Refactoring in eXtreme Programming:

Komposition von Transformationen

Hermann Wessels

Quellen:

Composite Refactorings for Java Programs

Mel O Cinneide, Paddy Nixon

Ein Modell für invasive Softwareadaption

Volker Kuttruff

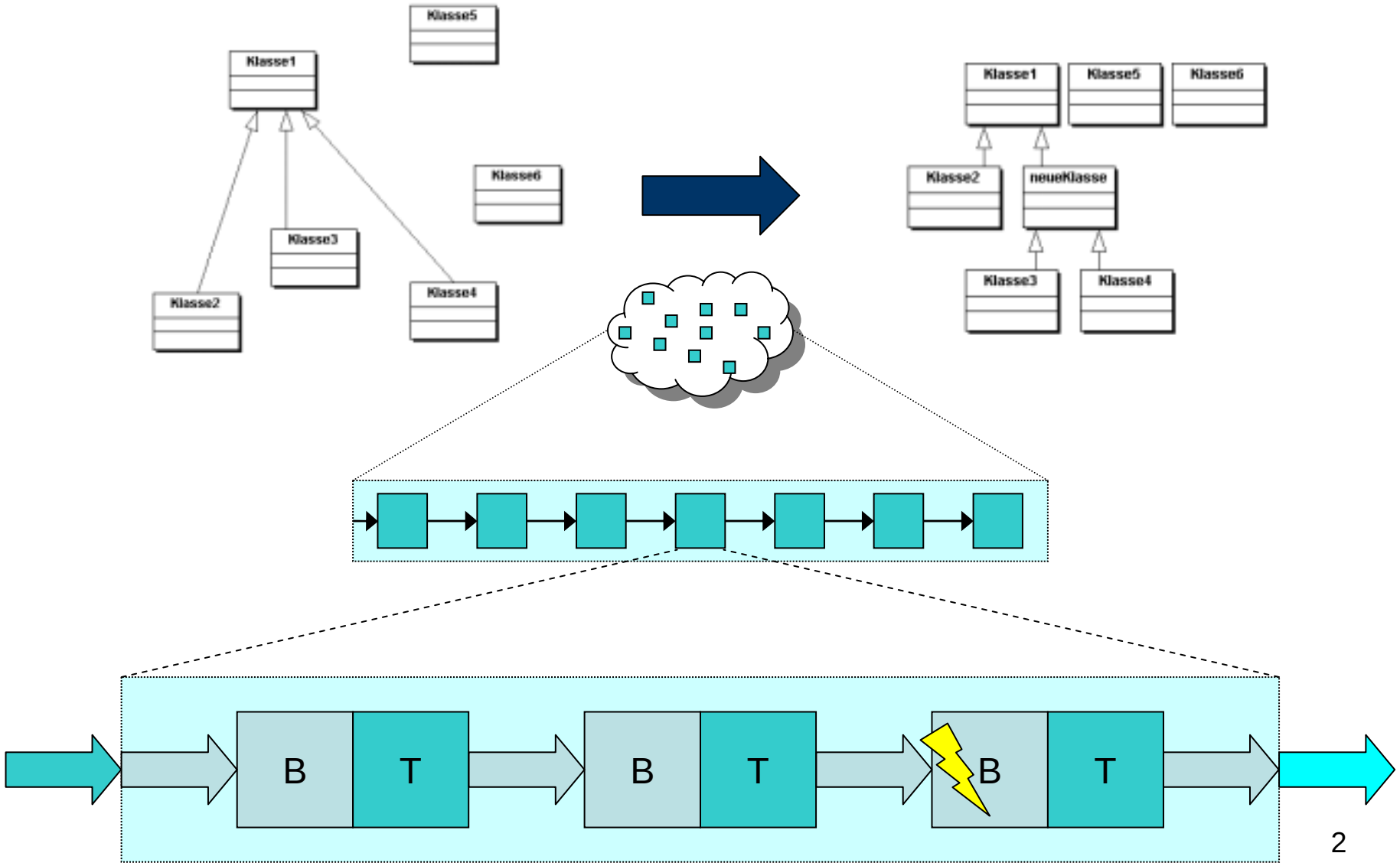
Practical Analysis for Refactoring

Donald Bradley Roberts

Refactoring Object-Oriented Frameworks

William F. Opdyke

Motivation


Klasse5
Klasse1
Klasse6
Klasse3
Klasse2
Klasse4
Klasse1
Klasse5
Klasse6
Klasse2
neueKlasse
Klasse3
Klasse4
B
T
B
T
B
T

Gliederung

- Verbesserung der Struktur
  - Vorbedingungen
    - Nachbedingungen
    - Basistransformationen
  - Kompositionen
    - Abhängigkeiten
    - Berechnung von Bedingungen
  - Nutzen und Anwendung

Verbesserung der Struktur

- Hier: Beschränkung auf Änderungen der Programmstruktur
- Programmverhalten bleibt erhalten
- Änderungen können in Basistransformationen zerlegt werden
- Basistransformationen sind bekannt
  - Erzeugen von Programmeigenschaften
  - Löschen von Programmeigenschaften
  - Verändern von Programmeigenschaften
  - Verschieben von Variablen
  - einfache Transformationen

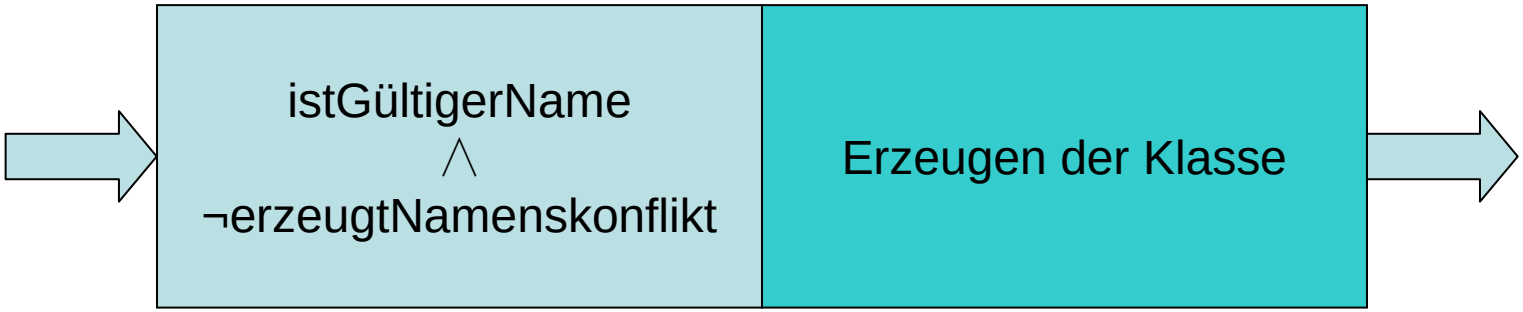
Vorbedingungen - Einführung

Eine Basistransformation ist ein geordneter Tupel, bestehend aus:

- den Vorbedingungen
- der Transformation

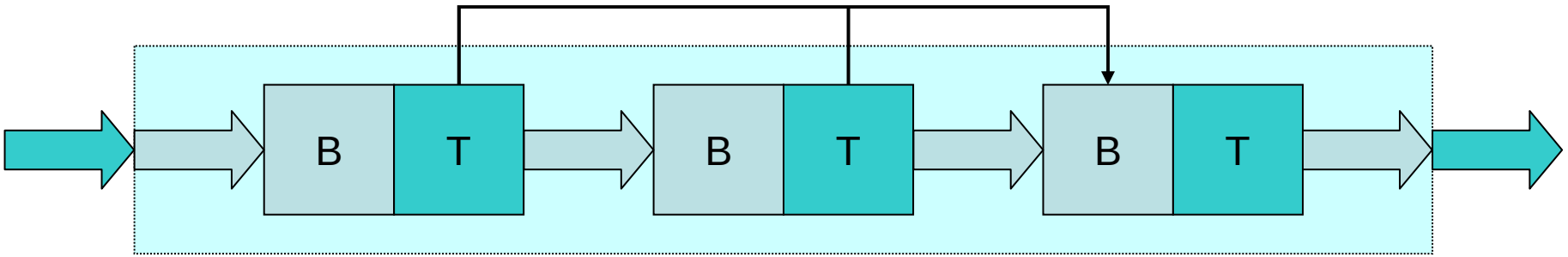
Nur wenn die Vorbedingungen wahr sind, ist die Basistransformation anwendbar und das Programmverhalten bleibt erhalten.

Beispiel:



Vorbedingungen - Ketten

- Basistransformationen werden kaum isoliert angewandt
- Änderungen sind Ketten von Basistransformationen
- Ketten werden sequentiell abgearbeitet
- Vorbedingungen werden oft erst durch vorhergehenden Basistransformationen erfüllt



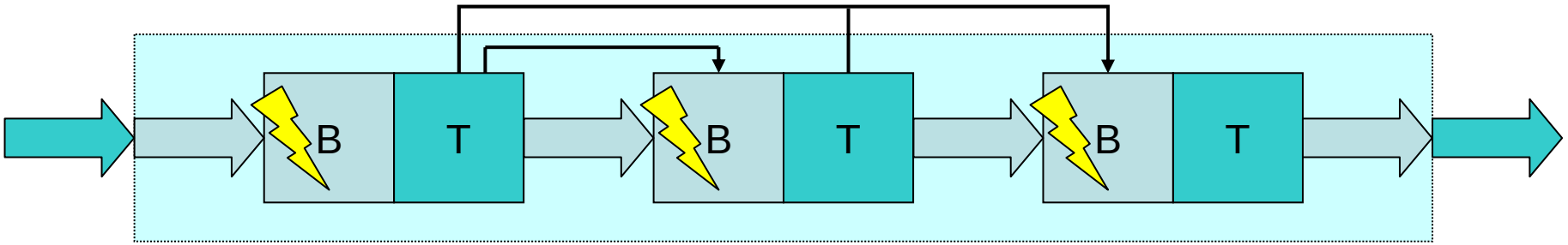
Vorbedingungen – In der Anwendung

Beispiel:

neue Klasse soll bestehendes Attribut kapseln

Ablauf:

1. Erzeugen der neuen Klasse
2. Erzeugen der Variablen
3. Verschieben der Zugriffsmethoden



Problem:

Vor jeder Basistransformation muss geprüft werden ob deren Vorbedingungen erfüllt sind

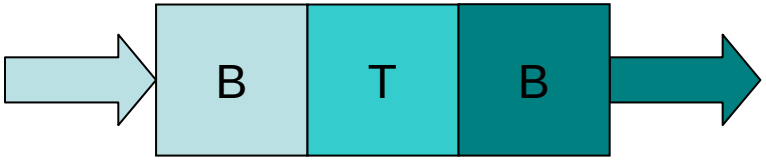
Gliederung

- Verbesserung der Struktur
  - Vorbedingungen
  - Nachbedingungen
  - Basistransformationen
- Kompositionen
  - Abhängigkeiten
  - Berechnung von Bedingungen
- Nutzen und Anwendung

Nachbedingungen - Einführung

Eine Basistransformation ist ein geordneter Tripel, bestehend aus:

- Vorbedingungen
- Transformation
- Nachbedingungen

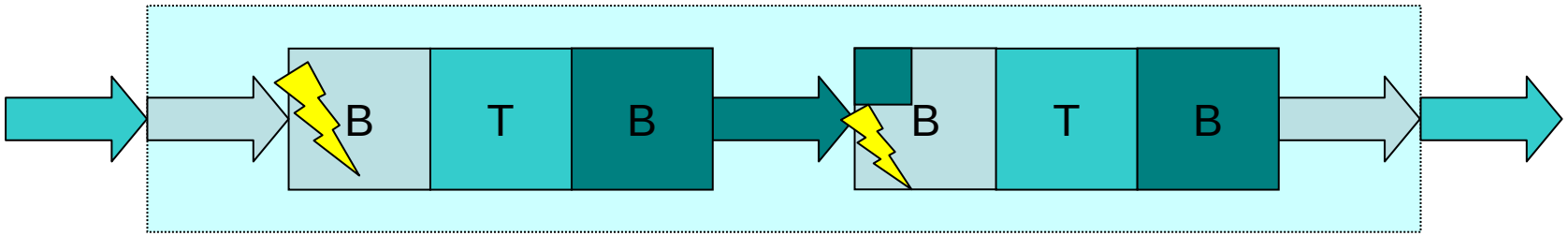


Mit Anwendung der Basistransformation, werden die Nachbedingungen wahr.

Beispiel:

Nachbedingungen – Anwendung

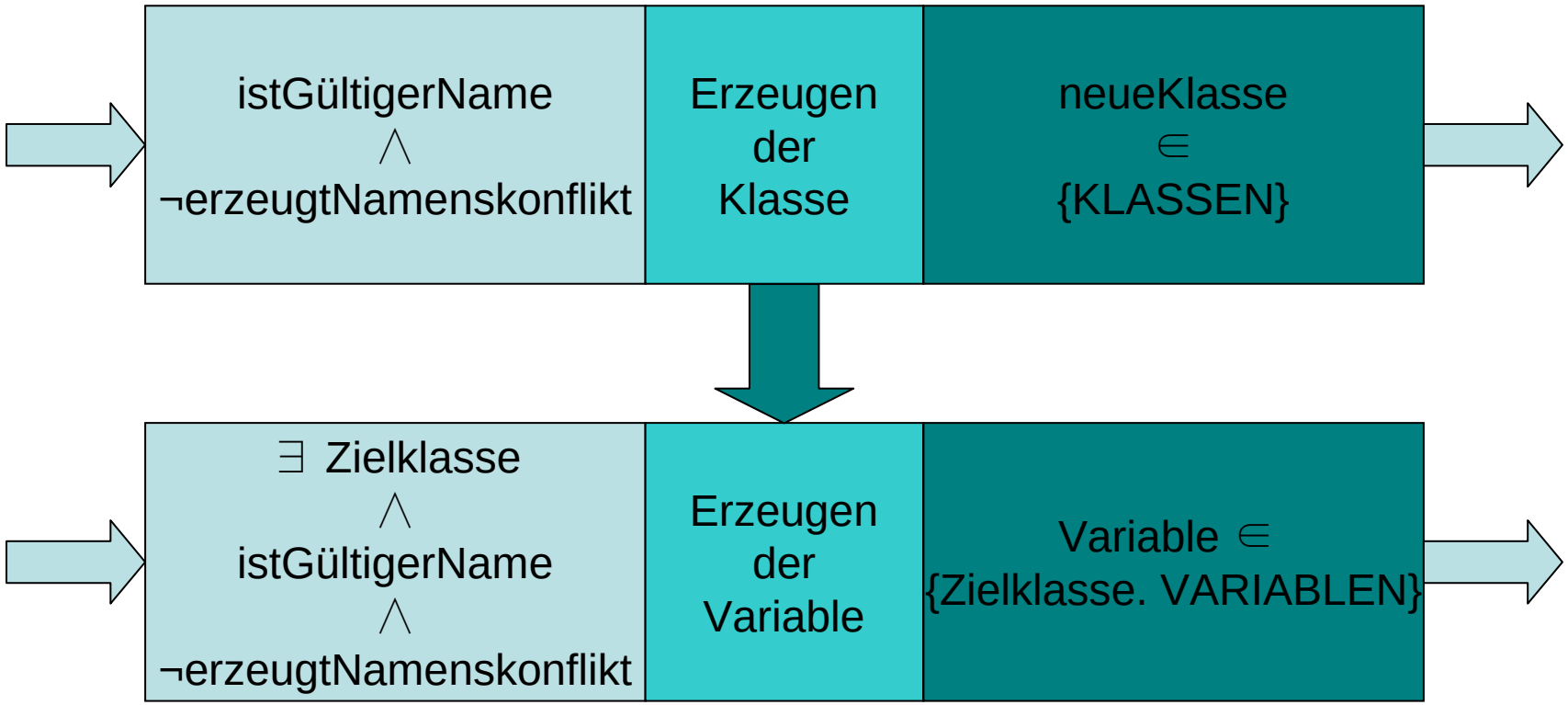
- Nachbedingungen können den Analyseaufwand verringern
- Strikt sequentielle Abarbeitung der Änderungen $\Rightarrow \exists$ eindeutige Menge gültiger Bedingungen zwischen Änderungen



Nachbedingungen - Beispiel

Beispiel:

Erzeugen einer Klasse mit einer Variable



Gliederung

- Verbesserung der Struktur
  - Vorbedingungen
  - Nachbedingungen
  - Basistransformationen
- Kompositionen
  - Abhängigkeiten
  - Berechnung von Bedingungen
- Nutzen und Anwendung

Basistransformationen – Beispiel (1)

erzeugeKlasse(MOD,(kontext,name)):Klasse

$$
\begin{aligned}
&\text{istGültigerKlassenname(name)}\\
&\wedge\\
&\neg\text{erzeugtNamenskonflikt(MOD,(kontext,name))}\\
&\wedge\\
&\text{kontext} \in \text{Pakete} \vee \text{kontext} \in \{\text{KLASSEN}\}\\
&\wedge\\
&((\text{„public“} \in \text{MOD} \Rightarrow \text{kontext} \in \text{Pakete} \vee \text{kontext} \in \{\text{KLASSEN}\})\\
&\wedge (\text{„protected“} \in \text{MOD} \Rightarrow \text{kontext} \in \{\text{KLASSEN}\} \wedge \text{„public“} \notin \text{MOD}\\
&\qquad \wedge \text{„private“} \notin \text{MOD})\\
&\wedge (\text{„private“} \in \text{MOD} \Rightarrow \text{kontext} \in \{\text{KLASSEN}\} \wedge \text{„public“} \notin \text{MOD}\\
&\qquad \wedge \text{„protected“} \notin \text{MOD})\\
&\wedge (\text{„static“} \in \text{MOD} \Rightarrow \text{kontext} \in \{\text{KLASSEN}\})\\
&\wedge (\text{„abstract“} \in \text{MOD} \Rightarrow (\text{kontext} \in \text{Pakete} \vee \text{kontext} \in \{\text{KLASSEN}\})\\
&\qquad \wedge (\text{„private“} \notin \text{MOD} \Lambda \text{„final“} \notin \text{MOD})))
\end{aligned}
$$

Basistransformationen – Beispiel (2)

kontext $\in$ Pakete $\Rightarrow$ erzeuge neue Klasse name im Paket kontext mit Zugriffsrechten MOD

$\vee$

kontext $\in$ {KLASSEN} $\Rightarrow$ erzeuge innere Klasse name in der Klasse kontext mit Zugriffsrechten MOD

{Klassen} = {Klassen} $\cup$ neueKlasse

$\wedge$

neueKlasse.name = name

$\wedge$

neueKlasse.qualifizierterName = (kontext,name)

$\wedge$

MOD.neueKlasse = MOD

$\wedge$

(Klassenzugriffe.neueKlasse = Ø $\wedge$ Superklassen.neueKlasse = Ø $\wedge$ innereKlassen.neueKlasse = Ø $\wedge$ Konstruktoren.neueKlasse = Ø $\wedge$ Attribute.neueKlasse = Ø $\wedge$ Methoden.neueKlasse = Ø)

$\wedge$

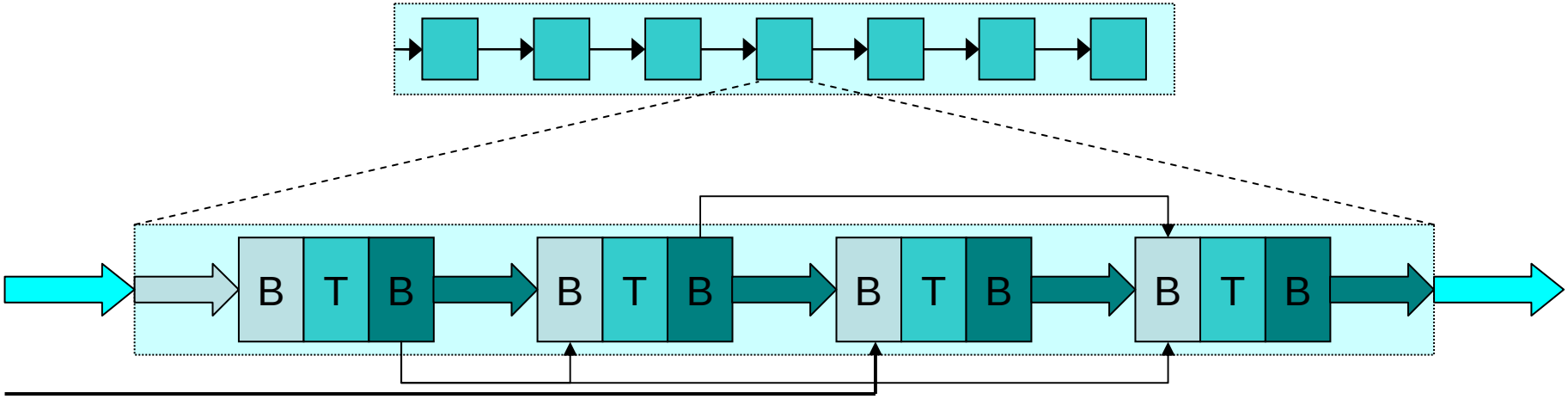
istSchnittstelle(neueKlasse) = false

Gliederung

- Verbesserung der Struktur
  - Vorbedingungen
  - Nachbedingungen
  - Basistransformationen
- Kompositionen
  - Abhängigkeiten
    - Berechnung von Bedingungen
- Nutzen und Anwendung

Abhängigkeiten - Einführung

Änderungen werden zerlegt



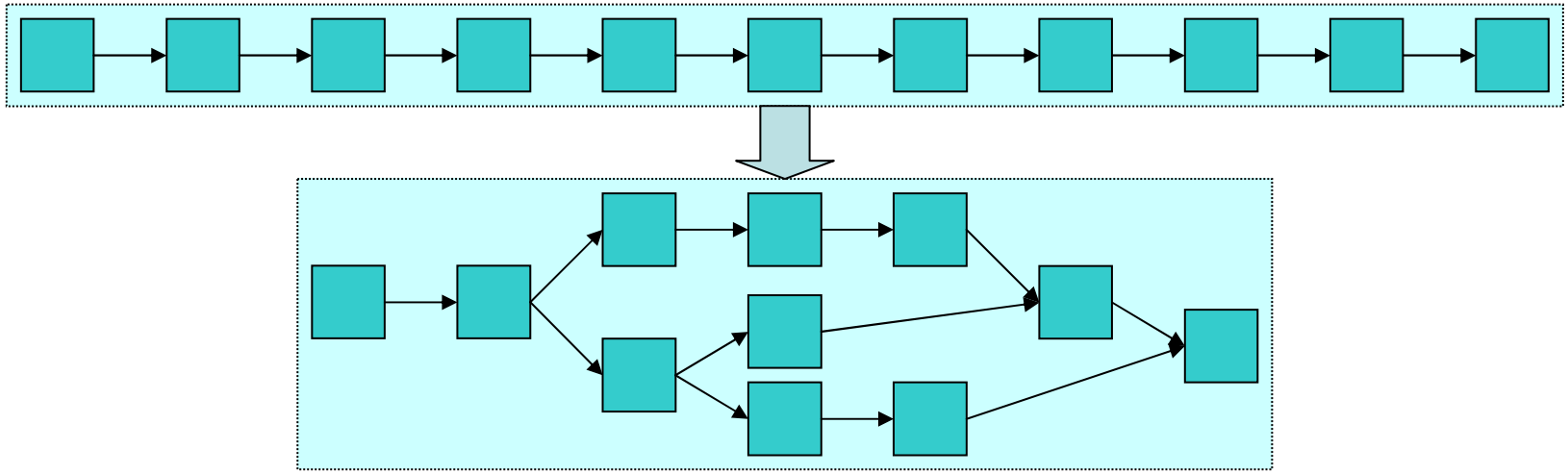
Es ∃ von einander unabhängige Änderungen

Beispiel:

Beim Erzeugen mehrerer Variablen ist die Reihenfolge irrelevant

Abhängigkeiten - Nutzen

Analyse der Abhängigkeiten erlaubt die Erstellung von Abhängigkeitsgraphen



Vorteile:

- Parallelisierung
- Komposition von Transformationen

Problem:

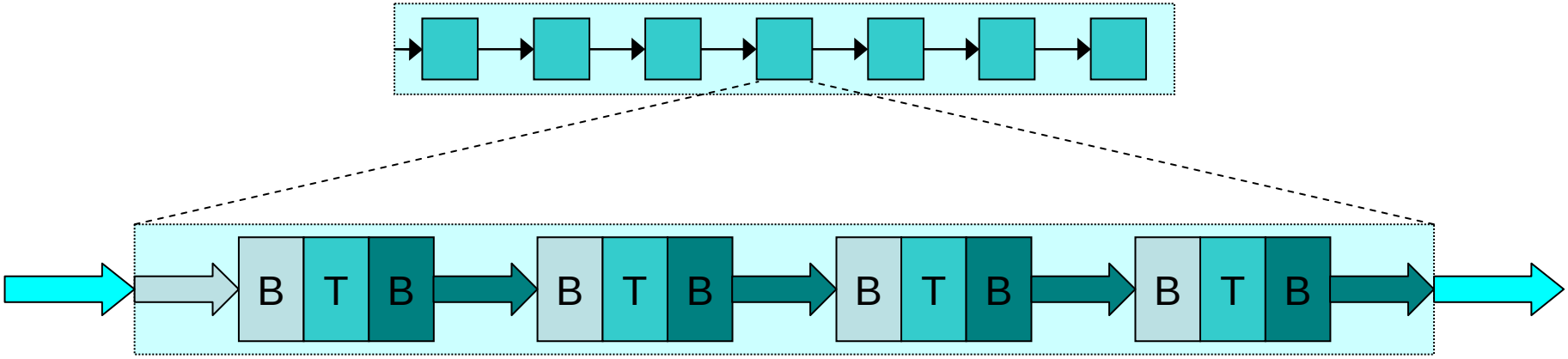
Methodik ist unklar

Gliederung

- Verbesserung der Struktur
  - Vorbedingungen
  - Nachbedingungen
  - Basistransformationen
- **Kompositionen**
  - Abhängigkeiten
  - **Berechnung von Bedingungen**
- Nutzen und Anwendung

Berechnung von Bedingungen

Änderungen werden in Basistransformationen zerlegt



Problem:

Nicht jede beliebige Sequenz von Basistransformationen ergibt eine Kette

Berechnung von Bedingungen

Wie kann für eine beliebige Sequenz von Basistransformationen geprüft werden ob es sich um eine Kette handelt?

Einfache $\bigwedge$-Verknüpfung der Vorbedingungen funktioniert nicht

Gegenbeispiel:

Erzeuge neue Klasse mit einer Variable.

istGültigerName $\wedge$ ¬erzeugtNamenskonflikt	$\exists$ Zielklasse $\wedge$ istGültigerName $\wedge$ ¬erzeugtNamenskonflikt	Erzeugen der Klasse	Erzeugen der Variable

Berechnung von Bedingungen

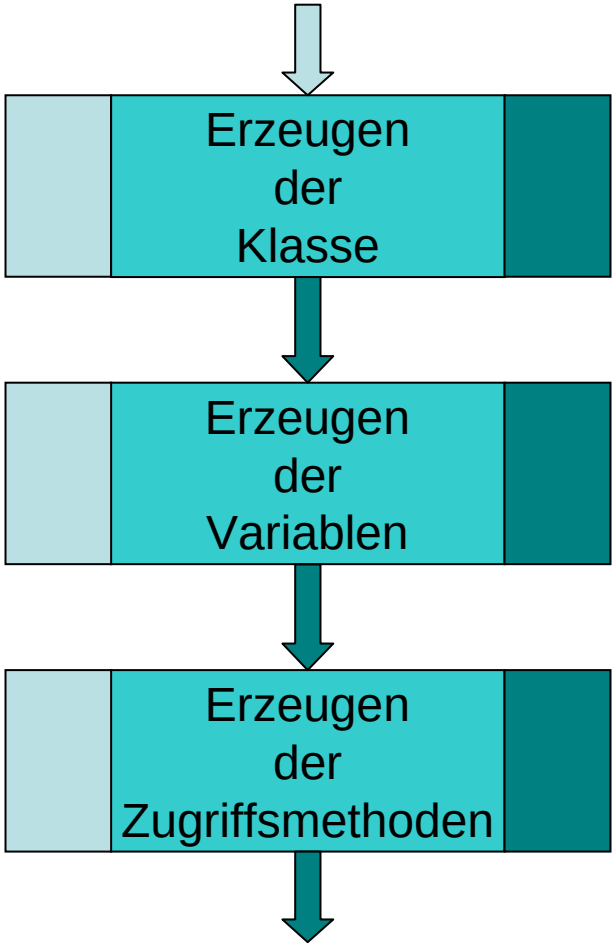
1. Analysen ob die Vorbedingung(en) der ersten Basistransformation wahr sind
   a) Ja: Initiale Formel = Analyseergebnisse
   b) Nein: Sequenz ist keine Kette
2. Verknüpfung der Nachbedingung(en) mit der Formel
3. Überprüfung ob Vorbedingung(en) der folgenden Basistransformation wahr sind
   a) Ja: Wiederholen der Schritte zwei und drei
   b) Nein:
      1) Vorbedingung(en) falsch $\Rightarrow$ Sequenz ist keine Kette
      2) Vorbedingung(en) „unbekannt“ $\Rightarrow$ Analysen
         i. Vorbedingung wahr $\Rightarrow$ Formel = Formel $\cup$ Analyseergebnisse, wiederholen der Schritte zwei und drei
         ii. Vorbedingung falsch $\Rightarrow$ Sequenz ist keine Kette

Berechnung von Bedingungen

Beispiel:

neue Klasse soll bestehendes Attribut kapseln

Sequenz:



Berechnung von Bedingungen

Initiale Formel:

Analyseergebnisse der Vorb.
Erzeugen der Klasse

istGültigerName
$\wedge$
¬erzeugtNamenskonflikt

Nachbedingungen:

Erzeugen der Klasse

neueKlasse
$\in$
{KLASSEN}

Neue Formel:

nach Erzeugen der Klasse

neueKlasse
$\in$
{KLASSEN}

Vorbedingungen:

Erzeugen der Variablen

$\exists$ Zielklasse
$\wedge$
istGültigerName
$\wedge$
¬erzeugtKonflikt

Berechnung von Bedingungen

Nachbedingungen:

Erzeugen der Variablen

$$neueVariable \in \{Zielklasse.\ VARIABLEN\}$$

Neue Formel:

nach Erzeugen der Variablen

$$neueKlasse \in \{KLASSEN\} \land neueVariable \in \{neueKlasse.VARIABLEN\}$$

Vorbedingungen:

Erzeugen (Kopieren) der Zugriffsmethoden

$$\exists\ Zielklasse \land \exists\ Zielvariable \land \exists\ Methode \land \neg erzeugtNamenskonflikt$$

Gliederung

- Verbesserung der Struktur
  - Vorbedingungen
  - Nachbedingungen
  - Basistransformationen
- Kompositionen
  - Abhängigkeiten
  - Berechnung von Bedingungen
- Nutzen und Anwendung

Nutzen und Anwendung

- Für Änderungen kann sicher gestellt werden, dass sie das Programmverhalten nicht verändern
- Änderungen können automatisiert werden
- Durch Ermittlung von Abhängigkeiten können Teile der Änderungen parallel ausgeführt werden

Vielen Dank für Ihre Aufmerksamkeit