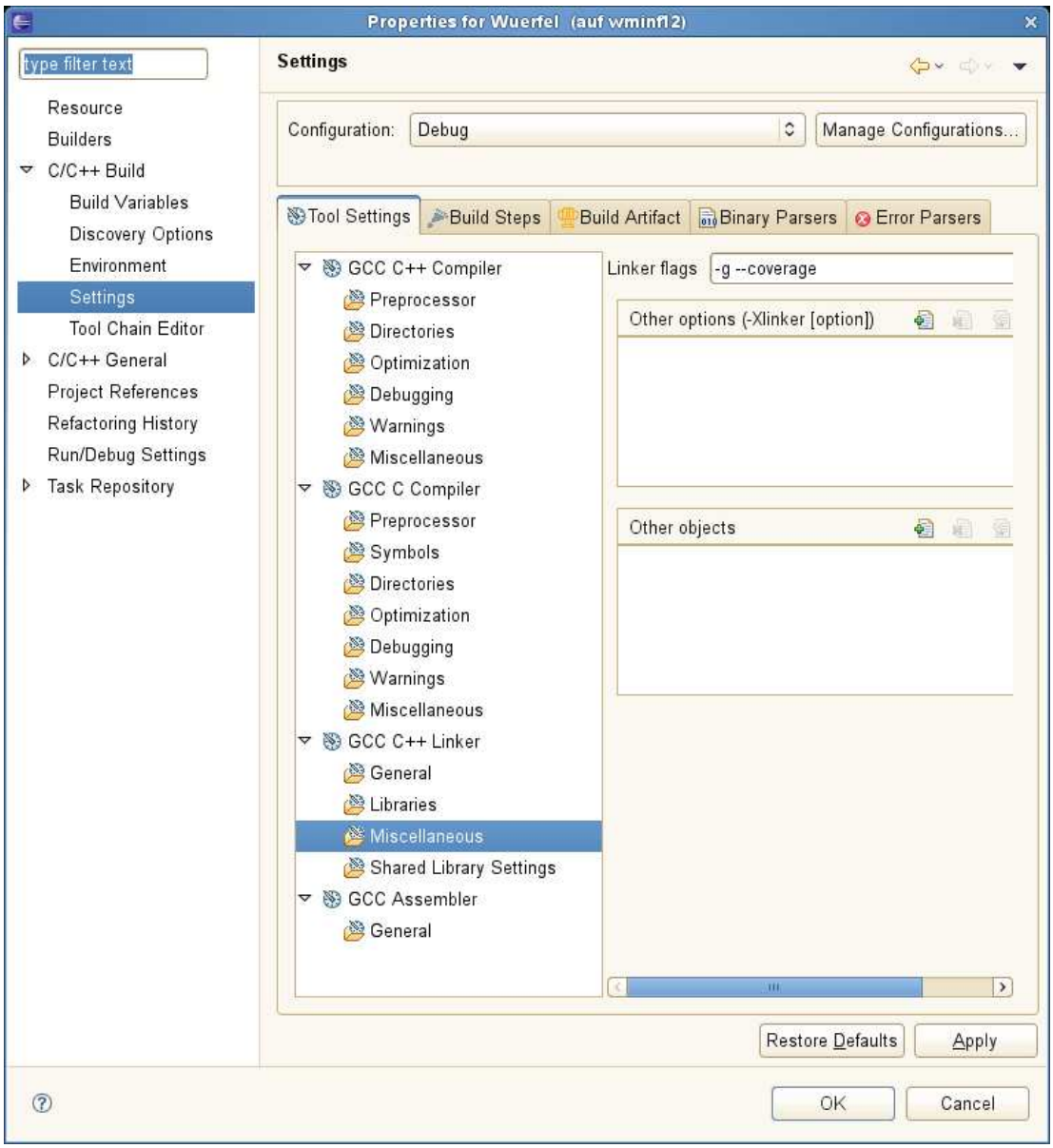
```
    1                :  /*
    2                :   * Wuerfel.cpp
    3                :   *
    4                :   *  Created on: 22.04.2009
    5                :   *      Author: buhl
    6                :   */
    7                :
    8                :  #include <sstream>
    9                :  #include <cmath>
   10                :  #include "Wuerfel.h"
   11                :
   12              1 :  Wuerfel::Wuerfel(double Seite): Seite(Seite) {
   13              1 :  }
   14                :
   15              1 :  Wuerfel::~Wuerfel() {
   16                :          // TODO Auto-generated destructor stub
   17              1 :  }
   18                :
   19              1 :  std::string Wuerfel::toString()
   20                :  {
   21              1 :          std::ostringstream help;
   22              1 :          help << Seite;
   23              1 :          return help.str();
   24                :  }
   25                :
   26              1 :  double Wuerfel::Oberflaeche()
   27                :  {
   28              1 :          return 6.0 * pow(Seite, 2);
   29                :  }
   30                :
   31              0 :  double Wuerfel::Volumen()
   32                :  {
   33              0 :          return pow(Seite,3);
   34                :  }
   35                :
   36                :
```

Ein `makefile` zur einfacheren Benutzung:

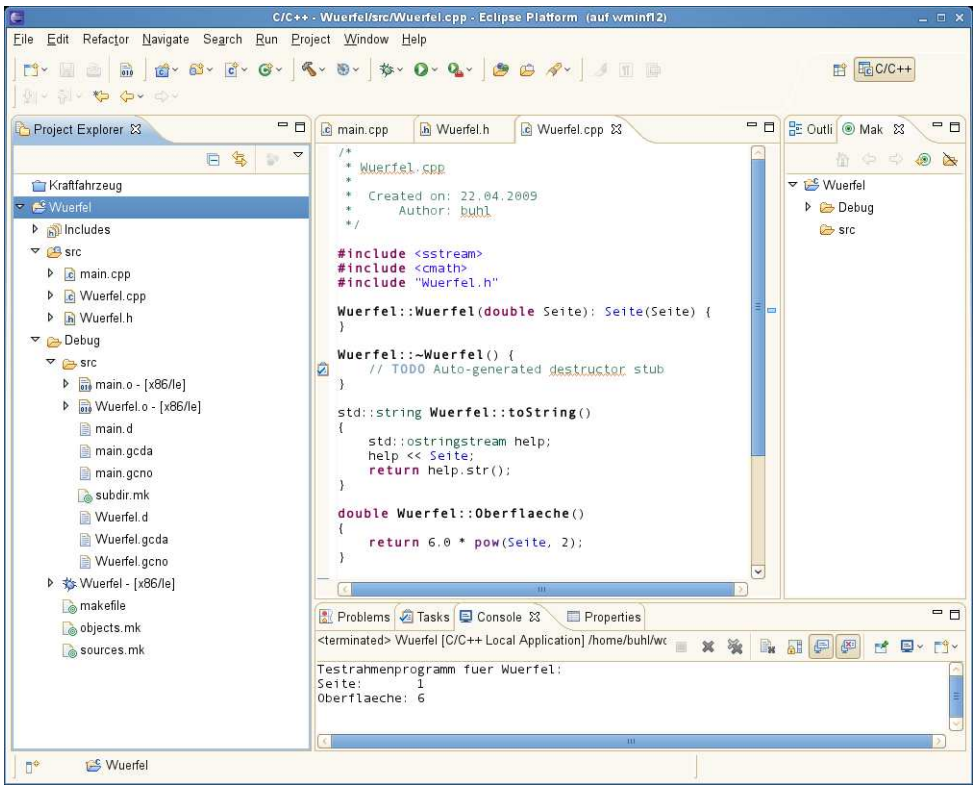
```
# For coverage infos
COV_FLAGS = -g --coverage

main: Wuerfel.o
        g++ -o main main.cpp Wuerfel.o $(COV_FLAGS)

Wuerfel.o: Wuerfel.cpp
        g++ -c Wuerfel.cpp $(COV_FLAGS)

lcov-html:
        lcov -d ‘pwd‘ -c -o gcov.info
        genhtml -o html gcov.info

clean:
        rm -f *.o main *.gc*
        rm -rf html
        rm gcov.info
```

Nun zur Integration in `eclipse`:
Selektieren Sie den Projektnamen und wechseln Sie zu den Eigenschaften des Projekts, um die Optionen `-g --coverage` sowohl fürs C++-Übersetzen als auch fürs Linken zu ergänzen:

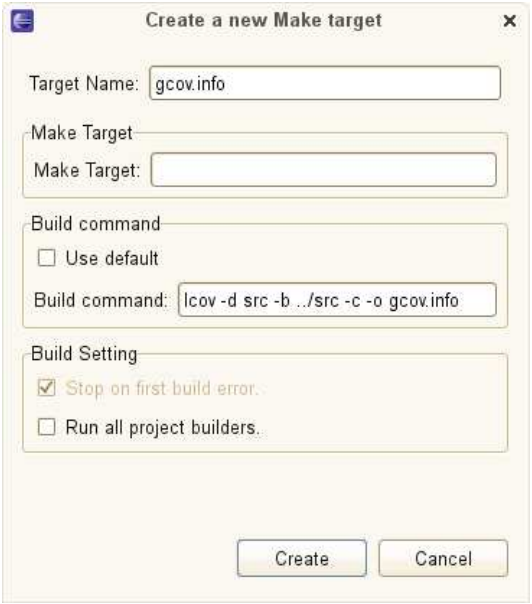

Properties for Wuerfel (auf wminf12)
type filter text
Resource
Builders
C/C++ Build
Build Variables
Discovery Options
Environment
Settings
Tool Chain Editor
C/C++ General
Project References
Refactoring History
Run/Debug Settings
Task Repository
Settings
GCC C++ Compiler
Preprocessor
Directories
Optimization
Debugging
Warnings
Miscellaneous
GCC C Compiler
Preprocessor
Symbols
Directories
Optimization
Debugging
Warnings
Miscellaneous
GCC C++ Linker
General
Libraries
Miscellaneous
Shared Library Settings
Other flags
-c -fmessage-length=0 -g --coverage
Verbose (-v)
OK
Cancel



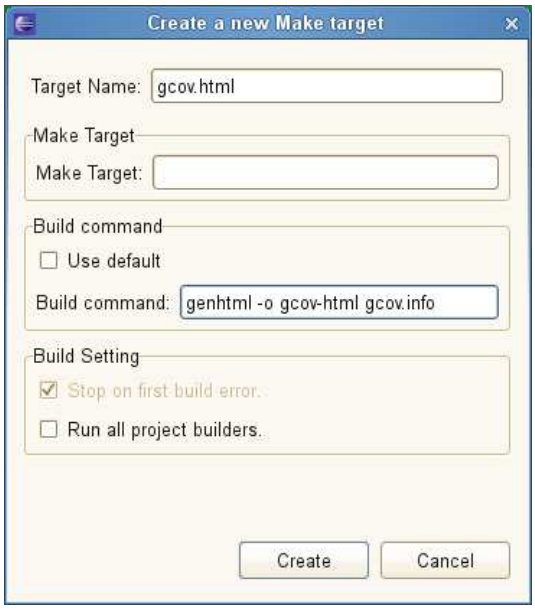

Nach erneutem `Build` und mindestens einmaligem Ausführen des Testrahmenprogramms `main()` sind die `*.gcno`- und `*.gcda`-Dateien angelegt worden:



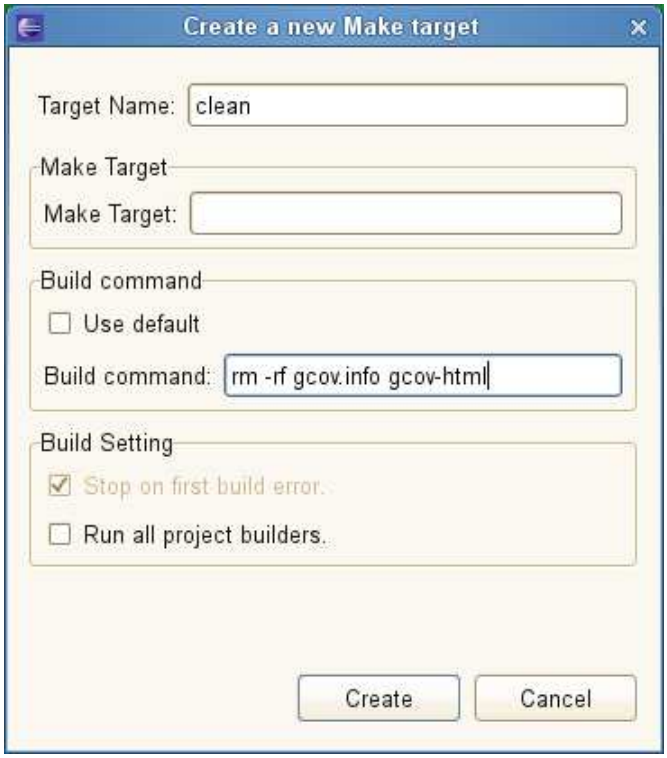

Die Ausführung der `lcov`- und `genhtml`-Kommandos kann man als Make-Targets konfigurieren:
Nach Selektion von `Wuerfel`, `Project`, `Make Target` und dann `Create...` auswählen sowie folgendes eintragen:



Analog




und




als `Make Target` konfigurieren. Jetzt sind die drei `Make Target`s durch Doppelklick im `Make Targets`-View startbar.

## 3.4 valgrind (Speicherzugriffsüberprüfung)

Benutzung in `eclipse` gemäß der Seiten 377 bis 387 der zweiten Auflage des Buchs *Eclipse für C/C++-Programmierer (Sebastian Bauer), dpunkt.verlag,2011.*

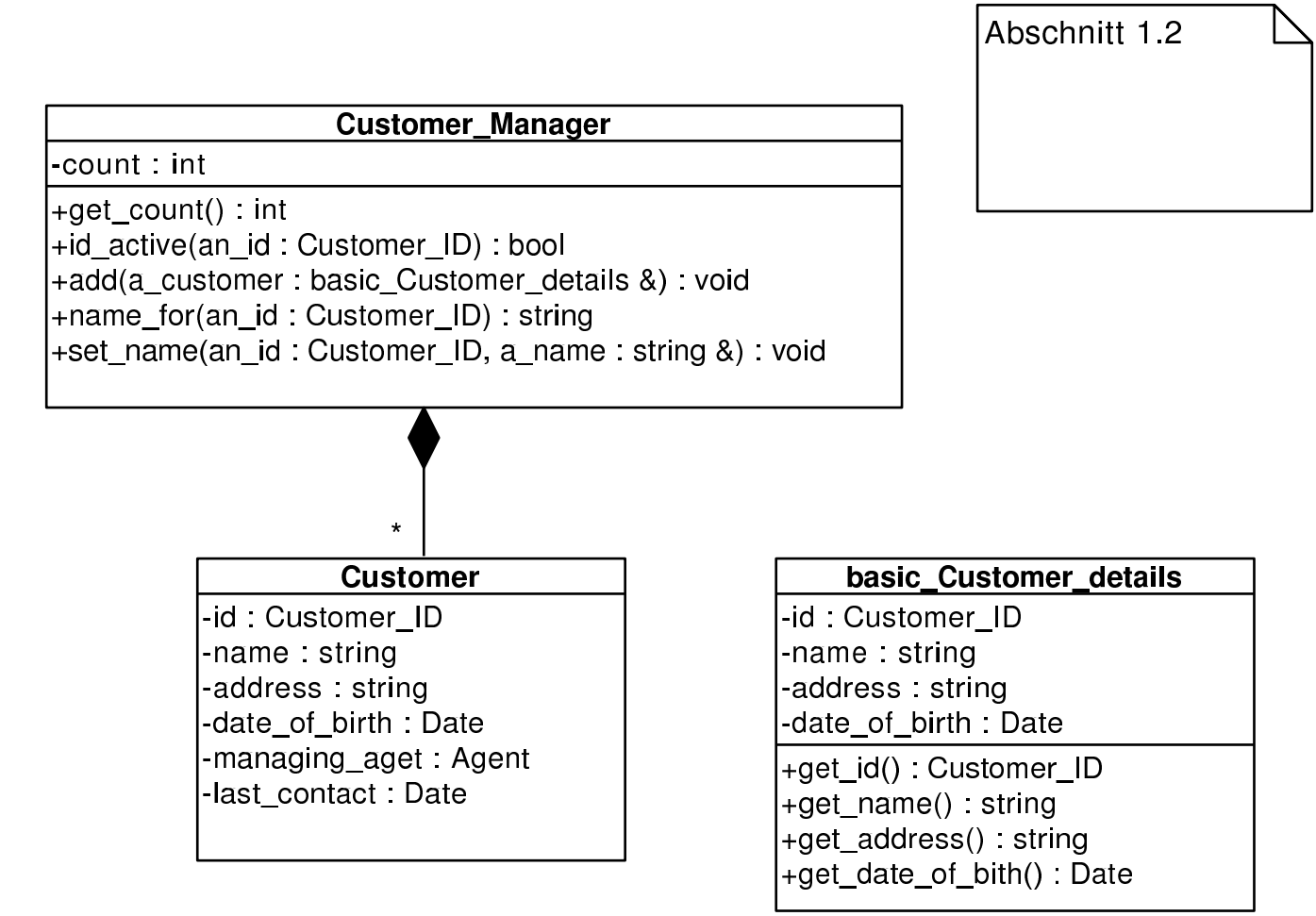
## 3.5 GProf (Laufzeitanalyse)

Benutzung in `eclipse` gemäß der Seiten 397 bis 388 der zweiten Auflage des Buchs *Eclipse für C/C++-Programmierer (Sebastian Bauer), dpunkt.verlag,2011.*

# A Degign by Contract, by Example, in C++ with nana

## A.1 A first Taste of Design by Contract






```
// Kapitel 7.1.2
```

```
//
// c++ -c  DbCrev.cc -I$HOME/include -L$HOME/lib -lnana
//
// eventuell mit
//                   -DWITHOUT_NANA
// und/oder
//                   -DNDEBUG
//
// und/oder
//                   -DEIFELL_CHECK=CHECK_....
//
// ------------------------------------------------------------------
// evtl. myexcept.o mit angeben

#define EIFFEL_DOEND

#ifndef EIFFEL_CHECK
#define EIFFEL_CHECK CHECK_ALL
//                                          Makros CHECK() und folgende
//                     CHECK_LOOP           Makros INVARIANT() und folgende
//                      CHECK_INVARIANT     Methode invariant() und folgende
//                       CHECK_ENSURE       Nachbedingungen und folgende
//                        CHECK_REQUIRE     Vorgedingungen
//                         CHECK_NO
#endif
#include "eiffel.h"
#include "nana.h"



#include <string>
#include <vector>
#include <exception>

using namespace std;

// ------------------------------------------------------------------

class Date
{
private: int day;
private: int month;
private: int year;
```

```
public:
  // ...
};

class Customer_ID
{
  // ...
};

class Agent
{
  // ...
};

class Customer_Manager;

// ----------------------------------------------------------------

        class Customer
        {
                private: Customer_ID id;
                private: string name;
                private: string address;
                private: Date date_of_birth;
                private: Agent managing_aget;
                private: Date last_contact;

                private: Customer_Manager* customer_manager;
        };


// ----------------------------------------------------------------

       class basic_Customer_details
        {
                private: Customer_ID id;
                private: string name;
                private: string address;
                private: Date date_of_birth;

                public: Customer_ID get_id() const;
                public: string get_name() const;
                public: string get_address() const;
                public: Date get_date_of_bith() const;
```

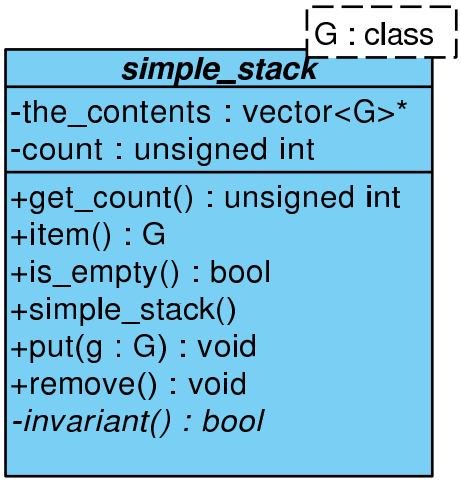
```
        };

// ------------------------------------------------------------

        class Customer_Manager
        {
                private: int count;
                private: virtual bool invariant() const;

                private: vector<Customer*> customer;

                public: int get_count() const;
                public: bool id_active(Customer_ID an_id) const;
                public: void add(const basic_Customer_details& a_customer);
                public: string name_for(Customer_ID an_id) const;
                public: void set_name(Customer_ID an_id, const string& a_name);
        };

bool Customer_Manager::invariant() const
{
  return get_count() >= 0;
};


// Anzahl der Kunden, die vom Custom_Manager verwaltet werden
//
int Customer_Manager::get_count() const
DO
        throw "Not yet implemented";
END;

// Existiert ein Kunde mit der Kennung an_id?
//
bool Customer_Manager::id_active(Customer_ID an_id) const
DO
        throw "Not yet implemented";
END;

// Fuege a_customer der Customer_Manager-Datenbasis hinzu
//
void Customer_Manager::add(const basic_Customer_details& a_customer)
DO
    REQUIRE(!id_active(a_customer.get_id()));
    ID(int old_count = get_count());
```

```
        throw "Not yet implemented";
    ENSURE(get_count() == old_count + 1 );
    ENSURE(id_active(a_customer.get_id()));
END;

// was ist der Name des mit an_id gekennzeichneter Kunden?
//
string Customer_Manager::name_for(Customer_ID an_id) const
DO
    REQUIRE(id_active(an_id));
        throw "Not yet implemented";
END;

// Setze/Ändere den Namen des mit an_id gekennzeichneten Kunden auf a_name
//
void Customer_Manager::set_name(Customer_ID an_id, const string& a_name)
DO
    REQUIRE(id_active(an_id));
        throw "Not yet implemented";
    ENSURE(name_for(an_id) == a_name);
END;



// ---------------------------------------------------------------------

int main(){

  exit (0);
}
```

## A.2 Elementary Principles of Design by Contract

### A.2.1 First Trial

Visual Paradigm for UML Standard Edition(University




```
//
// simple_stack0.cc
//
// g++ -g -o simple_stack0 simple_stack0.cc -I$HOME/include -L$HOME/lib -lnana
//
//
// eventuell mit
//                   -DWITHOUT_NANA
// und/oder
//                   -DNDEBUG
//
// und/oder
//                   -DEIFFEL_CHECK=CHECK_....
//
// ------------------------------------------------------------------
// evtl. myexcept.o mit angeben
// ------------------------------------------------------------------
//
// oder: nana-c++lg simple_stack0.cc
//


#define EIFFEL_DOEND
#ifndef EIFFEL_CHECK
#define EIFFEL_CHECK  CHECK_ALL
//                                              Makros CHECK() und folgende
```

```
//                    CHECK_LOOP              Makros INVARIANT() und folgende
//                     CHECK_INVARIANT        Methode invariant() und folgende
//                      CHECK_ENSURE          Nachbedingungen und folgende
//                       CHECK_REQUIRE        Vorgedingungen
//                        CHECK_NO
#endif


#include <iostream>
#include <vector>

#include <eiffel.h>
#include <nana.h>


using namespace std;


///////////////// class declaration ////////////////////////

template<class G>
class simple_stack{

private:

  vector<G>* the_contents;
  unsigned int count;

public:

  /////////// basic queries:

  unsigned int get_count() const;    // number of items in stack

  G item() const;                    // get top item

  /////////// class invariant:

private:
  virtual bool invariant() const;
public:

  /////////// derived queries:
```

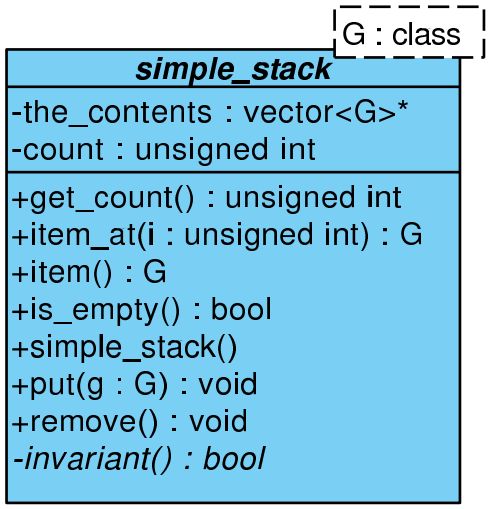
```
  bool is_empty() const;

  /////////// constructors:

  simple_stack();

  /////////// (pure) modificators:

  void put(G g);                  // Push 'g' onto the stack

  void remove();                  // delete the top item

};

///////////////// class definition ///////////////////////

/////////// basic queries:

template<class G>
unsigned int simple_stack<G>::get_count() const{
    return count;
};

template<class G>
G simple_stack<G>::item() const{         // the item on the top of stack
  REQUIRE( /* stack not empty */ get_count() > 0 );
  return the_contents->at(count-1);
};

/////////// class invariant:

template<class G>
bool simple_stack<G>::invariant() const {
  return (count >= 0) && (the_contents != 0);
};
/////////// derived queries:



template<class G>
bool simple_stack<G>::is_empty() const{  // does the stack contain no items?
  bool Result = (get_count() == 0);
  ENSURE( /* consistend with count */       Result == (get_count() == 0) );
  return Result;
```

```
};

////////////// constructors:

template<class G>
simple_stack<G>::simple_stack(){
  count = 0;
  the_contents = new vector<G>(100);

  ENSURE(/* stack is empty */       (get_count() == 0));
  ENSURE(invariant());
};

////////////// (pure) modificators:

template<class G>
void simple_stack<G>::put(G g)     // Push 'g' onto the stack
DO
  ID(int count_old = get_count());
  count++;
  the_contents->at(count-1) = g;
  ENSURE(/* count incremented */     get_count() == count_old + 1 );
  ENSURE(/* g on top */              item() == g );
END;

template<class G>
void simple_stack<G>::remove()     // delete the top item;
DO
  ID(int count_old = get_count());
  REQUIRE(/* stack not empty */      get_count() > 0 );
  count--;
  ENSURE(/* count decremented */     get_count() == count_old - 1 );
END;


///////////////// class test main() program ///////////////

int main(){

  cout << "\nTest der Klasse simple_stack: ----------------" << endl;

  simple_stack<long> s;

  s.put(10);
```

```
  cout << s.item() << endl;
  s.put(20);
  cout << s.item() << endl;
  s.put(30);
  std::cout << s.item() << endl;

  std::cout << endl;

  // Exercise 'remove'
  cout << s.item() << endl;
  s.remove();
  cout << s.item() << endl;
  s.remove();
  cout << s.item() << endl;

  // Exercise contract violation
  s.remove();
  cout << s.item() << endl;

  cout << "------------------ Testende ------------------" << endl << endl;

}
```

### A.2.2 Redesign

Visual Paradigm for UML Standard Edition(University of '




```
//
// simple_stack.cc
//
```

```
// g++ -g -o simple_stack simple_stack.cc -I$HOME/include -L$HOME/lib -lnana
//
//
// eventuell mit
//                   -DWITHOUT_NANA
// und/oder
//                   -DNDEBUG
//
// und/oder
//                   -DEIFFEL_CHECK=CHECK_....
//
// ------------------------------------------------------------------------
// evtl. myexcept.o mit angeben
// ------------------------------------------------------------------------
//
// oder: nana-c++lg simple_stack0.cc
//


#define EIFFEL_DOEND
#ifndef EIFFEL_CHECK
#define EIFFEL_CHECK  CHECK_ALL
//                                          Makros CHECK() und folgende
//                    CHECK_LOOP            Makros INVARIANT() und folgende
//                     CHECK_INVARIANT      Methode invariant() und folgende
//                      CHECK_ENSURE        Nachbedingungen und folgende
//                       CHECK_REQUIRE      Vorgedingungen
//                        CHECK_NO
#endif

#include <iostream>
#include <vector>

#include <eiffel.h>
#include <nana.h>


using namespace std;

///////////////// class declaration ///////////////////////

template<class G>
class simple_stack{
```

```
private:

  vector<G>* the_contents;
  unsigned int count;

public:

  /////////// basic queries:

  unsigned int get_count() const;    // number of items in stack

  G item_at(unsigned int i) const;

  /////////// class invariant:

private:
  virtual bool invariant() const;

public:

  /////////// derived queries:

  G item() const;                        // the item on the top of stack

  bool is_empty() const;                 // does the stack contain no items?

  /////////// constructors:

  simple_stack();

  /////////// (pure) modificators:

  void put(G g);                    // Push 'g' onto the stack

  void remove();                   // delete the top item;

};

///////////////// class definition ///////////////////////

/////////// basic queries:

template<class G>
G simple_stack<G>::item_at(unsigned int i) const {
```

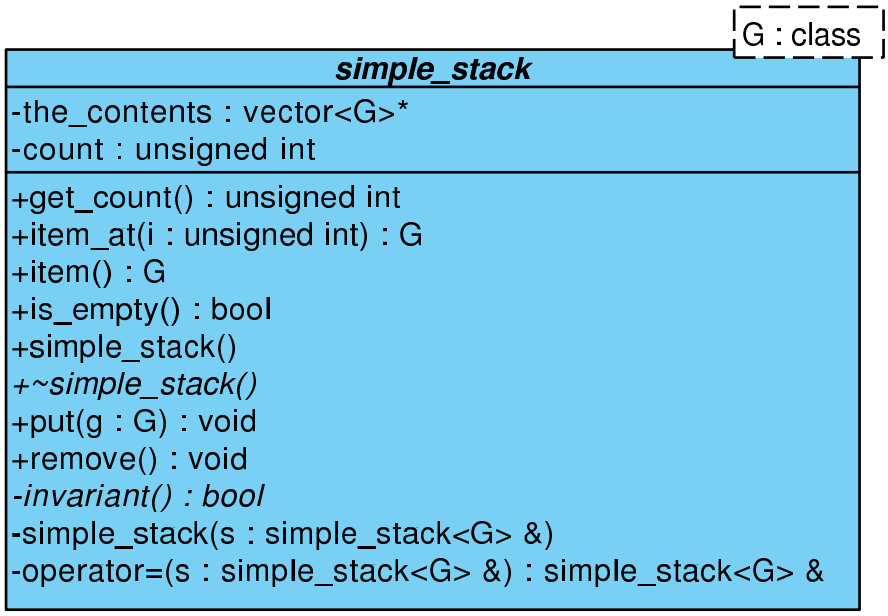
```
  REQUIRE( /* big enough */    i >= 1);
  REQUIRE( /* small enough */  i <= get_count() );

  return the_contents->at(i-1);
};

template<class G>
unsigned int simple_stack<G>::get_count() const{
    return count;
};

////////////// class invariant:

template<class G>
bool simple_stack<G>::invariant() const {
  return (get_count() >= 0) && (the_contents != 0);
};
////////////// derived queries:

template<class G>
G simple_stack<G>::item() const{         // the item on the top of stack
  REQUIRE( /* stack not empty */ get_count() > 0 );

  G result = the_contents->at(count-1);
  ENSURE( /* consistend with item_at() */   result == item_at(get_count()) );
  return result;
};

template<class G>
bool simple_stack<G>::is_empty() const{  // does the stack contain no items?

  bool result = (count == 0);
  ENSURE( /* consistend with count */       result == (get_count() == 0) );
  return result;

};

////////////// constructors:

template<class G>
simple_stack<G>::simple_stack(){

  count = 0;
  the_contents = new vector<G>(100);
```

```
  ENSURE(/* stack is empty */       (get_count() == 0));
  ENSURE(invariant());
};

////////////// (pure) modificators:

template<class G>
void simple_stack<G>::put(G g)     // Push 'g' onto the stack
DO
  ID(unsigned int count_old = get_count());
  count++;
  the_contents->at(count-1) = g;
  ENSURE(/* count incremented */     get_count() == count_old + 1 );
  ENSURE(/* g on top */              item_at(get_count()) == g );
END;

template<class G>
void simple_stack<G>::remove()     // delete the top item;
DO
  ID(unsigned int count_old = get_count());
  REQUIRE(/* stack not empty */      get_count() > 0 );
  count--;
  ENSURE( /* count decremented */        get_count() == count_old - 1 );
END;

///////////////// class test main() program ///////////////

int main(){

  cout << "\nTest der Klasse simple_stack: ----------------" << endl;

  simple_stack<long> s;

  s.put(10);
  cout << s.item() << endl;
  s.put(20);
  cout << s.item() << endl;
  s.put(30);
  cout << s.item() << endl;

  cout << endl;

  // Exercise 'remove'
```

```
  cout << s.item() << endl;
  s.remove();
  cout << s.item() << endl;
  s.remove();
  cout << s.item() << endl;

  // Exercise contract violation
  s.remove();
  cout << s.item() << endl;

  cout << "------------------ Testende ------------------" << endl << endl;

}
```

### A.2.3 Destruktor, Kopierkonstruktor und Wertzuweisung

Visual Paradigm for UML Standard Edition(University of Wuppertal)




```
...
#define EIFFEL_DOEND
#define EIFFEL_CHECK  CHECK_ALL
...
#include <iostream>
#include <vector>
#include <eiffel.h>
#include <nana.h>
...
class simple_stack{
```

```
private:

  vector<G>* the_contents;
  unsigned int count;

public:

  /////////// basic queries:

  unsigned int get_count() const;  // number of items in stack

  G item_at(unsigned int i) const;

  /////////// class invariant:

private:
  virtual bool invariant() const;
public:

  /////////// derived queries:

  G item() const;                          // the item on the top of stack

  bool is_empty() const;                   // does the stack contain no items?

  /////////// constructors:

  simple_stack();

  virtual ~simple_stack();                 // notwendig wegen new in Konstruktor

private:                                  // default copy-Konstruktor
                                          // und default operator=
                                          // fragwuerdig

  simple_stack<G>(const simple_stack<G>& s);
  simple_stack<G>& operator=(const simple_stack<G>& s);

public:

/////////// (pure) modificators:

void put(G g);                    // Push 'g' onto the stack
```

```
void remove();                  // delete the top item;

};
...
template<class G>
G simple_stack<G>::item_at(unsigned int i) const {
  REQUIRE( /* big enough */    i >= 1);
  REQUIRE( /* small enough */  i <= get_count() );
  ...
};
...
template<class G>
unsigned int simple_stack<G>::get_count() const{
    ...
};
...
template<class G>
bool simple_stack<G>::invariant() const {
  ...
};
...
template<class G>
G simple_stack<G>::item() const{         // the item on the top of stack
  REQUIRE( /* stack not empty */ get_count() > 0 );
  ...
  ENSURE( /* consistend with item_at() */   result == item_at(get_count()) );
  ...
};
...
template<class G>
bool simple_stack<G>::is_empty() const{  // does the stack contain no items?
  ...
  ENSURE( /* consistend with count */       result == (get_count() == 0) );
  ...
};
...
template<class G>
simple_stack<G>::simple_stack(){
  ...
  ENSURE(/* stack is empty */       (get_count() == 0));
  ENSURE(invariant());
};
...
```

```
template<class G>
simple_stack<G>::~simple_stack<G>(){
  REQUIRE (/* pointer not null */ the_contents != 0);
  ...
};
...
template<class G>
void simple_stack<G>::put(G g)     // Push 'g' onto the stack
DO
  ID(int count_old = get_count());
  ID(vector<G> old(*the_contents));
  ...
  ENSURE(/* count incremented */     get_count() == count_old + 1 );
  ENSURE(/* g on top */              item_at(get_count()) == g );
  ENSURE(/* old contents unchanged */
                A(int k=1, k<get_count(), k++, item_at(k)== old.at(k-1) ));
  ...
END;
...
template<class G>
void simple_stack<G>::remove()     // delete the top item;
DO
  REQUIRE(/* stack not empty */      get_count() > 0 );
  ID(int count_old = get_count());
  ID(vector<G> old(*the_contents));
  ...
  ENSURE( /* count decremented */        get_count() == count_old - 1 );
  ENSURE( /* consistancy with item() */  item_at(get_count()) == old.at(count_old-2) );
  ENSURE( /* old contents unchanged */
                A(int k=1, k<get_count(), k++, item_at(k)== old.at(k-1) ));
END;
...
int main(){
...
s.put(10);
cout << s.item() << endl;
s.put(20);
cout << s.item() << endl;
s.put(30);
cout << s.item() << endl;

cout << endl;

// Exercise 'remove'
```

```
cout << s.item() << endl;
s.remove();
cout << s.item() << endl;
s.remove();
cout << s.item() << endl;

// Test copy-Konstruktor ...
//
// simple_stack<long> s2(s);
simple_stack<long> s3;
// s3 = s;

cout << "------------------ Testende ------------------" << endl << endl;

}
```

Voller Code: simple_stack4.cc

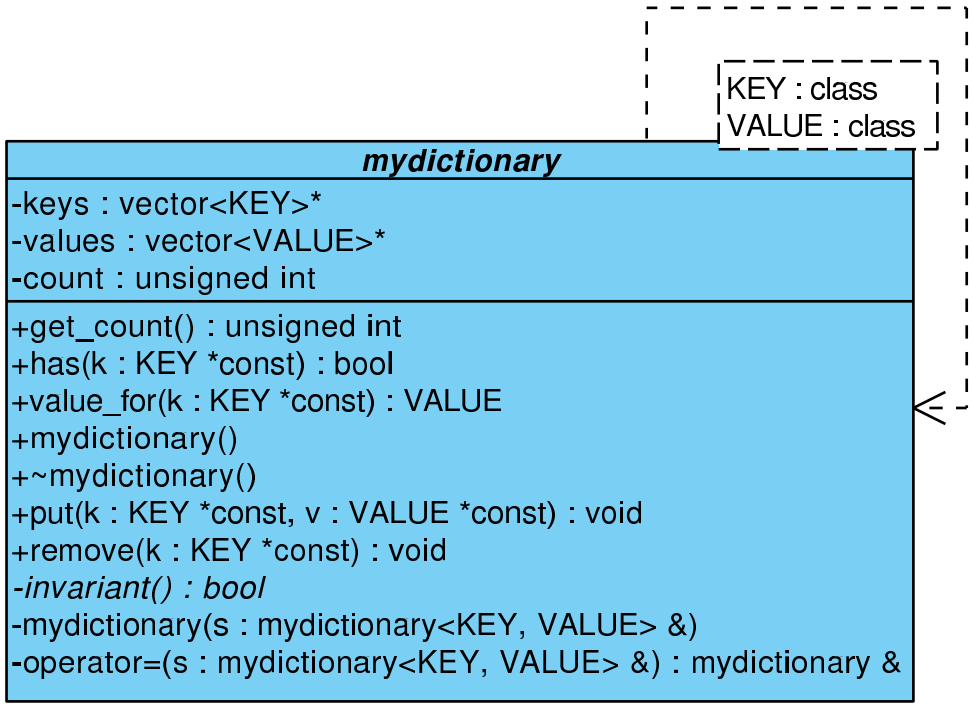
## A.3 Applying the Six Principles

**PbC-Regeln:**

- Vermeide statusändernde Methoden, die einen Wert liefern! (Observator **oder** Modifikator)
- Unterscheide grundlegende von abgeleiteten (redundanten) Observatoren.
- Schreibe für jeden abgeleiteten Observator eine Nachbedingung mit Hilfe der (aller) grundlegenden Observatoren.
- Schreibe für jeden Modifikator Nachbedingungen, die mit Hilfe der (aller) grundlegenden Observatoren den Inhalt des Klassenexemplars nach Methodenende in seiner Relation zum Exemplarinhalt bei Methodenbeginn **exakt** beschreiben. Nutze implizite (als Kommentar) oder explizite Frame-Bedingungen.
- Schreibe für alle Methoden Vorbedingungen (an Parameter bzw. Exemplarinhalt).
- Schreibe und benutze Invarianten, die gültige von ungültigen Exemplaren trennen.

### A.3.1 Design und Contracts






```
...
#define EIFFEL_DOEND
#define EIFFEL_CHECK  CHECK_ALL
```

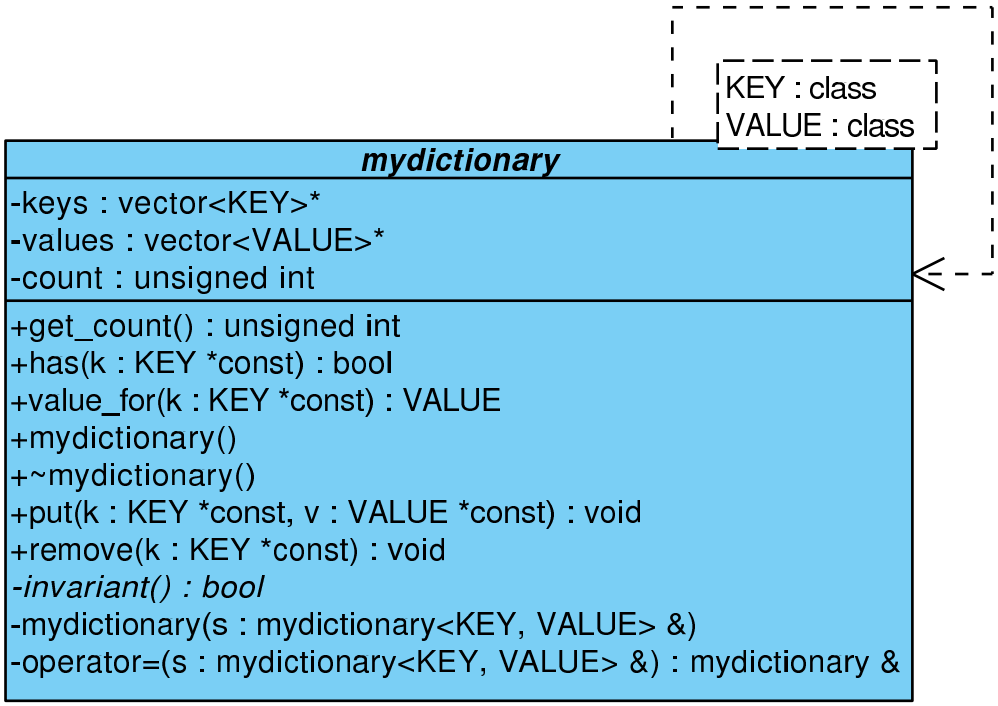
```
...
#include <iostream>
#include <vector>
#include <eiffel.h>
#include <nana.h>
...
class mydictionary{

  vector<KEY>* keys;
  vector<VALUE>* values;
  unsigned int count;

public:
  ///////////////// basic queries:

  unsigned int get_count() const;     // number of key/value-pairs in dict.

  bool has(const KEY *const k) const;              // key in dictionary?

  VALUE value_for(const KEY *const k) const;      // lookup value for key

  /////////////// class invariant:
private:
  virtual bool invariant() const;
public:
  /////////////// derived queries:

  // not yet necessary

  /////////////// constructors and destructors:

  mydictionary();

  ~mydictionary();

  /////////////// deactivate default copy constructor and operator=

private:
  mydictionary(const mydictionary<KEY, VALUE>& s);
  mydictionary& operator=(const mydictionary<KEY, VALUE>& s);
public:

  /////////////// (pure) modificators
```

```
  void put(const KEY *const k, const VALUE *const v);
                                              // put key/value-pair in dict.

  void remove(const KEY *const k);            // remove key/value-pair

};
...
template<class KEY, class VALUE>
unsigned int mydictionary<KEY, VALUE>::get_count() const{
  ...
};
...
template<class KEY, class VALUE>
bool mydictionary<KEY, VALUE>::has(const KEY *const k) const {
  REQUIRE( /* key exists */      k != 0);
  ...
  ENSURE( /* consistent with count */  (get_count() != 0) || ! result);
  ...
};
...
template<class KEY, class VALUE>
VALUE mydictionary<KEY, VALUE>::value_for(const KEY *const k) const{
  REQUIRE(/* key exists */       k != 0);
  REQUIRE(/* key in dict. */     has(k));
  ...
};
...
template <class KEY, class VALUE>
bool mydictionary<KEY, VALUE>::invariant() const{
  ...
};
...
template<class KEY, class VALUE>
mydictionary<KEY, VALUE>::mydictionary(){
  ...
  ENSURE(invariant());
};
...
template<class KEY, class VALUE>
mydictionary<KEY, VALUE>::~mydictionary(){
  REQUIRE(/* keys exist */      keys != 0);
  REQUIRE(/* values exist */    values != 0);
  ...
};
```

```
...
template<class KEY, class VALUE>
void mydictionary<KEY, VALUE>::put(const KEY *const k, const VALUE *const v)
DO
  ID(unsigned int count_old=get_count());
  REQUIRE(/* key exists */         k != 0);
  REQUIRE(/* key not in dict. */   ! has(k));
  ...
  ENSURE(/* count incremented */   get_count() == count_old + 1);
  ENSURE(/* key in dict. */        has(k) );
  ENSURE(/* correct value */       value_for(k) == *v);
END;
...
template<class KEY, class VALUE>
void mydictionary<KEY, VALUE>::remove(const KEY *const k)
DO
  ID(unsigned int count_old = get_count());
  REQUIRE(/* key exists */      k != 0);
  REQUIRE(/* key in dict. */    has(k));
  ...
  ENSURE(/* count decremented */ get_count() == count_old - 1);
  ENSURE(/* key not in dict. */ ! has(k));
  ...
END;
...
int main(){
  ...
}
```

### A.3.2 Implementierung und Tests






```
// dictionary2.cc
//
// g++ -g -o dictionary2 dictionary2.cc -I$HOME/include -L$HOME/lib -lnana
//
//
// eventuell mit
//                   -DWITHOUT_NANA
// und/oder
//                   -DNDEBUG
//
// und/oder
//                   -DEIFFEL_CHECK=CHECK_....
//
// ------------------------------------------------------------------------
// evtl. myexcept.o mit angeben
// ------------------------------------------------------------------------
//
// oder: nana-c++lg simple_stack4.cc
//


#define EIFFEL_DOEND
```

```
#ifndef EIFFEL_CHECK
#define EIFFEL_CHECK  CHECK_ALL
//                                          Makros CHECK() und folgende
//                    CHECK_LOOP            Makros INVARIANT() und folgende
//                     CHECK_INVARIANT      Methode invariant() und folgende
//                      CHECK_ENSURE        Nachbedingungen und folgende
//                       CHECK_REQUIRE      Vorgedingungen
//                        CHECK_NO
#endif

#include <iostream>
#include <vector>

#include <eiffel.h>
#include <nana.h>


using namespace std;

///////////////// class declaration ////////////////////////

template<class KEY, class VALUE>
class mydictionary{

  vector<KEY>* keys;
  vector<VALUE>* values;
  unsigned int count;

public:
  ///////////////// basic queries:

  unsigned int get_count() const;     // number of key/value-pairs in dict.

  bool has(const KEY *const k) const;              // key in dictionary?

  VALUE value_for(const KEY *const k) const;      // lookup value for key

  /////////////// class invariant:
private:
  virtual bool invariant() const;
public:
  /////////////// derived queries:

  // not yet necessary
```

```
  /////////////// constructors and destructors:

  mydictionary();

  ~mydictionary();

  /////////////// deactivate default copy constructor and operator=

private:
  mydictionary(const mydictionary<KEY, VALUE>& s);
  mydictionary& operator=(const mydictionary<KEY, VALUE>& s);
public:

  /////////////// (pure) modificators

  void put(const KEY *const k, const VALUE *const v);
                                          // put key/value-pair in dict.

  void remove(const KEY *const k);        // remove key/value-pair

};

///////////////// class definition ////////////////////////

///////////////// basic queries:

template<class KEY, class VALUE>
unsigned int mydictionary<KEY, VALUE>::get_count() const{
  return count;
};

template<class KEY, class VALUE>
bool mydictionary<KEY, VALUE>::has(const KEY *const k) const {

  REQUIRE( /* key exists */      k != 0);

  unsigned int i = 0;
  do {
    i++;
  }while((i <= count) && (keys->at(i-1) != *k) );

  bool result = (i <= count) && (keys->at(i-1) == *k);
  ENSURE( /* consistent with count */  (get_count() != 0) || ! result);
```

```
  return result;
};

template<class KEY, class VALUE>
VALUE mydictionary<KEY, VALUE>::value_for(const KEY *const k) const{
  REQUIRE(/* key exists */       k != 0);
  REQUIRE(/* key in dict. */     has(k));

  unsigned int i = 0;
  do {
    i++;
  }while(keys->at(i-1) != *k);
  return values->at(i-1);

};

////////////// class invariant:

template <class KEY, class VALUE>
bool mydictionary<KEY, VALUE>::invariant() const{

  return  (get_count() >= 0) && (keys != 0) && (values != 0);

};

/////////////// constructors and destructors:

template<class KEY, class VALUE>
mydictionary<KEY, VALUE>::mydictionary(){

  count = 0;
  keys = new vector<KEY>(100);
  values = new vector<VALUE>(100);
  ENSURE(invariant());
};

template<class KEY, class VALUE>
mydictionary<KEY, VALUE>::~mydictionary(){

  REQUIRE(/* keys exist */      keys != 0);
  REQUIRE(/* values exist */    values != 0);
  delete keys;
  delete values;
};
```

```
/////////////// (pure) modificators

template<class KEY, class VALUE>
void mydictionary<KEY, VALUE>::put(const KEY *const k, const VALUE *const v)
DO
  ID( unsigned int count_old=get_count());
  REQUIRE(/* key exists */         k != 0);
  REQUIRE(/* key not in dict. */   ! has(k));

  count++;
  keys->at(count-1) = *k;
  values->at(count-1) = *v;

  ENSURE(/* count incremented */   get_count() == count_old + 1);
  ENSURE(/* key in dict. */        has(k) );
  ENSURE(/* correct value */       value_for(k) == *v);
END;

template<class KEY, class VALUE>
void mydictionary<KEY, VALUE>::remove(const KEY *const k)
DO
  ID(unsigned int count_old = get_count());
  REQUIRE(/* key exists */      k != 0);
  REQUIRE(/* key in dict. */    has(k));

  unsigned int i = 0;
  do {
    i++;
  }while(keys->at(i-1) != *k);
  CHECK(i <= count);
  if (i < count){
    keys->at(i-1) = keys->at(count-1);
    values->at(i-1) = keys->at(count-1);
  };
  count--;

  ENSURE(/* count decremented */  get_count() == count_old - 1);
  ENSURE(/* key not in dict. */ ! has(k));
  // REQUIRE precondition for value_for() is false for k
END;

/////////////// class test main() program ///////////////
```

```
int main(){

  cout << "Test der Klasse dictionary: ----------------" << endl;

  mydictionary<string, string> d;

  string k("Denver"); string v("Colorado");
  d.put(&k, &v);
  k = "London"; v = "Ontario";
  d.put(&k, &v);
  k = "Austin"; v = "Texas";
  d.put(&k, &v);
  k = "Boston"; v = "Massachusetts";
  d.put(&k, &v);
  k = "Mobile"; v = "Alabama";
  d.put(&k, &v);

  cout << endl << "List of 5 pairs:" << endl;
  {
   string k1("Denver");
   if (d.has(&k1)) cout << k1 << " " << d.value_for(&k1) << endl;
   string k2("London");
   if (d.has(&k2)) cout << k2 << " " << d.value_for(&k2) << endl;
   string k3("Austin");
   if (d.has(&k3)) cout << k3 << " " << d.value_for(&k3) << endl;
   string k4("Boston");
   if (d.has(&k4)) cout << k4 << " " << d.value_for(&k4) << endl;
   string k5("Mobile");
   if (d.has(&k5)) cout << k5 << " " << d.value_for(&k5) << endl;
  };

  string l2("Mobile");
  d.remove(&l2);
  cout << endl << "List of " << d.get_count() << " pairs, last removed:" << endl;
   {
   string k1("Denver");
   if (d.has(&k1)) cout << k1 << " " << d.value_for(&k1) << endl;
   string k2("London");
   if (d.has(&k2)) cout << k2 << " " << d.value_for(&k2) << endl;
   string k3("Austin");
   if (d.has(&k3)) cout << k3 << " " << d.value_for(&k3) << endl;
   string k4("Boston");
   if (d.has(&k4)) cout << k4 << " " << d.value_for(&k4) << endl;
   string k5("Mobile");
```

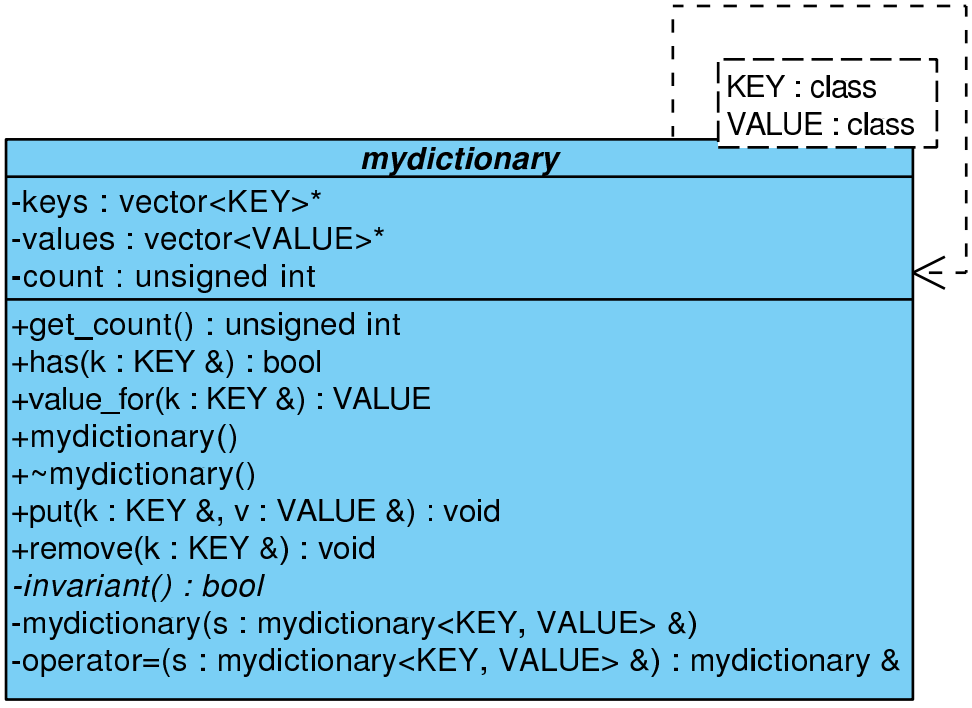
```
 if (d.has(&k5)) cout << k5 << " " << d.value_for(&k5) << endl;
};

string l1("Denver");
d.remove(&l1);
cout << endl << "List of " << d.get_count() << " pairs, 1st removed:" << endl;
 {
 string k1("Denver");
 if (d.has(&k1)) cout << k1 << " " << d.value_for(&k1) << endl;
 string k2("London");
 if (d.has(&k2)) cout << k2 << " " << d.value_for(&k2) << endl;
 string k3("Austin");
 if (d.has(&k3)) cout << k3 << " " << d.value_for(&k3) << endl;
 string k4("Boston");
 if (d.has(&k4)) cout << k4 << " " << d.value_for(&k4) << endl;
 string k5("Mobile");
 if (d.has(&k5)) cout << k5 << " " << d.value_for(&k5) << endl;
};

string l3("Austin");
d.remove(&l3);
cout << endl << "List of " << d.get_count() << " pairs, middle removed:" << endl;
{
 string k1("Denver");
 if (d.has(&k1)) cout << k1 << " " << d.value_for(&k1) << endl;
 string k2("London");
 if (d.has(&k2)) cout << k2 << " " << d.value_for(&k2) << endl;
 string k3("Austin");
 if (d.has(&k3)) cout << k3 << " " << d.value_for(&k3) << endl;
 string k4("Boston");
 if (d.has(&k4)) cout << k4 << " " << d.value_for(&k4) << endl;
 string k5("Mobile");
 if (d.has(&k5)) cout << k5 << " " << d.value_for(&k5) << endl;
};

// d.remove(&string("Austin"));

d.put(&k, &v);
cout << endl << "List of " << d.get_count() << " pairs after put:" << endl;
{
 string k1("Denver");
 if (d.has(&k1)) cout << k1 << " " << d.value_for(&k1) << endl;
 string k2("London");
 if (d.has(&k2)) cout << k2 << " " << d.value_for(&k2) << endl;
```

```
   string k3("Austin");
   if (d.has(&k3)) cout << k3 << " " << d.value_for(&k3) << endl;
   string k4("Boston");
   if (d.has(&k4)) cout << k4 << " " << d.value_for(&k4) << endl;
   string k5("Mobile");
   if (d.has(&k5)) cout << k5 << " " << d.value_for(&k5) << endl;
  };

  //  d.put(&k, &v);

  cout << "------------------ Testende ------------------" << endl;

}
```

Zum Download: dictionary2.cc

### A.3.3 konstante Referenzparameter/private Hilfsmethoden für die Spezifikation/old-Wert durch Kopie in Form eines geeigneten STL-Container-Exemplars






```
...
#define EIFFEL_DOEND
#define EIFFEL_CHECK  CHECK_ALL
```

```
...
#include <iostream>
#include <vector>
#include <set>
#include <eiffel.h>
#include <nana.h>
...
// private Spezifikations-Hilfsmethoden: fuer STL-Container
template <class T>
set<T> operator+(const set<T>& s, const T& e){
    ...
};
...
template <class T>
set<T> operator-(const set<T>& s, const T& e){
    ...
};
...
class mydictionary{

  vector<KEY>* keys;
  vector<VALUE>* values;
  unsigned int count;

public:
  ///////////////// basic queries:

  unsigned int get_count() const;     // number of key/value-pairs in dict.

  bool has(const KEY& k) const;              // key in dictionary?

  VALUE value_for(const KEY& k) const;      // lookup value for key

  /////////////// class invariant:
private:
  virtual bool invariant() const;
public:

  /////////////// derived queries:
  // not yet necessary

  /////////////// constructors and destructors:

  mydictionary();
```

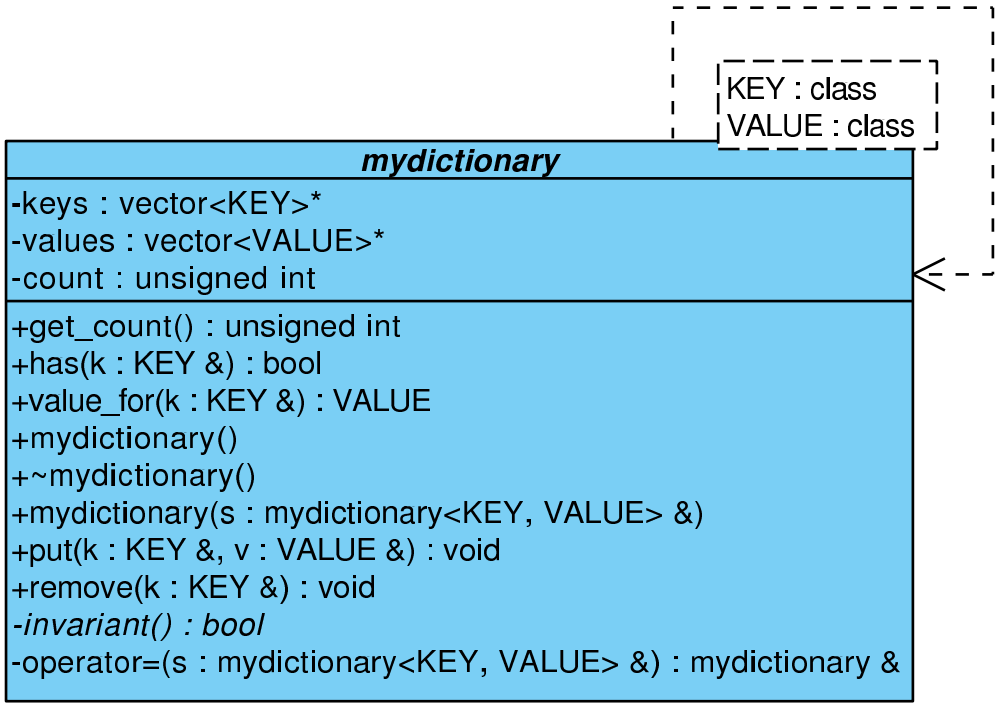
```
  ~mydictionary();

  /////////////// deactivate default copy constructor and operator=

private:
  mydictionary(const mydictionary<KEY, VALUE>& s);
  mydictionary& operator=(const mydictionary<KEY, VALUE>& s);
public:

  /////////////// (pure) modificators

  void put(const KEY& k, const VALUE& v);
                                              // put key/value-pair in dict.

  void remove(const KEY& k);                  // remove key/value-pair

};
...

/// basic queries:

template<class KEY, class VALUE>
unsigned int mydictionary<KEY, VALUE>::get_count() const{
  ...
};
...
template<class KEY, class VALUE>
bool mydictionary<KEY, VALUE>::has(const KEY& k) const {
  ...
  ENSURE( /* consistent with count */  (get_count() != 0) || ! result);
  ...
};
...
template<class KEY, class VALUE>
VALUE mydictionary<KEY, VALUE>::value_for(const KEY& k) const{
  REQUIRE(/* key in dict. */     has(k));
  ...
};
...

/// Invariante:

template <class KEY, class VALUE>
```

```
bool mydictionary<KEY, VALUE>::invariant() const{
  ...
};
...
/// Konstruktor/Destruktor:

template<class KEY, class VALUE>
mydictionary<KEY, VALUE>::mydictionary(){
  ...
  ENSURE(invariant());
  ENSURE(count == 0);
};
...
template<class KEY, class VALUE>
mydictionary<KEY, VALUE>::~mydictionary(){
  REQUIRE(/* keys exist */      keys != 0);
  REQUIRE(/* values exist */    values != 0);
  ...
};
...
/// Modifikatoren:

template<class KEY, class VALUE>
void mydictionary<KEY, VALUE>::put(const KEY& k, const VALUE& v)
DO
  REQUIRE(/* key not in dict. */   ! has(k));
  ID(unsigned int count_old=get_count());
  ID(set<KEY> old_keys(keys->begin(), keys->begin()+count_old));
  ...
  ENSURE(/* count incremented */    get_count() == count_old + 1);
  ENSURE(/* key in dict. */         has(k) );
  ENSURE(/* correct value */        value_for(k) == v);
  ID(set<KEY> new_keys(keys->begin(),keys->begin()+count));
  ENSURE(old_keys + k == new_keys);
  ...
END;
...
template<class KEY, class VALUE>
void mydictionary<KEY, VALUE>::remove(const KEY& k)
DO
  REQUIRE(/* key in dict. */    has(k));
  ID(unsigned int count_old = get_count());
  ID(set<KEY> old_keys(keys->begin(), keys->begin()+count_old));
  ...
```

```
  ENSURE(/* count decremented */  get_count() == count_old - 1);
  ENSURE(/* key not in dict. */   ! has(k));
  ...
  ID(set<KEY> new_keys(keys->begin(),keys->begin()+count));
  ENSURE(old_keys - k == new_keys);
  ...
END;
...
int main(){
  ...
}
```

### A.3.4 old-Wert durch den Kopierkonstruktor






Mit Hilfe des Kopierkonstruktors

```
...
template<class KEY, class VALUE>
mydictionary<KEY, VALUE>::mydictionary(const mydictionary<KEY, VALUE>& s){
    count = s.count;
    keys = new vector<KEY>(100);
    values = new vector<VALUE>(100);
    for (int i=1; i<=count; i++){
        keys->at(i-1) = s.keys->at(i-1);
        values->at(i-1) = s.values->at(i-1);
```

```
    };
    ENSURE(count == s.count);
    ENSURE(A(int i=1, i<=count, i++, keys->at(i-1) == s.keys->at(i-1)));
    ENSURE(A(int i=1, i<=count, i++, values->at(i-1) == s.values->at(i-1)));
    ENSURE(invariant());
};
...
```

können Sie den alten Exemplarwert vollständig in einem Stück als konstante Variable memorieren und in den Nachbedingungen aller Modifikatoren benutzen.

AUFGABE: Ändern Sie dictionary4.cc so ab, dass put() und remove() so spezifiziert wird!

Es ist noch unschön, dass die Nachbedingungen des Kopierkonstruktors auf die Implemantierungsdetails Bezug nehmen. Deshalb:

### A.3.5 Redesign

Ein nächster Schritt sollte die Einführung eines neuen grundlegenden Observators `set<KEY> keys()` sein. Führen Sie die notwendigen Änderungen durch!

Alternativ könnte man einen Iterator in der neuen Containerklasse mydictionary implementieren (siehe folgender Abschnitt).

## A.4 Immutable Lists

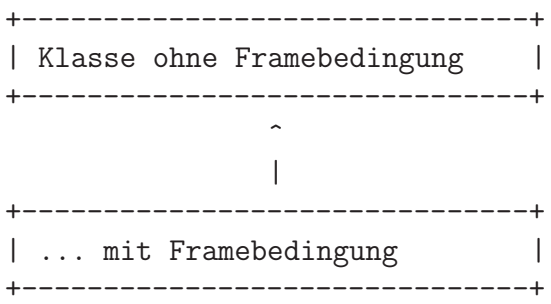
... entfällt in C++ wegen der Existenz der STL

## A.5 Using Immutable Lists

... entfällt in C++ wegen der Existenz der STL

## Leitlinien:

- Nutze technische Einschränkungen, wo immer erforderlich: z.B. Zeiger != 0, nichtleere Container, nichtidentische Exemplare, ...
- In Vorbedingungen genutzte Observatoren sollten effizient berechnet werden. Notfalls führe neue effiziente abgeleitete Observatoren ein und benutze sie in den Vorbedingungen. Die neuen effizienten Observatoren benötigen Nachbedingungen, die ihre Konsistenz zu den grundlegenden Observatoren sicherstellen.
- Attribute haben keine Nachbedingungen. Benutze deshalb die Klasseninvariante für Contracts (oder Nachbedingungen von get-Methoden).
- Nachbedingungen von virtuellen Methoden sollten die Form
  `ENSURE (!Vorbedingung || Nachbedingung)` haben;
  Invarianten sollten `protected` als `virtual bool invariant() const` deklariert werden.
- Nutze die Vererbung:



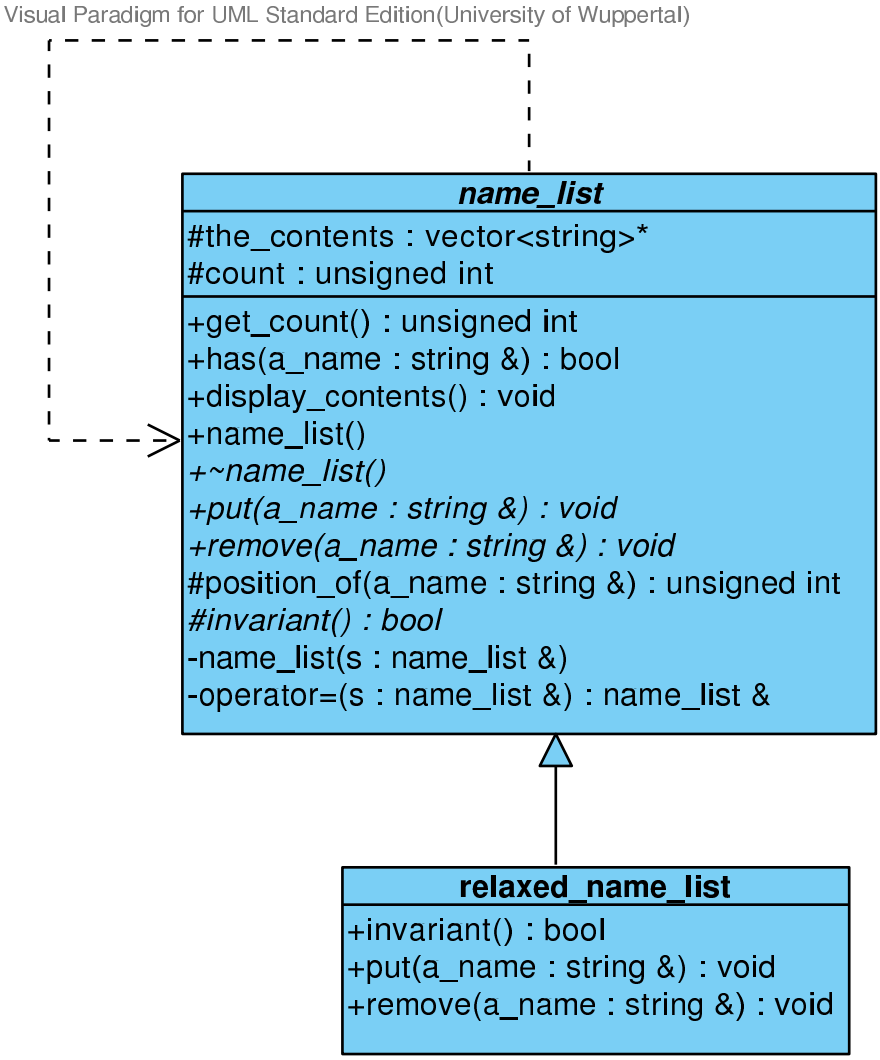

  um dem wiederverwendenden Nutzer die Wahl zwischen der Verwendung der effizienten oberen oder sicheren unteren Klasse zu überlassen.
- Nutze die Vererbung analog z.B. für:

```
Klasse ohne Framebedingungen

Klasse mit Framebedingungen

... mit unzugänglichen (private) Observatoren,
    die weiter oben lediglich für die Contracts
    genutzt werden
```

```
... mit benutzerfreundlichen Methodenvarianten
    ohne Vorbedingungen

...
```

## A.6 Subcontracting in Design by Contract in Nana

### A.6.1 name_list-Design (Subcontracting)




```
...
#define EIFFEL_DOEND
#define EIFFEL_CHECK  CHECK_ALL
...
#include <iostream>
#include <vector>
```

```
#include <eiffel.h>
#include <nana.h>
...
class name_list{

protected:

  vector<string>* the_contents;
  unsigned int count;

  // private support function

  unsigned int position_of(const string& a_name) const;

public:

  /////////// basic queries:

  unsigned int get_count() const;        // number of items in stack

  bool has(const string& a_name) const;

  /////////// class invariant:

protected:
  virtual bool invariant() const;
public:

  /////////// derived queries:

  void display_contents() const;         // print contents of list to cout

  /////////// constructors:

  name_list();

  virtual ~name_list();              // notwendig wegen new in Konstruktor

private:                             // diable default methods

  name_list(const name_list& s);
  name_list& operator=(const name_list& s);

public:
```

```
////////////// (pure) modificators:

  virtual void put(const string& a_name);       // Push a_name into list

  virtual void remove(const string& a_name);    // delete a_name in list

};
...
unsigned int name_list::position_of(const string& a_name) const{
    ...
    ENSURE(/* a_name in list */
                                ((1<=result)&&(result<=get_count())&&
                                 (the_contents->at(result-1)==a_name)) ||
           /* otherwise: */
                                (0 == result)                          );
    ...
};
...
unsigned int name_list::get_count() const{    // number of items in stack
    ...
    ENSURE(result == count);
    ...
};
bool name_list::has(const string& a_name) const{
    ...
    ENSURE((get_count()>0) || !result);
    ...
};
...
bool name_list::invariant() const{
    ...
};
...
void name_list::display_contents() const{     // print contents of list to cout
    ...
};
...
name_list::name_list(): count(0), the_contents(new vector<string>(100)){
    ENSURE(invariant());
};
name_list::~name_list(){                  // notwendig wegen new in Konstruktor
    ...
};
```

```
...
void name_list::put(const string& a_name)    // Push a_name into list
DO
   ID(bool pre = !has(a_name));
   ID( unsigned int count_old = get_count());
   REQUIRE(/* name not in list */    pre );
   ...
   ENSURE(has(a_name));
   ENSURE((!pre) || (get_count() == count_old + 1));
   ...
END;
void name_list::remove(const string& a_name) // delete a_name in list
DO
   ID(bool pre(has(a_name)));
   ID( unsigned int count_old = get_count());
   REQUIRE(/* name in list */     has(a_name));
   ...
   ENSURE(! has(a_name));
   ENSURE((!pre) || (get_count() == count_old -1));
   ...
END;
...
class relaxed_name_list : public name_list{

public:

    ///////// invariant:

    virtual bool invariant() const;

    ///////// (pure) modificators: (redefined)

    virtual void put(const string& a_name);       // Push a_name into list

    virtual void remove(const string& a_name);    // delete a_name in list

};
...
bool relaxed_name_list::invariant() const{
    ...
};
...
void relaxed_name_list::put(const string& a_name)    // Push a_name into list
DO
```

```
   ID(bool pre_parent(!has(a_name)));
   ID(unsigned int count_old = get_count());
   REQUIRE(/* nothing */  true);       // pre_parent || has(a_name)
   ...
   ENSURE(has(a_name));
   ENSURE((!pre_parent) || (get_count() == count_old + 1));  // &&
   ENSURE(  pre_parent  || (get_count() == count_old));
   ...
END;
void relaxed_name_list::remove(const string& a_name) // delete a_name in list
DO
   ID(bool pre_parent(has(a_name)));
   ID(unsigned int count_old = get_count());
   REQUIRE(/* nothing */ true);        // pre_parent || !has(a_name)
   ...
   ENSURE(! has(a_name));
   ENSURE((!pre_parent) || (get_count() == count_old -1));  // &&
   ENSURE(  pre_parent  || (get_count() == count_old));
   ...
END;
...
int main(){
    ...
}
```

Zum Download: name_list.cc

### A.6.2 Implementierung und Tests

... der Contracts und der Prototypinstallation:

```
...
unsigned int name_list::position_of(const string& a_name) const{

    unsigned int index(1);
    unsigned int result;
    for(; (index<=count) && (the_contents->at(index-1)!=a_name); index++);
    if (index <= count)
        result = index;
    else
        result = 0;

    ENSURE(/* a_name in list */
```

```
                                ((1<=result)&&(result<=get_count())&&
                                 (the_contents->at(result-1)==a_name)) ||
          /* otherwise: */
                                (0 == result)                          );
    return result;
};
...
////////////// basic queries:

unsigned int name_list::get_count() const{    // number of items in stack

    unsigned int result = count;
    return result;
};

bool name_list::has(const string& a_name) const{

    bool result = (position_of(a_name) > 0);
    ENSURE((get_count()>0) || !result);   // consistency
    return result;
};
...

////////////// class invariant:

bool name_list::invariant() const{
    return (count >= 0) && (the_contents != 0);
};


...
////////////// derived queries:

void name_list::display_contents() const{     // print contents of list to cout

    cout << endl << "Anzahl der Listenelemente: " << count << endl;
    for (unsigned int i=1; i<=count; i++)
        cout << "  " << the_contents->at(i-1);
    cout << endl << endl;
    // ENSURE('Drucke alle Namen in Name_list');
};

////////////// constructors:

name_list::name_list(): count(0), the_contents(new vector<string>(100)){
```

```
    ENSURE(invariant());
};

name_list::~name_list(){                    // notwendig wegen new in Konstruktor
    delete the_contents;
};

////////////// (pure) modificators:

void name_list::put(const string& a_name)    // Push a_name into list
DO
   ID(bool pre(!has(a_name)));
   ID(unsigned int count_old = get_count());

   REQUIRE(/* name not in list */    pre);


   count++;
   the_contents->at(count-1) = a_name;

   ENSURE(has(a_name));
   ENSURE( (!pre) || (get_count() == count_old + 1));
   // ENSURE: Fuer alle s in list_old: has(s)
   // ENSURE: Fuer alle s in list:     (s == a_name) || s in list_old
END;
...
///////////////// class test main() program ///////////////

int main(){

    cout << "\nTest der Klasse name_list: ----------------" << endl;

    { name_list s;
    s.display_contents();

    s.put("Richard"); s.display_contents();
    //s.put("Richard"); s.display_contents(); // Test fuer pre-Verletzung
    s.put("Helen"); s.display_contents();
    s.put("Yu"); s.display_contents();
    s.put("Jim"); s.display_contents();
    s.put("Chen"); s.display_contents();
...
```

Zum Download: name_list2.cc

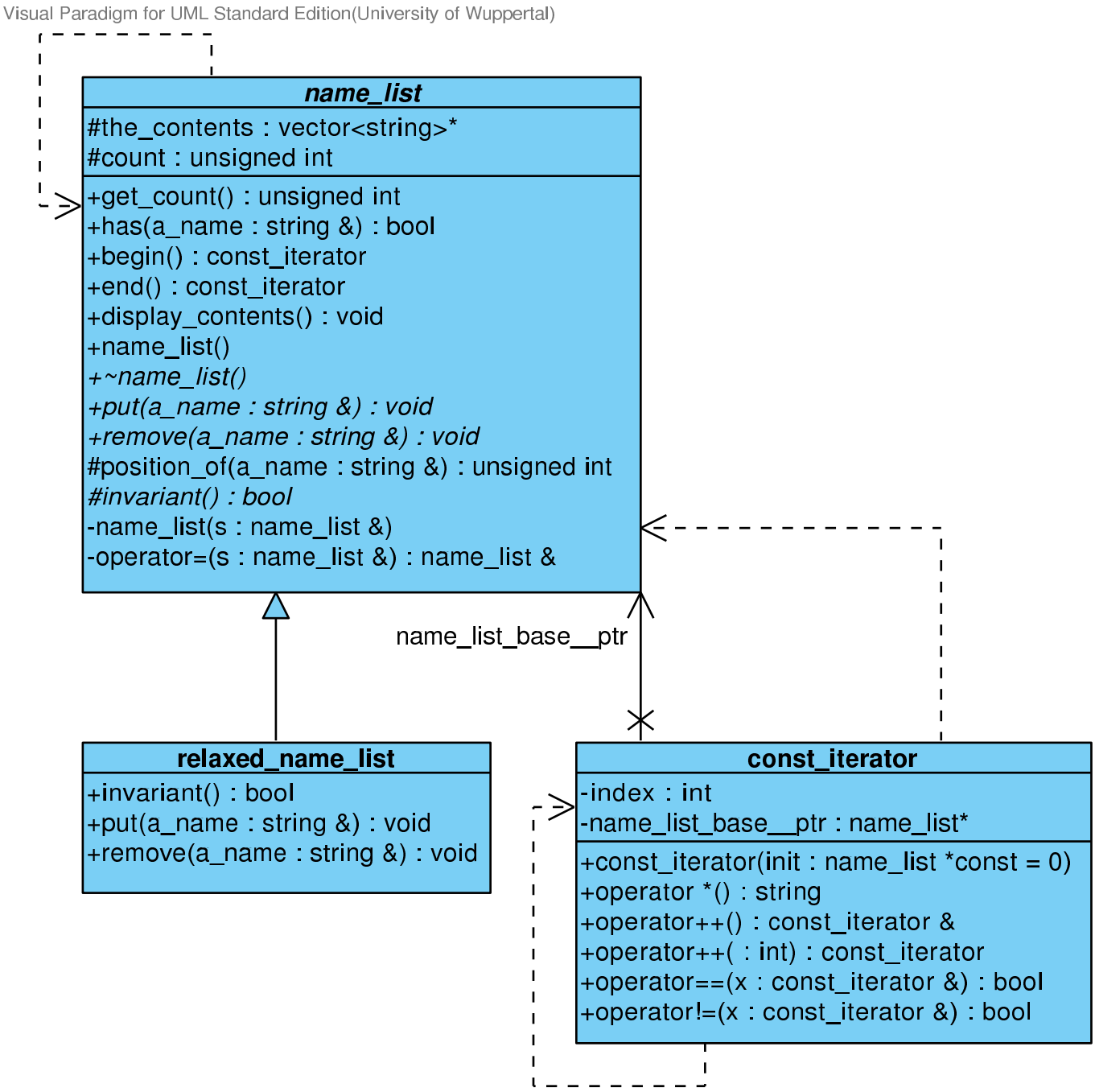
### A.6.3 Mit Frameregel

```
unsigned int name_list::position_of(const string& a_name) const{

    unsigned int index(1);
    unsigned int result;
    for(; (index<=count) && (the_contents->at(index-1)!=a_name); index++);
    if (index <= count)
        result = index;
    else
        result = 0;
    ENSURE(!(result < 0));
    ENSURE(!(result >get_count()));
    ID(set<string> values(the_contents->begin(),the_contents->begin()+get_count()));
    ENSURE(/* a_name in list */
                                ((1<=result)&&(result<=get_count())&&
                                 (the_contents->at(result-1)==a_name)) ||
           /* otherwise: */
               ( ((0 == result) && (values.find(a_name) == values.end())))
         );
    return result;
};
...
unsigned int name_list::get_count() const{    // number of items in stack

    unsigned int result = count;
    CHECK(result == count);
    return result;
};

bool name_list::has(const string& a_name) const{

    bool result = (position_of(a_name) > 0);
    ENSURE((get_count()>0) || !result);
    ID(set<string> values(the_contents->begin(),
                  the_contents->begin()+get_count()));
    CHECK(result == (values.find(a_name) != values.end( )));
    return result;
};
...
void name_list::put(const string& a_name)    // Push a_name into list
DO
   ID(bool pre);
   IS(pre = !has(a_name));
```

```
   // DS($pre_false = has(a_name));    // funktioniert nicht
   ID(set<string> values_old(the_contents->begin(),
                             the_contents->begin()+get_count()));
   REQUIRE(/* name not in list */    pre);
   DS($count_old = get_count());

   count++;
   the_contents->at(count-1) = a_name;

   ENSURE(has(a_name));
   DI( (!pre) || (this->get_count() == $count_old + 1));
   ID(set<string> values(the_contents->begin(),
                         the_contents->begin()+get_count()));
   IS(values_old.insert(a_name));    // stilistisch unschoen!
   ENSURE(values_old == values);
END;
...
```

Zum Download: name_list3.cc

### A.6.4 Mit Iterator-Methode (Design)




```
...
class name_list{
...
  /////////// Iteratoren:

  class const_iterator;
```

```
  friend class const_iterator;

  const_iterator begin() const;
  const_iterator end() const;
...
}
...
///////////////// class name_list::const_iterator /////////

class name_list::const_iterator{

    int index;
    const name_list* name_list_base__ptr;

public:

    const_iterator(const name_list* const init = 0);

    string operator*() const;        // Zugriff auf Elemet am Iterator-Ort
    const_iterator& operator++();    // Praefix-Inkrement
    const_iterator  operator++(int); // Postfix-Inkrement

    bool operator==(const const_iterator& x) const; // Vergleich von Iteratoren
    bool operator!=(const const_iterator& x) const;
};

/// contract for Iterator:  (as a basic query)

name_list::const_iterator::const_iterator(const name_list* const init){

    // ...
    // (*this)=="first" element of name_list (*init) if (init != 0),
    // this is an iterator not in any name_list if (init == 0)
};

name_list::const_iterator name_list::begin() const {
    // return ...
    // returns const_iterator pointing to "first" element of name_list
};

name_list::const_iterator name_list::end() const {
    // return ...
    // returns const_iterator denoting to be not any more in name_list
};
```

```
string name_list::const_iterator::operator*() const {
    // return ...
    // return element const_iterator is pointing to
};

name_list::const_iterator& name_list::const_iterator::operator++(){ // Praefix
   // return ...
   // increment position of const_iterator and return reference to this
   //       incremented const_iterator afterwards
};

name_list::const_iterator  name_list::const_iterator::operator++(int){//Postfix
    // return ...
    // return copy of const_iterator and as a side effect increment position
    // of the actual const_iterator
};

bool name_list::const_iterator::operator==(const name_list::const_iterator& x) const{
   // return ...
   // return if const_itrator and x point to the same element in the
   // same name_list
};

bool name_list::const_iterator::operator!=(const name_list::const_iterator& x) const{
    bool result; // = ...
    ENSURE(!((*this)==x));          // Konsistenzbedingung
};
...
```

Zum Download: name_list4.cc

### A.6.5 Implementierung der Iterator-Methode

```
...
/// contract for Iterator:  (as a basic query)

name_list::const_iterator::const_iterator(const name_list* const init){

    name_list_base_ptr = init;
    if ((name_list_base_ptr != 0)&&(name_list_base_ptr->count > 0)){
        index = 0;
    } else
        index = -1;
```

```
    if (index == -1) name_list_base_ptr = 0;
    // (*this)=="first" element of name_list (*init) if (init != 0),
    // this is an unique iterator not in any name_list if (init == 0)
};

name_list::const_iterator name_list::begin() const{
    return const_iterator(this);
    // returns const_iterator pointing to "first" element of name_list
};


name_list::const_iterator name_list::end() const{
    return const_iterator();
    // returns const_iterator denoting to be not any more in name_list
};

string name_list::const_iterator::operator*() const {
    REQUIRE(index != -1);
    return name_list_base_ptr->the_contents->at(index);
    // return element const_iterator is pointing to
    // name_list of const_iterators not changed
};

...
```

Zum Download: name_list5.cc

### A.6.6 Test des Iterators in display_contents() und main()

```
...
////////////// derived queries:

void name_list::display_contents() const{     // print contents of list to cout

    cout << endl << "Anzahl der Listenelemente: " << count << endl;
    for (name_list::const_iterator i = begin(); i != end(); i++)
        cout << "  " << *i;
    cout << endl << endl;
    // ENSURE('Drucke alle Namen in Name_list');
};
...
///////////////// class test main() program ///////////////

int main(){
```

```
    cout << "\nTest der Klasse name_list: ----------------" << endl;

    { name_list s;
    s.display_contents();

    s.put("Richard"); s.display_contents();
    // s.put("Richard"); s.display_contents(); // Test fuer pre-Verletzung
    s.put("Helen"); s.display_contents();
    s.put("Yu"); s.display_contents();
    s.put("Jim"); s.display_contents();
    s.put("Chen"); s.display_contents();
    s.put("Moira"); s.display_contents();
...
```

Zum Download: name_list6.cc

### A.6.7 Qstl.h bei Contracts und Klassen mit eigenen Iteratoren: Framebedingungen mit Hilfe eines Iterators

```
...

#include <eiffel.h>
#include <nana.h>
#include <Qstl.h>     //<<<////////////////////////////////////// NEU!

using namespace std;
...
unsigned int name_list::position_of(const string& a_name) const{

    unsigned int index(1);
    unsigned int result;
    for(; (index<=count) && (the_contents->at(index-1)!=a_name); index++);
    if (index <= count)
        result = index;
    else
        result = 0;

    ENSURE(/* a_name in list */
                                ((1<=result)&&(result<=get_count())&&
                                 (the_contents->at(result-1)==a_name)) ||
           /* otherwise: */
                               ((0 == result)&&(!EO(i,(*this),(*i)==a_name))));
    return result;
```

```
};
...
bool name_list::has(const string& a_name) const{

    bool result = (position_of(a_name) > 0);
    ENSURE((get_count()>0) || !result);              // Konsistenz
    ENSURE(result == EO(i, (*this), (*i)==a_name));  // Konsistenz
    return result;
};
...
////////////// (pure) modificators:

void name_list::put(const string& a_name)    // Push a_name into list
DO
   REQUIRE(/* name not in list */    !has(a_name));
   ID(set<string> contents_old(begin(),end()));
   ID(int count_old = get_count());
   ID(bool not_in_list = !has(a_name));

   count++;
   the_contents->at(count-1) = a_name;

   ENSURE(has(a_name));
   ENSURE( (!not_in_list) || (get_count() == count_old + 1));
   ID(set<string> contents(begin(),end()));
   IS(contents_old.insert(a_name));
   ENSURE(contents == contents_old);
END;
...
```

Zum Download: name_list7.cc

### A.6.8 Hilfsoperatoren für die STL

```
...
///////////////// Hilfsfunktionen ////////////////////////

template <class T>
set<T> operator+(const set<T>& s, const T& e){
    set<T> result(s);
    result.insert(e);
    return result;
};

template <class T>
set<T> operator-(const set<T>& s, const T& e){
    set<T> result(s);
    result.erase(e);
    return result;
};
...
void relaxed_name_list::remove(const string& a_name) // delete a_name in list
DO
   REQUIRE(/* nothing */ true);        // pre_parent || !has(a_name)
   ID(set<string> contents_old(begin(),end()));
   ID(int count_old = get_count());
   ID(bool pre_parent = has(a_name));

   if (has(a_name)){
      the_contents->at(position_of(a_name)-1) = the_contents->at(count-1);
      count--;
   };

   ENSURE(!has(a_name));
   ENSURE((!pre_parent) || (get_count() == count_old -1));  // &&
   ENSURE(  pre_parent  || (get_count() == count_old));
   // Menge der Eintraege in list == Menge der Eintraege in list_old ohne a_name
   ID(set<string> contents(begin(),end()));
   ENSURE(  pre_parent  || (contents == contents_old));;
   ENSURE((!pre_parent) || (contents == contents_old - a_name));
END;
...
```

Zum Download: name_list9.cc

## A.7 Neuformulierung: Regeln und Leitlinien für PbC in C++

1. Formuliere *grundlegende Observatoren*, die den Zustand eines Exemplars vollständig beschreiben können und eine *Klasseninvariante*, die gültige von ungültigen Exemplaren trennt. Falls grundlegende Observatoren mit Parametern existieren, so gib den zur vollständigen Exemplarbeschreibung nötigen Parameter-Wertebereich an. Im Contract sollte kein Bezug auf Implementierungsdetails sondern lediglich auf die grundlegenden Observatoren genommen werden! Nachbedingungen von grundlegenden Observatoren spezifizieren deshalb lediglich Konsistenzbedingungen zwischen den Methoden. Observatoren sind const-Methoden.
2. *Abgeleitete Observatoren* sind i.a. besser lesbar als eine (komplizierte) Kombination grundlegender Observatoren, können evtl. effizienter implementiert sein und sollten dann in Vorbedingungen unbedingt statt der grundlegenden Observatoren benutzt werden. Sie sind const-Methoden. Ihre Nachbedingungen sollten die Return-Werte mit Hilfe der grundlegenden Observatoren vollständig spezifizieren.

3. Konstruktoren (default, Kopier-): ...
4. Zuweisungsoperator: ...
5. `operator==`, `operator!=`: ...
6. Destruktor: ...
7: Modifikatoren: ...
8. `friend`-Methoden und Operatoren: ...
9. Iteratoren: ...