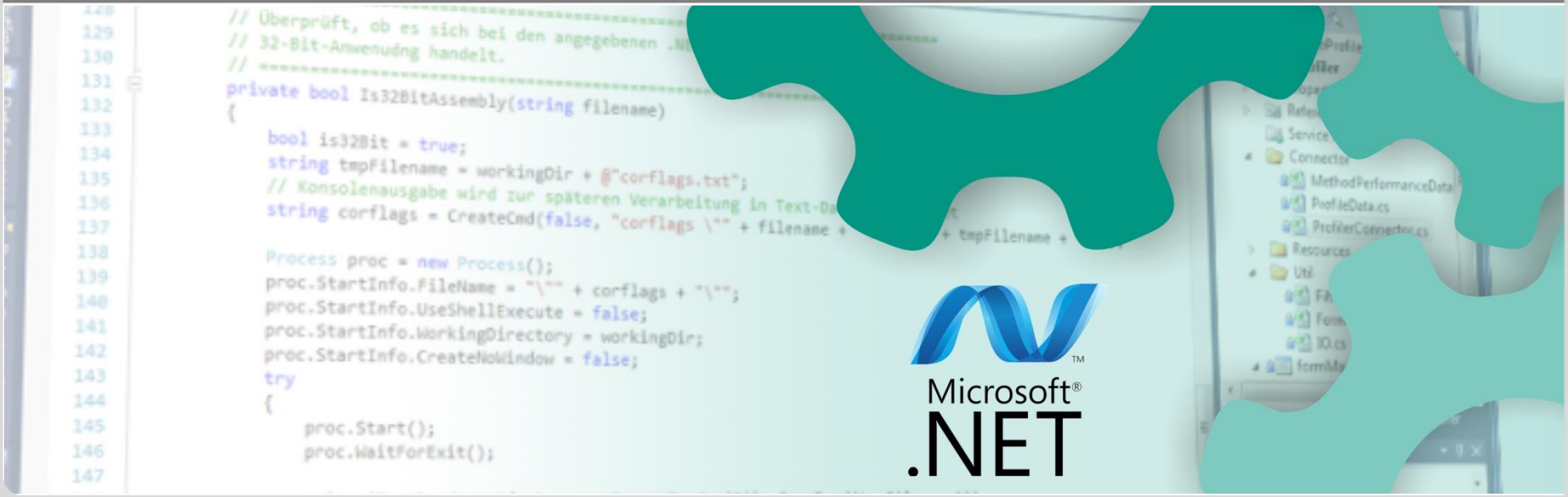
Automatische Korrektur von Wettlaufbedingungen

Diplomarbeit von Atanas Dimitrov
Betreuer: Jochen Schimmel, Korbinian Molitorisz

IPD Tichy – Lehrstuhl für Programmiersysteme



Motivation und Grundlagen

```
1.  private int balance = 0;

2.  public void Deposit(int amount)
3.  {
4.
5.          balance = balance + amount;
6.
7.  }
```

Nach einer sequenziellen Ausführung: balance = ?
Nach einer parallelen Ausführung: balance = ?

Motivation und Grundlagen

```
1.  private int balance = 0;

2.  public void Deposit(int amount)
3.  {
4.
5.          balance = balance + amount;
6.
7.  }
```

Nach einer sequenziellen Ausführung: balance = 10
Nach einer parallelen Ausführung: balance = 10 oder balance = **5**

Parallele Fehler (Datenwettläufe und Verklemmungen):

- treten nur unter bestimmten Zeitbedingungen auf
- können auch während längeren QA-Phasen unerkannt bleiben
- können eine unvorhersagbare Auswirkung auf das Codeverhalten haben
- die Korrektur von erkannten Problemen ist kompliziert

Motivation und Grundlagen

```
1.  private int balance = 0;

2.  public void Deposit(int amount)
3.  {
4.      lock (balanceLock) {
5.          balance = balance + amount;
6.      }
7.  }
```

Nach einer sequenziellen Ausführung: balance = 10
Nach einer parallelen Ausführung: balance = 10 ~~oder balance = 5~~

Parallele Fehler (Datenwettläufe und Verklemmungen):

- treten nur unter bestimmten Zeitbedingungen auf
- können auch während längeren QA-Phasen unerkannt bleiben
- können eine unvorhersagbare Auswirkung auf das Codeverhalten haben
- die Korrektur von erkannten Problemen ist kompliziert

Ziel der Arbeit

- Ziel: Entwickler bei der Korrektur von Datenwettläufen unterstützen
  - Korrekturvorschläge werden für Wettlaufbedingungen im parallelen und getesteten Code automatisch erzeugt
  - Die erzeugten Lösungen werden automatisch eingebaut und anhand Komponententests verifiziert

- Das Verfahren
  - betrachtet Datenwettläufe sowohl im Programmcode als auch in externen Bibliotheken
  - korrigiert Fehler durch sichere Synchronisationsblöcke und Austausch von parallel unsicheren Datenstrukturen
  - betrachtet Atomizitätsverletzungen, die durch parallele Zugriffe auf dieselbe Variable entstehen können

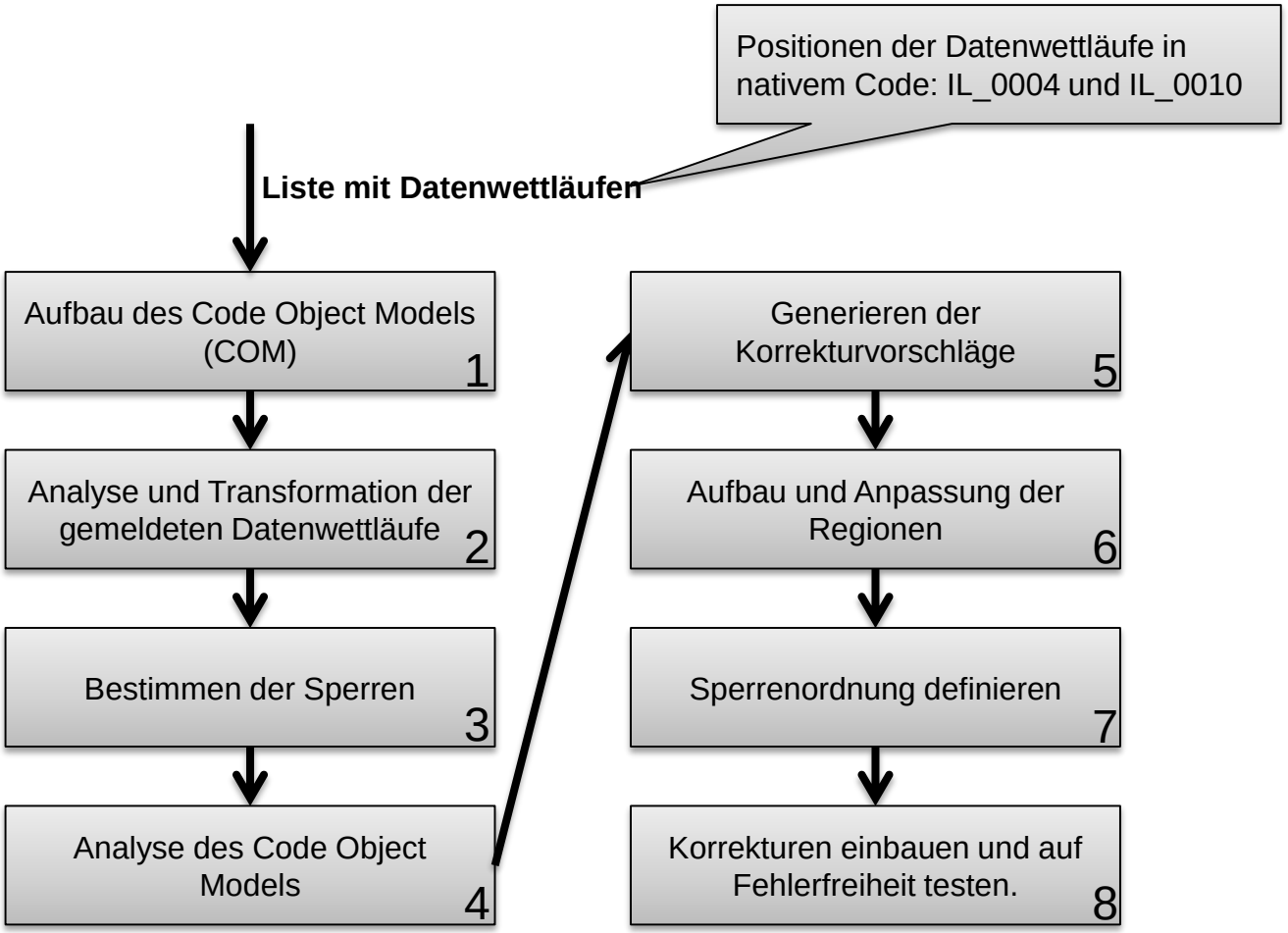
Verwandte Arbeiten

- Testgenerierung
  - **Automatische Testgenerierung für parallele .NET-Anwendungen**
    Diplomarbeit von Filip Dimitrov, IPD Tichy
- Automatisches Erkennen von Wettlaufbedingungen
  - Statische Werkzeuge: RacerX, Jlint, Extended Static Checker for Java, Coverity Static Analysis
  - Dynamische Werkzeuge: Coverity Dynamic Analysis, Eraser, Helgrind, Helgrind+
  - Testbasierte Werkzeuge: **Microsoft Research CHESS**, Relacy Race Detector, Concurrent Testing
- Korrektur und Vermeidung von Datenwettläufen
  - **Automatische Quellcodekorrektur von Wettlaufsituationen**
    Diplomarbeit von Felix Bondarenko, IPD Tichy
  - **Automatische Synchronisationskorrektur**
  - GenProg: Genetische Software-Reparatur
  - Selbstheilen zur Laufzeit, TachoRace, Asymetrische Datenwettläufe, Schichten, …

Verwandte Arbeiten: Im Vergleich zu Anderen

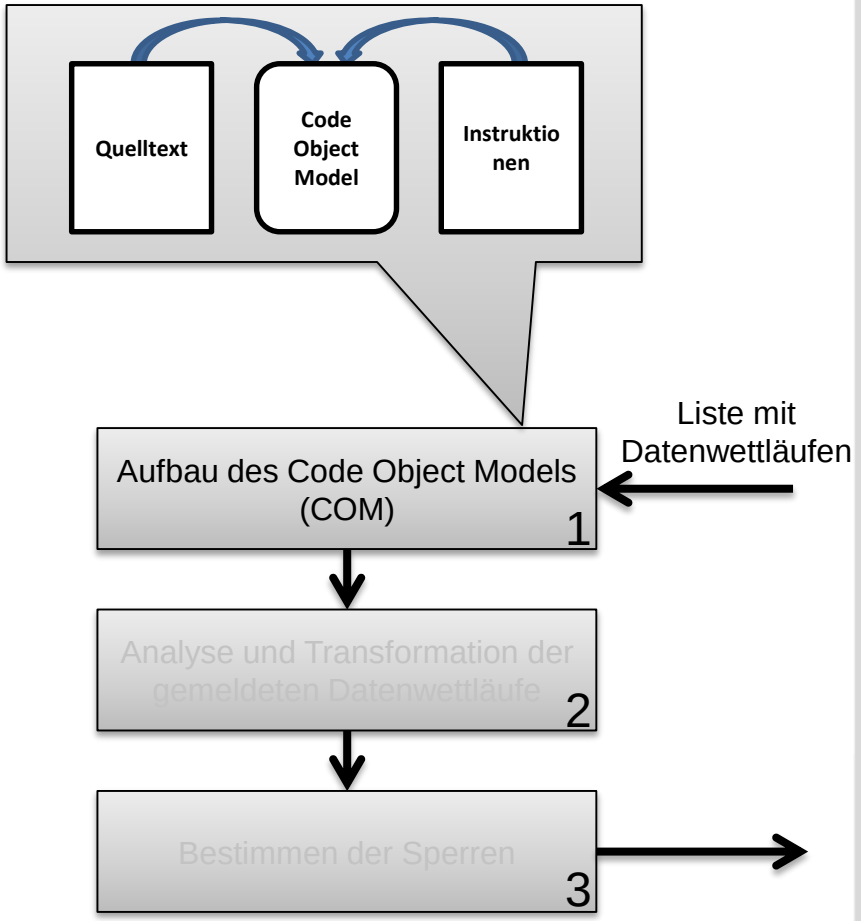
Konzept	Bondarenko [Bon11]	Flanagan und Freud [FF05]	Diese Arbeit
Allgemeine Codeannahme	Isoliertes System	---	Teil eines komplexen Systems
Wiederverwenden von Sperren	Sperreninferenz		Codeannotation
Korrektur von Datenwettläufen im Programmcode durch:	Explizites Akquirieren und Freigeben der Sperren	Synchronisationsblöcke	Synchronisationsblöcke
Korrektur von Datenwettläufen im externen Code durch:	Keine		Synchronisationsblöcke und Austausch von Datenstrukturen
Korrekturvorschläge	Liste zahlreicher Korrekturvarianten	Eine Lösung	Eine Lösung
Codeanalyse	Verwendet nur Quelltext		Quelltext und nativer Code
Behandlung von Verklemmungen	Sperrenordnung	Keine	Regionenanpassung und Sperrenordnung

Entwurf: Das Verfahren im Überblick



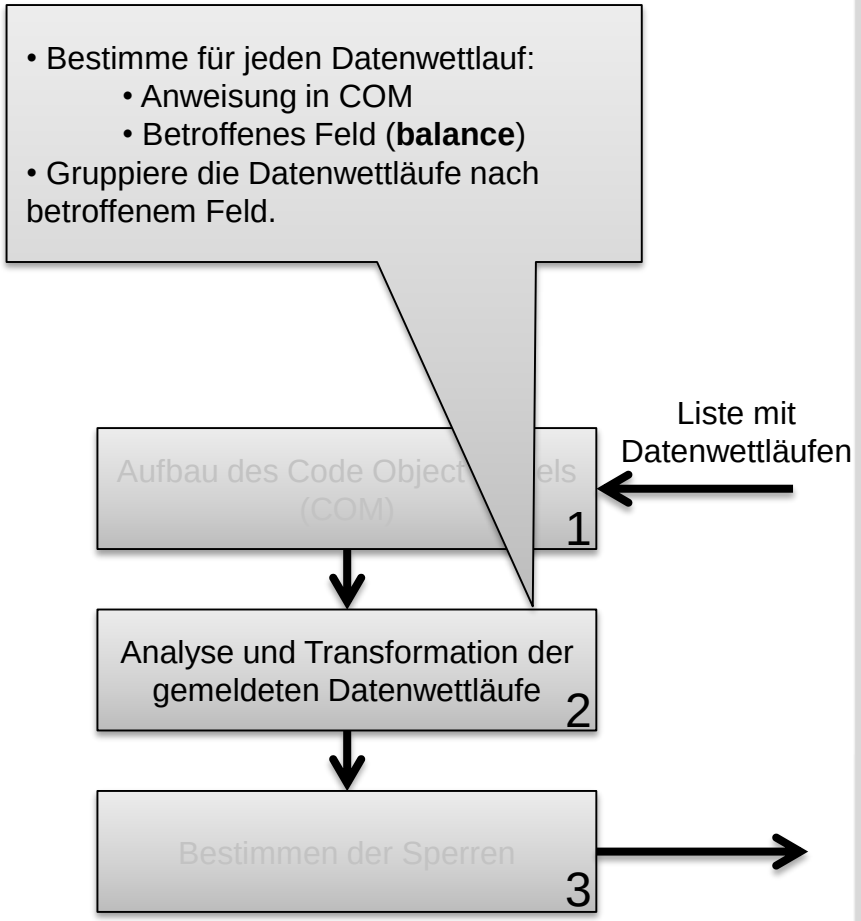
Entwurf: Code Object Model (COM)

```
1.  public void Withdraw(int amount) {
2.
3.          int temp = Read();
4.          balance = temp - amount;
5.
6.  }
7.  public int Read() {
8.
9.          if (loggingIsActive) {
10.             Console.Write("Current: " + balance);
11.         }
12.         return balance;
13.
14.  }
15. public void Deposit(int amount) {
16.
17.         balance = balance + amount;
18.
19. }
```



Entwurf: Analyse der Datenwettläufe

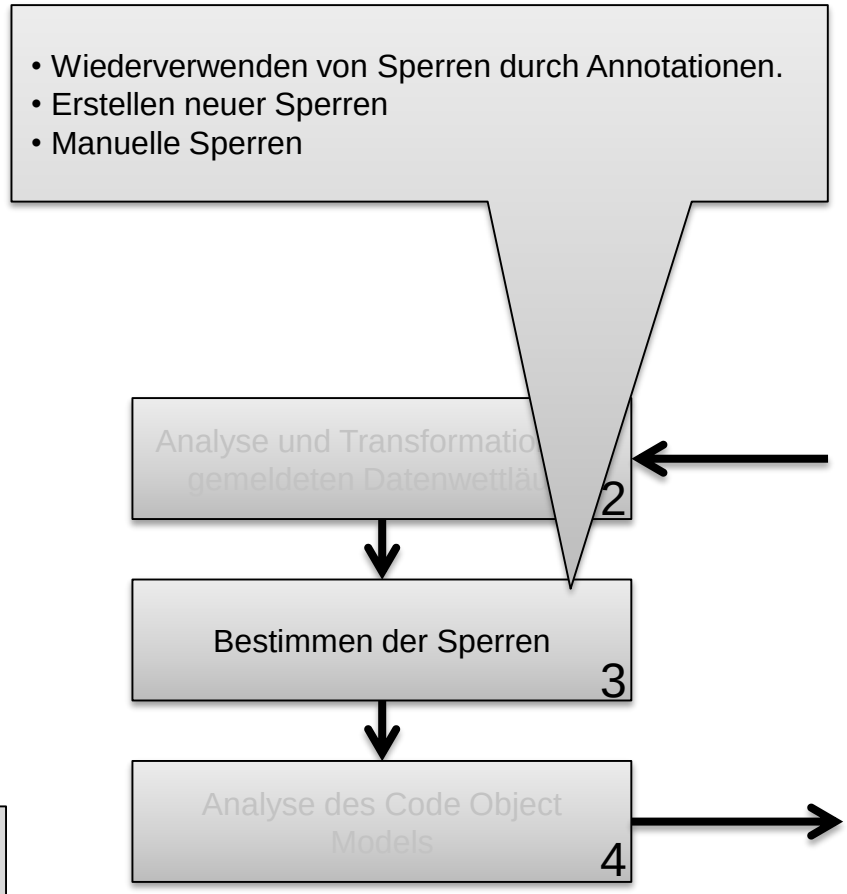
```
1.  public void Withdraw(int amount) {
2.
3.      int temp = Read();
4.      balance = temp - amount;
5.
6.  }
7.  public int Read() {
8.
9.      if (loggingIsActive) {
10.         Console.Write("Current: " + balance);
11.     }
12.     return balance;
13.
14.  }
15. public void Deposit(int amount) {
16.
17.     balance = balance + amount;
18.
19. }
```



Entwurf: Bestimmen der Sperren

```
1.  public void Withdraw(int amount) {
2.
3.      int temp = Read();
4.      balance = temp - amount;
5.
6.  }
7.  public int Read() {
8.
9.      if (loggingIsActive) {
10.         Console.Write("Current: " + balance);
11.     }
12.     return balance;
13.
14. }
15. public void Deposit(int amount) {
16.
17.     balance = balance + amount;
18.
19. }
```

```
[LockDescriptor("Bank.Account.balance")]
public static object StaticLockObject_balance = new Object();
```

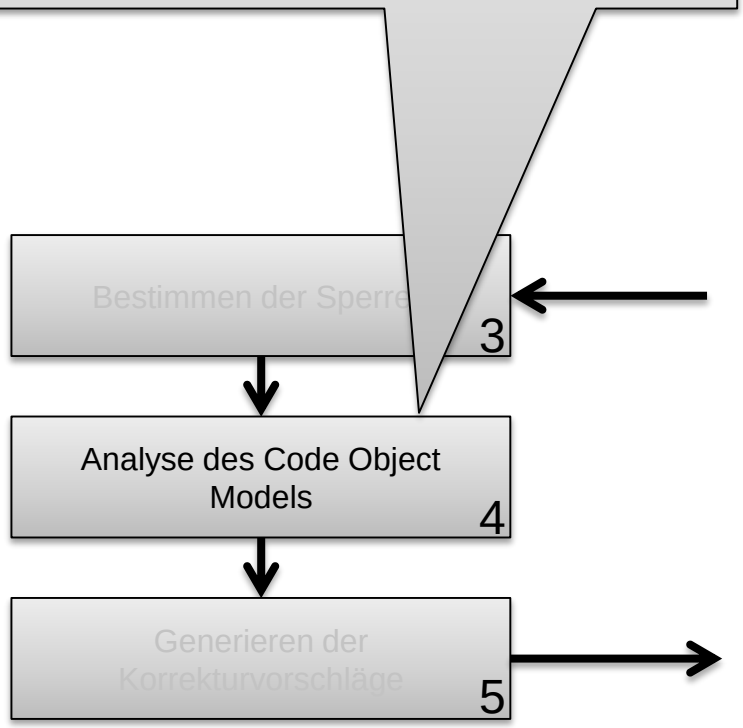


Entwurf: Analyse des Code Object Models

```
1.  public void Withdraw(int amount) {
2.
3.      int temp = Read();
4.      balance = temp - amount;
5.
6.  }
7.  public int Read() {
8.
9.      if (loggingIsActive) {
10.         Console.Write("Current: " + balance);
11.     }
12.     return balance;
13.
14. }
15. public void Deposit(int amount) {
16.
17.     balance = balance + amount;
18.
19. }
```

```
[LockDescriptor("Bank.Account.balance")]
public static object StaticLockObject_balance = new Object();
```

Vorbereitungsschritt: Gewinnen von Codeinformationen
- Aufrufhierarchien
- Sperrenbeziehungen
- Kontroll- und Datenfluss
- Datenabhängigkeiten
- Verfolgen von Objekten

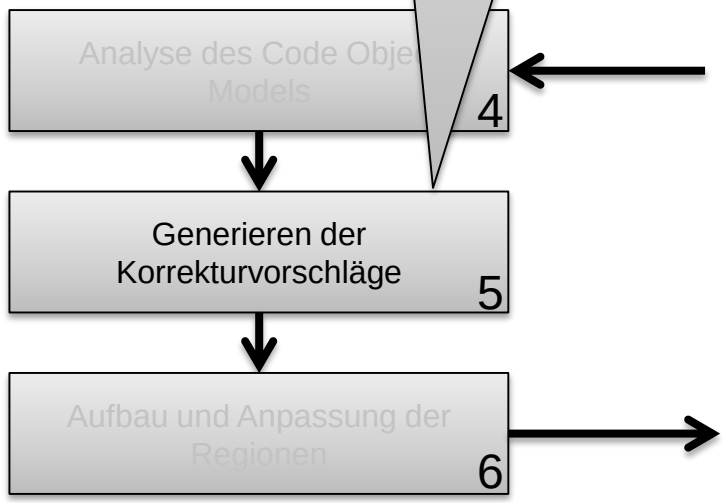


Entwurf: Generieren der Korrekturvorschläge

```
1.  public void Withdraw(int amount) {
2.
3.      int temp = Read();
4.      balance = temp - amount;
5.
6.  }
7.  public int Read() {
8.
9.      if (loggingIsActive) {
10.         Console.Write("Current: " + balance);
11.     }
12.     return balance;
13.
14. }
15. public void Deposit(int amount) {
16.
17.     balance = balance + amount;
18.
19. }
```

```
[LockDescriptor("Bank.Account.balance")]
public static object StaticLockObject_balance = new Object();
```

Bestimme für jede Problemstelle eine Liste mit Anweisungen, die geschützt werden müssen, um den Datenwettlauf zu entfernen.

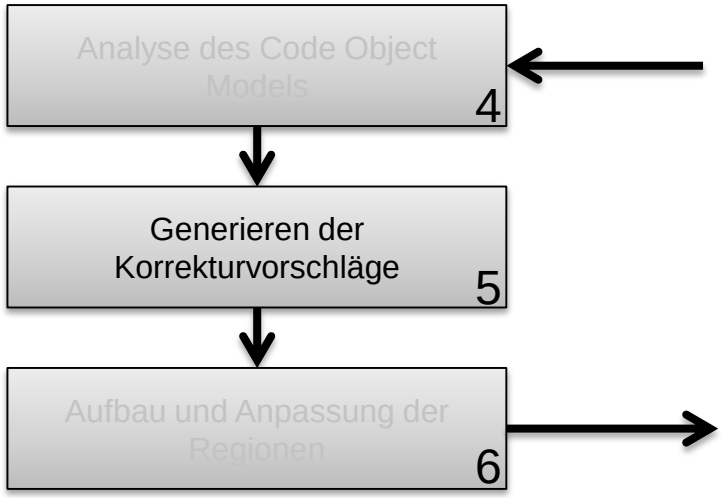


Entwurf: Generieren der Korrekturvorschläge

```
1.  public void Withdraw(int amount) {
2.
3.      int temp = Read();
4.      balance = temp - amount;
5.
6.  }
7.  public int Read() {
8.
9.      if (loggingIsActive) {
10.         Console.Write("Current: " + balance);
11.     }
12.     return balance;
13.
14. }
15. public void Deposit(int amount) {
16.
17.     balance = balance + amount;
18.
19. }
```

Erkennen von Atomizitätsstellen: **Das Stopper-Konzept**
- Thread.Sleep
- Thread.Join
- Monitor.Wait
- Process.WaitForExit

…

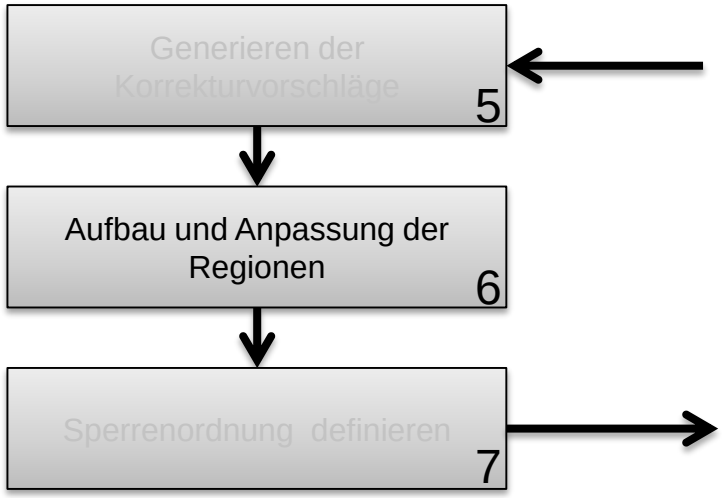



```
[LockDescriptor("Bank.Account.balance")]
public static object StaticLockObject_balance = new Object();
```

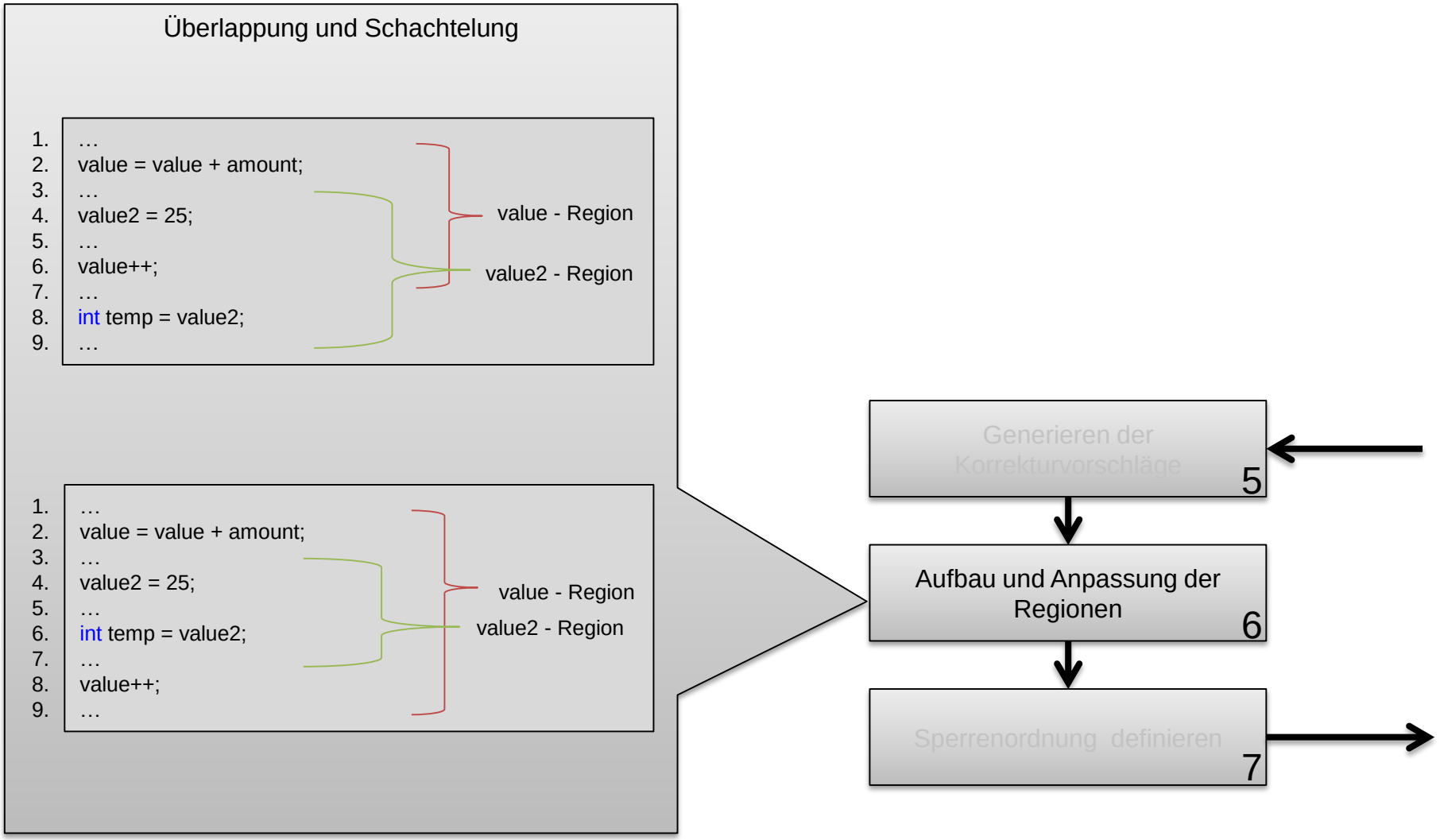
Entwurf: Aufbau und Anpassung der Regionen

```
1.  public void Withdraw(int amount) {
2.
3.          int temp = Read();
4.          balance = temp - amount;
5.
6.  }
7.  public int Read() {
8.
9.          if (loggingIsActive) {
10.             Console.Write("Current: " + balance);
11.         }
12.         return balance;
13.
14. }
15. public void Deposit(int amount) {
16.
17.         balance = balance + amount;
18.
19. }
```

```
[LockDescriptor("Bank.Account.balance")]
public static object StaticLockObject_balance = new Object();
```



Entwurf: Aufbau und Anpassung der Regionen

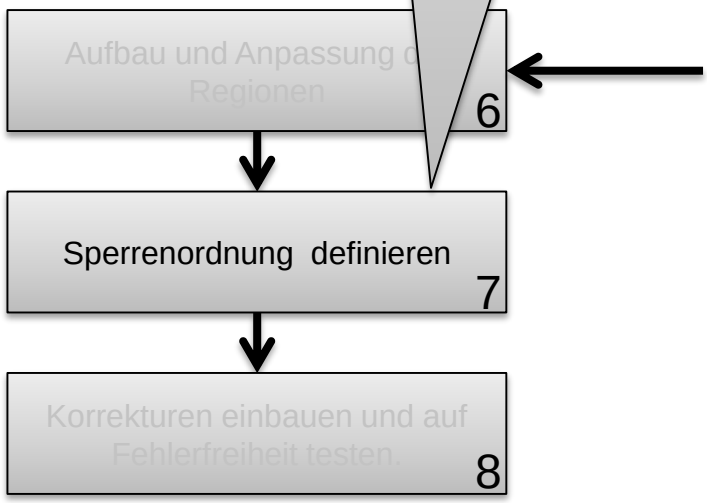


Entwurf: Sperrenordnung

```
1.  public void Withdraw(int amount) {
2.
3.          int temp = Read();
4.          balance = temp - amount;
5.
6.  }
7.  public int Read() {
8.
9.          if (loggingIsActive) {
10.             Console.Write("Current: " + balance);
11.         }
12.         return balance;
13.
14.  }
15. public void Deposit(int amount) {
16.
17.         balance = balance + amount;
18.
19. }
```

```
[LockDescriptor("Bank.Account.balance")]
public static object StaticLockObject_balance = new Object();
```

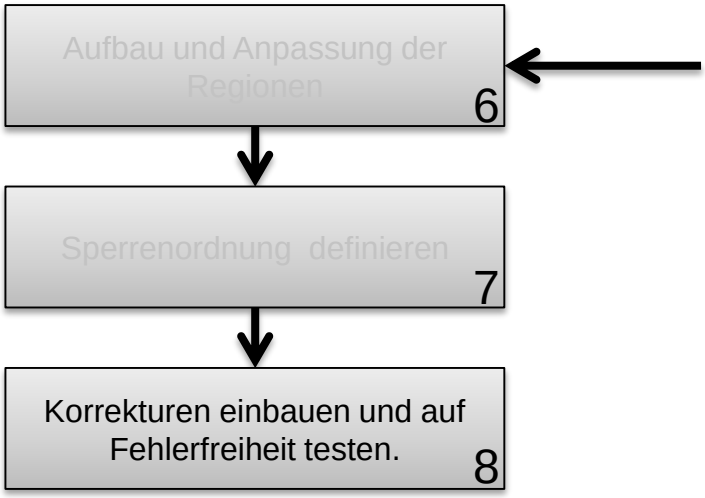
- Dieselben Sperren schützen mehrere Regionen -> Verklemmungsgefahr.
- Definiere eine feste Sperrenreihenfolge.



Entwurf: Einbau der Korrekturen

```
1.  public void Withdraw(int amount) {
2.      lock (LockClass.StaticLockObject_balance) {
3.          int temp = Read();
4.          balance = temp - amount;
5.      }
6.  }
7.  public int Read() {
8.      lock (LockClass.StaticLockObject_balance) {
9.          if (loggingIsActive) {
10.             Console.Write("Current: " + balance);
11.         }
12.         return balance;
13.     }
14.  }
15. public void Deposit(int amount) {
16.     lock (LockClass.StaticLockObject_balance) {
17.         balance = balance + amount;
18.     }
19. }
```

```
[LockDescriptor("Bank.Account.balance")]
public static object StaticLockObject_balance = new Object();
```



Entwurf: Korrekturmuster

- Schützen einfacher Anweisungen und Regionen (Bank Account Beispiel)
- Zusammengesetzten Anweisungen
  - Schützen im Körper (Bank Account Beispiel)
  - Schützen der Bedingung

```
1.
2.
3.
4.
5.
6.
7.
```

```
if (condition)
{
    this.doWork();
}
```

```
1.
2.
3.
4.
5.
6.
7.
8.
9.
```

```
for (i = 0; i < 10; i++)
{
    ...
    value = 20;
    …
}
```

Entwurf: Korrekturmuster

- Schützen einfacher Anweisungen und Regionen (Bank Account Beispiel)
- Zusammengesetzten Anweisungen
  - Schützen im Körper (Bank Account Beispiel)
  - Schützen der Bedingung

```
1.  lock (conditionLock)
2.  {
3.      if (condition)
4.      {
5.          this.doWork();
6.      }
7.  }
```

```
1.  lock (iLock)
2.  {
3.      for (i = 0; i < 10; i++)
4.      {
5.          ...
6.          value = 20;
7.          …
8.      }
9.  }
```

Entwurf: Korrekturmuster

- Schützen einfacher Anweisungen und Regionen (Bank Account Beispiel)
- Zusammengesetzten Anweisungen
  - Schützen im Körper (Bank Account Beispiel)
  - Schützen der Bedingung
  - Schützen durch Bedingungs- oder Sperrenextrahierung

```
1.
2.
3.
4.
5.
6.
7.
8.  if (condition)
9.  {
10.     this.doWork();
11. }
```

```
1.
2.
3.
4.
5.
6.  lock (lockObject)
7.  {
8.      ...
9.      value = 20;
10.     …
11. }
```

Entwurf: Korrekturmuster

- Schützen einfacher Anweisungen und Regionen (Bank Account Beispiel)
- Zusammengesetzten Anweisungen
  - Schützen im Körper (Bank Account Beispiel)
  - Schützen der Bedingung
  - Schützen durch Bedingungs- oder Sperrenextrahierung

```
1.  bool localCond = false;
2.
3.  lock (conditionLock)
4.  {
5.      localCond = condition;
6.  }
7.
8.  if (localCond)
9.  {
10.     this.doWork();
11. }
```

```
1.  object newLock = null;
2.  lock (lockLockObject)
3.  {
4.      newLock = lockObject;
5.  }
6.  lock (newLock)
7.  {
8.      ...
9.      value = 20;
10.     …
11. }
```

Entwurf: Schützen im externen Code

- Nur korrigierbare Situationen werden betrachtet
- Gruppieren: nach gemeinsamen Codestellen

Entwurf: Schützen im externen Code

- Nur korrigierbare Situationen werden betrachtet
- Gruppieren: nach gemeinsamen Codestellen
- Schützen durch Ersatz von Datenstrukturen
  - Vorteile

```
1. var myDict = new Dictionary<A, B>();
2. …
3. myDict.Add(key, value);
```

```
1. var myDict = new Dictionary<A, B>();
2. …
3. lock (lockObject)
4. {
5.     myDict.Add(key, value);
6. }
```

```
1. var myDict = new ThreadSafeDictionary<A, B>();
2. …
3. myDict.Add(key, value);
```

Entwurf: Schützen im externen Code

- Nur korrigierbare Situationen werden betrachtet
- Gruppieren: nach gemeinsamen Codestellen
- Schützen durch Ersatz von Datenstrukturen
  - Vorteile
  - Definition und Wahl der Datenstruktur, die Adapter-Datenstruktur

```
1.  public void Evaluate(ICollection<String> collection)
2.  {
3.      …
4.      if (collection is List<String>)
5.      {
6.          //Evaluate as List
7.      }
8.      else if (collection is Queue<String>)
9.      {
10.         // Evaluate  as Queue
11.     }
12.     else if (collection is Stack<String>)
13.     {
14.         // Evaluate as Stack
15.     }
16.     …
17. }
```

Entwurf: Schützen im externen Code

- Nur korrigierbare Situationen werden betrachtet
- Gruppieren: nach gemeinsamen Codestellen
- Schützen durch Ersatz von Datenstrukturen
  - Vorteile
  - Definition und Wahl der Datenstruktur, die Adapter-Datenstruktur
  - Erkennen von Wettlaufsituationen in Datenstrukturen
  - Austausch der Datenstruktur

```
1. var myDict = new Dictionary<A, B>();
2. …
3. myDict.Add(key, value);
```

```
1. var myDict = new Dictionary<A, B>();
2. …
3. lock (lockObject)
4. {
5.     myDict.Add(key, value);
6. }
```

```
1. var myDict = new ThreadSafeDictionary<A, B>();
2. …
3. myDict.Add(key, value);
```

Evaluierung

Programmname	Gemeldete Datenwettläufe	Korrigierte Datenwettläufe	Betroffene Felder	Erkannte Datenwettlauf-gruppen
Bank Account	6	6	1	1
Dekker	8	8	2	2
Parallel Printing	8	8	1	1
Limited Queue	15 interne 20 externe	35	2 + 2 externe Aufrufe	3
Master/Worker	4	4	1	1
RGB	40	40	4	4
Statement Diversity	10	10	4	4
Summe	111	111	15 + 2 externe Aufrufe	16

Ausblick

- Erweiterung des Code Object Models
- Virtuelle Methoden
- Erweiterung des Stopper-Konzepts
- Editieren bereits vorhandener Synchronisierung
- Weitere Synchronisationsmechanismen
- Schützen von Schleifen

```
1.
2.
3.    while (count < 1000000)
4.    {
5.        value++;
6.        count++;
7.    }
8.
```

```
1.  while (count < 1000000)
2.  {
3.
4.
5.      value++;
6.
7.      doAlotOfWork();
8.      count++;
9.  }
```

Ausblick

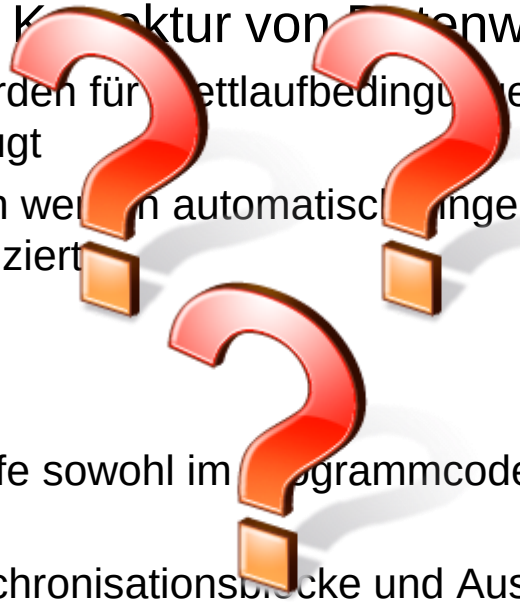
- Erweiterung des Code Object Models
- Virtuelle Methoden
- Erweiterung des Stopper-Konzepts
- Editieren bereits vorhandener Synchronisierung
- Weitere Synchronisationsmechanismen
- Schützen von Schleifen

```
1. lock (valueLock)
2. {
3.     while (count < 1000000)
4.     {
5.         value++;
6.         count++;
7.     }
8. }
```

```
1. while (count < 1000000)
2. {
3.     lock (valueLock)
4.     {
5.         value++;
6.     }
7.     doAlotOfWork();
8.     count++;
9. }
```

Zusammenfassung

- Ziel: Entwickler bei der K[illegible]ktur von D[illegible]tenwettläufen zu unterstützen
  - Korrekturvorschläge werden für [illegible]ettlaufbedingu[illegible]en im parallelen und getesteten Code automatisch erzeugt
  - Die erzeugten Lösungen we[illegible]n automatisc[illegible]ingebaut und anhand Komponententests verifiziert



- Das Verfahren
  - betrachtet Datenwettläufe sowohl im P[illegible]ogrammcode als auch in externen Bibliotheken
  - verwendet sichere Synchronisationsb[illegible]cke und Austausch von parallel unsicheren Datenstrukturen
  - betrachtet Atomizitätsverletzungen, die durch parallele Zugriffe auf dieselbe Variable entstehen können

```
1.  public class Account {
2.      private int balance;
3.      public Account(int amount) {
4.          balance = amount;
5.      }
6.      public void Withdraw(int amount) {
7.          int temp = Read();
8.          balance = temp - amount;
9.      }
10.     public int Read() {
11.         int temp;
12.         temp = balance;
13.         return temp;
14.     }
15.     public void Deposit(int amount) {
16.         balance = balance + amount;
17.     }
18. }
```

(a)

```
1.  public void Withdraw(int amount) {
2.      lock (LockClass.StaticLockObject_balance) {
3.          int temp = Read();
4.          balance = temp - amount;
5.      }
6.  }
7.  public int Read() {
8.      int temp;
9.      lock (LockClass.StaticLockObject_balance) {
10.         temp = balance;
11.     }
12.     return temp;
13. }
14. public void Deposit(int amount) {
15.     lock (LockClass.StaticLockObject_balance) {
16.         balance = balance + amount;
17.     }
18. }
```

(b)

```
1.  [AttributeUsage(AttributeTargets.Field)]
2.  public class LockDescriptor : Attribute {
3.      public LockDescriptor(string protectedField) {
4.          this.protectedField = protectedField;
5.      }
6.      string protectedField;
7.  }
8.  public class MC_LockClass {
9.      [LockDescriptor("Bank.Account.balance")]
10.     public static object StaticLockObject_balance = new Object();
11. }
```

(c)

```
1.  public static volatile bool t1_is_entering;
2.  public static volatile bool t2_is_entering;
3.
4.  public static void thread1() {
5.      t1_is_entering = true;
6.      if (!t2_is_entering) {
7.          // (mutually exclusive section)
8.      }
9.      t1_is_entering = false;
10. }
11.
12. public static void thread2() {
13.     t2_is_entering = true;
14.     if (!t1_is_entering) {
15.         // (mutually exclusive section)
16.     }
17.     t2_is_entering = false;
18. }
```

(a)

```
1.  public static void thread1() {
2.      lock (LockClass.StaticLockObject_t2_is_entering) {
3.          lock (LockClass.StaticLockObject_t1_is_entering){
4.              t1_is_entering = true;
5.              if (!t2_is_entering) {
6.                  // (mutually exclusive section)
7.              }
8.              t1_is_entering = false;
9.          }
10.     }
11. }
12. public static void thread2() {
13.     lock (LockClass.StaticLockObject_t2_is_entering) {
14.         t2_is_entering = true;
15.         bool local_t1_is_entering = false;
16.         lock (LockClass.StaticLockObject_t1_is_entering) {
17.             local_t1_is_entering = !t1_is_entering;
18.         }
19.         if (local_t1_is_entering) {
20.             // (mutually exclusive section)
21.         }
22.         t2_is_entering = false;
23.     }
24. }
```

(b)

```
1.  private string consoleOutput = "";
2.  public void Print() {
3.      Random r = new Random();
4.      int threadId = Thread.CurrentThread.ManagedThreadId;
5.      consoleOutput = threadId + ": started printing: " + System.DateTime.Now;
6.      Console.WriteLine(consoleOutput);
7.      consoleOutput = "";
8.      for (int i = 0; i < 25; i++) {
9.          consoleOutput += string.Format("({0}:{1}), ", threadId, i);
10.         Console.WriteLine(consoleOutput);
11.     }
12.     consoleOutput = threadId + ": finished printing: " + System.DateTime.Now;
13.     Console.WriteLine(consoleOutput);
14. }
```

(a)

```
1.  private string consoleOutput = "";
2.  public void Print() {
3.      Random r = new Random();
4.      int threadId = Thread.CurrentThread.ManagedThreadId;
5.      lock (LockClass.StaticLockObject_consoleOutput) {
6.          consoleOutput = threadId + ": started printing: " + System.DateTime.Now;
7.          Console.WriteLine(consoleOutput);
8.          consoleOutput = "";
9.          for (int i = 0; i < 25; i++) {
10.             consoleOutput += string.Format("({0}:{1}), ", threadId, i);
11.             Console.WriteLine(consoleOutput);
12.         }
13.         consoleOutput = threadId + ": finished printing: " + System.DateTime.Now;
14.         Console.WriteLine(consoleOutput);
15.     }
16. }
```

(b)

```
1.  public class SafeLimitedQueue {
2.      private Queue<Object> queue;
3.      private int numberOfOps = 0;
4.      private int current = 0;
5.      private int limit;
6.      public SafeLimitedQueue(int limit) {
7.          this.limit = limit;
8.          queue = new Queue<Object>();
9.      }
10.     public void Enqueue(Object item) {
11.         if (current < limit) {
12.             numberOfOps++;
13.             current = current + 1;;
14.             queue.Enqueue(item);
15.         }
16.     }
17.     public Object Dequeue() {
18.         if (current > 0) {
19.             numberOfOps++;
20.             current = current -1;
21.             return queue.Dequeue();
22.         }
23.         return null;
24.     }
25. }
```

(a)

```
1.  public void Enqueue(Object item) {
2.      lock (LockClass.StaticLockObject_current) {
3.          if (current < limit) {
4.              lock (LockClass.StaticLockObject_numberOfOps) {
5.                  numberOfOps++;
6.              }
7.              current = current + 1;
8.              lock (LockClass.StaticLockObject_1) {
9.                  queue.Enqueue(item);
10.             }
11.         }
12.     }
13. }
14. public Object Dequeue() {
15.     lock (LockClass.StaticLockObject_current) {
16.         if (current > 0) {
17.             lock (LockClass.StaticLockObject_numberOfOps) {
18.                 numberOfOps++;
19.             }
20.             current = current - 1;
21.             lock (LockClass.StaticLockObject_1) {
22.                 return queue.Dequeue();
23.             }
24.         }
25.     }
26.     return null;
27. }
```

(b)

```
1.  public class MasterWorker {
2.      private int workDone = 0;
3.      public void StartMaster() {
4.          Thread s = null;
5.          Console.WriteLine("Work done: " + workDone);
6.          workDone += 10;
7.          s = new Thread(l => {
8.              (l as MasterWorker).DoSomeWork();
9.          } );
10.         s.Start(this);
11.         workDone += 10;
12.         s.Join();
13.         Console.WriteLine("Work done: " + workDone);
14.     }
15.     private void DoSomeWork() {
16.         workDone += 10;
17.         for (int i = 0; i < 10; i++) {
18.             workDone += 10;
19.         }
20.     }
21. }
```

(a)

```
1.  private int workDone = 0;
2.  public void StartMaster() {
3.      Thread s = null;
4.      Console.WriteLine("Work done: " + workDone);
5.      lock (LockClass.StaticLockObject_workDone){
6.          workDone += 10;
7.          s = new Thread(l => {
8.              (l as MasterWorker).DoSomeWork();
9.          } );
10.         s.Start(this);
11.         workDone += 10;
12.     }
13.     s.Join();
14.     Console.WriteLine("Work done: " + workDone);
15. }
16. private void DoSomeWork() {
17.     lock (LockClass.StaticLockObject_workDone) {
18.         workDone += 10;
19.         for (int i = 0; i < 10; i++) {
20.             workDone += 10;
21.         }
22.     }
23. }
```

(b)

```
1.  private int red;
2.  private int green;
3.  private int blue;
4.  private String name;
5.  public void SetRGB(int red, int green, int blue, String name)
6.  {
7.      if (Check(red, green, blue)) {
8.          this.red = red;
9.          this.green = green;
10.         this.blue = blue;
11.         this.name = name;
12.     }
13. }
14. public int GetRGB() {
15.     return ((red << 16) | (green << 8) | blue);
16. }
17. public String GetName() {
18.     return name;
19. }
20. public void Invert() {
21.     red = 255 - red;
22.     green = 255 - green;
23.     blue = 255 - blue;
24.     name = "Inverse of " + name;
25. }
```

(a)

```
1.  public void SetRGB(int red, int green, int blue, String name) {
2.      if (Check(red, green, blue)) {
3.          lock (LockClass.StaticLockObject_red) {
4.              this.red = red; }
5.          lock (LockClass.StaticLockObject_green) {
6.              this.green = green; }
7.          lock (LockClass.StaticLockObject_blue) {
8.              this.blue = blue; }
9.          lock (LockClass.StaticLockObject_name) {
10.             this.name = name; }
11.     }
12. }
13. public int GetRGB() {
14.     lock (LockClass.StaticLockObject_red) {
15.         lock (LockClass.StaticLockObject_green) {
16.             lock (LockClass.StaticLockObject_blue) {
17.                 return ((red << 16) | (green << 8) | blue);
18.             }
19.         }
20.     }
21. }
22. public String GetName() {
23.     lock (LockClass.StaticLockObject_name) {
24.         return name; }
25. }
26. public void Invert() {
27.     lock (LockClass.StaticLockObject_red) {
28.         red = 255 - red; }
29.     lock (LockClass.StaticLockObject_green) {
30.          green = 255 - green; }
31.     lock (LockClass.StaticLockObject_blue) {
32.         blue = 255 - blue; }
33.     lock (LockClass.StaticLockObject_name) {
34.         name = "Inverse of " + name; }
35. }
```

(b)

```
1.  public bool ifCond = true;
2.  public int raceInIf = 0;
3.  public int raceInWhile = 0;
4.  public int raceInCondition = 5;
5.  public int doWhileRace = 5;
6.
7.  public void ExecStatements() {
8.      if (ifCond) {
9.          raceInIf++;
10.     } else {
11.         raceInIf--;
12.     }
13.     int loops = 5;
14.     while (loops > 0) {
15.         raceInWhile++;
16.         loops--;
17.     }
18.     while (raceInCondition > 0) {
19.         raceInCondition--;
20.     }
21.     do {
22.        doWhileRace--;
23.     } while (doWhileRace > 0);
24. }
```

(a)

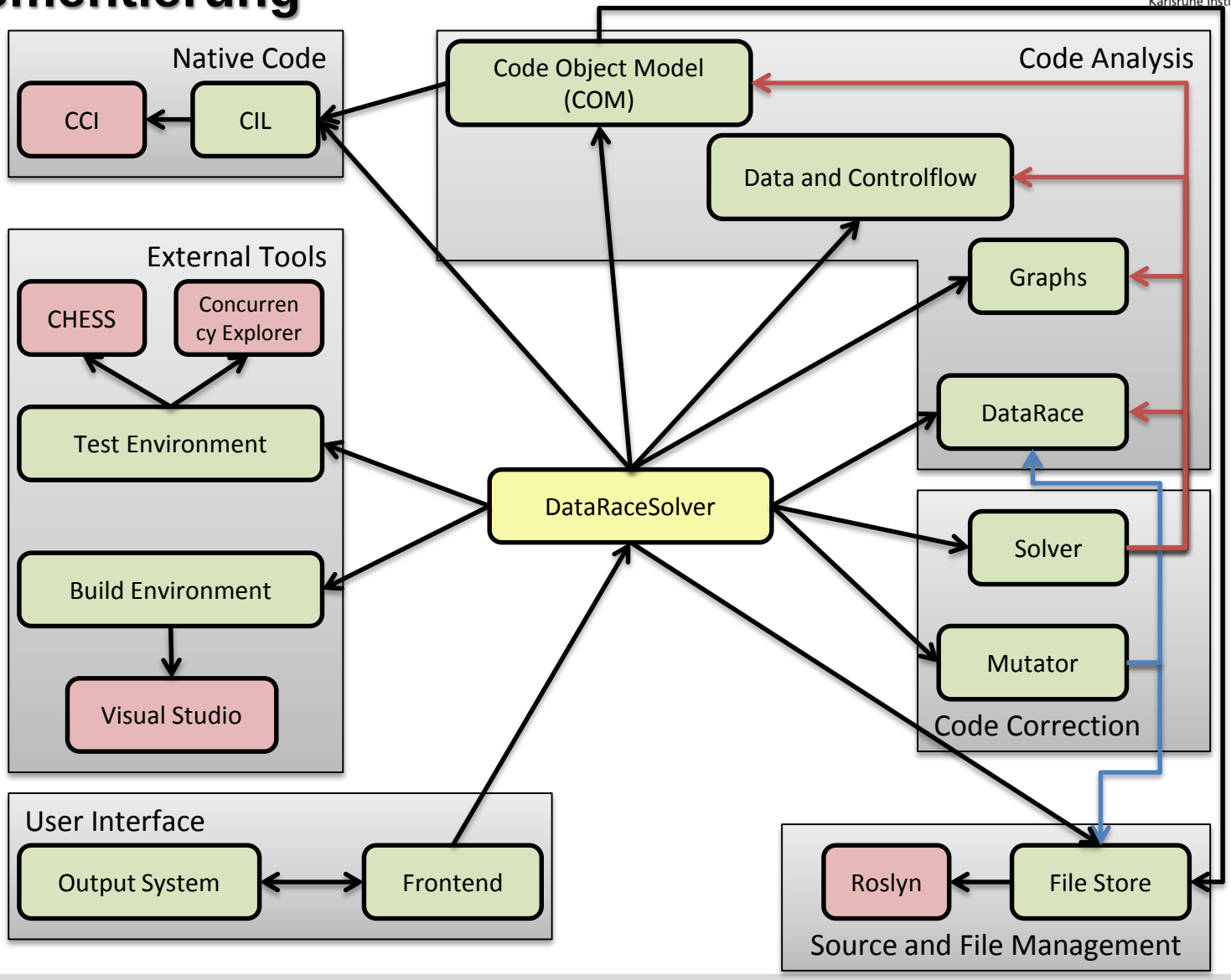
```
1.  public void  ExecStatements() {
2.      if (ifCond) {
3.          lock (LockClass.StaticLockObject_raceInIf) {
4.              raceInIf++;
5.          }
6.      } else {
7.          lock (LockClass.StaticLockObject_raceInIf) {
8.              raceInIf--;
9.          }
10.     }
11.     int loops = 5;
12.     while (loops > 0) {
13.         lock (LockClass.StaticLockObject_raceInWhile) {
14.             raceInWhile++;
15.         }
16.         loops--;
17.     }
18.     lock (LockClass.StaticLockObject_raceInCondition) {
19.         while (raceInCondition > 0) {
20.             raceInCondition--;
21.         }
22.     }
23.     lock (LockClass.StaticLockObject_doWhileRace) {
24.         do {
25.             doWhileRace--;
26.         } while (doWhileRace > 0);
27.     }
28. }
29.
30.
31.
```

(b)

Grundlagen: Verwendete Technologien

- Microsoft .NET und die Common Language Infrastructure
- Visual Studio
  - Entwicklungsumgebung
- Common Compiler Infrastructure (CCI)
  - Schnittstelle zum IL-Code.
  - Auslesen, Editieren und Erzeugen von Metadaten und Intermediate Language Code.
- Roslyn
  - Schnittstelle zum Quelltext.
  - Stellt das gesamte Quelltextwissen des Übersetzers und seine „Funktionalität zur Verfügung.
- Microsoft Research CHESS
  - Testumgebung.
  - Ausführen paralleler Komponententests unter verschiedener Schedulles.
  - Wurde angepasst, um mehrere Informationen über die Datenwettläufe zu gewinnen.

Implementierung



Verwandte Arbeiten: Im Vergleich zu Anderen

- Betrachtet den Code als Teil eines komplexen Systems
- Alle öffentliche Methoden sind Startpunkte, die parallel ausgeführt werden können
- Korrektur von Fehlern im eigenen Code durch Lock-Blöcke
- Korrektur von Fehlern im externen Code durch Lock-Blöcke und Austausch von Thread-unsicheren Datenstrukturen
- Generiert keine Liste von Lösungen, sondern nur eine
- Codeanalyse: Verwendet sowohl Quelltext als auch nativen Code
- Anpassung der geschützten Regionen, um Verklemmungen zu vermeiden