

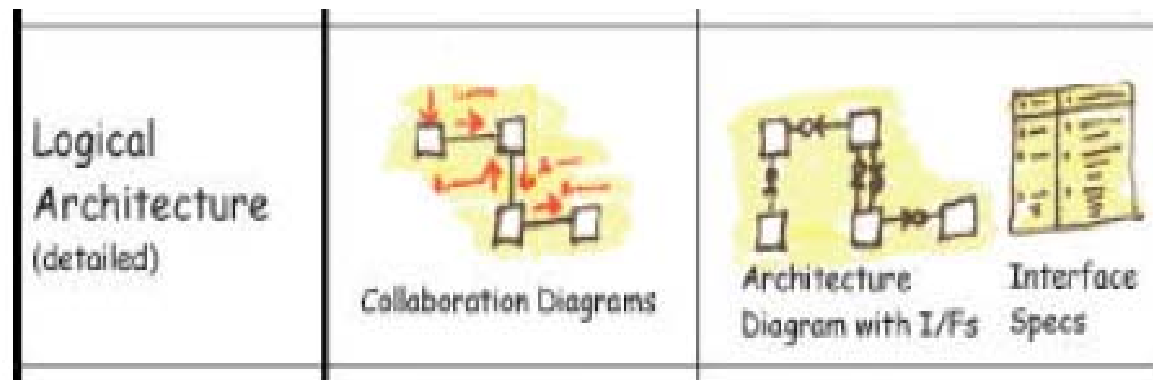
Code Contracts

Bernhard Hollunder

Department of Computer Science
Furtwangen University of Applied Sciences
Robert-Gerwig-Platz 1
D-78120 Furtwangen
hollunder@hs-furtwangen.de
<http://www.hs-furtwangen.de/>

Design by Contract (1)

- Konzept aus dem Bereich der Software-Entwicklung
- Grundlagen
 - Trennung von Schnittstelle und Implementierung einer Komponente
 - Definition formaler Verträge zur Verwendung von Schnittstellen



- Eingeführt durch Bertrand Meyer in der Programmiersprache Eiffel

Design by Contract (2)

- Abgeschwächte Form in typisierten Programmiersprachen:
 - `Student getStudent(int nr);`
 - `Collection<Student> getEnrolledStudents(Course c);`
- Im Typsystem *nicht* oder *nicht adäquat* abbildbar:
 - Matrikelnummern sind nicht negativ und haben genau 6 Ziffern
 - `Course` muss ungleich `null` sein
 - Zusammenhänge zwischen Parametern (z.B. die Summe von zwei Parameterwerten darf nicht größer `MAX_INT` sein)
 - Invariante Zustände (z.B. jeder Student hat eine Postadresse)
 - ...

Pragmatische Lösung

- Überprüfung der Gültigkeit von Parameterwerten in der Implementierung der Methode.
- Beispiel:

```
Student getStudent(int nr) {  
  
    // Überprüfung von Vorbedingungen  
    if (nr < 0 || nr > 999999) {  
        throw new IllegalArgumentException("...");  
    }  
  
    // hier beginnt die eigentliche Logik:  
    // z.B. Auffinden der Daten in einer DB  
    ...  
  
}
```

Systematische Lösung: Design by Contract

- Integration in Programmiersprachen / Spracherweiterungen
- Explizite Ausdrucksmittel für die Spezifikation von
 - Vorbedingungen
 - Nachbedingungen
 - Invarianten
- Tool-Unterstützung
- Ausgewählte Ansätze
 - JContract für Java
 - Spec# für C#
 - Code Contracts für .NET

Code Contracts

- Teil des .NET 4 Frameworks
- Integration in Visual Studio 2010
- Programmiermodell
 - Klasse **Contract** enthält statische Methoden für die Spezifikation von Vorbedingungen, Nachbedingungen und Invarianten
- Leistungsumfang
 - Überprüfung der Bedingungen zur Laufzeit (*Runtime checking*)
 - Statische Code-Analyse zur Compilezeit
 - Dokumentation

Ein Beispiel

```
class Konto {
    public void gutschrift(float betrag, string text)
        {...}
    public void abbuchung(float betrag, string text)
        {...}
    private float kontoStand;
    ...
}
```

```
class Bank {
    static void Main(string[] args)
    {
        Konto k = new Konto("Peter");
        k.gutschrift(50, "Weihnachtsgeld Oma");
        k.abbuchung(20, "Kino");
    }
}
```

Gutschrift über 50 EUR: Weihnachtsgeld Oma
Abbuchung über 20 EUR: Kino

Abbuchung: der 2. Versuch

```
public void abbuchung(float betrag, string text)
{
    if (betrag <= 0)
    {
        throw new ArgumentException(„Positiver Betrag ...“);
    }
    if (betrag > kontostand)
    {
        throw new ArgumentException("Abbuchung über ...");
    }

    kontostand -= betrag;
    Console.WriteLine("Abbuchung über " + betrag +
        " EUR: " + text + "\n");
}
```

Abbuchung mit Code Contract (1)

```
public void abbuchung(float betrag, string text)
{
    Contract.Requires(betrag > 0);

    Contract.Requires(betrag <= konstostand);

    kontostand -= betrag;

    Console.WriteLine("Abbuchung über " + betrag +
        " EUR: " + text + "\n");
}
```

Vorbedingungen

- Vorbedingungen werden bei Aufruf einer Methode ausgewertet.

- Beispiele

1. `Contract.Requires(betrag > 0);`

2. `Contract.Requires(x != null);`

3. `Contract.Requires<ArgumentNullException>(x != null, "x");`

4. `Contract.Requires<ArgumentException>(`
 `betrag > 0,`
 `"Negativer Betrag nicht erlaubt");`

Abbuchung: mit Code Contract (2)

```
public void abbuchung(float betrag, string text)
{
    Contract.Requires(betrag > 0);

    Contract.Requires(betrag <= kontostand);

    Contract.Ensures(
        kontostand == Contract.OldValue(kontostand) - betrag);

    kontostand -= betrag;

    Console.WriteLine("Abbuchung über " + betrag +
        " EUR: " + text + "\n");
}
```

Nachbedingungen

- Nachbedingungen werden vor dem Verlassen der Methode ausgewertet.
- Beispiele
 1. `Contract.Ensures(kontostand >= 0);`
 2. `Contract.Ensures(kontostand == Contract.OldValue(kontostand) - betrag);`
 3. `Contract.Ensures(Contract.Result<float>() > 0);`

Invarianten

- „Der Kontostand darf nicht negativ sein.“ ist eine Eigenschaft jeder Konto-Instanz, die nicht verletzt werden soll.
- Invarianten für Objekte sind Bedingungen, die aus Sicht des Aufrufers erfüllt sein müssen.
- Invarianten werden am Ende jeder öffentlichen Methode überprüft.

```
[ContractInvariantMethod]
private void AccountInvariant()
{
    Contract.Invariant(this.kontostand >= 0);
}
```

Quantoren

■ All-Quantor

- Alle Elemente einer Collection erfüllen eine bestimmte Eigenschaft .
- Die Eigenschaft wird über ein einstelliges Prädikat mit Rückgabebetyp `bool` festgelegt.
- Für alle Elemente der Collection wird das Prädikat angewendet.
- `ForAll` liefert `true`, wenn für alle Elemente das Prädikat erfüllt ist, ansonsten `false`.
- Beispiel:

```
public void reicheKonten(IEnumerable<Konto> konten)
{
    Contract.Requires(
        Contract.ForAll(konten, k => k.kontostand > 40));
}
```

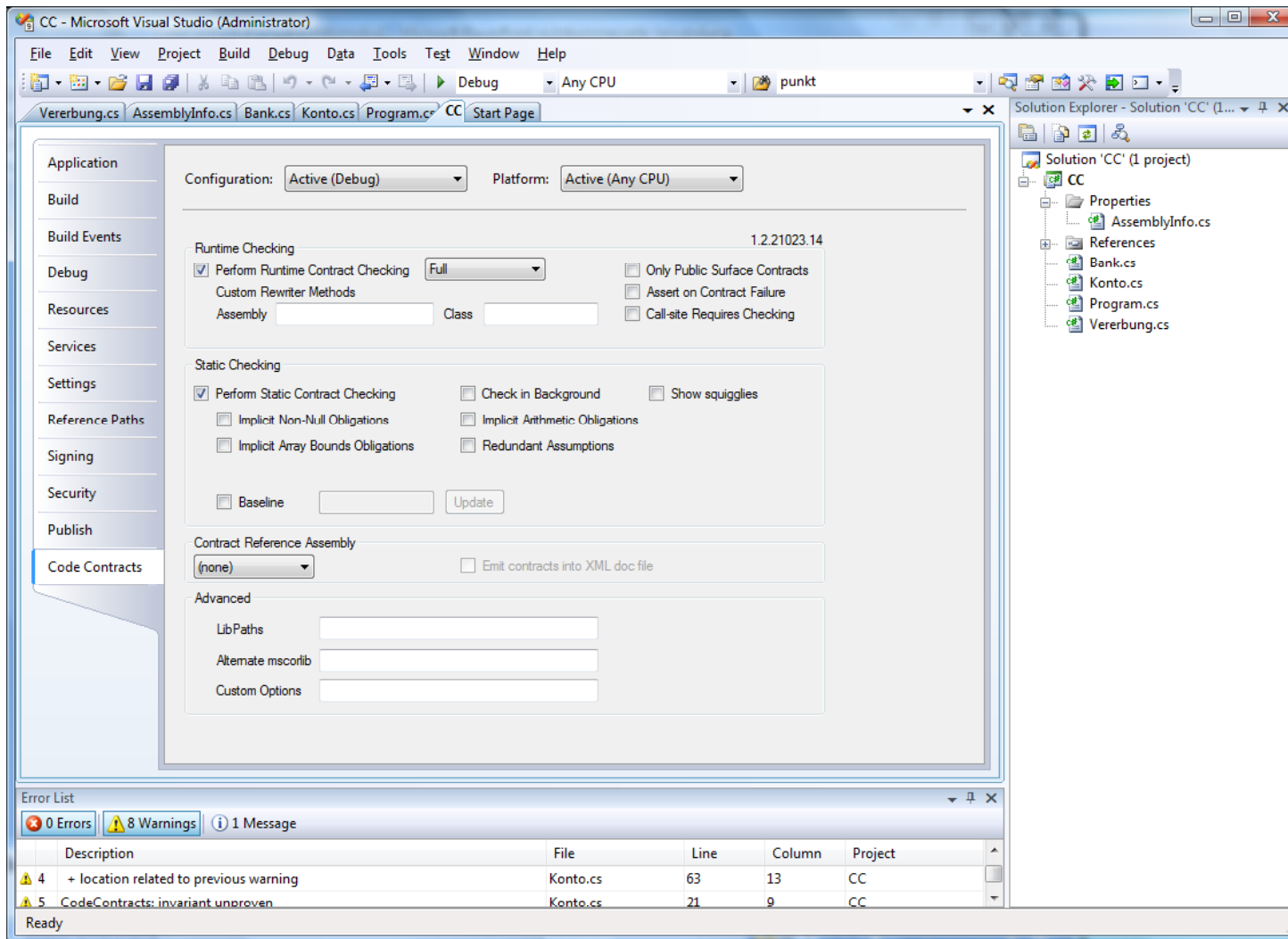
■ Analog wird ein Existenz-Quantor bereit gestellt.

Interface Contracts

- Keine Methodenimplementierung in Interfaces möglich.
- Ansatz: Definition einer Contract-Klasse für ein Interface.

```
[ContractClass(typeof(IFooContract))]  
interface IFoo  
{  
    void Put(int value);  
}  
  
[ContractClassFor(typeof(IFoo))]  
class IFooContract : IFoo  
{  
    void IFoo.Put(int value)  
    {  
        Contract.Requires(0 <= value);  
    }  
}
```

Visual Studio Integration



Zusammenfassung

- Code Contracts
 - bringen zusätzliche Qualität bei der Komponentenerstellung und -nutzung
 - mehr als nur „Spielerei“ (→ offizieller Support)
 - Toolunterstützung
 - Code Contracts für Interfaces (naja)
 - Unterstützung von Vererbung
 - Keine Unterstützung der Windows Communication Foundation (WCF)
- Systematische Unterstützung in der Java Welt steht noch aus
 - erste Schritte in JSE 7