Parallel Helper

Erkennung von Fehlermuster hinsichtlich Nebenläufigkeit, Asynchronität und Parallelisierung in C#

Christoph Amrein und Luc Bläser

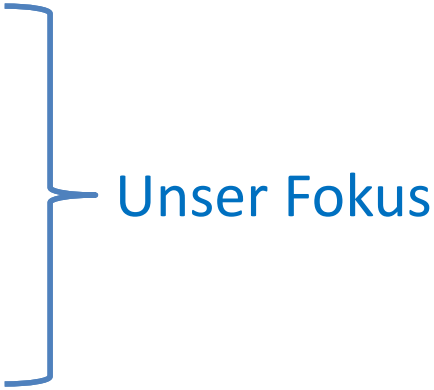
OST – Ostschweizer Fachhochschule

Nebenläufigkeitskonzepte

- Wachsendes Spektrum in Programmiersprachen
  - Multi-Threading
  - Synchronisationen
  - Atomics, Fences
  - Thread Pooling
  - Task Parallelisierung
  - Datenparallelisierung
  - Asynchrone Programmierung
- Vielfältige Fehlermöglichkeiten
  - Heimtückisch, da nicht-deterministisch

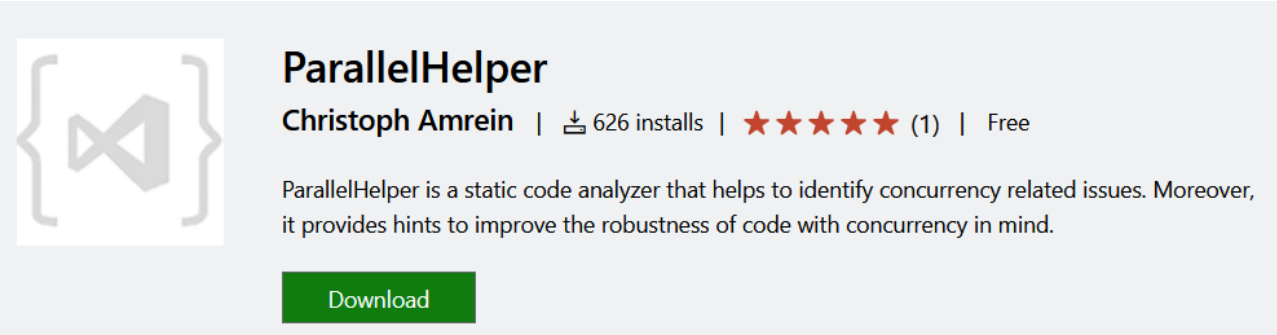
Nebenläufigkeitsfehler erkennen

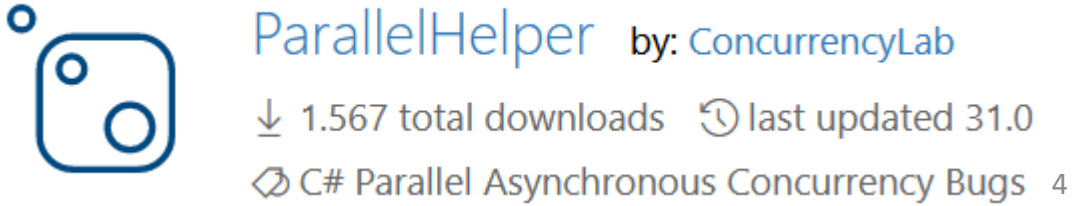
- Komplexe Fehler
  - Generelle Fehlerklassen: Race Conditions, Data Races, Deadlocks, Livelocks, Starvation
  - Umfassende Programm-Analyse, statisch oder dynamisch
  - Tradeoff: Präzision <-> Vollständigkeit <-> Analysezeit
- Bug Patterns
  - Spezifische Fehlertypen
  - Auch sprachabhängige Fallen
  - Lokaler Analyse, einfache Muster
  - Schnelle Erkennung



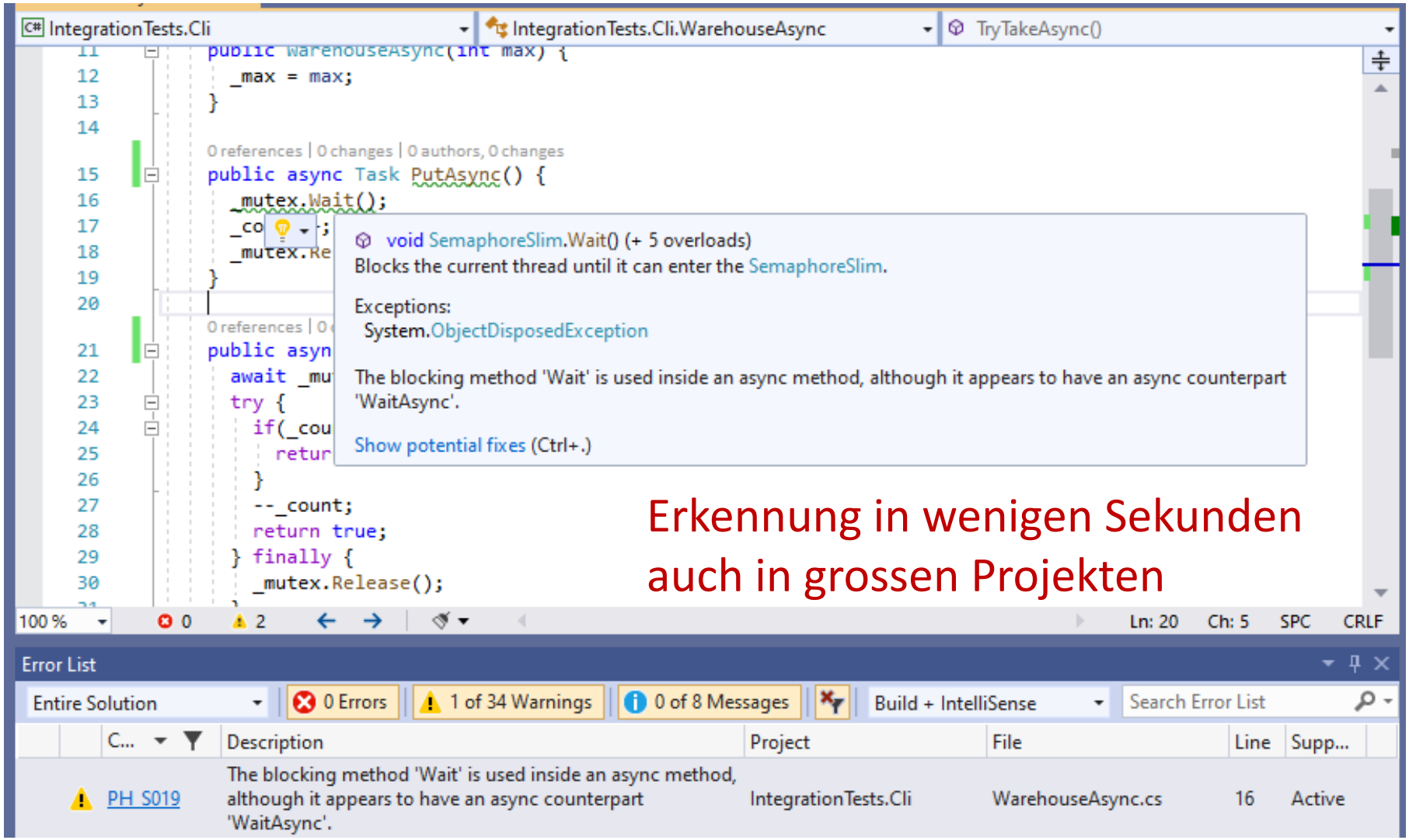
Parallel Helper

- Statische Analyse von Bug Patterns in C# 9
- Nische Concurrency, Asynchronität & Parallelisierung
- Roslyn-basierter Checker, als Plugin in Visual Studio, VS Code oder NuGet Analyzer
- Open Source, >2k Downloads kombiniert





Feedback zur Entwicklungszeit



Erkennung in wenigen Sekunden auch in grossen Projekten

Concurrency Bug Patterns

- Stetig ausgebautes Repertoire
- Aus >5 Jahren Praxisprojekten gesammelt
  - Code Reviews, Muster aus allgemeinen Analysen
- Startpunkt: Parallele Code Smells: Top 10 Liste (Artikel Heise Developer)
- Aktuell 56 Bug Patterns implementiert
- Klassifiziert nach
  - Bugs - wahrscheinlich falsch
  - Smells - tendenziell falsch
  - Bad practice - unsauber, fehleranfällig

Auswahl interessanter Muster

1. Missing Monitor Synchronization
2. Non-Atomic Read/Writes on Volatile
3. Pulse on Param-Dependent Wait
4. Fire-And-Forget Tasks
5. Timer Scheduled Upon Instantiation
6. Blocking Wait in Async

Auf andere Sprachen übertragbar (z.B. Java)

C#-spezifisch

1. Missing Monitor Synchronization

PH_B010 (Bug)

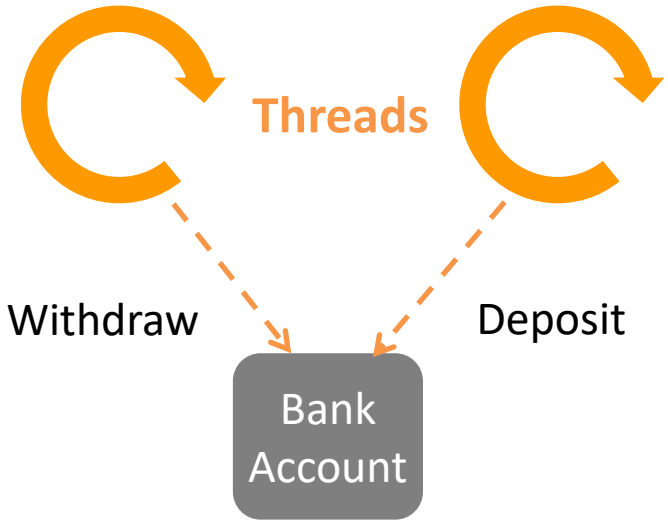
```
class BankAccount {
   private readonly object sync = new();
   private int balance;

   public void Deposit(int amount) {
      lock (sync) {
          balance += amount;
      }
   }

   public bool Withdraw(int amount) {
      if (amount > balance) return false;
      balance -= amount;
      return true;
   }
}
```

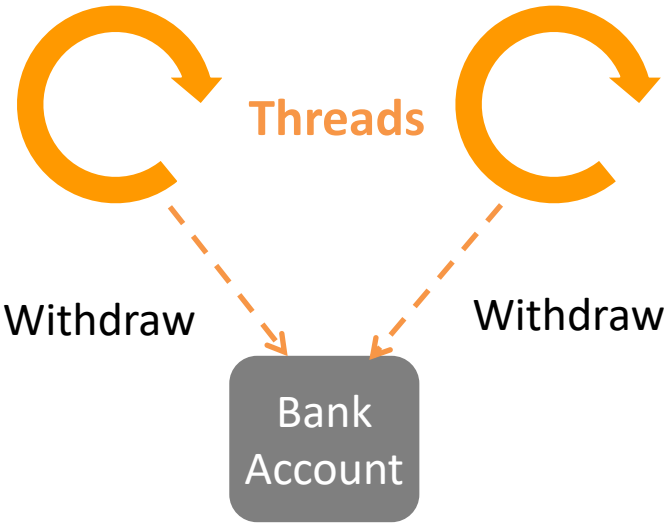
synchronisiert

unsynchronisiert

Problem: Halb Thread-Safe

- Nur nebenläufige Deposit/Deposit sind sicher
- Andere Kombinationen nicht





Data Races & Race Conditions

Musterbeschreibung

PH_B010 - Missing Monitor Synchronization

Problem

A field is accessed multiple times inside a monitor lock. However, this field is also accessed outside of a monitor lock. Since there is at least one write-access, it incorporates the risk of a race condition.

Solution

Enclose the block with a monitor lock.

Options

```
# Report volatile fields: ignore (default) / report
dotnet_diagnostic.PH_B010.volatile = ignore
```

2. Non-Atomic Read/Write on Volatile

PH_B006 (BUG)

Einzelzugriffe synchronisiert (partial fences)

```
volatile int count;

int Next() {
  return count++;
}
```

Operation aber nicht atomar

Race Conditions

3. Pulse for Param-Dependent Wait

PH_B004 (Bug)

```
void Take(int amount) {
  lock(syncObject) {
   while(amount > count) {
     Monitor.Wait(syncObject);
   }
   count -= amount;
  }
}

void Put(int amount) {
  lock(syncObject) {
    count += amount;
    Monitor.Pulse(syncObject);
  }
}
```

Parameter-variabel

Verschiedene Wartebedingungen

PulseAll nötig

4. Fire-and-Forget Tasks

PH_S004 (Smell)

```
Task.Run(() => {
   ...
});
```

Keine Continuation, kein Abwarten des Ende

1. Exceptions in Task werden ignoriert

2. Task wird allenfalls abgebrochen

(weil Background Worker Threads)

5. Timer Scheduled Upon Instantiation

PH_S008 (Smell)

Timer startet im Konstruktor

```
timer = new Timer(Tick, null, 10, Timout.Infinite);
```

```
void Tick(object arg) {
  if (timer != null) {
    timer.Dispose();
  }
}
```

Läuft evtl. nebenläufig zur Zuweisung im Konstruktor

Data Races & Race Conditions

Muster bei Auswertung von statischer Data Race Analyse-Tools erkannt [ISSTA18]

6. Blocking Wait in Async

PH_S026 (Smell)

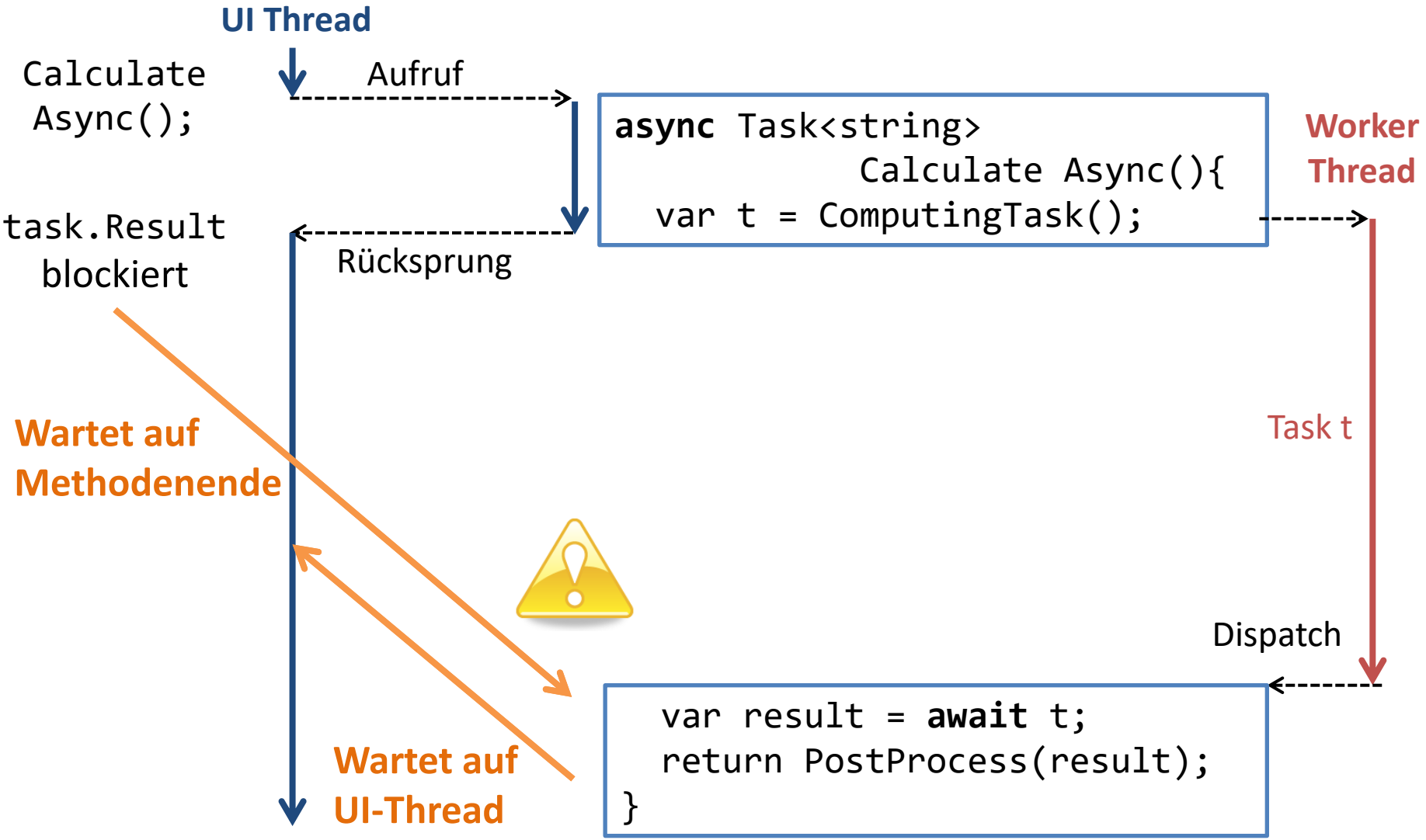
- Aufruf von Result/Wait in async Methode

```
label = CalculateAsync().Result

DownloadAsync().Wait()
```

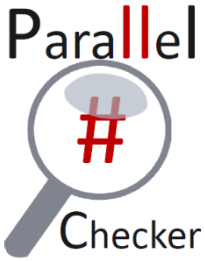
Blockierendes Warten

Deadlock in UI und ASP.NET

Deadlock Analyse



Ergänzung Parallel Checker



- Statische Analyse von allgemeinen Nebenläufigkeitsfehlern in C#
  - Data Races
  - Deadlocks
  - Thread-Unsafe API Calls
- Bounded Randomized Abstract Interpretation
- Damit in Projekten auch Fehlermuster erkannt
- >3.7k Downloads (VS Marketplace und NuGet)

[ISSTA18] L. Bläser. Practical Detection of Concurrency Issues at Coding Time. International Symposium on Software Testing and Analysis (ISSTA) 2018.

Schlussfolgerung

- Concurrency Bug Patterns als nützliches Mittel
  - Schnell erkennbar, spezifische Fallen
  - Aber beschränkt (lokale und simple Patterns)
  - Komplementär zu komplexer Programm-Analyse
- Work in Progress – weitere Ideen
  - Experimentelle Analyse
  - Automatische Refactorings/Fixes
  - Sammeln/Lernen neuer Bugs
  - Anwendung auf andere Sprachen

Weitere Informationen

- Parallel Helper
  - https://github.com/Concurrency-Lab/ParallelHelper
  - [Heise16] L. Bläser, Parallel Code Smells: Eine Hitparade, Heise Developer, Juni 2016.
- Parallel Checker
  - https://parallel-checker.com
  - [ISSTA18] L. Bläser. Practical Detection of Concurrency Issues at Coding Time. Intl. Symp. on Software Testing and Analysis (ISSTA), 2018.

Issue Overview

- 56 issues
  - 15 bugs
  - 28 smells (incontiguous numbering)
  - 13 bad practices
- * means: “very specific to .NET or C#”
- (*) means “somewhat specific to .NET or C#”
- **Bold** means “discussed in talk”

Threads / Tasks / Syncs

ID	Description
PH_P003	Discouraged Thread Method
PH_P004	CancellationToken not Passed Through (*)
PH_P007	Unused Cancellation Token (*)
PH_S004	**Fire-and-Forget Tasks**
PH_S007	Thread Start in Constructor
PH_S011	Invalid use of Non-Blocking Collection
PH_P008	Missing OperationCanceledException in Task
PH_P012	Prefer Slim Synchronization *

Race Conditions

ID	Description
PH_B003	Unsynchronized collection access
PH_B006	**Non-Atomic Read/Write on Volatile Field**
PH_B007	Non-Atomic Access to Concurrent Collection
PH_B008	LINQ To* Operation on ConcurrentDictionary *
PH_S008	**Timer Scheduled upon Instantiation**
PH_S009	PLINQ Side-Effects
PH_S010	Parallel.For Side-Effects (*)

Monitor

ID	Description
PH_B001	Nested Monitor Locks
PH_B002	Monitor Lock inside Different Nested Locks
PH_B004	**Monitor Wait with Parameter-Dependent While Loop**
PH_B005	Monitor Signal Without Conditional Loop Effect
PH_B009	**Missing Monitor Synchronization (multiple fields)**
PH_B010	Missing Monitor Synchronization (single field)
PH_S002	Discouraged Sync Object
PH_S003	SyncObject Change
PH_S016	Monitor.Pulse with multiple Monitor.Wait
PH_S017	Monitor.Pulse is used in combination with Monitor.PulseAll
~~PH_S018~~	~~Monitor.Pulse with multiple Monitor.Wait (deprecated)~~

Monitor (cont.)

ID	Description
PH_S022	Parallel.For with Monitor Synchronization
PH_S023	Monitor Lock in Async Method *
PH_S027	Leaked Outbound Collection
PH_S028	Leaked Inbound Collection
PH_S029	ThreadLocal in Async Method *
PH_S031	Monitor.Wait Inside Async Method or Task *
PH_P001	Writeable SyncObject
PH_P002	Monitor.Wait without Conditional Loop
PH_P006	Discouraged Monitor Method

Async

ID	Description
PH_B011	Returning a Task based on a Disposed Value
PH_B012	Multiple Awaits on the Same ValueTask *
PH_B013	Null-Check Against Task Instead of Value
PH_B014	Await On Conditional Access
PH_B015	Disposed Task Instead of Value
PH_S005	Fake Async Methods
PH_S006	Unnecessarily Async Methods
PH_S012	Task only returning value
PH_S013	Thread.Sleep in Async Method
PH_S014	Task.Factory.StartNew with async delegate *
PH_S019	Blocking Method in Async Method
PH_S020	Awaiting Synchronous Task Completions

Async (cont.)

ID	Description
PH_S021	Async Naming Confusion *
PH_S025	Unused Synchronous Task Result
PH_S026	**Blocking Wait in Async Method ***
PH_S030	Async Void Method Invocation (*)
PH_S032	Throws in Potentially Async Method *
PH_P005	Blocking Wait on Async Method *
PH_P009	Synchronous Dispose in Async Method
PH_P010	Async Instead of Continuation (*)
PH_P011	Replace With Async Stream *
PH_P013	Discouraged EntityFramework Method *