# Studienarbeit 2011
# Weiterentwicklung einer KSM

David Henn

Mai 2011

Huppladuppla

## 1 Danksagung

Schubert
Danke, dass wir selbstständig arbeiten durften und sie ihr Ego zurückgehalten haben

Dreher
Merci for the work

Mischa
Danke fürs mitmachen

Yves
Danke für den Input von außen und die Motivation

# 2 Einleitung

zweiter Teild er Studienarbeit, die sich um KSM dreht Schwerpunkt Bugfixing und Weiterentwicklung

# 3 Projektplanung

Bei der Projektplanung sind für KSM mehrere Dinge zu berücksichtigen.

- Der Zustand des Projekts
- Die knappe Zeit
- Die Aufgaben der Teammitglieder
- Die Anforderungen des Projektinhabers Herr Prof. Schubert

Aufgrund des chaotischen Zustands des Quellcodes, sowie der restlichen Projektartefakte wurde in Absprache mit Herrn Prof. Schubert ein Zwei-Stufen-Plan[1] verabschiedet.

Während der ersten, längeren Phase wird vom Projektteam Bugfixing und Refactoring betrieben, um das Projekt weiter zu ordnen. Um dieses Ziel zu erreichen wird nach dem agilen Entwicklungsmodell *Scrum* vorgegangen.

Nachdem das Projekt stabilisiert wurde, soll in der zweiten Phase die Funktionalität des KSM weiterentwickelt werden.

# 4 Erste Phase

Wie im Kapitel 3 beschrieben, wurde in der ersten Phase des Projektes versucht den Zustand des Gesamtprojektes zu verbessern. Da dies im Allgemeinen viele kurze Aufgaben sind, sowie während dieser Phase ständig neue Aufgaben identifiziert werden, wurde mit Scrum ein äußerst Vorgehensmodell gewählt, das es dem Team ermöglicht diese Aufgaben äußerst flexibel abzuarbeiten.

[1] siehe Anhang

## 4.1 Das Scrum Modell

In diesem Abschnitt soll kurz auf das Scrum Modell, sowie der Einsatz in diesem Projekt eingegangen werden.

Im Gegensatz zu traditionellen Entwicklungsmodellen geht Scrum iterativ vor. Einzelne Iterationen werden in Scrum "Sprint" genannt. Vor jedem Sprint legt das Projektteam fest, welche Aufgaben in dieser Zeit abgearbeitet werden sollen. Diese Aufgaben werden im Sprint-Backlog festgehalten. Am Ende eines solchen Sprints muss eine lauffähige Version stehen. Da ein Sprint nur eine Laufzeit von zwei bis vier Wochen hat, kann es durchaus passieren, dass einzelne Aufgaben innerhalb eines Sprints nicht fertiggestellt werden können. Dies ist jedoch nicht schlimm, solange die Version trotzdem lauffähig bleibt. Nicht fertiggestellte Aufgaben werden im nächsten Sprint weitergeführt.

Während eines Sprints stehen die Entwicklern in engem Kontakt zueinander. Jeden Tag gibt es ein kurzes Scrum Meeting, in dem jeder Entwickler kurz darlegt, was er am Tag davor geschafft hat und was er an diesem Tag zu tun gedenkt. Dies stellt sicher, dass stets das gesamte Team weiß, wo sich das Projekt zu diesem Zeitpunkt befindet und was für Entscheidungen getroffen wurden.

Der gesamte Scrum Prozess wird in [DHV11] genau beschrieben.

In KSM Projekt wurde eine Sprintdauer von jeweils zwei Wochen festgelegt. Das Sprint-Backlog, sowie das übergeordnete Projekt-Backlog wurden in Form eines Excel-Dokuments[2] geführt. Die Aufgaben in den Backlogs wurden von den einzelnen Teammitgliedern definiert. Als Scrum Master, der den Prozess überwacht und koordiniert fungierte Herr Tobias Dreher, der durch das Thema seiner Studienarbeit den größten Überblick über die Baustellen des Programms besitzt.

## 4.2 Ergebnis

Während dieser Stabilisierungsphase wurde der Produktreifegrad des KSM Projekts deutlich verbessert.

- Durch viele kleine und größere Änderungen wurden Inkonsistenzen im Design, sowie der Benutzerführung beseitigt

[2] Siehe Anhang

- Die Icons des Programms wurden ausgetauscht und durch aussagekräftigere Symbole ersetzt
- Es wurde ein Build Prozess definiert, der es ermöglicht, eine auslieferbare Version des Programms zu generieren
- Es wurden viele Codeleichen entfernt
- eine Projekthomepage wurde erstellt
- Das Projektverzeichnis wurde strukturiert und vereinheitlicht

Durch diese und viele weitere Änderungen wurde die Benutzbarkeit des Programms gesteigert, sowie der Entwicklungsprozess geordnet und vereinheitlicht.

# 5 Weiterentwicklung

In der zweiten Phase des Projekts soll die Funktionalität des KSM erweitert werden. Besonderes Augenmerk liegt hierbei auf der Verbesserung der Anzeige, welche Knoten im aktiven/passiven, oder kritischen Bereich liegen. Diese Anzeige wird intern auch SumChart genannt. Hier wurde bereits in einer der letzten Studienarbeiten von Frau Klein angefangen, es dem Benutzer zu ermöglichen, Bereiche zu definieren und zu markieren [Kle10]. Bereiche waren in diesem Fall Ellipsen, die vom Benutzer aufgezogen wurden. Je größer diese wurden, desto transparenter wurde die Füllfarbe. Die Abbildung 1 zeigt den letzten Stand dieser Implementierung.

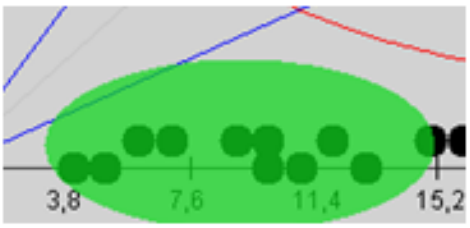


Abbildung 1: Bereichsmarkierung nach Klein [Kle10]

Nach Evaluation dieser Implementierung kam Herr Professor Schubert zu dem Schluss, dass Ellipsen auf der einen Seite zu unflexibel sind, um einen Bereich adäquat ausbilden zu können. Auf der anderen Seite jedoch auf keinerlei Begrenzungen, wie zum Beispiel

aktiv, passiv Geraden Rücksicht nehmen. Dazu wurde die Füllfarbe trotz Transparenz teilweise als zu deckend empfunden.

## 5.1 Anforderungen

Die neuen Anforderungen an eine Bereichsmarkierung lauteten nach diesen Erkenntnissen wie folgt:

- Der Benutzer muss in der Lage sein, einen Bereich mittels Eckpunkten definieren zu können
- Es dürfen beliebig viele Eckpunkte sein
- Die Eckpunkte dürfen nur auf Bereits vorhandenen Linien laufen:
  - X-Achse
  - Y-Achse
  - Q-Achsen zur Abgrenzung der aktiven/passiven Bereiche, sowie Q = 1 als Mitte
  - Hyperbeln zur Markierung des stabilen/kritischen Bereichs
- Punkte (Knoten des System Graphen), die innerhalb des markierten Bereichs liegen, müssen aufgelistet werden
- Die Füllfarbe des Bereichs soll transparenter sein, als bisher

Dazu wurde noch von den Projektmitgliedern entschieden, dass erstellte Bereiche benannt werden sollen. Dieser Name soll innerhalb des Bereichs sichtbar sein.

## 5.2 Ergebnis

Die gestellten Anforderungen wurden vollständig umgesetzt. Der Benutzer kann nun durch setzen mehrerer Eckpunkte, die sich auf den vorgegebenen Linien befinden ein Polygon definieren.

Damit der Benutzer immer weiß, wo der nächste Eckpunkt gesetzt werden würde, wurde ein roter Punkt implementiert, der sich nur auf den erlaubten Linien bewegt und so den nächsten Eckpunkt markiert (Abbildung 2).

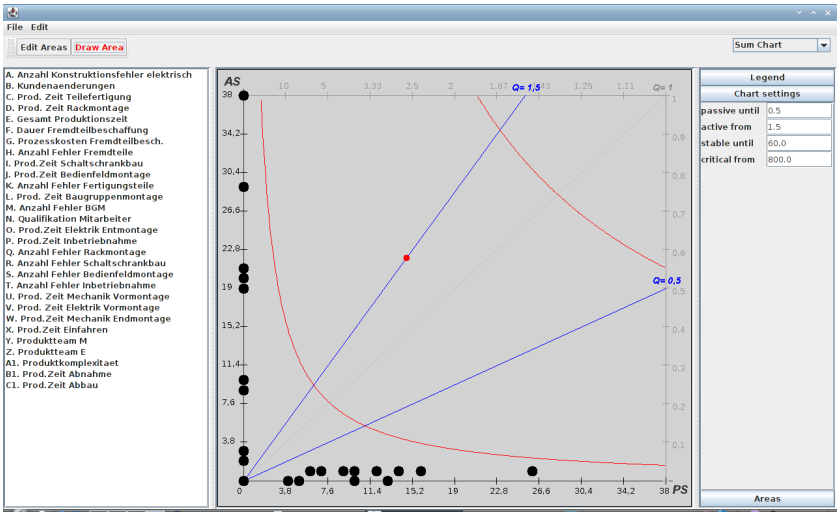


Abbildung 2: roter Punkt

Mittels Klick wird der Punkt gespeichert und ein vorläufiges Polygon gezeichnet. Will der Benutzer die Eingabe beenden, kann er das Polygon mit einem Doppelklick abschließen. Anschließend wird einen Auswahldialog präsentiert, ob das Polygon übernommen, oder verworfen werden soll. Abbildung 3 zeigt ein abgeschlossenes Polygon und das Auswahlmenü.

Nachdem ein Polygon akzeptiert und benannt wurde, wird ermittelt das Programm, welche Knoten sich innerhalb dieses Polygons befinden und listet diese in einer Baumstruktur auf. Diese Auflistungen können im rechten Panel unter dem Eintrag “Areas” gefunden werden. Abbildung 4 zeigt diese Auflistung für das Polygon “Beispiel”.

Die Anforderung der zu geringen Transparenz wurde etwas anders gelöst, als anfangs geplant. Die ursprüngliche Implementierung von Frau Klein sah vor, dass die Transparenz mit der Größe der Fläche zunimmt. Dieser Umstand führte in der neuen Implementierung dazu, dass obwohl der Transparenzwert erhöht wurde, kleine Flächen immer noch viel zu deckend waren. Große Flächen jedoch kaum noch sichtbar waren. Deswegen wurde die Kopplung der Transparenz mit der Fläche aufgehoben und stattdessen eine starke Transparenz der Füllfarbe definiert und die Fläche zudem mit einer deckenden Umrandung versehen. Dies ermöglicht eine gute Sichtbarkeit der Knoten und Werte innerhalb des Polygons, bei gleichzeitiger guter Sichtbarkeit des Polygons selbst, wie auf den Abbildungen 4 und 3 zu sehen ist.

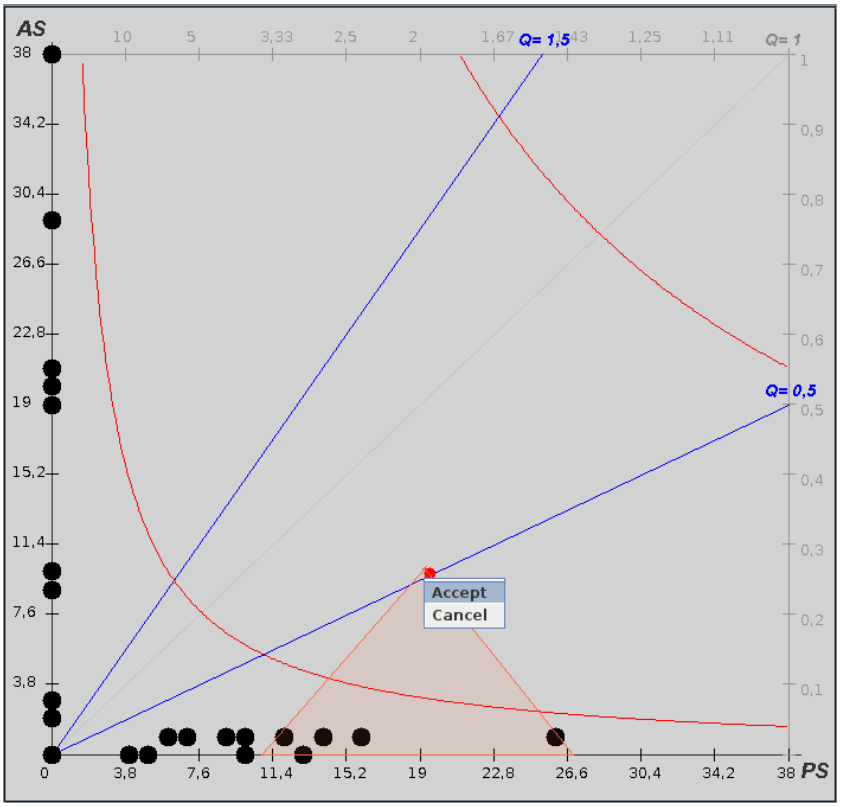


Abbildung 3: abgeschlossenes Polygon mit Auswahldialog

## 5.3 Implementierung

Die Implementierung der neuen Funktionalitäten lässt sich in mehrere Aspekte unterteilen.

1. die gültigen Linien
2. Ermittlung der nächsten gültigen Linie zur Mausposition
3. Zeichnen des aktuellen Polygons
4. Speicherung und Anzeige akzeptierter Polygone
5. Ermittlung der Punkte innerhalb eines Polygons und Darstellung in einem Baum

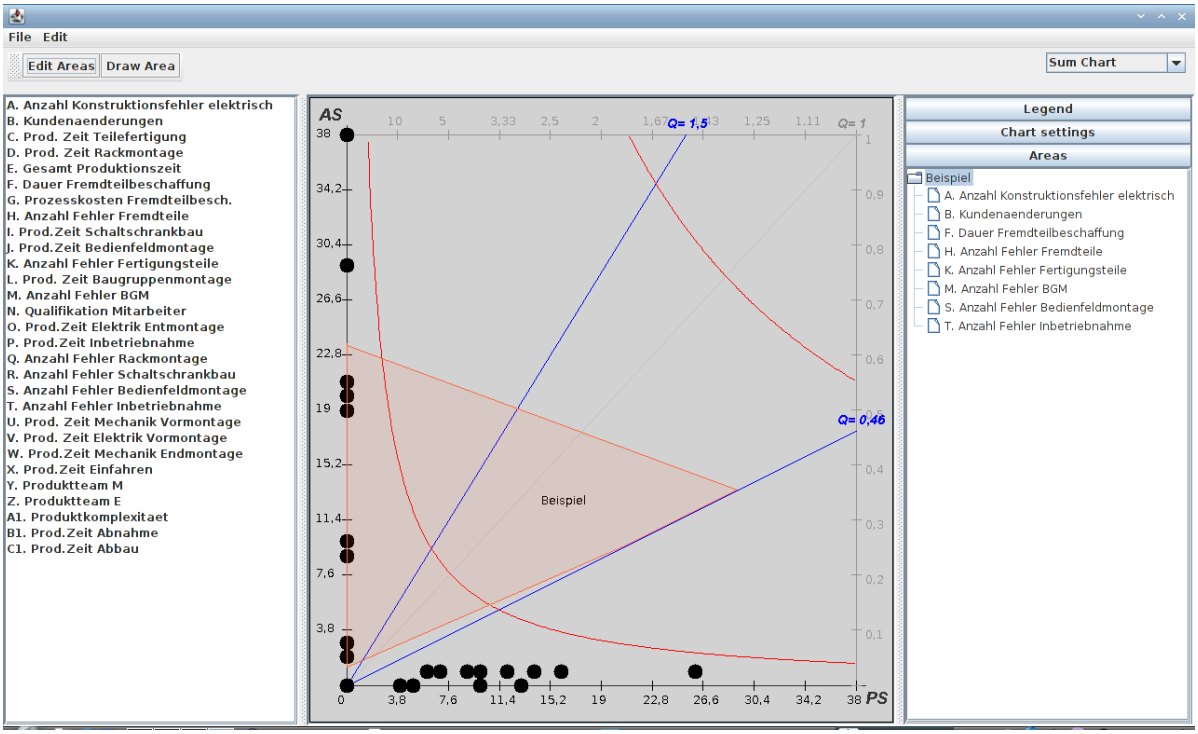


Abbildung 4: Baumdarstellung der Knoten innerhalb eines Polygons

Da diese Aufgabe im Team mit Herr Vogt gemeinsam gelöst werden sollte, wurden die Aspekte aufgeteilt. Herr Vogt bearbeitete Aspekt Nr. 1, ich die Aspekte 4 - 6. Im folgenden sollen nun die wichtigsten Details meiner Implementierung erläutert werden.

**Ermittlung der nächsten gültigen Linie zur Mausposition:** Um die nächste Linie zu ermitteln wurde ein Interface definiert, das alle Klassen, die eine Linie repräsentieren implementieren müssen. Anhand dieses Interfaces kann gesammelt über alle Linien des Diagramms iteriert werden und Funktionswerte verglichen werden. Listing 1 zeigt die Definition des Interfaces.

```
1 public interface Function {
2     public Point calcY(int x, int y);
3 }
```

Listing 1: Interface für die Linien

Wie in Listing 1 zu sehen, wird immer ein Punkt zurückgegeben. Dieser Punkt definiert den Punkt auf der Linie, der am ehesten der aktuellen Mausposition

entsprechen würde. Diese Punkte können nun verglichen werden.

```
Point bestPoint = new Point();
        int bestDist = 1000000;
        for (Function function : functions) {
            Point curPoint = function.calcY(x, y);

            int dist = Math.abs(x - curPoint.x) + Math.abs(y -
                curPoint.y);
            if(dist < bestDist){
                bestDist = dist;
                bestPoint = curPoint;
            }
        }
```

Listing 2: Methode zum ermitteln des nächsten Punktes

Listing 2 zeigt die Ermittlung der nächsten Linie. Der Abstand des Punktes auf der Linie zur aktuellen Mausposition wird durch die Addition der absoluten Abstände der X- und Y-Werte berechnet. Ist dieser Abstand kleiner als der bisherige kleinste Abstand, so wird dieser übernommen und der Punkt gespeichert.

Damit diese Methode bei jeder Mausbewegung aufgerufen wird, ist es noch notwendig einen eigenen MouseMotionListener zu implementieren. Dieser reagiert auf Ereignisse, die durch die Maus ausgelöst werden. Wie zum Beispiel, wenn die Maus bewegt wird. Dieser MouseMotionListener ist in Listing 3 abgebildet. Dort wird bei jeder Mausbewegung die Methode *setPoint* aufgerufen, die die Berechnung aus Listing 2 beinhaltet. Im Anschluss wird das gesamte Bild neu gezeichnet.

```
class drawAreaMouseMotionListener extends MouseMotionAdapter{
        @Override
        public void mouseMoved(MouseEvent event){
            setPoint(event.getX(), event.getY());
            repaint();
        }
}
```

Listing 3: MouseMotionListener

**Zeichnen des aktuellen Polygons:** Das aktuelle Polygon muss immer dann neu gezeichnet werden, wenn der Benutzer durch einen Klick einen neuen Eckpunkt hinzufügt. Um diesen Klick zu registrieren, muss wieder ein MouseListener implementiert werden. Dieser reagiert allerdings nicht auf Mausbewegungen, wie der

vorherige MouseListener, sondern auf Klicks. Somit wird in der Methode *mouseClicked* der im vorherigen Schritt ermittelte Punkt zu dem aktuellen Polygon hinzugefügt (Listing 4).

```
polygon.addPoint(currentPoint.x, currentPoint.y);
repaint();
```

Listing 4: Neuen Punkt zum Polygon hinzufügen

Mit dem Aufruf der Methode *repaint* wird unter anderem die Methode *paint* ausgeführt. Dort wird das Polygon neu gezeichnet.

```
g.setColor(ovalColor[2-x_index][2-y_index]);
g.drawPolygon(polygon);

Composite c = AlphaComposite.getInstance(AlphaComposite.SRC_OVER
    ,0.1f);
g2d.setComposite(c);

g.fillPolygon(polygon);
```

Listing 5: Zeichnen des Polygons

In Listing 5 wird die Zeichenroutine aufgeführt. Es ist zu sehen, dass zuerst mit der Methode *drawPolygon* nur die Umrisse des Polygons gezeichnet werden. Erst im Anschluss wird der Alphakanal verändert, um die Transparenz der Farbe zu beinflussen, damit die Füllung des Polygons die darunterliegenden Knoten nicht überdeckt.

**Speicherung und Anzeige akzeptierter Polygone:** Die Definition eines neuen Polygons wird mit einem Doppelklick abgeschlossen. Dieser Doppelklick wird ebenfalls mit dem vorherigen MouseListener registriert. Daraufhin wird dem Benutzer ein Dialog präsentiert, in dem er wählen kann, ob das Polygon gespeichert, oder verworfen werden soll. Soll das Polygon behalten werden, so muss der Benutzer einen Namen vergeben. Listing 6 zeigt, wie die Speicherroutine implementiert wurde.

```
String description = JOptionPane.showInputDialog(parent, "Give in
    a description..", "Area Description", 3);
if(description == null){
  polygon = new Polygon();
  repaint();
  return;
}
NamedPolygon pol = new NamedPolygon(polygon.xpoints, polygon.
    ypoints, polygon.npoints);
```

```
8  pol.setName(description);
9  searchInnerPoints(pol);
10 polygons.add(pol);
11 polygon = new Polygon();
12 repaint();
```

Listing 6: Routine zum Speichern eines Polygons

Die Zeilen 1 - 6 Erzeugen einen Eingabedialog für den Namen. Gibt der Benutzer keinen Namen ein, wird die Routine abgebrochen. Im Anschluss wird ein neues Polygon vom selbst erzeugten Typ NamedPolygon (Listing 7) instanziiert, dem der vom Benutzer eingegebene Name zugewiesen wird. Daraufhin wird das neue Polygon in einer Liste gespeichert und das alte Polygon wieder zurückgesetzt.

```
1  public class NamedPolygon extends Polygon{
2      String name;
3
4      public NamedPolygon(int xpoints[], int ypoints[], int npoints)
           {
5          super(xpoints, ypoints, npoints);
6      }
7      public NamedPolygon(){
8          super();
9      }
10     public void setName(String name){
11         this.name = name;
12     }
13     public String getName(){
14         return this.name;
15     }
16
17 }
```

Listing 7: Klassendefinition des NamedPolygon

**Ermittlung der Punkte innerhalb eines Polygons und Darstellung in einem Baum:**
Zeile 9 des Listings 6 zeigt schon den Aufruf der Methode *searchInnerPoints*. Listing 8 zeigt diese Methode.

```
1  public void searchInnerPoints(NamedPolygon pol){
2          DefaultMutableTreeNode root = new DefaultMutableTreeNode(
               pol.getName());
3          List<String> innerPoints = new ArrayList<String>();
4          for(int i=0; i<points.size();i++){
5              if(pol.contains(points.elementAt(i))){
```

```
6                   String name = data.getListModelNodes().
                        getElementAt(i).toString();
7                   innerPoints.add(name);
8                   DefaultMutableTreeNode child = new
                        DefaultMutableTreeNode(name,false);
9                   root.add(child);
10              }
11          }
12
13          data.getPolyModel().insertNodeInto(root, data.getRoot(),
                0);
14  }
```

Listing 8: Methode searchInnerPoints

Listing 9: Mehr Text

# 6 Nachfolger

Run for the Hills idiots!!!

# 7 Fazit

Jippieeee. Vorbei!!!!

# 8 Literatur


[Kri07] David Kriesel: **Ein kleiner Überblick über Neuronale Netze**. *www.dkriesel.com*, 2007, Abgerufen am 15.04.2011

[DHV11] Tobias Dreher, Mischa Vogt, David Henn: **Agile Softwareentwicklung - Einsatz und Werkzeuge im Open Source Umfeld.** Seminararbeit Software-Engineering II, 2011

[Kle10] Marita Klein: **Graphische Oberfläche einer KSM.** Studienarbeit 5. Semester, 2010

# 9 Anhang

## Listings



## Abbildungsverzeichnis



Projektplan
CD