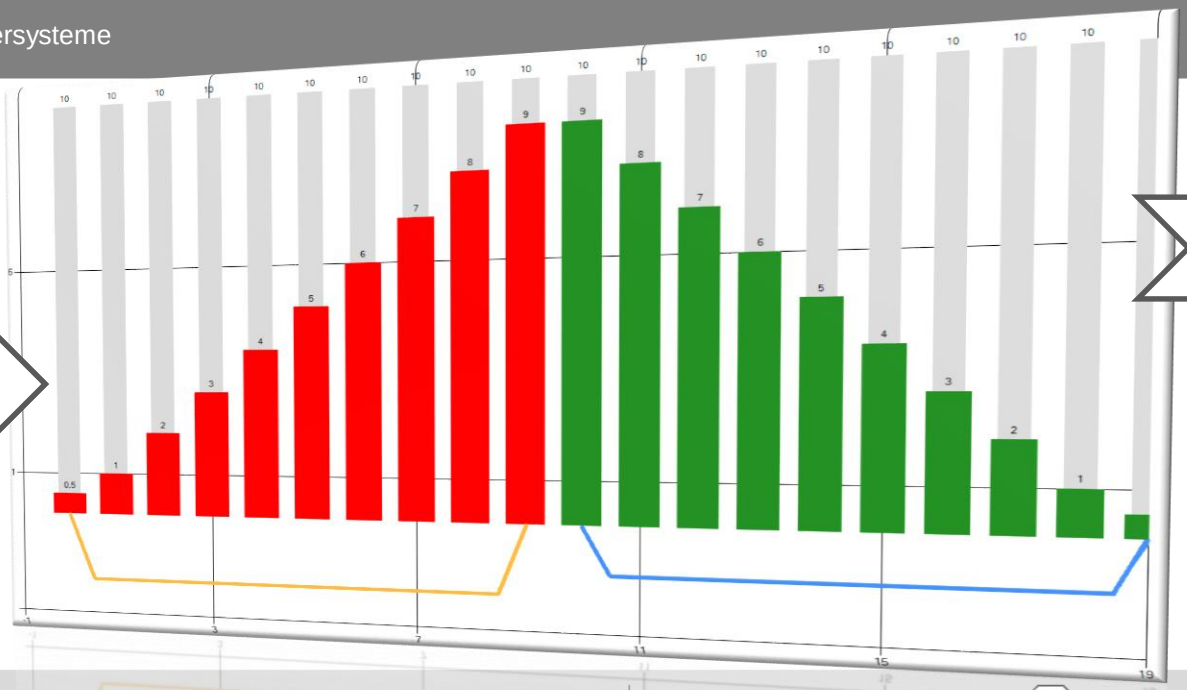
# Werkzeugunterstützte Ortung von Parallelisierungspotenzial

**Diplomarbeit von Alexander Bieleš**

**Betreuer: Korbinian Molitorisz, Thomas Karcher**

IPD Tichy – Lehrstuhl für Programmiersysteme

```
int[] arr = new int[10];

for (int i=0; i<10; i++)
    arr[i]=i;

for (int i=9; i>=0; i--)
    Debug.Write(arr[i]);

//9876543210
```



# ein nicht bekannter Quelltext…

```
namespace MandelbrotDemo
{
   public class MandelbrotEquation
   {
      public static int Solve(double x0,
                              double y0, int maxIterations)
      {
         int iteration = 0;
         double x = 0;
         double y = 0;
         while (x * x + y * y <= 4
                && iteration < maxIterations)
         {
            double xtemp = x * x - y * y + x0;
            y = 2 * x * y + y0;
            x = xtemp;
            iteration++;
         }
         return iteration;
      }
   }
}
```

```
      w.Start();
      Initialize();
      long timeInit = w.ElapsedTicks;
      w.Reset();
      w.Start();
      maxIter = Solver.Solve(x, y, maxIter);
      long timeSolve = w.ElapsedTicks;
      // Trace.WriteLine(t1 + "/" + t2 + "=" + (double)t1 / t2);
   }

   /// <summary>
   /// Initialize the x/y arrays (using a Parallel.For loop)
   /// </summary>
   private void Initialize()
   {
      //Stopwatch sw = Stopwatch.StartNew();
      x = new double[Width * Height];
      y = new double[Width * Height];

      double dx = (X1 - X0) / (Width - 1);
      double dy = (Y1 - Y0) / (Height - 1);
      double x0 = X0;
      double y0 = Y0;
      int w = Width;

#if ParallelInitialization
      Parallel.For(0, Height, j =>
#else
      for (int j=0;j<Height;j++)
#endif
      {
         int ij = j * w;
         for (int i = 0; i < w; i++)
         {
            x[ij] = x0 + dx * i;
            y[ij++] = y0 + dy * j;
         }
      }
#if ParallelInitialization
      );
#endif

      //sw.Stop();
      //Debug.WriteLine(sw.ElapsedMilliseconds);
   }

   public void Zoom(double u, double v, double f)
   {
      double x = X0 + (X1 - X0) * u;
      double y = Y0 + (Y1 - Y0) * v;
      X0 += (x - X0) * f;
      X1 += (x - X1) * f;
      Y0 += (y - Y0) * f;
      Y1 += (y - Y1) * f;
   }

   public void Pan(double du, double dv)
   {
      double w = X1 - X0;
      double h = Y1 - Y0;
      X0 += w * du;
      X1 += w * du;
      Y0 += h * dv;
      Y1 += h * dv;
   }

}
}
```

```
   // .NET 4.0
   // var rangePartitioner = Partitioner<int>.Create(0, source.Length);
   Parallel.For(0, i => result[i] = MandelbrotEquation.Solve(x[i], y[i], maxit));
   return result;
}
}

public class Parallel3MandelbrotSolver : IEquationSolver
{
   int parts = 8;
   public int[] Solve(double[] x, double[] y, int maxit)
   {
      int n = x.Length;
      var result = new int[n];
      int[] l = new int[n];
      for (int i = 0; i < parts - 1; i++)
         l[i] = i * n / parts;
      l[parts - 1] = n;

      Parallel.For(0, parts - 1, p =>
      {
         for (int i = l[p]; i < l[p + 1]; i++)
            result[i] = MandelbrotEquation.Solve(x[i], y[i], maxit);
      });
      return result;
   }
}

/// <summary>
/// Interface for the solvers
/// </summary>
public interface IEquationSolver
{
   int[] Solve(double[] x, double[] y, int maxit);
}

/// <summary>
/// Solving the equation in a non-parallel loop
/// </summary>
public class MandelbrotSolver : IEquationSolver
{
   public int[] Solve(double[] x, double[] y, int maxit)
   {
      int n = x.Length;
      var result = new int[n];
      for (int i = 0; i < n; i++)
         result[i] = MandelbrotEquation.Solve(x[i], y[i], maxit);
      return result;
   }

}
}
```

```
   var handler = PropertyChanged;
   if (handler != null)
      handler(this, new PropertyChangedEventArgs(property));
}
#endregion

private void Image_MouseWheel(object sender, MouseWheelEventArgs e)
{
   // zoom the image
   var img = (Image)sender;
   var pos = e.GetPosition(sender as Image);
   frame.Zoom(pos.X / img.ActualWidth, pos.Y / img.ActualHeight, e.Delta * 0.001);
   UpdateFrame();
}

Point lastPoint;
private void Image_MouseDown(object sender, MouseButtonEventArgs e)
{
   var img = (Image)sender;
   img.CaptureMouse();
   lastPoint = e.GetPosition(img);
}

private void Image_MouseMove(object sender, MouseEventArgs e)
{
   var img = (Image)sender;
   if (img.IsMouseCaptured)
   {
      // pan the image
      var pt = e.GetPosition(img);
      frame.Pan((lastPoint.X - pt.X) / img.ActualWidth, (lastPoint.Y - pt.Y) / img.ActualHeight);
      lastPoint = pt;
      UpdateFrame();
   }

}

private void Image_MouseUp(object sender, MouseButtonEventArgs e)
{
   var img = sender as Image;
   img.ReleaseMouseCapture();
}

private void Grid1_SizeChanged(object sender, SizeChangedEventArgs e)
{
   // Update when the window has been resized
   UpdateFrame();
}

protected override void OnKeyDown(KeyEventArgs e)
{
   base.OnKeyDown(e);
   if (e.Key == Key.A)
      Zoom(0.2);
   if (e.Key == Key.Z)
      Zoom(-0.2);
}

void Zoom(double p)
{
   frame.Zoom(0.5, 0.5, p);
   UpdateFrame();
}

}
}
```

```
public static BitmapPalette CreateGradientPalette(int n1, Color c1, Color c2, Color c3)
{
   Color[] colors = new Color[256];
   for (int i = 0; i < n1; i++)
      colors[i] = BlendColor(c1, c2, (double)i / n1);
   for (int i = n1; i < 255; i++)
      colors[i] = BlendColor(c2, c3, (double)(i - n1) / (255 - n1));
   colors[0] = Colors.Black;

   return new BitmapPalette(colors);
}

/// <summary>
/// Blend two colors (1-p) of color1 and (p) of color2
/// </summary>
/// <param name="color1"></param>
/// <param name="color2"></param>
/// <param name="p"></param>
/// <returns></returns>
private static Color BlendColor(Color color1, Color color2, double p)
{
   return Color.FromArgb(255,
                         (byte)(color1.R * (1 - p) + color2.R * p),
                         (byte)(color1.G * (1 - p) + color2.G * p),
                         (byte)(color1.B * (1 - p) + color2.B * p));
}

public static BitmapPalette CreateBlusihPalette(byte rotate)
{
   return CreatePalette(BluishPalette, rotate);
}
//         var p = CreateGradientPalette(180, Colors.DarkBlue, Colors.White, Colors.Y
//         var p = CreateGradientPalette(180, Colors.DarkBlue, Colors.Cyan, Colors.W

static readonly byte[,] BluishPalette = {
                        {0, 0, 7, 100},
                        {40, 32, 107, 203},
                        {107, 237, 255, 255},
                        {163, 255, 170, 0},
                        {218, 0, 0, 0}
                     };

private static BitmapPalette CreatePalette(byte[,] c, byte rotate)
{
   Color[] colors = new Color[256];
   int b = c.GetUpperBound(0)+1;
   for (int i = 0; i < b; i++)
   {
      int i2 = (i + 1) % b;
      var c1 = Color.FromArgb(255, c[i, 1], c[i, 2], c[i, 3]);
      var c2 = Color.FromArgb(255, c[i2, 1], c[i2, 2], c[i2, 3]);
      int n0 = c[i, 0];
      int n1 = c[i2, 0];
      if (n1 == 0) n1 = 256;
      int n = n1 - n0;
      for (int j = n0; j < n1; j++)
      {
         int jr = (j + rotate)%255;
         colors[jr] = BlendColor(c1, c2, (double)(j-n0) / n);
      }
   }
   colors[0] = Colors.Black;
   return new BitmapPalette(colors);
}

}
}
```

# ein nicht bekannter Quelltext…

```
            // .NET 4.0
            // var rangePartitioner = Partitioner<int>.Create(0, source.Length);
            Parallel.For(0, n, i => result[i] = MandelbrotEquation.Solve(x[i], y[i], maxit));
            return result;
        }
    }

    public class Parallel3MandelbrotSolver : IEquationSolver
    {
        int parts = 8;
        public int[] Solve(double[] x, double[] y, int maxit)
        {
            int n = x.Length;
            var result = new int[n];
            int[] l = new int[n];
            for (int i = 0; i < parts - 1; i++)
                l[i] = i * n / parts;
            l[parts - 1] = n;

            Parallel.For(0, parts - 1, p =>
            {
                for (int i = l[p]; i < l[p + 1]; i++)
                    result[i] = MandelbrotEquation.Solve(x[i], y[i], maxit);
            });
            return result;
        }
    }
/// <summary>
    /// Interface for the solvers
    /// </summary>
    public interface IEquationSolver
    {
        int[] Solve(double[] x, double[] y, int maxit);
    }

    /// <summary>
    /// Solving the equation in a non-parallel loop
    /// </summary>
    public class MandelbrotSolver : IEquationSolver
    {
        public int[] Solve(double[] x, double[] y, int maxit)
        {
            int n = x.Length;
            var result = new int[n];
            for (int i = 0; i < n; i++)
                result[i] = MandelbrotEquation.Solve(x[i], y[i], maxit);
            return result;
        }

    }
}
```

# ein nicht bekannter Quelltext…

Parallelisierungspotenzial

```
/// <summary>
/// Solving the equation in a non-parallel loop
/// </summary>
public class MandelbrotSolver : IEquationSolver
{
    public int[] Solve(double[] x, double[] y, int maxit)
    {
        int n = x.Length;
        var result = new int[n];
        for (int i = 0; i < n; i++)
            result[i] = MandelbrotEquation.Solve(x[i], y[i], maxit);
        return result;
    }

}
}
```

Ziel der Diplomarbeit ist

den Nutzer an parallelisierungsrelevante Codestellen heranführen,

indem das Laufzeitverhalten von Datenstrukturen

in objektorientierten, sequentiellen Programmen analysiert wird.

# Verwandte Arbeiten

Zhang – “Optimizing Data Layouts for Parallel Computation on Multicores”

Rane – “Performance optimization of data structures using memory access characterization”

Jung – “Brainy: effective selection of data structures”

Software-Visualisierung

| | Zhang | Rane | Jung | SW-Visualisierung | diese Arbeit |
|---|---|---|---|---|---|
| zeitlicher Verlauf von skalaren Messgrößen | ○ | + | – | + | ○ |
| zeitlicher Verlauf der Zugriffe | + | – | – | ○ | + |
| Ableiten von Optimierungspotenzial im Quelltext | – | – | + | – | + |

# Anforderungen

Eingreifbarkeit des Nutzers

→ Instrumentierung des Quelltextes

Datenstrukturkompatibilität

→ Einsatz von Stellvertretern

Warum keine Protokollierungsaufrufe vor Zugriffen?

→ Stellvertreter entkoppeln Instrumentierung von Protokollierung

Unverändertes Nutzungsverhalten

→ Quelltext wird kompiliert

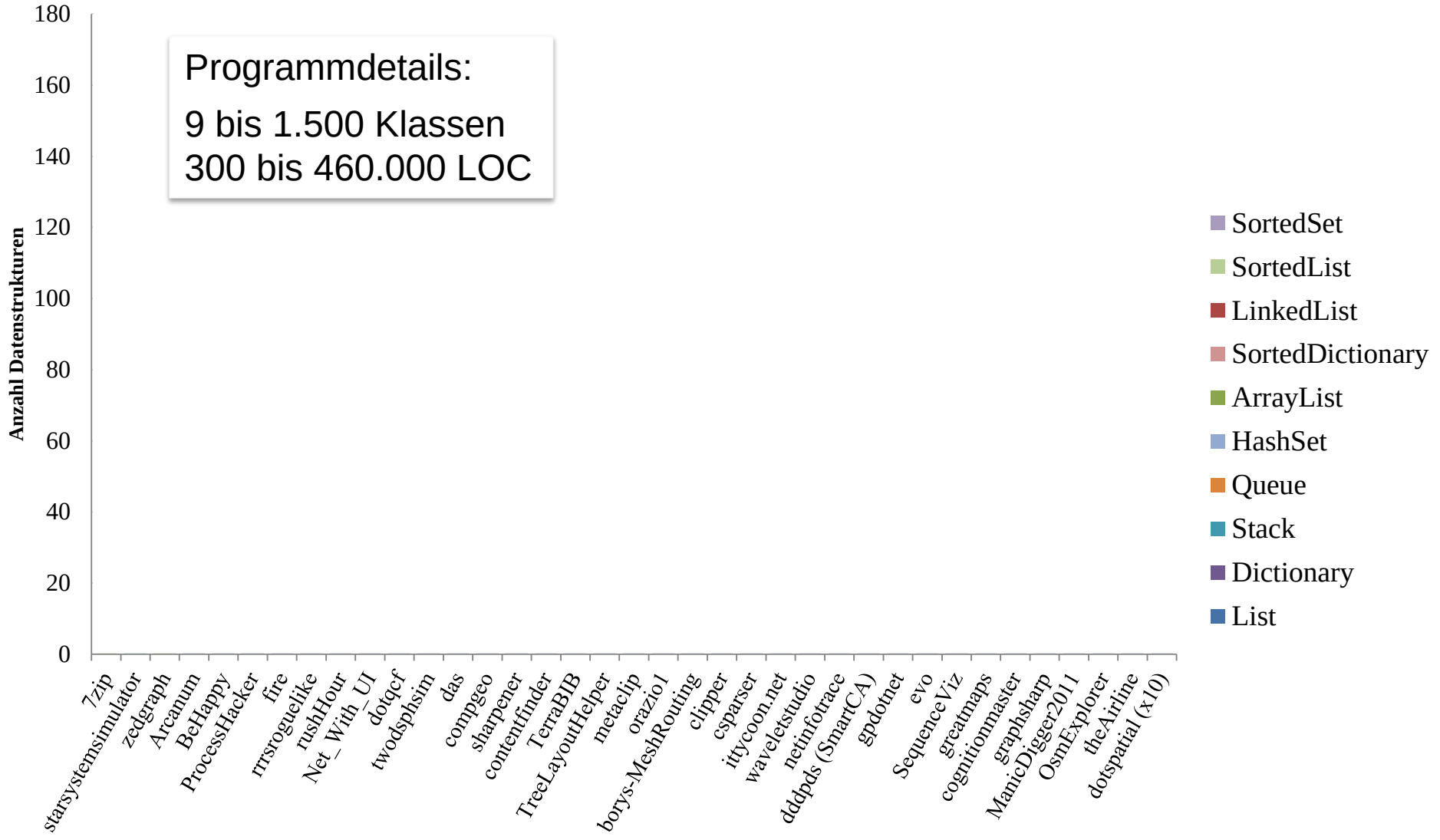
# VORUNTERSUCHUNGEN

# Voruntersuchungen

**Welche Datenstrukturen** werden von Programmierern verwendet?

Treten **Regelmäßigkeiten** in Historien **wiederkehrend** auf?

# Voruntersuchung – welche Datenstrukturen?



# Voruntersuchung – welche Datenstrukturen?

**List<T>** mit fast 1.300 Vorkommen

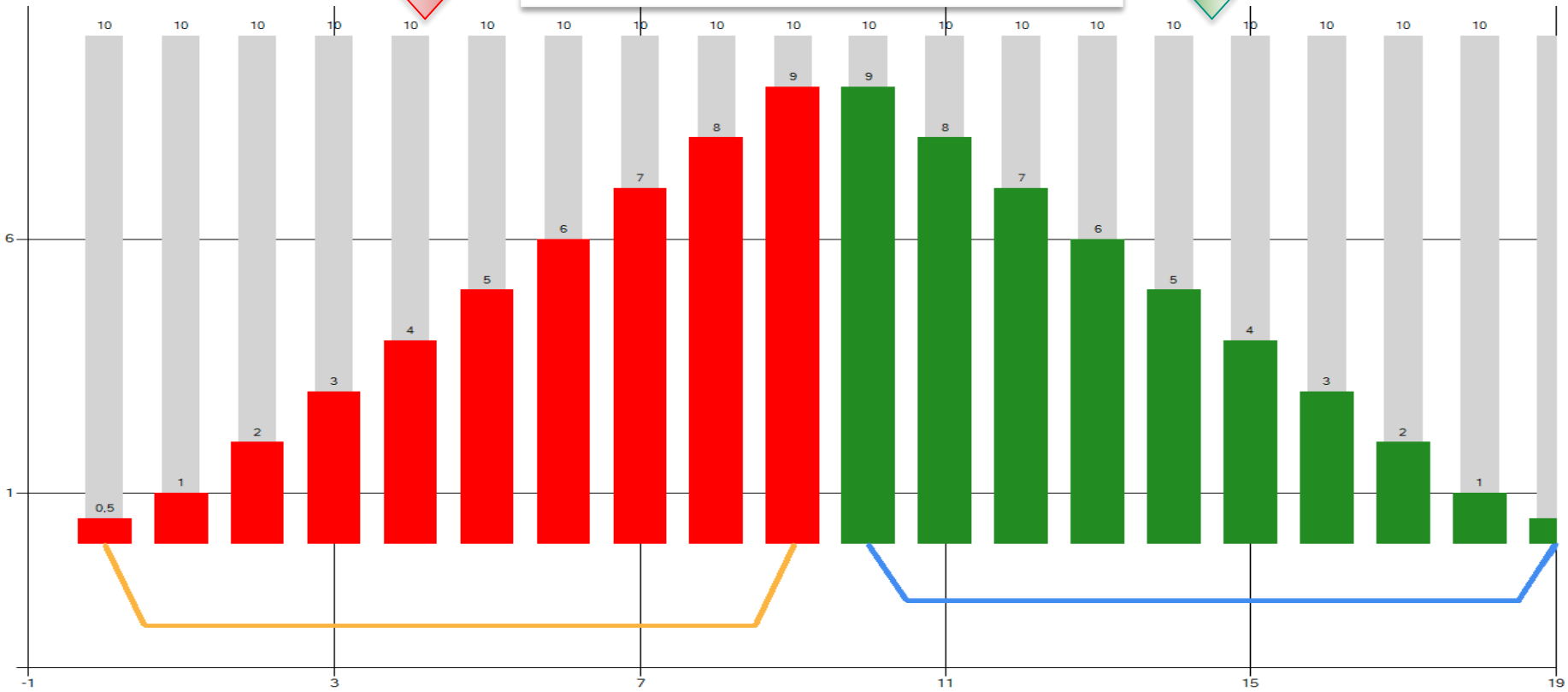
vier Mal häufiger als zweithäufigste Datenstruktur (Dictionary)

Durchschnittlich enthält jede dritte Klasse eine List

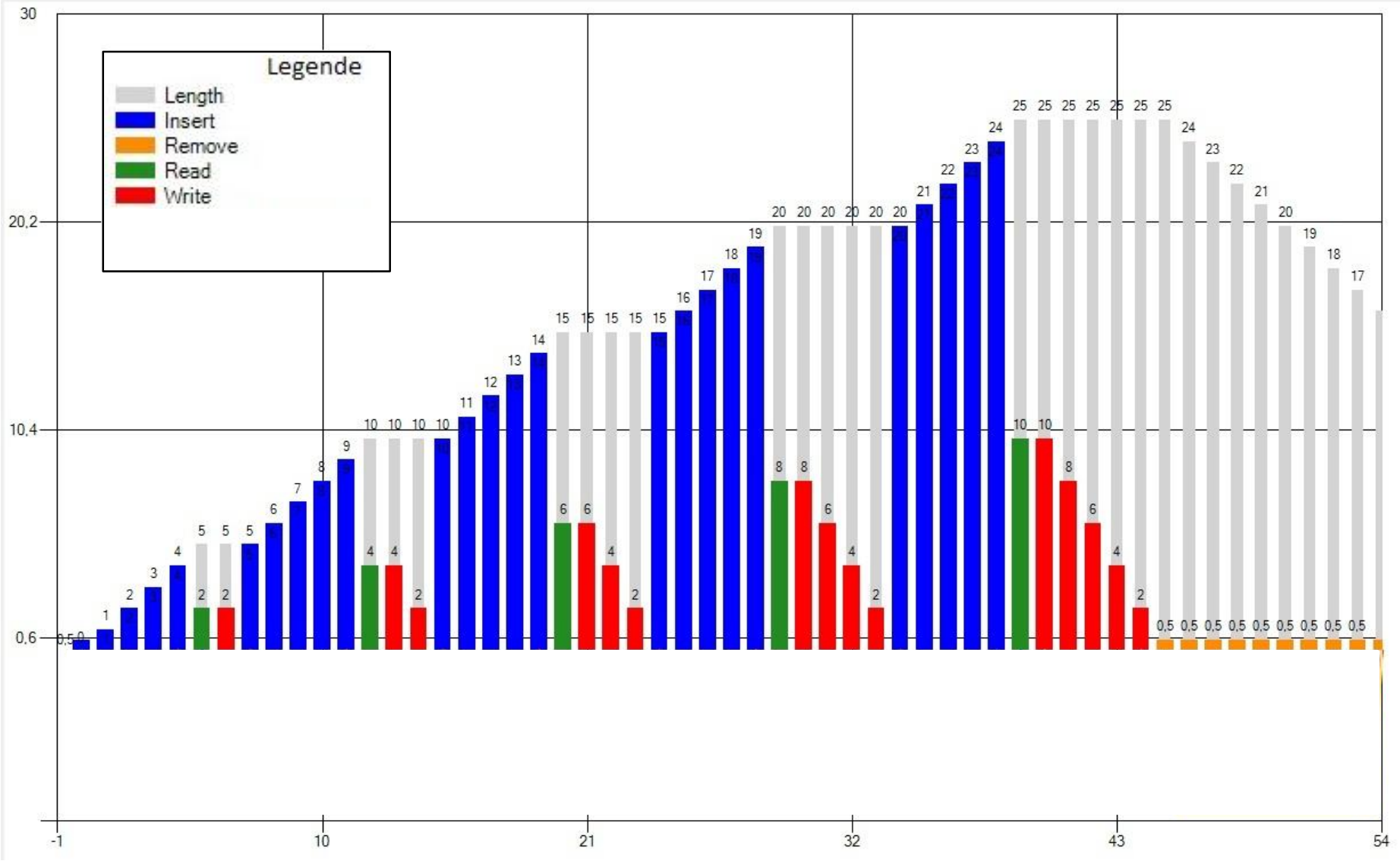
sieben Mal mehr als das Dictionary

# Voruntersuchung – Regelmäßigkeiten?

```
int[] arr = new int[10];
for (int i=0; i<10; i++)
    arr[i]=i;
for (int i=9; i>=0; i--)
    Debug.Write(arr[i]);
```



# Voruntersuchung – Regelmäßigkeiten?



# Voruntersuchung – Regelmäßigkeiten?

| Anzahl Historien… Programm | mit erkennbaren Regelmäßigkeiten | mit redundanten Regelmäßigkeiten | ohne verwertbare Regelmäßigkeiten |
|---|---|---|---|
| **astrogrep** | | 2 | 1 |
| **borys-MeshRouting** | 4 | 3 | 7 |
| **clipper** | 3 | 9 | 1 |
| **compgeo** | 2 | | |
| **contentfinder** | | 2 | |
| **csparser** | 2 | 5 | 3 |
| **‚dsa‘** | | 5 | |
| **dotqcf** | 4 | 2 | |
| **fire** | 1 | 1 | 1 |
| **ManicDigger2011** | 1 | 6 | 7 |
| **MidiSheetMusic** | 4 | 14 | |
| **Net_With_UI** | 3 | 11 | 3 |
| **netinfotrace** | 4 | 13 | 4 |
| **rrrsroguelike** | 1 | 1 | |
| **TerraBIB** | 2 | 1 | 1 |
| **TreeLayoutHelper** | | 6 | 2 |
| ∑ | **31** | **81** | **30** |

# Voruntersuchungen – Zusammenfassung

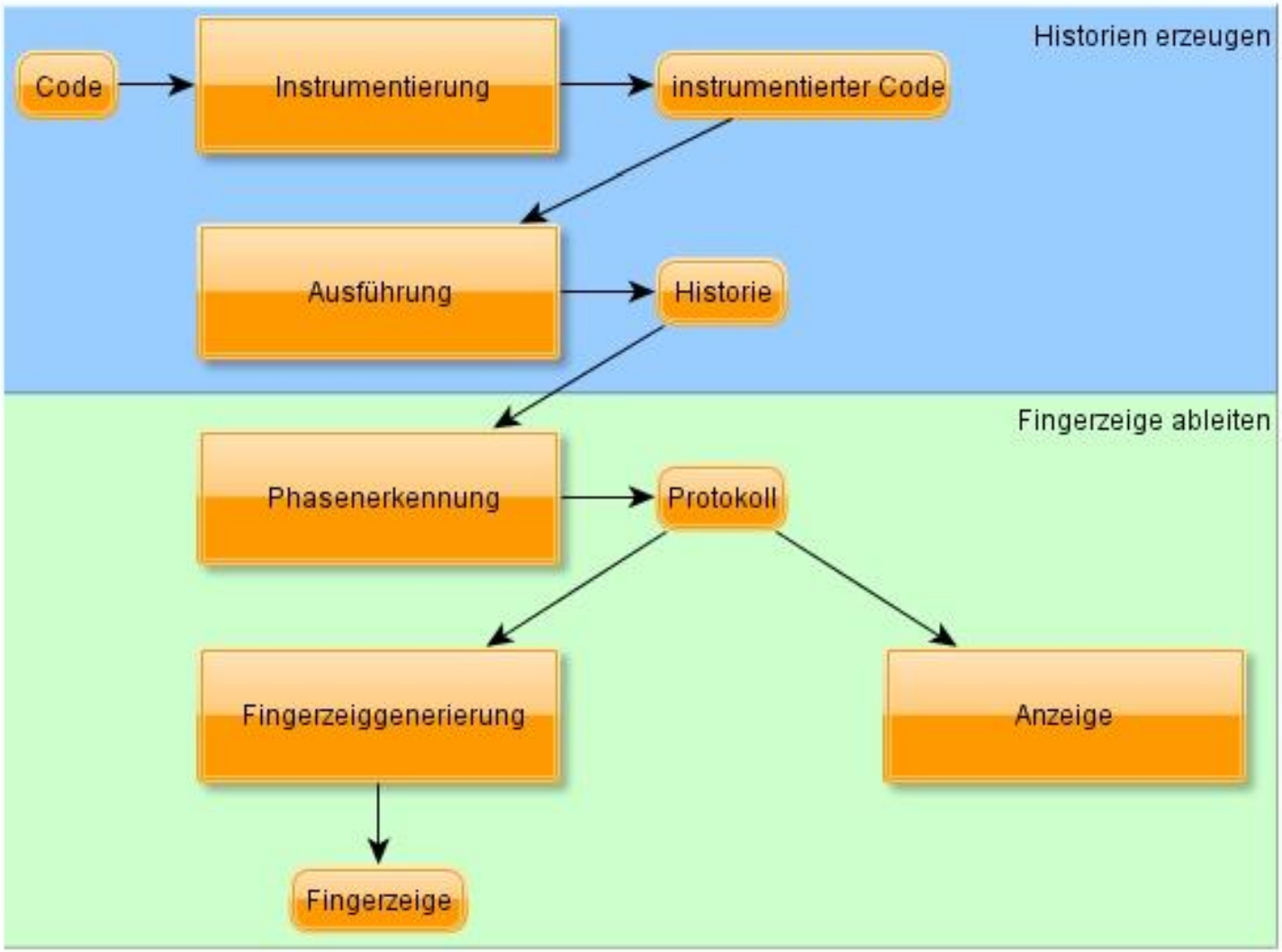
Regelmäßigkeiten in Historien erkennbar

Regelmäßigkeiten kommen wiederkehrend vor

List und Array am meisten benutzt

# KONZEPT

# Lösungsansatz



# Instrumentierung

```
using System;
using System.Collections.Generic;
using System.Linq;
using MyCollectionNS;

namespace BeispielNS {
    class Program {

        private   List<int> l;

          List<int> methode1(  List<int> eingabe) {
              List<   string[]    > liste = new   List<   string[]   >();
            l = methode2(0).ToList();
            return l;
        }

        int[]        methode2(int x) {
            return new int[arr.Length];
        }
}}
```

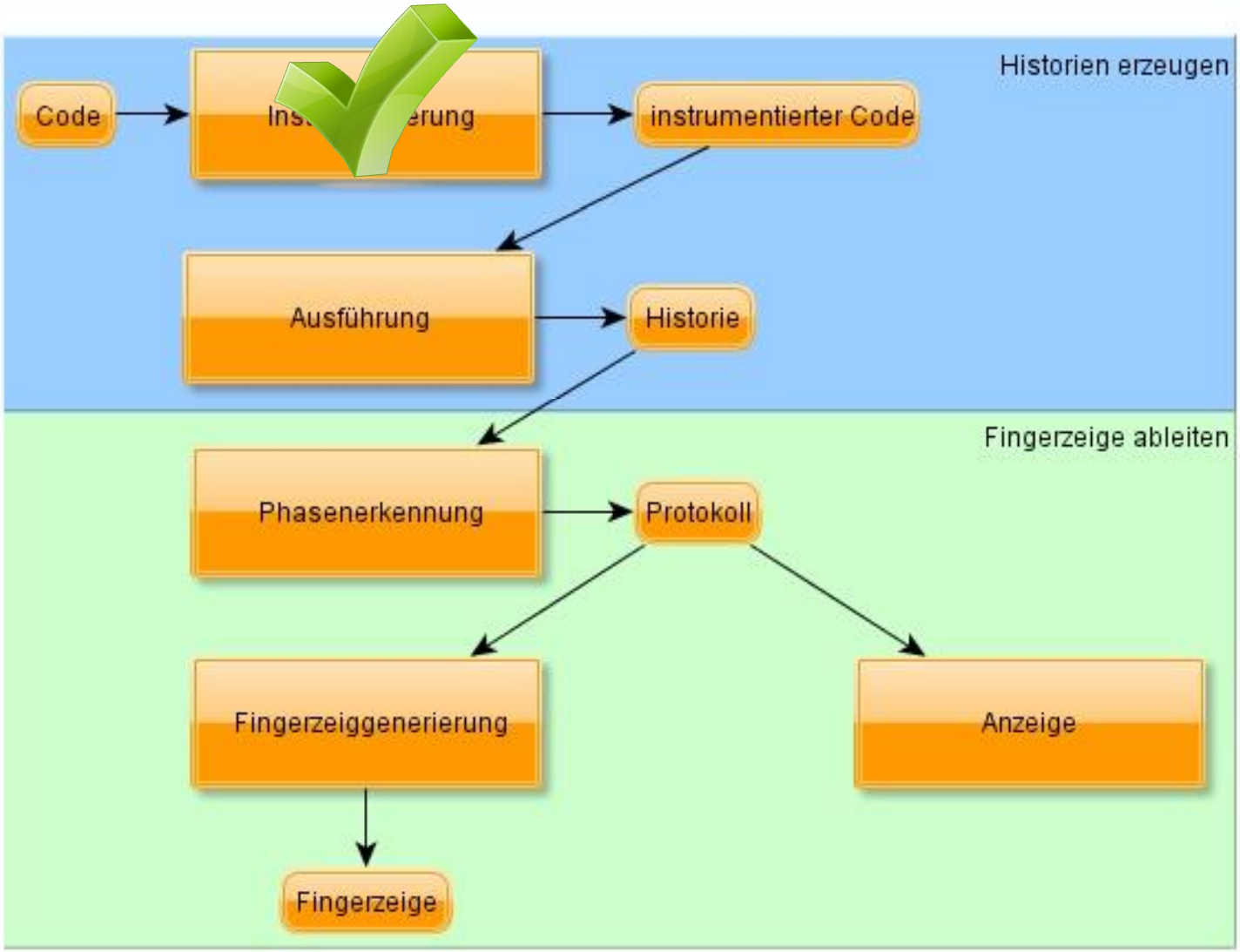
# Instrumentierung

```
using System;
using System.Collections.Generic;
using System.Linq;
using MyCollectionNS;

namespace BeispielNS {
    class Program {

        private   List<int> l;

          List<int> methode1(List<int> eingabe) {
              List<MyArray<string>> liste = new   List<MyArray<string>>();
            l = methode2(0).ToList();
            return l;
        }

        MyArray<int> methode2(int x) {
            return new int[arr.Length];
        }
}}
```

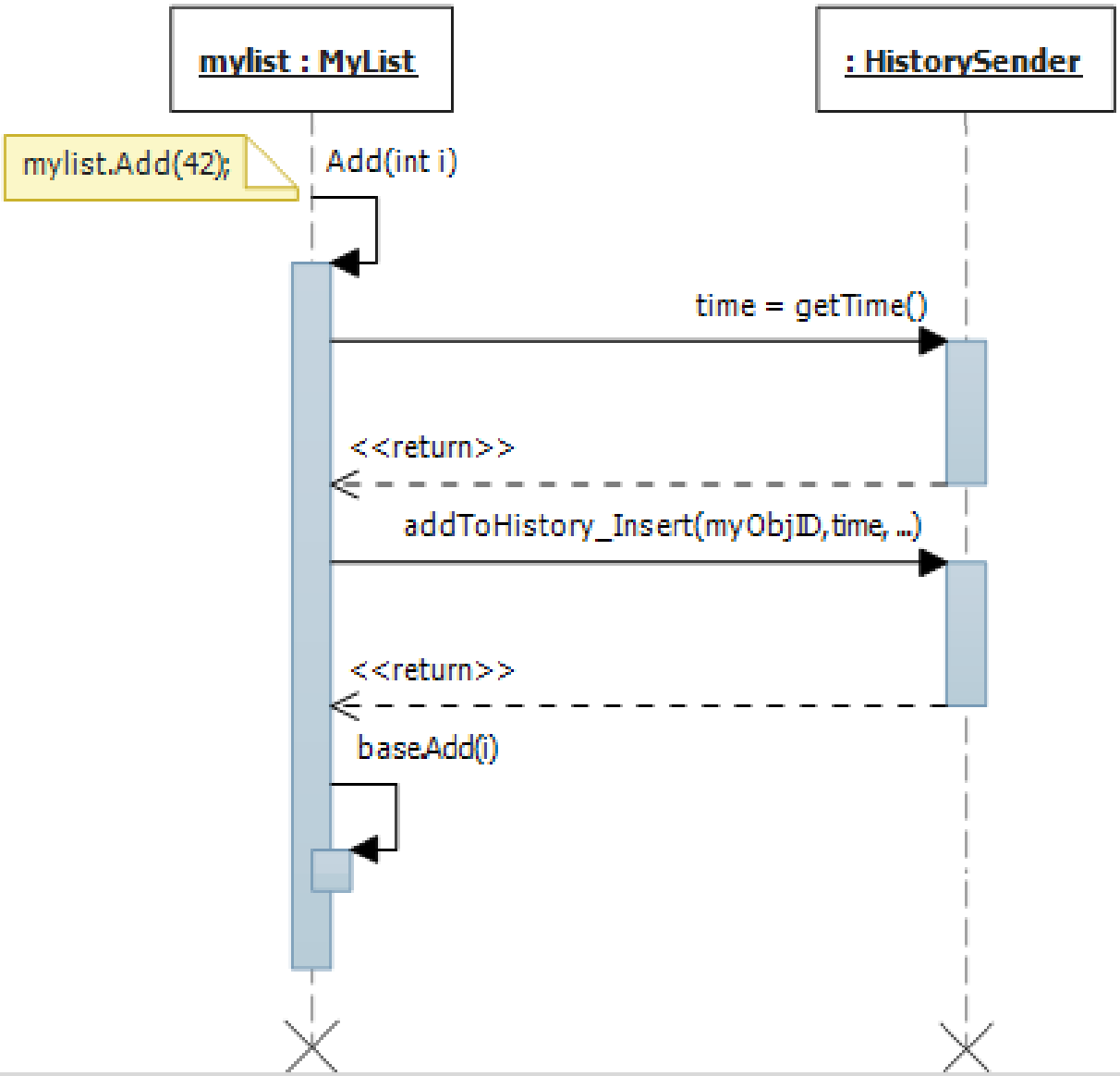
# Instrumentierung

```
using System;
using System.Collections.Generic;
using System.Linq;
using MyCollectionNS;

namespace BeispielNS {
    class Program {

        private MyList<int> l;

        MyList<int> methode1(MyList<int> eingabe) {
            MyList<MyArray<string>> liste = new MyList<MyArray<string>>();
            l = new MyList<int>(methode2(0).ToList());
            return l;
        }

        MyArray<int> methode2(int x) {
            return new int[arr.Length];
        }
}}
```

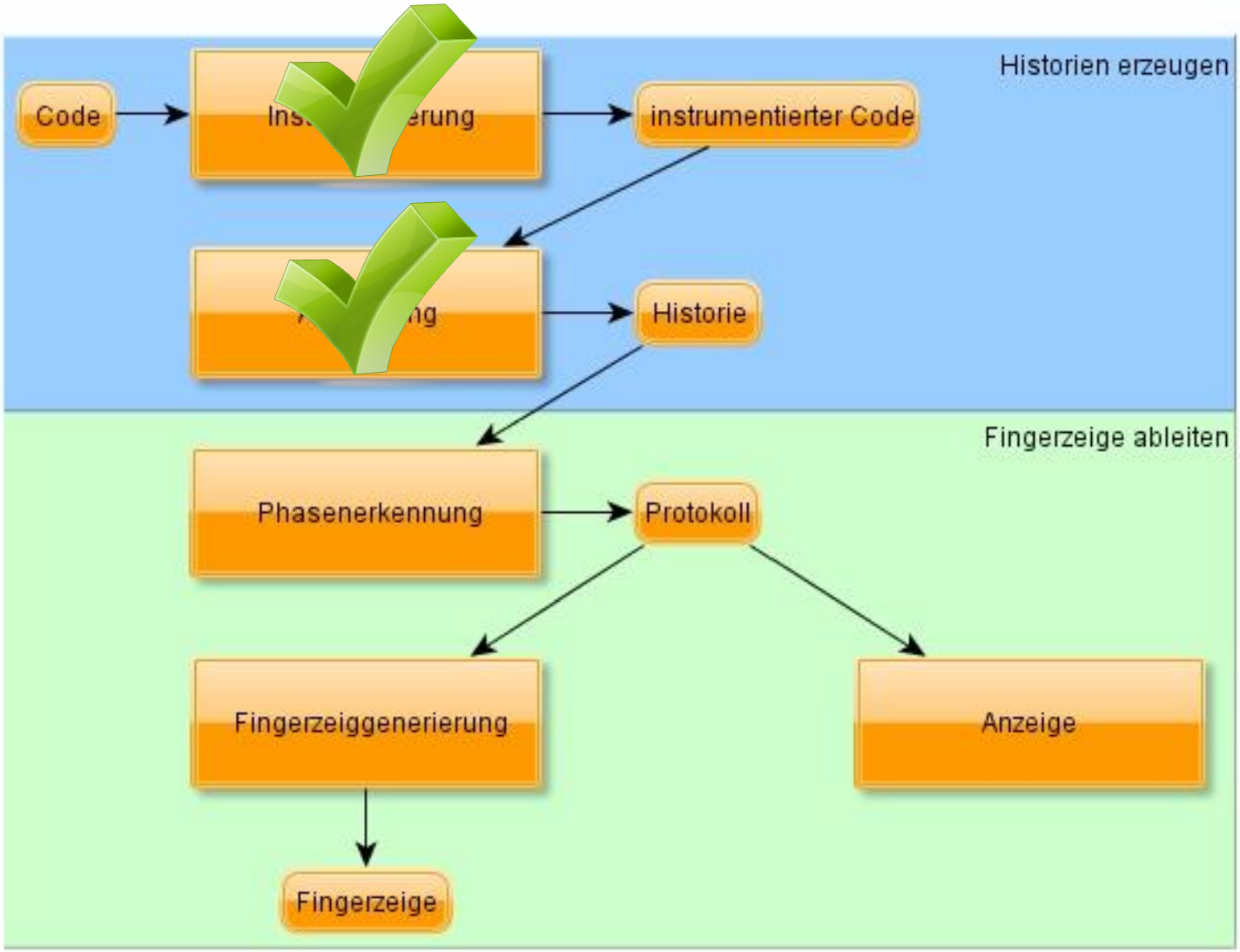
# Lösungsansatz



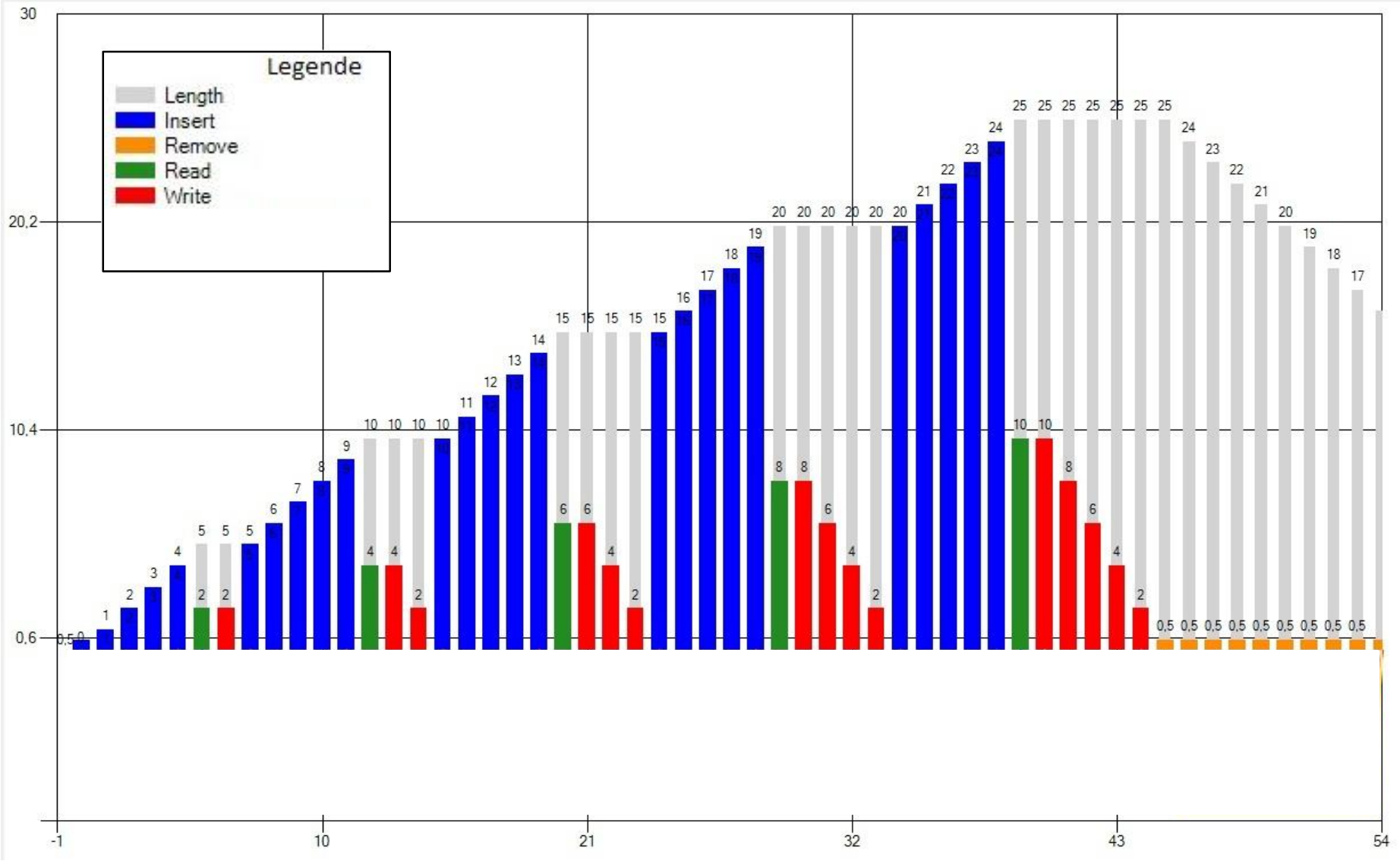
# Protokollierung durch Stellvertreter



# Lösungsansatz



# Phasenerkennung



# Phasencharakteristikum

Lineares-Lesen-Vorwärts

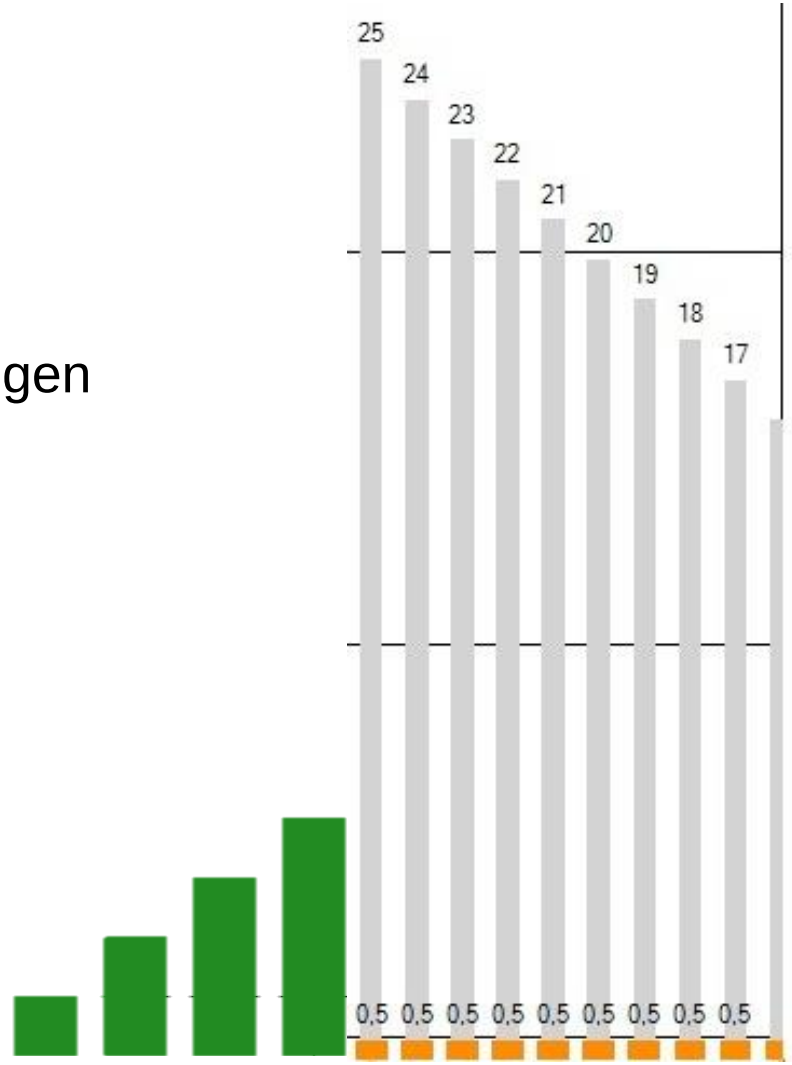
Zugriffe gruppieren wenn:

- ✓ direkt benachbart
- ✓ lesend
- ✓ Zugriffsindizes mit der Zeit steigen

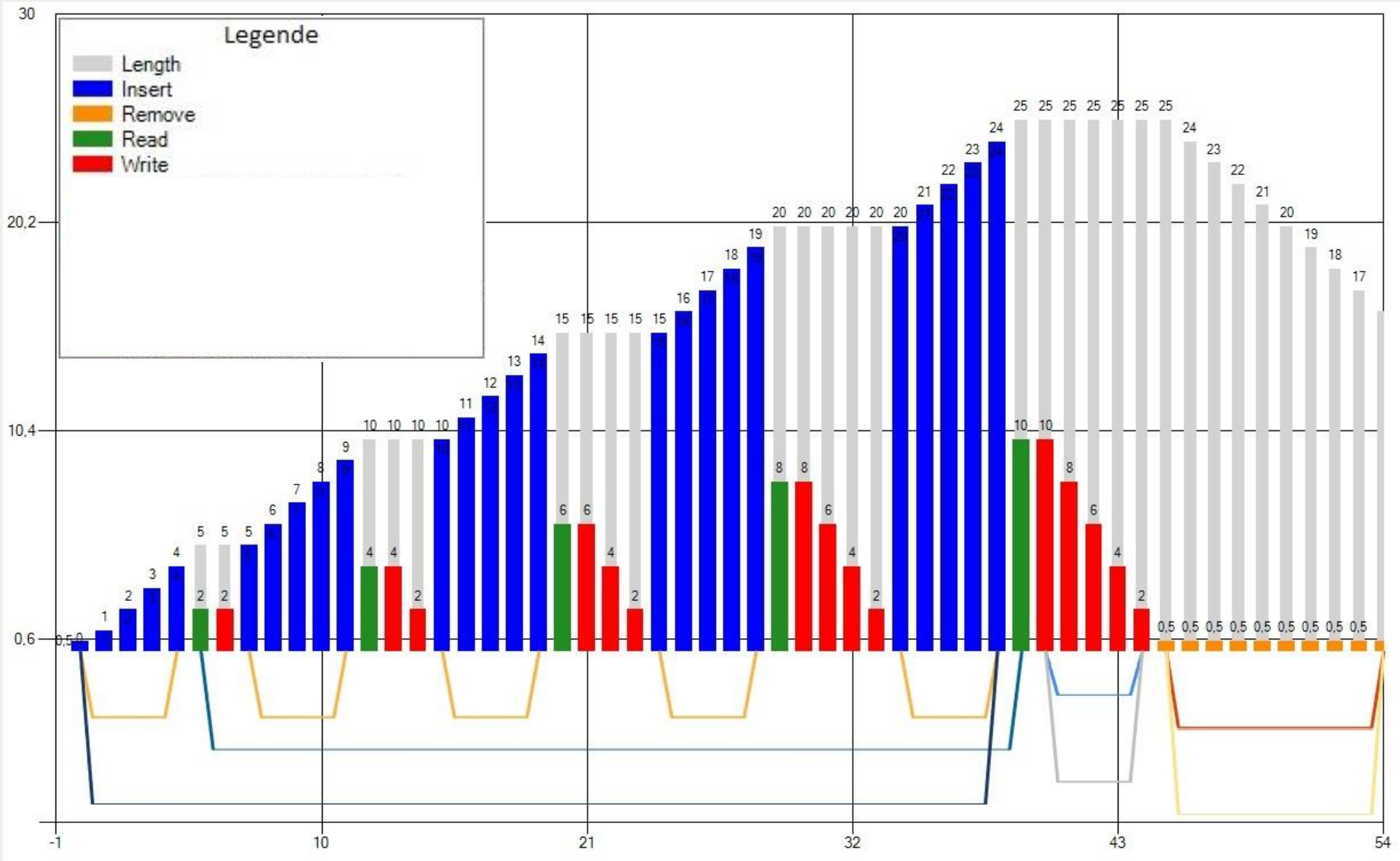
Lineares-Schreiben-Rückwärts

Einfügen-Hinten

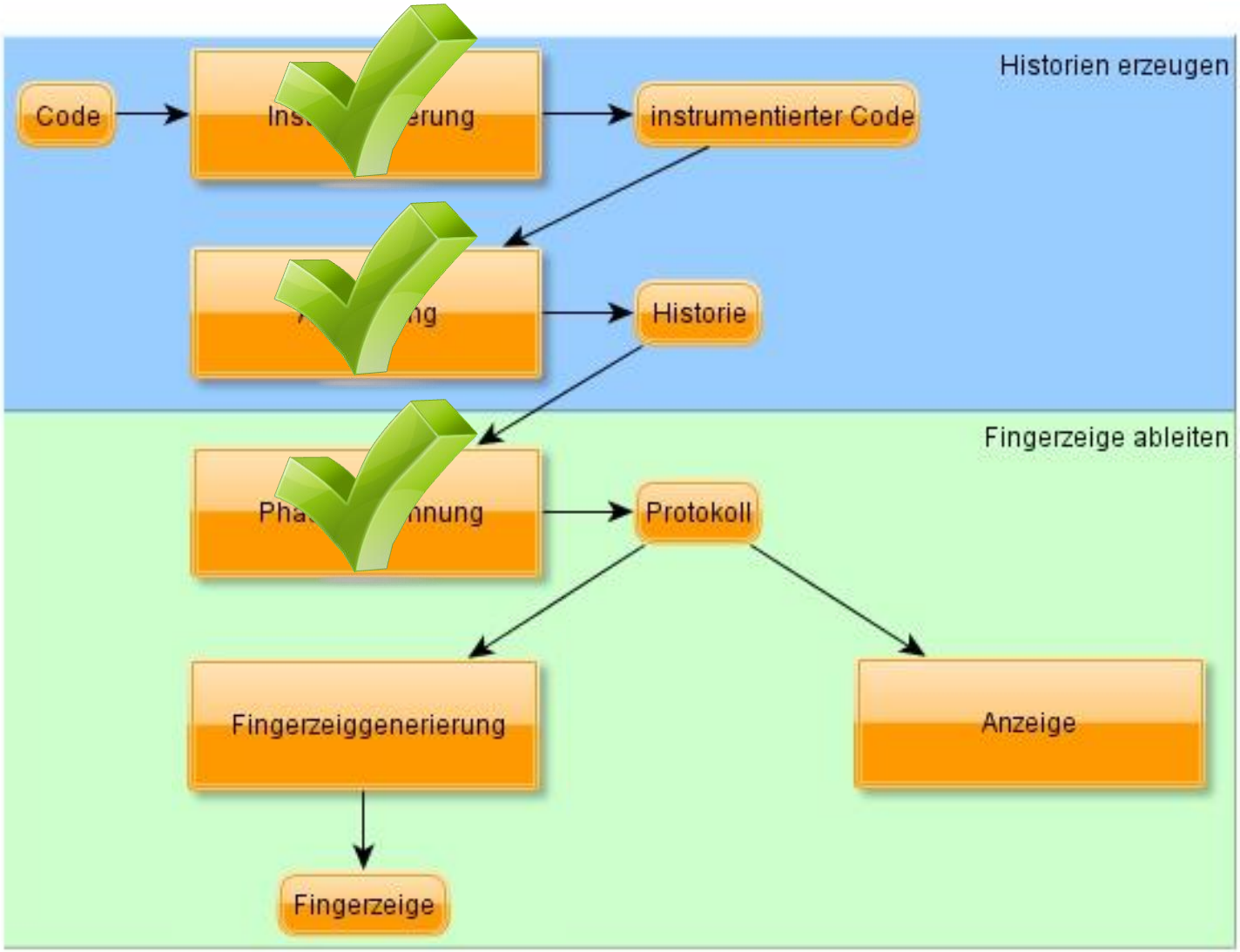
Löschen-Vorn



# Phasenerkennung



# Lösungsansatz



# Fingerzeige

Langes Einfügen

Häufiges Suchen

Häufig langes Lesen

Implementierung einer Schlange

Sortieren nach dem Einfügen

# Handlungsempfehlung

Fingerzeig

**Klasse:**

**Methode, Position:**

**Datenstruktur:**

**Grund, Ordnungskennzahl:**

# Handlungsempfehlung (Bsp.)

Fingerzeig 1

**Klasse:** GPdotNET.Engine.CHPopulation

**Methode, Position:** .ctor, 14

**Datenstruktur:** List<GPdotNET.Core.IChromosome>

**Grund, Ordnungskennzahl:** Häufig langes Lesen, 67187

Fingerzeig 2

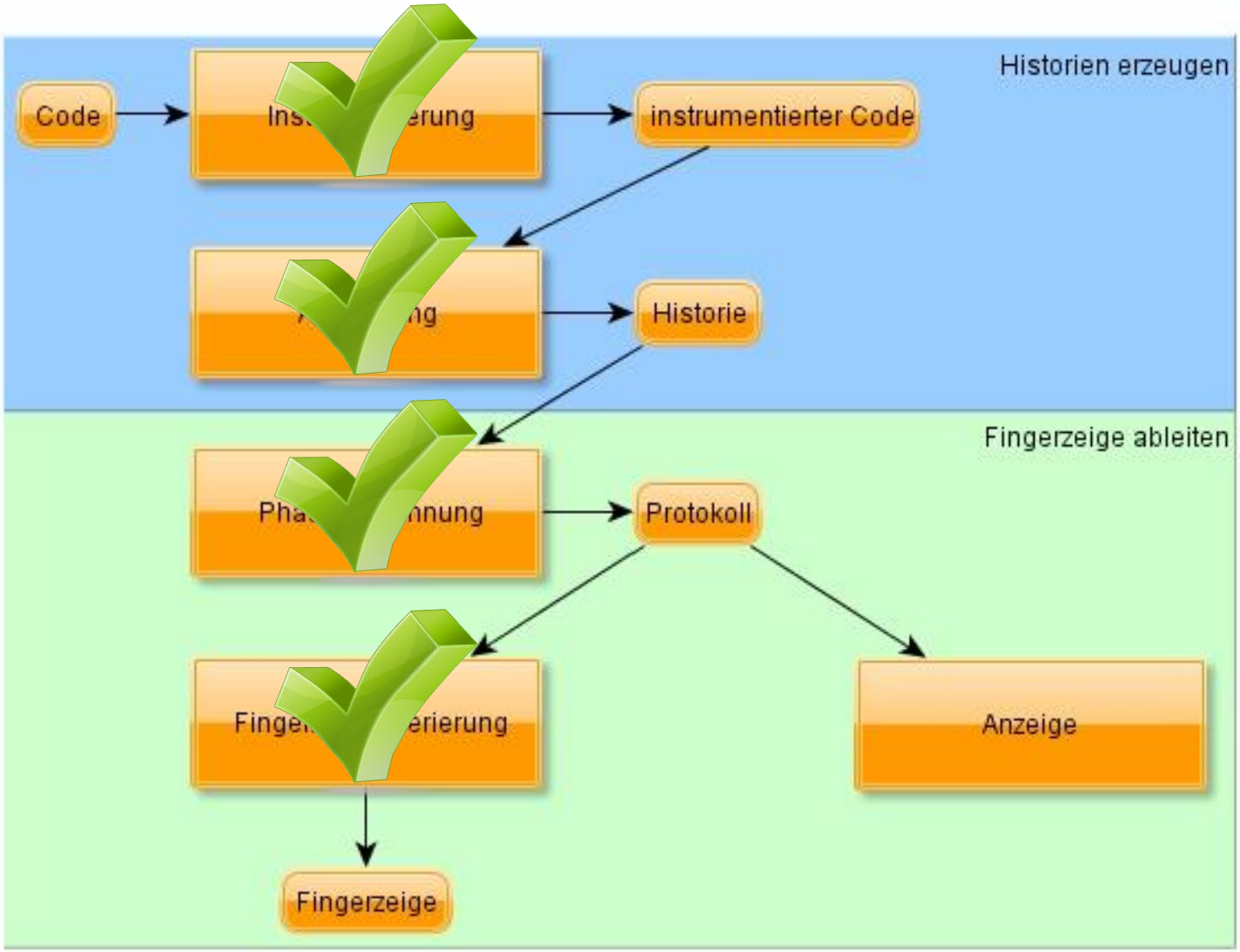
**Klasse:** GPdotNET.Engine.CHPopulation

**Methode, Position:** .ctor, 14

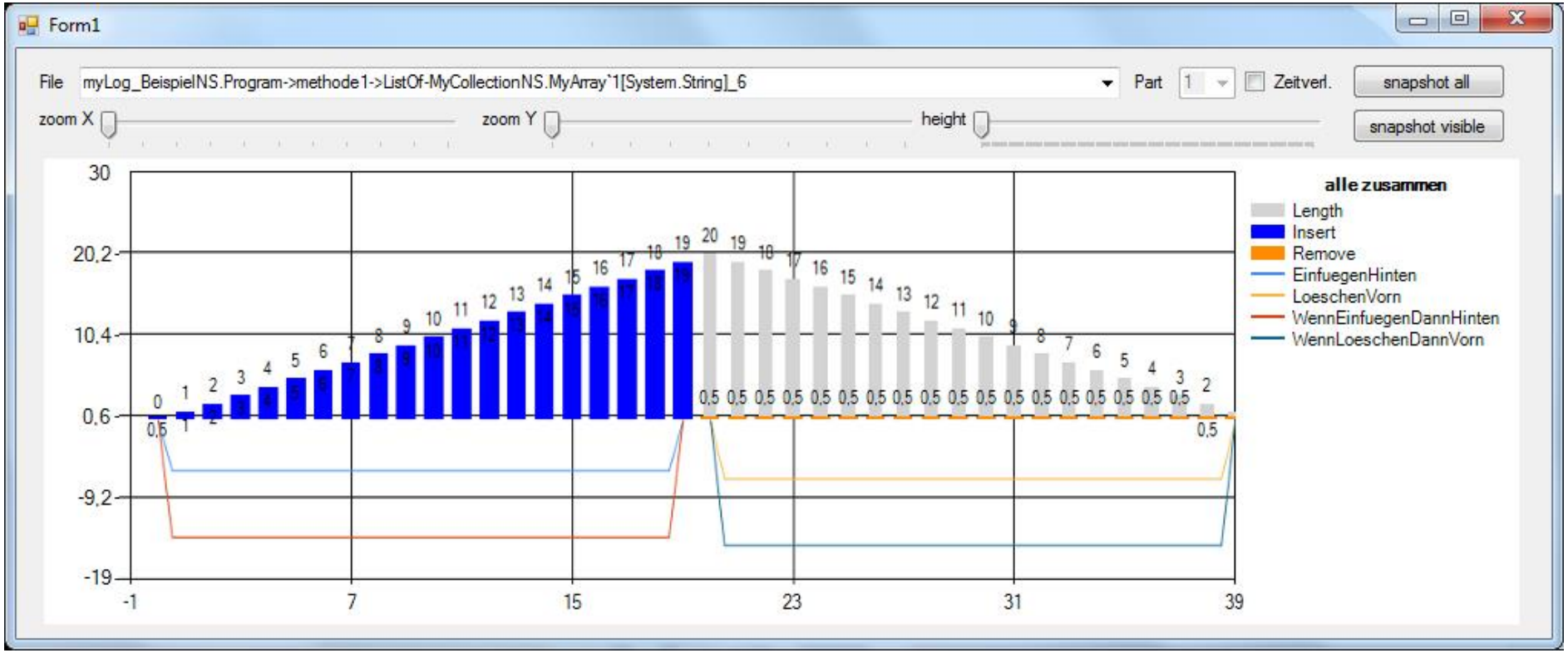
**Datenstruktur:** List<GPdotNET.Core.IChromosome>

**Grund, Ordnungskennzahl:** Langes Einfügen, 67187

# Lösungsansatz



# Visualisierung



# EVALUATION

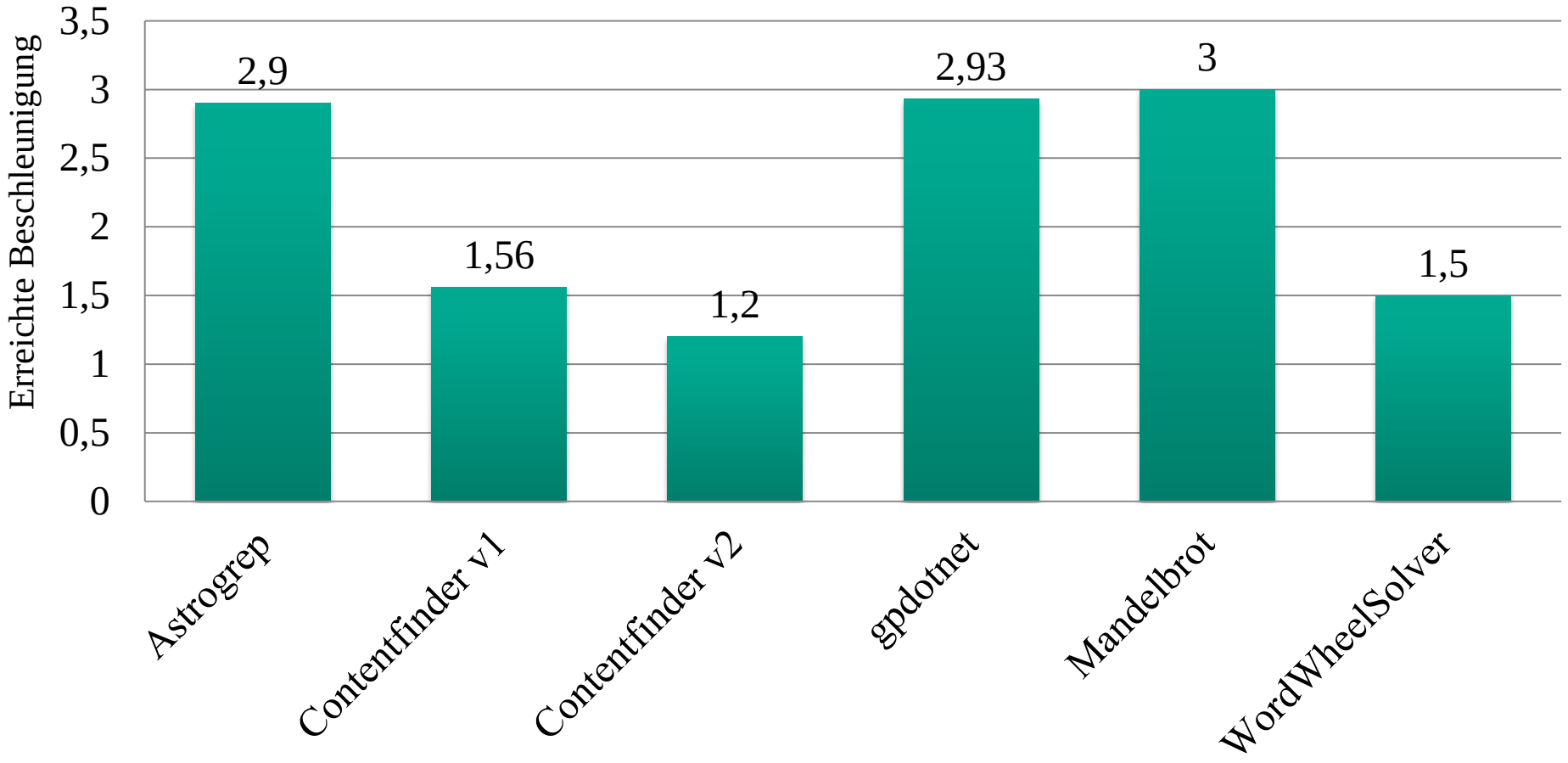
# Evaluation: Testprogramme

| Programm | LOC | Domäne | Textuelle Ausgabe | Grafische Visualisierung | Art der Ausführung | #Instanziierungsorte (List/Array) |
|---|---|---|---|---|---|---|
| Algorithmia | 2800 | Bibliothek | - | - | UnitTest | 16 (15/1) |
| Astrogrep | 4800 | Dateisuche | + | - | Manuell | 21 (6/15) |
| Contentfinder | 290 | Dateisuche | + | - | Manuell | 11 (5/6) |
| CPU-Benchmark | 400 | Benchmark | + | - | Manuell | 7 (0/7) |
| Gpdotnet | 7000 | Simulation | + | + | Manuell | 33 (14/19) |
| Mandelbrot | 150 | Problemlöser | + | + | Manuell | 7 (0/7) |
| WordWheel-Solver | 110 | Problemlöser | + | - | Manuell | 5 (5/0) |

# Evaluation: Ergebnisse

| Programm | #Fingerzeige (untersch. Orte) | #richtig-positive Orte | #Parallelitäts-umsetzungen | #falsch-positive Orte |
|---|---|---|---|---|
| Algorithmia | 4 (3) | 2 | 2 | 1 |
| Astrogrep | 2 (2) | 1 | 1 | 1 |
| Contentfinder | 2 (2) | 2 | 1 | 0 |
| CPU-Benchmark | 5 (4) | 4 | 1 | 0 |
| Gpdotnet | 5 (3) | 2 | 2 | 1 |
| Mandelbrot | 4 (4) | 4 | 3 | 0 |
| WordWheel-Solver | 2 (2) | 1 | 1 | 1 |

# Evaluation: Ergebnisse



# Zusammenfassung

List und Array werden am meisten verwendet

Wiederkehrende Regelmäßigkeiten in Zugriffshistorien erkennbar

Zugriffsverhalten durch Phasen und Fingerzeige beschreibbar

Werkzeug mit Ausgabe von Handlungsempfehlungen

Fingerzeige orten reales Parallelisierungspotenzial

# Ausblick

Zugriffsarten, Phasen, Fingerzeige erweitern/ergänzen

Hinzunahme weiterer Laufzeitdaten

Weitere Studien zum Vergleich mit anderen Werkzeugen und Ansätzen

Wechselspiel mehrerer Datenstrukturen

Einsatz im automatischen Parallelisierungsprozess

Lassen sich

aus dem Zugriffsverhalten von Datenstrukturen

Handlungsempfehlungen automatisiert ableiten,

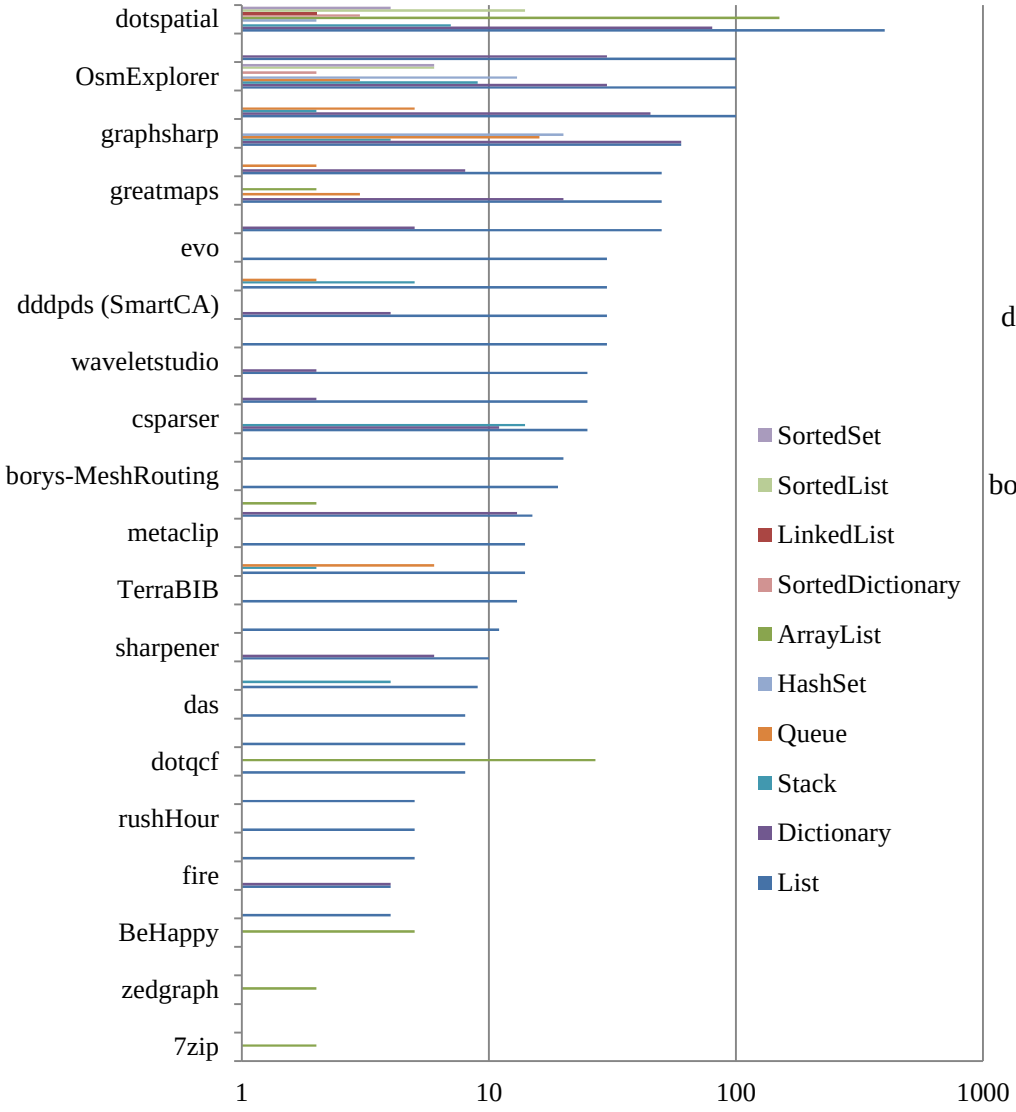
sodass der Nutzer an parallelisierungsrelevante Codestellen herangeführt wird?

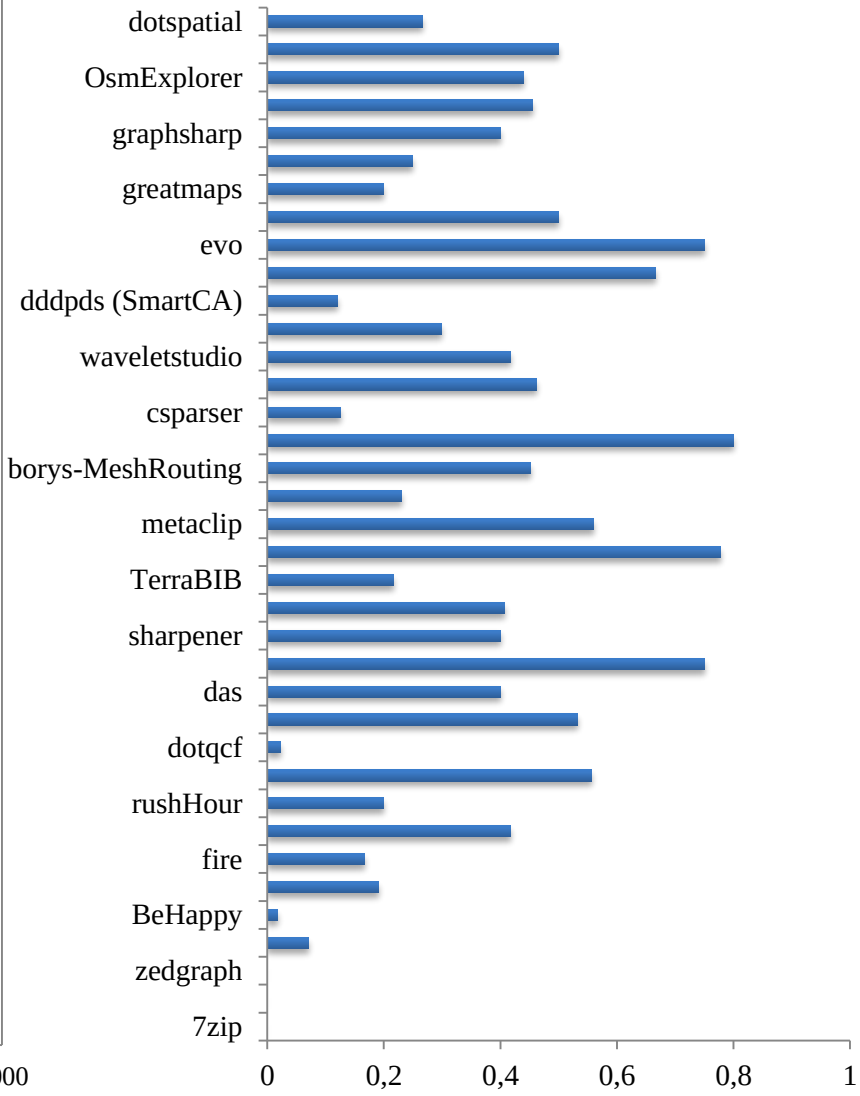
→ JA

# Evaluation: Verlangsamung

| Programm | Einheit | Laufzeit (orig.) | Laufzeit (instr.) | Verlangs.-Faktor |
|---|---|---|---|---|
| Algorithmia | sek | 0,50 | 2,40 | 4,80 |
| Astrogrep | sek | 4,80 | 5,80 | 1,21 |
| Contentfinder(v1) | sek | 3,20 | 17,00 | 5,31 |
| Contentfinder(v2) | sek | 1,80 | 5,20 | 2,89 |
| CPU-Benchmark | ms | 10,00 | 550,00 | 55,00 |
| Gpdotnet(SC) | sek | 0,36 | 78,00 | 216,67 |
| Gpdotnet(MC) | sek | 0,30 | 64,38 | 214,60 |
| Mandelbrot(SC) | ms | 110,00 | 1200,00 | 10,91 |
| Mandelbrot(MC) | ms | 50,00 | 800,00 | 16,00 |
| WordWheelSolver | ms | 39,00 | 1500,00 | 38,46 |

# Voruntersuchungen - Datenstrukturen



# Zugriffsarten

- Einfügen
- Suchen
- Löschen
- Lesen
- Schreiben
- Leeren*
- Kopieren/Konvertieren*
- Umdrehen*
- Sortieren*
- Auf-Alle-Anwenden-Operator* (ForEach)

# Phasencharakteristika 1

- Lineares-Lesen-Vorwärts
  direkt benachbarte Lesezugriffe, deren Zugriffsindizes mit der Zeit (monoton) steigen
- Lineares-Lesen-Rückwärts
  direkt benachbarte Lesezugriffe, deren Zugriffsindizes mit der Zeit (monoton) fallen
- Lineares-Schreiben-Vorwärts
  direkt benachbarte Schreibzugriffe, deren Zugriffsindizes mit der Zeit (monoton) steigen
- Lineares-Schreiben-Rückwärts
  direkt benachbarte Schreibzugriffe, deren Zugriffsindizes mit der Zeit (monoton) fallen
- Einfügen-Vorn
  direkt benachbarte Einfügeoperationen, die stets vorn stattfinden
- Einfügen-Hinten
  direkt benachbarte Einfügeoperationen, die stets hinten stattfinden
- Löschen-Vorn
  direkt benachbarte Löschoperationen, die stets vorn stattfinden
- Löschen-Hinten
  direkt benachbarte Löschoperationen, die stets hinten stattfinden

# Phasencharakteristika 2

- Wenn-Lesen-Dann-Linear-Vorwärts
- Wenn-Lesen-Dann-Linear-Rückwärts
- Wenn-Schreiben-Dann-Linear-Vorwärts
- Wenn-Schreiben-Dann-Linear-Rückwärts
- Wenn-Einfügen-Dann-Vorn
- Wenn-Einfügen-Dann-Hinten
- Wenn-Löschen-Dann-Vorn
- Wenn-Löschen-Dann-Hinten

# Evaluation: parallelisierbare Zeitanteile