

*Fachstudie Nr. 208*

---

# Kategorisierung und Evaluierung von Graphvisualisierungssystemen

G. Matheis – J. Strotzer – C. Völker

<b>Studiengang:</b>	Softwaretechnik
<b>Prüfer/in:</b>	Prof. Dr. Daniel Weiskopf
<b>Betreuer/in:</b>	Dr. rer. nat. Michael Burch Dr. Fabian Beck
<b>Beginn:</b>	16. Dezember 2014
<b>Beendet am:</b>	16. Juni 2015
<b>CR-Nummer:</b>	C.4, D.2.2, E.1, H.5.2

## Inhaltsverzeichnis

1. Einleitung .....	4
a) Motivation .....	4
b) Technischer Hintergrund .....	4
Grunddefinitionen .....	4
Graph Darstellungen .....	5
Anwendungsbereiche .....	5
Graphen Probleme .....	5
Algorithmen (Laufzeit/Darstellung) .....	6
Unterschiede: Bibliothek und Werkzeug („Tool“) .....	6
2. Test-Suite .....	6
a) Testkriterien .....	6
b) Testdaten .....	9
3. Graph Bibliotheken .....	9
a) Auswahlkriterien .....	9
b) Warum keine Tools? .....	10
c) Gefundene Bibliotheken und Begründung für die Auswahl .....	10
d) Testmaschine .....	11
4. Bewertung der Java-Bibliotheken .....	12
a) Einzelbewertungen .....	12
JGraphX .....	12
Prefuse .....	17
b) Bewertung .....	21
5. Bewertung der JavaScript-Bibliotheken .....	23
a) Einzelbewertungen .....	23
CytoScapeJS .....	23
D3JS .....	28
SigmaJS .....	33
yFiles HTML .....	38
b) Bewertung .....	41

6. Bewertung der C#-Bibliotheken.....	44
a) Einzelbewertungen .....	44
GraphX.....	44
yFiles WPF .....	48
b) Bewertung.....	52
7. Gesamtbewertung.....	54
a) Support.....	55
b) Community .....	55
c) Einarbeitungsaufwand .....	55
d) Performanz.....	56
e) Einlesen von (Standard-)Formaten .....	57
f) Vielfalt (Knoten- / Kantendarstellung).....	57
g) Layout (als Ranking) .....	57
h) Algorithmen.....	59
i) Vielfalt (nicht-)/kommerziell.....	60
8. Quellen .....	61

# 1. Einleitung

## a) Motivation

In der Informatik lassen sich viele Probleme mit Hilfe von Graphen modellieren und lösen. Situationsbedingt variieren diese mehr oder weniger stark und erfordern individuelle und problemangepasste Funktionen. Im Laufe der Jahre wurden darum diverse Bibliotheken für die Darstellung von Graphen oder deren Berechnung entwickelt. In Zeiten des Internets entsteht hierdurch natürlich eine breitgefächerte Vielfalt von Bibliotheken, welche sich teils deutlich unterscheiden, was sich sowohl durch die globale Vernetzung, als auch die große Vielfalt an Programmiersprachen erklären lässt. Ein weiterer Grund sind die unterschiedlichen Anforderungen welche sich aus den individuellen Problemen ergeben, die gelöst werden sollen. Hierbei kann unter anderem die Performanz oder auch die Art der Visualisierung im Vordergrund stehen. Durch diese Vielfalt an Anforderungen ist es unmöglich eine Bibliothek zu entwickeln, die allen Anforderungen gerecht wird, zumal diese sich teilweise gegenseitig ausschließen. Für die Nutzer dieser Bibliotheken ist es darum besonders wichtig zu wissen, was die Bibliothek kann, um entscheiden zu können, welche Bibliothek die individuellen Anforderungen am besten unterstützt.

Aus diesem Grund werden im Folgenden einige Bibliotheken auf ihren Funktionsumfang, ihre Performanz, Erlernbarkeit und Bedienbarkeit geprüft und diese anhand fest definierter Erwartungen bewertet, um ein möglichst objektives Bild zu ermöglichen.

## b) Technischer Hintergrund

### Grunddefinitionen

#### **Graph** - „Was ist ein Graph?“

Hinter dem Begriff Graph verstecken sich gleich mehrere Bedeutungen. Ein Graph, wie in der Mathematik bekannt, wird meist als Darstellung von verschiedenen Funktionen verwendet. Auch einzelne Bilder bzw. Darstellungen, vielmehr bekannt unter dem Namen Diagramm, werden oft auch als solche bezeichnet. Hier ist allerdings der Begriff gemeint, wie er in der Graphentheorie genutzt wird. Ein Graph ist vereinfacht formuliert eine Struktur, welche auf Knoten und Kanten basiert, wobei die übliche Schreibweise  $G = (V, E)$  entspricht.

#### **Knoten** – „Vertex“

Knoten sind eine abstrakte Repräsentation von Objekten, deren Relation durch den Graphen modelliert wird (Wikipedia, 2015). Für jeden Graphen gibt es eine ganze Liste bzw. Menge an solchen Knoten. Abgekürzt mit dem Buchstaben „V“ für das englische Wort „Vertex“. Bsp.:  $V = \{A, B, C, \dots, H\}$

#### **Kante** – „Edge“

Kanten stellen die Verbindungen zwischen den Knoten dar und symbolisieren einen Aufwand oder eine Relation zwischen Diesen. Wie auch bei den Knoten existiert für jeden Graphen eine Liste bzw. Menge solcher Kanten, abgekürzt mit dem Buchstaben „E“ für „Edge“. Jede Kante besitzt immer mindestens zwei Informationen: Ursprung- und Zielknoten. Alle Kanten „E“ sind dem entsprechend ein Tupel aus Elementen der Menge V bzw. ein Element der Menge  $V \times V$ . So steht z.B.  $e(A, B)$  für eine Kante zwischen den Knoten A und B.

## Schlinge

Eine Schlinge bezeichnet eine Kante, die von einem Knoten ausgeht und zu demselben Knoten führt.

## Graph Darstellungen

Für die Darstellung von Graphen gibt es unterschiedliche Möglichkeiten. Die prominentesten hierbei sind das „Knoten-Kanten-Diagramm“, die Adjazenzmatrix bzw. die Inzidenzmatrix und die Adjazenzliste.

	A	B	C	D	E	F
A	0	1	1	0	0	0
B	1	0	0	1	0	0
C	1	0	0	1	1	0
D	0	1	1	0	1	1
E	0	0	1	1	0	1
F	0	0	0	1	1	0

Abbildung 2: Adjazenzmatrix

Eingehend	Knoten	Ausgehend
C, B	<- A ->	B, C
D, A	<- B ->	A, D
E, D, E	<- C ->	A, D, E
F, E, C, B	<- D ->	B, C, E, F
F, D, C	<- E ->	C, D, F
E, D	<- F ->	D, E

Abbildung 3: Adjazenzliste

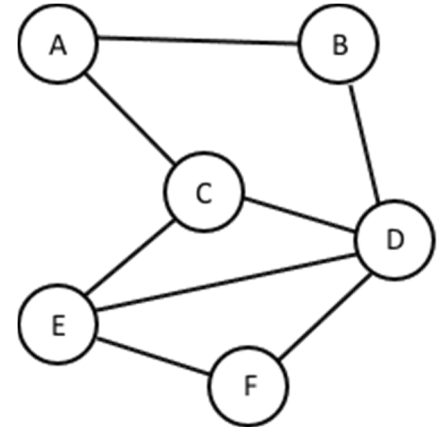


Abbildung 1: ungerichtetes Knoten-Kanten-Diagramm

## Anwendungsbereiche

Ein Graph kann immer dann genutzt werden, wenn ein Problem sich als Graph modellieren lässt. Dabei wird versucht die Lösung mittels einer problemangepassten Kombination aus algorithmischer und visueller Analyse zu finden. So werden Graphen zum Beispiel verwendet, um Straßennetze darzustellen. Hierbei werden die Straßen (oder Wege) durch Kanten und die Kreuzungen mittels Knoten modelliert. Durch zusätzliche Eigenschaften der Kanten lassen sich auch Straßenarten unterscheiden (Autobahn, etc.). Hier könnte man für die verschiedenen Eigenschaften zum Beispiel unterschiedliche Breiten, Linien oder Farben wählen. Anhand eines solchen Graphen ist es dann möglich sich in einer Stadt oder ähnlichem zurecht zu finden. Dies kann entweder durch einen Menschen (visuelle Analyse) oder automatisch durch einen Algorithmus (algorithmische Analyse) geschehen.

## Graphen Probleme

Die Modellierung mittels Graphen bringt allerdings auch ein paar Probleme, sowohl im algorithmischen, wie auch im visuellen Bereich, mit sich. Im algorithmischen Bereich spielt unter anderem die Komplexität eine Rolle, während im visuellen Bereich die große Anzahl der zu zeichnenden Elemente auf kleinem Raum leicht zu einer starken Unübersichtlichkeit führt. Dieser Effekt wird auch als „Visual Clutter“ bezeichnet.

Bedenkt man nun, dass in einem Graphen gegebenenfalls auch Kanten- und Knotengewichte visualisiert werden sollen, wird deutlich, dass sich die Probleme nur noch verschlimmern je mehr Informationen in den Graphen gepackt werden.

## Algorithmen (Laufzeit/Darstellung)

Um genau solchen Problemen entgegen zu wirken, kommen spezielle Algorithmen zum Einsatz. Diese sorgen nicht nur für eine bessere Lesbarkeit und Übersicht, sondern bestimmen auch meist über die Performanz und den Ressourcenverbrauch.

Gute Algorithmen zeigen eine gute Performanz, wenig Ram-Verbrauch und noch weniger, wenn ihre Aufgaben abgearbeitet wurden. Zudem sorgen sie für eine sinnvolle Verteilung der Objekte, sodass unterschiedliche Merkmale und Probleme sichtbar werden. Speziell bei Graphen bieten sich meist kräfte-basierte (force-directed) Algorithmen an. Diese verteilen und verschieben die Knoten basierend auf der Anzahl und Stärke der Kanten und simulieren physikalische Kräfte, ähnlich zu Springfedern. Wenn diese Layout-Algorithmen nach einer gewissen Zeit zum Stillstand kommen, so erhält man visualisierte Graphen, welche sich selbst verteilt und angeordnet haben.

## Unterschiede: Bibliothek und Werkzeug („Tool“)

Ein Werkzeug ist ein fertiges Programm, das bestenfalls noch durch sogenannte Plugins erweitert werden kann. Hierbei ist aber bereits eine bestimmte Struktur durch den Entwickler des Werkzeuges vorgegeben, sodass man nicht alle Freiheiten hat, um eigene Funktionalität in dieses Werkzeug zu integrieren. Aus diesem Grund wird für Entwickler zumeist eine Bibliothek veröffentlicht, die im Grunde genommen nicht mehr ist als eine Sammlung von Funktionen. Diese Funktionen sind aber bereits auf einem so hohen Level, dass dem Entwickler einer Software durch die Nutzung der Bibliothek viel Arbeit abgenommen wird. Auch hier hat der Entwickler nicht unbedingt alle Freiheit in die Funktionen einzugreifen, kann aber deutlich besser beeinflussen, wie sich die daraus resultierende Anwendung verhält. So kann der Entwickler auch völlig eigene Interaktionskonzepte in die Anwendung integrieren, was mit einem Tool in der Regel nicht möglich ist.

## 2. Test-Suite

### a) Testkriterien

Im Folgenden wird unser Bewertungsbogen vorgestellt, anhand dessen die Bibliotheken bewertet wurden. Die enthaltenen Leitfragen sollen das einheitliche und vergleichbare Bewerten erleichtern und sicherstellen. In jedem Fall werden zu dem, was getestet wurde, ein messbares Ergebnis sowie ein erklärender Kommentar festgehalten. Als Testgraph dient in jedem Fall der „Wicket“-Datensatz (vgl. 2b) Testdaten).

#### **Art und Anzahl der Importformate:**

Wie viele Standard-Importformate werden von der Bibliothek unterstützt?

#### **Einrichtung und Einbindung:**

Wie viel Zeit musste investiert werden, bis die Bibliothek den ersten Graph angezeigt hat?

**Render-Performanz:**

Wie viel Zeit vergeht zwischen Laden und Anzeigen des Graphen?

**Übersichtlichkeit des Graphen:**

Wie übersichtlich ist und bleibt der Graph bei größer werdenden Datensätzen?

Können Knoten und Kanten auf normaler Zoomstufe noch unterschieden werden?

Werden zu kleine Elemente ausgeblendet um die Übersichtlichkeit zu wahren?

Schafft es der Layout-Algorithmus die Elemente sinnvoll und übersichtlich anzuordnen?

Werden Kanten geclustert?

**Interaktion:**

Welche Arten der Interaktionen werden von der Bibliothek unterstützt?

**Interaktions-Performanz:**

Wie flüssig lässt sich mit einem Graphen arbeiten?

Ab welcher Graph-Größe werden beim Verschieben / Zoomen Lags wahrgenommen?

Wie stark sind die Lags bei einer Interaktion?

Wie machen sich diese Bemerkbar?

Können Maßnahmen ergriffen werden, dass Interaktionen bei großen Graphen flüssiger laufen?

Wie stark verschlechtert sich die Performanz des größten Graphen im Vergleich zum kleinsten Graphen?

**Hauptspeicherverbrauch:**

Wie viel RAM benötigt das Programm (mit und ohne Daten)?

**CPU-Last:**

Wie viel CPU-Leistung wird für das Rendern benötigt?

Wird die Grafikkarte zum Rendern mitgenutzt?

**Dokumentation:**

Wie gut und ausführlich wurde Dokumentiert?

Wie leicht verständlich ist die Dokumentation?

In welchen Sprachen ist die Dokumentation verfügbar?

Gibt es Beispiele und sind diese vollständig und hilfreich?

Werden verschiedene Fragestellungen (FAQ) aufgegriffen und erklärt?

Wird der Zusammenhang der einzelnen Komponenten (Viewer / Models / Renderer) der Bibliothek klar?

Werden auch fortgeschrittene Anforderungen erklärt (Einfügen von Interaktionen / Highlighting / ...)?

**Dokumentationsverfügbarkeit und -qualität:**

Wie einfach lässt sich die Dokumentation benutzen?

In welchem Format liegt die Dokumentation vor?

Ist die Dokumentation aktuell / wann wurde sie zuletzt überarbeitet?

**Code Dokumentation:**

Gibt es Code-Kommentare?  
Durchgehend oder nur partiell?  
Sind diese Kommentare verständlich?  
In welcher Sprache sind die Code Kommentare verfasst?  
Wie hilfreich sind die Kommentare?

**Live-Rendering:**

Werden Veränderungen im Graphen live dargestellt?  
Wie performant ist das Live-Rendering?  
Ist das Live-Rendering automatisch aktiviert oder einstellbar?  
Muss ein Live-Rendering simuliert werden (nach jeder Änderung wird der Graph neu gezeichnet)?

**Visuelle Besonderheiten:**

Welche visuellen Besonderheiten werden unterstützt? Gibt es Highlighting?  
Kann man die Knotenform, -farbe, -größe ändern und ändern sich diese automatisch?

**Graph-Eigenschaften:**

Können gerichtete Kanten dargestellt werden und werden diese erkennbar dargestellt?  
Werden Schlingen dargestellt?  
Sind Kantengewichte erkennbar?  
Wie werden die verschiedenen Graph-Informationen visualisiert?

**Layout-Algorithmen:**

Gibt es unterschiedliche Layout-Algorithmen?  
Sind die Algorithmen mittels Parameter anpassbar?  
Gibt es auch Layout-Algorithmen die nicht auf einem Kräftemodell basieren?

**Stabilität:**

Wie verhält sich die Bibliothek bei der Ausführung der Funktionen?  
Stürzt das Programm bei großen Graphen ab?  
Arbeitet es langsam oder nimmt so viele Ressourcen in Anspruch, dass der Computer spürbar verlangsamt wird?  
Wie anfällig ist die Bibliothek gegenüber Fehlern?  
Falls die Bibliothek Standardformate unterstützt: Wie anfällig ist die Bibliothek bei defekten Daten?  
Konnten Fehlermeldungen / Deadlocks oder ähnliches produziert werden?



## b) Testdaten

Unsere Testdaten bestehen aus Abhängigkeitsgraphen verschiedener Java-Software. Diese wurden uns vom VISUS zur Verfügung gestellt (On the Impact of Software Evolution on Software Clustering, 2012) und müssen auch nicht der jeweils aktuellen Softwareversion entsprechen. Da aber die Abhängigkeitsgraphen einer Software, je nach Umfang, auch sehr komplex werden können sind Sie ideal dafür geeignet die Layout-Qualitäten der verschiedenen Bibliotheken zu testen. Als Haupt-Testdatensätze haben sich hierbei die Abhängigkeitsgraphen von jFTP, jUnit und Wicket erwiesen.

**jFTP** ist mit 78 Knoten und 144 Kanten unser kleinster Testgraph. Dabei hat jeder Knoten im Schnitt 1,8 ausgehende Kanten.

Der Abhängigkeitsgraph von **jUnit** beinhaltet 119 Knoten und 356 Kanten und ist damit auch noch relativ klein. Allerdings hat hier jeder Knoten etwa 3 ausgehende Kanten, was für die Layout-Algorithmen wiederum eine größere Herausforderung darstellt.

**Wicket** ist der Graph mit den meisten Elementen. Er enthält 622 Knoten und 3057 Kanten, was im Schnitt 5 ausgehenden Kanten pro Knoten entspricht. Aus diesem Grund ist Wicket auch die Bewertungsgrundlage für die Fähigkeiten der Bibliotheken. Zum einen ist er groß genug um zu zeigen, wie die Algorithmen skalieren, zum anderen hat er eine hohe Kantenmenge, die auch schnell zu Visual Clutter führen kann.

## 3. Graph Bibliotheken

### a) Auswahlkriterien

Die Auswahl von Bibliotheken erfolgt mit Hilfe von bestimmten Kriterien. Diese Kriterien werden im weiteren Verlauf nicht mehr explizit getestet, da eine Bibliothek bzw. ein Tool nur dann aufgeführt wird, wenn alle Auswahlkriterien erfüllt sind. Dabei setzen sich die Auswahlkriterien wie folgt zusammen:

**Knoten-Kanten-Darstellung:** Es muss möglich sein, ein einfaches, ungerichtetes Knoten-Kanten-Diagramm zu visualisieren.

**Automatisches Layout:** Es muss die Möglichkeit bestehen, dass Knoten und Kanten automatisch angeordnet werden.

**Letzter Release:** Da wir möglichst aktuelle Bibliotheken und Tools testen wollen, darf der letzte Release höchstens fünf Jahre (2010) zurück liegen.

**Grundlegende Funktionen in adäquater Zeit nutzbar:** Die erste Implementierung darf nicht länger als 4 Stunden dauern. Danach soll es möglich sein einen ersten Graphen anzuzeigen, wobei danach noch nicht zwingend eigene Testdaten, sondern zunächst auch einen Standard-Graph angezeigt werden darf.

**Nutzung auf herkömmlichen Maschinen möglich:** Die Bibliothek sollte auf einem handelsüblichen PC oder Laptop ausführbar bzw. benutzbar sein.

## b) Warum keine Tools?

Nach einer ausführlichen Recherche im Internet über existierende Tools und Bibliotheken zur Graphvisualisierung, haben wir eine große Auswahl an Tools und Bibliotheken gefunden und in einer Liste die wichtigsten Eckdaten zusammengefasst. Von den gefundenen Ergebnissen schieden aber auch einige aufgrund der genannten Kriterien wieder aus. Ein weiteres Ausschlusskriterium war die Programmiersprache, welche wir gut genug beherrschen müssen, um später auch brauchbare Testresultate zu erhalten. Dennoch hatten wir mehr als genug Auswahl und nur sehr begrenzte Kapazitäten.

Mit der Zielsetzung die Bewertungen nicht oberflächlich sondern eher gründlicher machen, wurde beschlossen zunächst nicht mehr als acht Bibliotheken und Tools zu testen. Bei dieser geringen Menge erschien es uns aber nicht sinnvoll Tools und Bibliotheken zu testen, da sich Tools nur schwer mit Bibliotheken vergleichen lassen. So führte uns die Überlegung, dass man mit Bibliotheken einen deutlich größeren Freiheitsgrad hat und nicht nur die vorgegebenen Interaktionsmöglichkeiten nutzen kann, zu der Entscheidung ausschließlich Bibliotheken zu testen.

## c) Gefundene Bibliotheken und Begründung für die Auswahl

Aus unserer Liste der verfügbaren Bibliotheken (vgl. vorhergehender Abschnitt „Warum keine Tools?“) wurden nun einige wenige Bibliotheken ausgewählt, welche dann entsprechend näher betrachtet und getestet wurden. Hierbei war das erste Auswahlkriterium die Programmiersprache, für welche die Bibliothek entwickelt wurde. Die, aufgrund unserer Kompetenzen, zur Auswahl stehenden Programmiersprachen sind Java, JavaScript und C#.

Das zweite Kriterium war das Datum der letzten veröffentlichten Version der Bibliothek, da keine Bibliotheken getestet werden sollten, welche schon seit mehr als vier Jahren nicht mehr weiter entwickelt wurden. Damit jede Bibliothek in einem angemessenen Rahmen getestet werden kann, wurde die maximale Anzahl der zu testenden Bibliotheken zunächst auf acht beschränkt, was sich schnell als eine optimale Anzahl bezüglich unserer Kapazitäten herausgestellt hat.

In Abbildung 4 ist die Liste der gefundenen Bibliotheken zu sehen. Aus dieser Liste wurden die Bibliotheken ausgewählt, die von uns näher betrachtet und getestet wurden. Diese ausgewählten Bibliotheken sind in der Liste grün hinterlegt. Die grau hinterlegten Einträge sind ausgeschieden, da diese zu lange nicht mehr weiterentwickelt wurden und die orange hinterlegten Einträge scheiden aufgrund der Programmiersprache aus. Blau hinterlegte Einträge sind veraltete Versionen, für die bereits ein Nachfolger existiert. Die rot hinterlegten Einträge sind aufgrund besonderer Probleme ausgeschieden. So haben wir im Falle von Keylines keine Testlizenz erhalten, um die Bibliothek testen zu können und im Falle von NodeXL konnte die Evaluation aufgrund einer fehlenden Dokumentation (die CHM-Datei ist leider defekt, eine andere Dokumentation existiert nicht) nicht durchgeführt werden. Die restlichen, farblich nicht hinterlegten Einträge standen zwar zur Auswahl, wurden kapazitätsbedingt nicht für die Evaluation ausgewählt.

Name	Sprache	Datum	Lizenz	Letzter Release	URL
yFiles WPF	C#	26.11.2014	Kommerziell	April 2015	<a href="http://www.yworks.com/en/products_yfileswfp_about.html">http://www.yworks.com/en/products_yfileswfp_about.html</a>
GINY	?Java?	30.11.2014	LGPL	31.08.2005	<a href="http://csbi.sourceforge.net/index.html">http://csbi.sourceforge.net/index.html</a>
tceetree	C	30.11.2014	SDL	22.11.2014	<a href="http://sourceforge.net/projects/tceetree/">http://sourceforge.net/projects/tceetree/</a>
GraphX	C#	01.12.2014	Apache	17.07.2014	<a href="https://graphx.codeplex.com/">https://graphx.codeplex.com/</a>
NodeXL	C#	01.12.2014	MS-PL	23.01.2014	<a href="http://nodexl.codeplex.com/">http://nodexl.codeplex.com/</a>
Graph#	C#	26.11.2014	Apache	01.06.2009	<a href="https://graphsharp.codeplex.com/">https://graphsharp.codeplex.com/</a>
Tulip	C++	30.11.2014	LGPL	29.09.2014	<a href="http://tulip.labri.fr/TulipDrupal/">http://tulip.labri.fr/TulipDrupal/</a>
zoomhub	Coffee Script	01.12.2014	MIT	17.10.2014	<a href="http://zoomhub.org/">http://zoomhub.org/</a>
UbiGraph	Diverse	30.11.2014	Apache	24.06.2008	<a href="http://ubitylab.net/ubigraph/">http://ubitylab.net/ubigraph/</a>
IsaViz	Java	30.11.2014	?	Mai 2010	<a href="http://www.w3.org/2001/11/IsaViz/">http://www.w3.org/2001/11/IsaViz/</a>
JGraph	Java	30.11.2014	BSD	27.11.2014	<a href="https://www.jgraph.com">https://www.jgraph.com</a>
nytlabs Streamtools	Java	01.12.2014	Apache	20.11.2014	<a href="http://nytlabs.github.io/streamtools/">http://nytlabs.github.io/streamtools/</a>
JGraphX	Java	26.11.2014	BSD	10.11.2014	<a href="https://github.com/jgraph/jgraphx">https://github.com/jgraph/jgraphx</a>
prefuse	Java	26.11.2014	BSD	14.08.2013	<a href="http://prefuse.org/">http://prefuse.org/</a>
Jung	Java	26.11.2014	BSD	29.05.2013	<a href="http://jung.sourceforge.net/">http://jung.sourceforge.net/</a>
Cytoscape	Java	30.11.2014	LGPL v2	25.04.2013	<a href="http://www.cytoscape.org">http://www.cytoscape.org</a>
Cytoscape	Javascript	08.01.2015	LGPL v2	22.12.2014	<a href="http://www.cytoscape.org">http://www.cytoscape.org</a>
TouchGraph	Java	30.11.2014	Apache	08.04.2013	<a href="http://sourceforge.net/projects/touchgraph/">http://sourceforge.net/projects/touchgraph/</a>
TreeMap	Java	30.11.2014	MIT	08.04.2013	<a href="http://treemap.sourceforge.net">http://treemap.sourceforge.net</a>
Hypertree	Java	30.11.2014	MIT	22.03.2013	<a href="http://hypertree.sourceforge.net">http://hypertree.sourceforge.net</a>
Walrus	Java	30.11.2014	GPL	30.03.2005	<a href="http://www.caida.org/tools/visualization/walrus/">http://www.caida.org/tools/visualization/walrus/</a>
IVC Software Framework	Java	30.11.2014	Apache	23.08.2004	<a href="http://iv.slis.indiana.edu/sw/">http://iv.slis.indiana.edu/sw/</a>
D3js	JavaScript	01.12.2014	BSD	Oktober 2014	<a href="http://d3js.org/">http://d3js.org/</a>
Keylines	JavaScript	01.12.2014	Kommerziell	?	<a href="http://keylines.com/">http://keylines.com/</a>
sigmajs	JavaScript	01.12.2014	MIT	13.11.2014	<a href="http://sigmajs.org/">http://sigmajs.org/</a>
linkurious	neo4j, Java	01.12.2014	Kommerziell	?	<a href="http://linkurio.us">http://linkurio.us</a>
graph-tool	Python	30.11.2014	GPL v3	11.09.2014	<a href="http://graph-tool.skewed.de">http://graph-tool.skewed.de</a>
NodeBox OpenGL	Python	01.12.2014	BSD	06.05.2013	<a href="http://www.cityinabottle.org/nodebox/">http://www.cityinabottle.org/nodebox/</a>
NodeBox 1	Python	01.12.2014	MIT	18.04.2013	<a href="https://www.nodebox.net/code/index.php/Home">https://www.nodebox.net/code/index.php/Home</a>
NodeBox 3	Python (Java?)	01.12.2014	GNU	24.11.2014	<a href="https://www.nodebox.net/node/">https://www.nodebox.net/node/</a>
RDF - Gravity	Tool	30.11.2014	Haftungsausschluss	?2003?	<a href="http://semweb.salzburgresearch.at/apps/rdf-gravity/index.html">http://semweb.salzburgresearch.at/apps/rdf-gravity/index.html</a>
GraphViz	Tool	30.11.2014	EPL	13.04.2014	<a href="http://www.graphviz.org">http://www.graphviz.org</a>
Visual Browser	Tool	30.11.2014	NLP	02.09.2013	<a href="http://nlp.fi.muni.cz/projekty/visualbrowser/index.html#sec08">http://nlp.fi.muni.cz/projekty/visualbrowser/index.html#sec08</a>
Gephi	Tool	30.11.2014	CDDL + GPL v3	03.01.2013	<a href="http://gephi.github.io">http://gephi.github.io</a>
SemaSpace	Web-Tool	30.11.2014	CC-BY NC SA	Mai 2010	<a href="http://residence.aec.at/didi/FLweb/">http://residence.aec.at/didi/FLweb/</a>
yFiles HTML	HTML5 / JS	26.11.	Kommerziell	April 2015	<a href="http://www.yworks.com/en/products_yfiles/yfiles-for-html/">http://www.yworks.com/en/products_yfiles/yfiles-for-html/</a>

Abbildung 4: Liste der gefundenen Tools und Bibliotheken

## d) Testmaschine

Für den Test der Bibliotheken und Tools nutzen wir zwei (fast) baugleiche Laptops mit den folgenden Hardwarespezifikationen:

Tabelle 1: Hardwaredetails der Test-Laptops

Bezeichnung	Beschreibung
<b>Typ</b>	Acer E5-571G-539F mit modifizierter Hardware
<b>CPU</b>	Intel Core i5-4210U mit 1,7 GHz
<b>RAM</b>	8192 MB RAM DDR3 mit 1600 MHz
<b>Festplatte</b>	120 bzw. 512 GB SSD, 512 GB Hybrid
<b>Grafikkarte</b>	NVIDIA GeForce 840M
<b>Display</b>	1920 x 1080, 60Hz

## 4. Bewertung der Java-Bibliotheken

### a) Einzelbewertungen

#### JGraphX



Abbildung 5: Datensatz jFTP mit JGraphX

#### Kurzbeschreibung:

JGraphX ist eine freie Bibliothek der Entwickler, die auch mxGraph als JavaScript Bibliothek kommerziell vertreiben und zudem die Internetseite draw.io<sup>1</sup> verwalten. Da das Hauptaugenmerk dieser Firma auf ihrem Produkt mxGraph liegt und die Java Bibliothek JGraphX mit eingeschränkten Funktionen von dieser ableitet, tragen die enthaltenen Komponenten meist das Kürzel ‚mx‘ vor ihren Namen. Die fast gleichnamige Bibliothek JGraph, welche gegen Ende des Jahres 2014 nicht mehr weiterentwickelt wurde, stellt den Vorläufer dar. Da die Bibliothek von Grund auf neu geschrieben wurde, kam ein neuer Name zur Verwendung.

Aktuelle Versionen können auf der referenzierten GitHub-Seite heruntergeladen werden. Für die von uns durchgeführten Versuche wurde die Version 3.2 verwendet. Sollten von den Benutzern Fragen oder Vorschläge aufkommen, so wird ein offizielles Forum hierfür angeboten.

#### Art und Anzahl der Importformate:

**Ergebnis:** Es kann nur ein XML-Format importiert werden. (✓)

**Kommentar:** JGraphX selbst besitzt eine Klasse, welche das Ein- und Auslesen von XML Dateien in eine interne Graph Struktur ermöglicht. Anschließend können noch zusätzlich weitere Knoten oder Kanten hinzugefügt und angepasst werden. In unseren Versuchen und Zeitmessungen wurde jedoch eine eigene Struktur und Importfunktion verwendet.

<sup>1</sup> <https://www.draw.io>

## Einrichtung und Einbindung:

Ergebnis: Schnelle und einfache Einbindung. **Innerhalb einer Stunde** konnten die ersten Graphen angezeigt werden.

Kommentar: Die Bibliothek liegt sowohl als Kompilat, wie auch als Source-Code vor und erlaubt direkte Verwendung oder Anpassung, sofern benötigt. Durch unterschiedliche Anleitungen und dem beiliegenden Beispiel sind die notwendigen Schritte für die ersten Graphen schnell herausgefunden und umgesetzt.

## Render-Performanz:

Ergebnis: Bei kleinen Graphen benötigt die Bibliothek keine 50ms für das Laden und Layouten. Für größere Graphen aber auch mal 2 Sekunden. (+ +)

Kommentar: Abgesehen von wenigen Millisekunden, welche für das Laden benötigt werden, benötigt das Rendern der Graphen bei größeren Graphen nicht unverhältnismäßig länger. Als Ergebnis wird ein nicht animierter Graph angezeigt.

## Übersichtlichkeit des Graphen:

Ergebnis: Die Übersichtlichkeit ist mittelmäßig bis schlecht.

Kommentar: Trotz unterschiedlicher Layout Möglichkeiten sind die meisten Ansichten nicht übersichtlich. Die einzelnen Elemente sind häufig dicht beieinander, überlappen sich meist und werden nicht automatisch überarbeitet. Jedes Element im Graphen wird einzeln dargestellt. Des Weiteren sind die Kanten ohne Anpassung im Vordergrund, sodass größere Bereiche nicht les- bzw. interpretierbar sind.

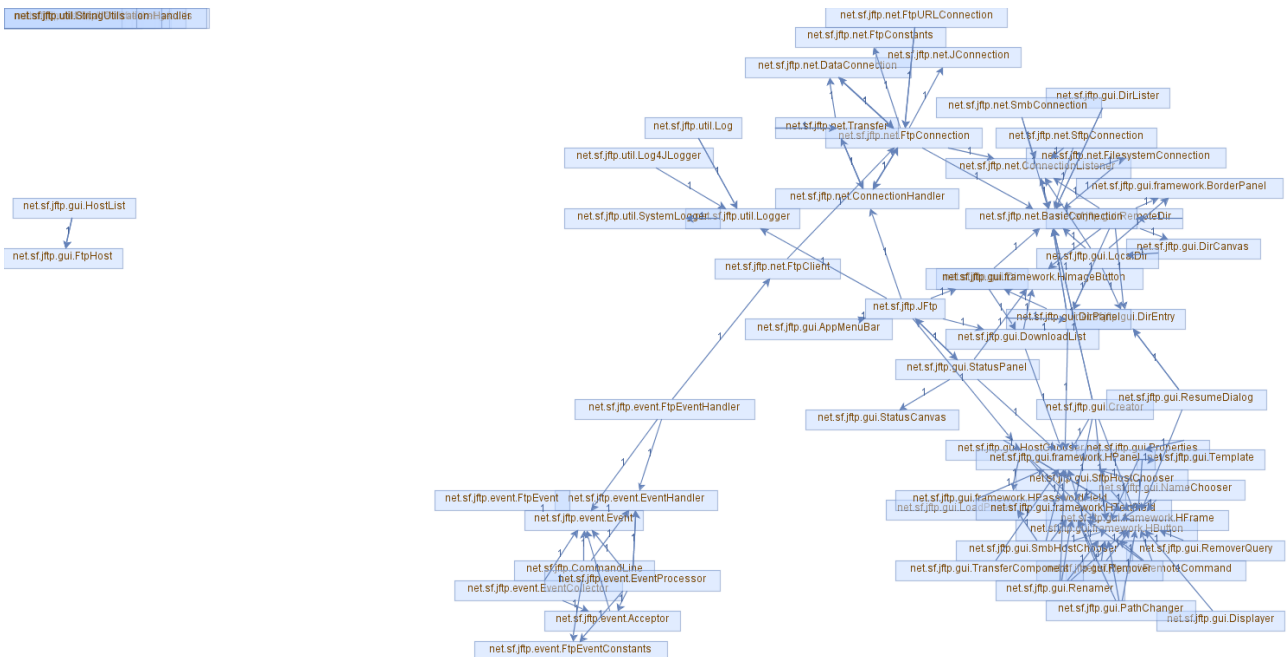


Abbildung 6: Datensatz jFTP in JGraphX

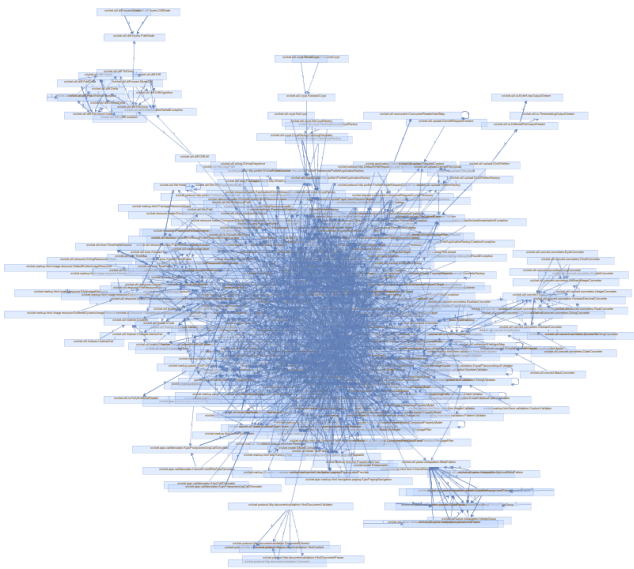


Abbildung 7: Datensatz Wicket mit JGraphX

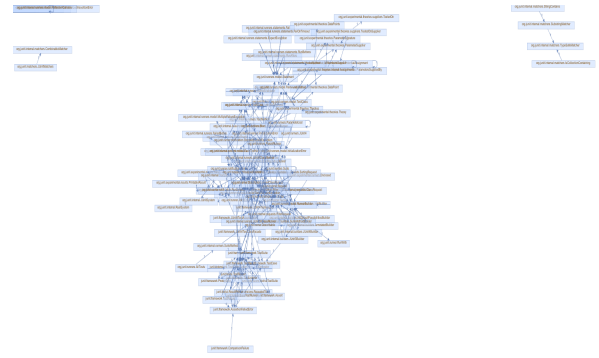


Abbildung 8: Datensatz JUnit mit JGraphX

### Interaktion:

**Ergebnis:** Verschieben der Ansicht und Objekten, Anpassen der Größe und Form von Objekten, Fensterübergreifendes klonen, Kombinieren und trennen von Objekten, Direktes Editieren der Texte, Selektion von mehreren Objekten. (++)

**Kommentar:** Von der Bibliothek selbst werden bereits einige Interaktionen unterstützt, wobei der Fokus auf dem Hinzufügen neuer Objekte und deren Anpassung liegt. Funktionen wie z.B. Zoom müssen von dem Entwickler selbst mit wenig Code hinzugefügt werden. Die dafür nötigen Funktionen und Methoden liegen bereits vor und müssen nur aufgerufen werden. Wenn ein Knoten verschoben wird, aktualisieren sich die Kanten automatisch. Dieses Verhalten kann aber auch abgeschaltet werden, sodass auch nur der Knoten verschoben werden kann. Das Duplizieren von Objekten sowie deren Verbindungen wird nativ unterstützt.

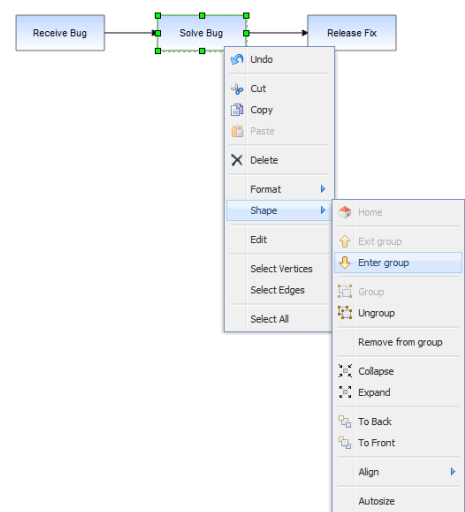


Abbildung 9: Visuelle Einstellungsmöglichkeiten (JGraphX, 2015)

### Interaktions-Performanz:

**Ergebnis:** Bei den Interaktionen reagiert JGraphX sehr gut. (++)

**Kommentar:** Bei kleinen Graphen sind die Interaktionen alle sehr flüssig. Bei größeren Graphen mit mehr Elementen gibt es durchaus kleinere Lags, die aber nicht allzu schwerwiegend sind.

### Hauptspeicherverbrauch:

**Ergebnis:** Für den größten Graphen werden etwa **75 MB** benötigt.

**Kommentar:** Beim Test wurde der Graph mehrfach hintereinander neu gerendert. Hierbei fällt auf, dass der Speicherverbrauch mit jedem Rendern leicht ansteigt.

### CPU-Last:

**Ergebnis:** Kurze Spitzen mit bis zu **100%** Auslastung.

**Kommentar:** Für das Rendern und für die Interaktionen wird die CPU-Last kurzzeitig erhöht.

### Dokumentation:

Ergebnis: Die Dokumentation ist sehr gut. (✓)

Kommentar: Es existiert eine umfassende und vollständige Javadoc Dokumentation online und auch in dem offline geladenen Paket. Die Dokumentation ist in Englisch verfasst und nicht allzu ausführlich. Außerdem gibt es ein englisches Handbuch in dem die grundlegenden Funktionen erklärt werden.

### Dokumentationsverfügbarkeit und –Qualität:

Ergebnis: Die Verfügbarkeit und die Qualität sind sehr gut. (++, Details siehe Tabelle 2)

Kommentar: Sowohl die Dokumentation als auch das Handbuch sind sowohl online Verfügbar, liegen aber auch dem Download bei. Dadurch ist die Dokumentation eigentlich immer zur Hand.

### Code Dokumentation:

Ergebnis: Nur im Quellcode, dort aber vollständig und verständlich. (++, Details siehe Tabelle 2)

Kommentar: Da der Quellcode selbst zur Verfügung steht, hat man auch Zugriff auf die Code-Kommentare. Zudem ist die Javadoc-Dokumentation vollständig.

### Live-Rendering:

Ergebnis: JGraphX unterstützt kein Live-Rendering. (✗)

Kommentar: Das Live-Rendering wird von der Bibliothek aus nicht unterstützt. Manche Interaktionen wie das Verschieben oder Klonen von Objekten wird jedoch transparent während der Durchführung angezeigt.

### Visuelle Besonderheiten:

Ergebnis: Umfangreiche Einstellungsmöglichkeiten.

Kommentar: Jedes Element im Graphen ist in fast allen visuellen Eigenschaften frei und direkt anpassbar. Sei es die Höhe oder Breite einzelner Knoten, welche direkt durch Interaktion manipuliert werden können oder unterschiedliche Formen, Farben, Linienkurven und Symbole an diesen. Jedes Element im Graph kann individuell dargestellt werden. Interaktionen werden durch farbiges hervorheben, zusätzlich angezeigten Rahmen oder transparenten Vorschauen unterstützt.

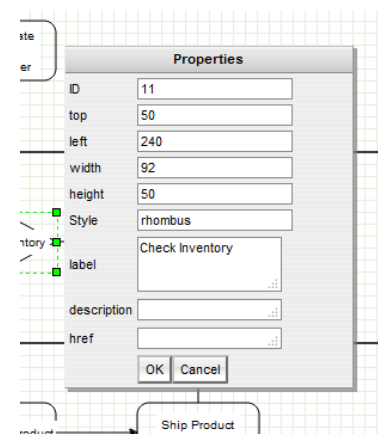


Abbildung 10: Einstellungsmöglichkeiten für die Spitze einer Linie (JGraphX, 2015)

### Graph-Eigenschaften:

Ergebnis: Es werden verschiedene Einstellungen gemacht werden.

Kommentar: Gerichtete bzw. ungerichtete Kanten können genauso angezeigt werden wie Schlingen. Auch die Kantengewichte kann man anzeigen lassen. Hierfür müssen lediglich für die entsprechenden Elemente entsprechende Styles angegeben werden.

Zusätzlich können weitere Informationen zu dem Graphen ermittelt werden, wie z.B. der kürzeste Weg zwischen zwei Knoten oder ob der Graph eine Baumstruktur aufweist.

## Layout-Algorithmen:

Ergebnis: Es stehen mehrere Algorithmen zur Auswahl.

Kommentar: Es stehen mehrere Layout-Algorithmen zur Auswahl. Unter anderem: Circle, CompactTree, Organic, FastOrganic und Hierarchisches Layout. Von der Performanz sind die meisten Layout-Algorithmen sehr schnell. Die einzige Ausnahme stellt hier der klassische Organic-Algorithmus dar, der sehr zeitintensiv ist. Zusätzlich sind noch Parameter für individuelle Anpassungen vorhanden.

## Stabilität:

Ergebnis: Die Stabilität von JGraphX ist sehr zuverlässig. (++, Details siehe Tabelle 2)

Kommentar: Sei es die Performanz oder der Verbrauch von Ressourcen, beides ist sehr gut. Auch bei größeren Graphen sind keine Fehler oder Abstürze aufgetreten. Die Bibliothek schreibt außerdem entwicklungsrelevante Warnungen und Informationen in die Java-Konsole um den Entwickler mit wichtigen Informationen zu unterstützen.

## Anmerkungen und Sonstiges:



Abbildung 11: Mögliche Symbole für gerichtete Kanten in JGraphX



Abbildung 12: Knotenformen von JGraphX



Abbildung 13: Kantenarten von JGraphX



## Prefuse

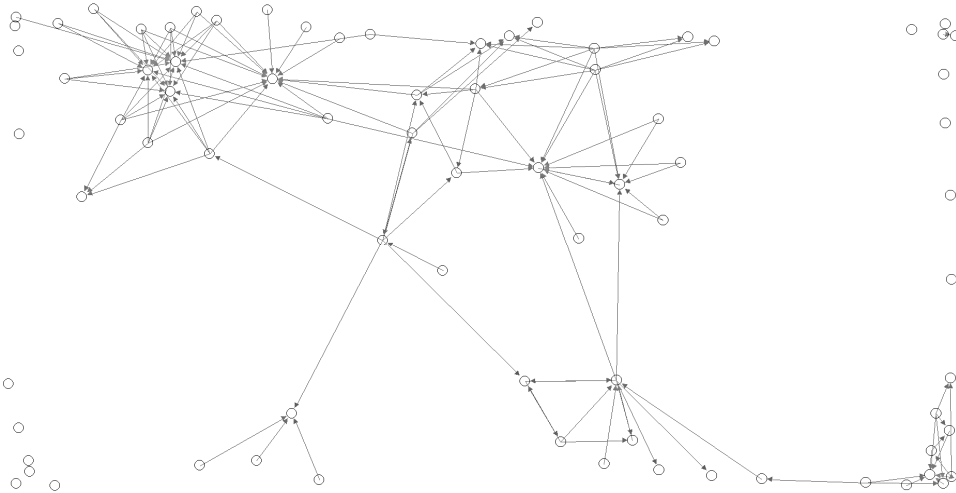


Abbildung 14: Datensatz jFTP mit Prefuse

### Kurzbeschreibung:

Prefuse, welches an der Universität von Kalifornien in Berkley entwickelt und später auf GitHub zur Verfügung gestellt wurde, bietet umfangreiche Möglichkeiten für interaktive und animierte Visualisierungen. Obwohl das letzte Update Ende April des Jahres 2014 erfolgte, ist die Funktionalität erwähnenswert. Trotz dessen ist eine starke Abnahme an Fragen und Beiträgen in Foren wie SourceForge (SourceForge, 2015) zu beobachten.

Überzeugend ist vor allem die Tatsache, dass in jeden einzelnen Schritt der Erstellung einer Visualisierung eingegriffen werden kann. Von dem Einlesen der Daten, über das Einrichten der Visualisierungsoberfläche, dem Einstellen der Renderer, den Interaktions- und Animationsmöglichkeiten und dem abschließenden Zusammenfügen, wird sehr viel Freiheit und Anpassungen angeboten.

### Art und Anzahl der Importformate:

Ergebnis: Prefuse unterstützt den Import von GraphML, TreeML, CSV-Dateien. (✓)

Kommentar: Die Bibliothek unterstützt sowohl das Lesen als auch das Schreiben mehrerer Formate. Für unseren Test ist dabei der GraphML-Reader bzw. -Writer der wichtigste. Ebenso lassen sich aber auch Bäume im TreeML-Format, CSV-Dateien und einfache Textdateien im- und exportieren. Dabei müssen aber die Textdateien nach einem bestimmten Format aufgebaut sein. Besonders erwähnenswert ist, dass auch direkt Daten aus einer Datenbank verarbeitet werden können.

### Einrichtung und Einbindung:

Ergebnis: Die Einbindung ist nicht ganz einfach und bedarf etwas Vorarbeit. (3h)

Kommentar: Aufgrund der Tatsache, dass man die Bibliothek als Quell-Code erhält muss man diese zuerst selbst kompilieren. Dies setzt voraus, dass man ein wenig Grundwissen besitzt, es werden jedoch alle benötigten Komponenten mitgeliefert. Eine rudimentäre Anleitung und ein kleines Skript werden auch mitgeliefert. Kleinere Anpassungen, die allerdings in der Anleitung nicht erläutert werden, müssen jedoch gemacht werden.

**Render-Performanz:**

Ergebnis: Das Rendering ist insgesamt gut. (+ +)

Kommentar: Das Rendern erfolgt ohne Verzögerung und braucht auch bei größeren Graphen nicht mehr als ein paar Sekunden. Sofern man das Live-Rendering aktiv hat, erfolgt für größer werdende Graphen keine flüssige Animation, da jeder einzelne Zeitschritt doch ein wenig Rechenzeit benötigt.

**Übersichtlichkeit des Graphen:**

Ergebnis: Stark abnehmend bei wachsender Anzahl von Elementen.

Kommentar: Trotz der Tatsache, dass jedes visuelle Merkmal beeinflusst werden kann, ist die Übersichtlichkeit bei größeren Graphen meist nicht gegeben. Mehrere Layout-Algorithmen verschaffen unterschiedliche Möglichkeiten den Graph zu visualisieren, jedoch lässt sich Visual Clutter nicht vermeiden. Vorteilhaft ist die automatische Skalierung des Renderns bezüglich der Fenstergröße und die Berechtigung über diese Grenzen hinaus zu gehen, wobei die Größe der Elemente beibehalten wird. Eine Überlagerung der Objekte wird allerdings nicht verhindert, sodass öfters mehrere Ebenen von Knoten und Kanten an derselben Stelle entstehen.

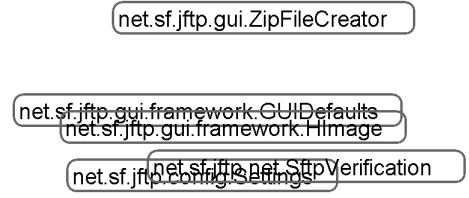


Abbildung 15: Überlappung auch bei wenigen Objekten



Abbildung 16: Datensatz JUnit mit Prefuse



Abbildung 17: Datensatz jFTP in Prefuse



Abbildung 18: Datensatz Wicket mit Prefuse

### **Interaktion:**

Ergebnis: Drag & Drop von Objekten, Verschieben des gesamten Graphen, Zoom, Reaktion auf Maus-Events. (+)

Kommentar: Interaktionen sind sehr schnell und einfach eingebaut. Je nach gewünschten Möglichkeiten muss lediglich eine Einstellung bei der Generierung angegeben werden und schon sind Interaktionen wie Zoom oder Drag & Drop eingebunden. Der Entwickler kann jedoch recht unkompliziert seine eigenen Interaktionen hinzufügen oder vorher eingerichtete Anpassungen automatisch auf Interaktion ausführen lassen.

### **Interaktions-Performanz:**

Ergebnis: Sehr gut und flüssig – auch bei großen Graphen. (+ +)

Kommentar: Auch bei Interaktionen ist die Performanz sehr gut. Jede Anpassung wird ohne Verzögerung durchgeführt und sofern während der Generierung angegeben auch animiert ausgeführt. Auch bei größeren Datensätzen ist der Performanzverlust kaum merkbar. Standardmäßig kann nur mit Knoten interagiert werden, nicht mit Kanten.

### **Hauptspeicherverbrauch:**

Ergebnis: Etwa **50 MB** für den Graphen mit den meisten Elementen.

Kommentar: Beim Test fiel auf, dass der Speicherverbrauch sich nach diversen Interaktionen bei etwa 120 MB eingependelt hat.

### **CPU-Last:**

Ergebnis: Die CPU wird für das Layouting **vollständig (100%)** ausgelastet. Die Interaktionen sind zwar auch Leistungshungrig, lasten aber die CPU nur zu maximal 40% aus.

### **Dokumentation:**

Ergebnis: Die englische Javadoc-Dokumentation ist nur bedingt Vollständig und zudem relativ knapp formuliert. (✓)

Kommentar: Prefuse stellt eine Online-Dokumentation zur Verfügung. Jede Methode oder Schnittstelle besitzt eine, wenn auch knappe Erklärung. Es liegen aber mehrere Demo-Anwendungen bei, welche wenige Kommentare beinhalten, sodass direkte Experimente dort durchgeführt werden können.

### **Dokumentationsverfügbarkeit und -qualität:**

Ergebnis: Die Dokumentation ist nur als Website verfügbar. (0, Details siehe Tabelle 2)

Kommentar: Die Javadoc-Dokumentation ist nur online verfügbar.

### **Code Dokumentation:**

Ergebnis: Bedingt, jedoch vollständig. (+ +, Details siehe Tabelle 2)

Kommentar: Zu unterscheiden ist zum einen die Bibliothek als JAR-Datei und der offenliegende Source Code. Während in dem Archiv selbst keine Dokumentation oder Kommentare zu finden sind, so sind diese ausführlich in dem beiliegenden Source Code enthalten. Zusätzliche Kommentare für das Verständnis sind dort ebenfalls enthalten.

## Live-Rendering:

Ergebnis: Live-Rendering wird von Prefuse unterstützt. (+ +, Details siehe Tabelle 2)

Kommentar: Neben dem einmaligen Layouten existiert zusätzlich die Möglichkeit diese für eine bestimmte Zeit, für bestimmte Events oder auch unendlich lange laufen zu lassen. Die Kombinationen sind ebenfalls umsetzbar, da Eventketten erstellt werden können. Mit größer werdenden Graphen sinkt jedoch stark die Performanz, da alle Einzelschritte berechnet werden müssen. Ruckelnde und springende Elemente bzw. Graphen sind die Folge.

## Visuelle Besonderheiten:

Ergebnis: Es gibt umfangreiche und vielfältige Einstellungsmöglichkeiten.

Kommentar: Durch die Bibliothek bestehen nur wenige Einschränkungen. Für jedes Element des Graphen (z.B. Knoten und Kanten) kann der Renderer angegeben werden, der dafür sorgt, dass das Element nach einem bestimmten Schema dargestellt wird. Viele visuelle Aspekte sind automatisierbar, sodass zum Beispiel anhand bestimmter Informationen die Farben angepasst werden oder nach einer Interaktion die Elemente ein anderes Aussehen bekommen. Sogenannte „DecorationItems“ ermöglichen zusätzlich das Erweitern von Objekten, sodass Knoten und Kanten visuell angepasst werden können ohne dafür einen separaten Renderer nutzen zu müssen.

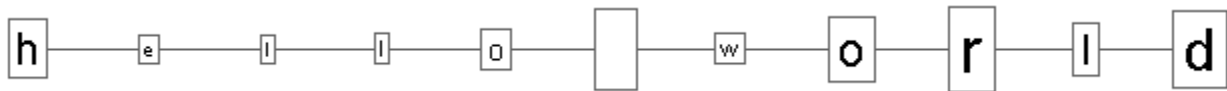


Abbildung 19: Automatisch an die Textgröße angepasste Knoten in Prefuse

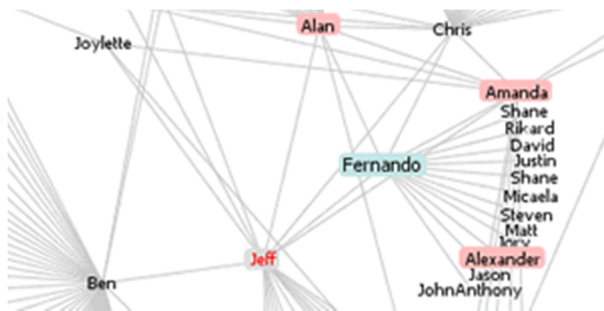


Abbildung 20: Beispiel für visuelle Besonderheiten in Prefuse (Prefuse, 2015)

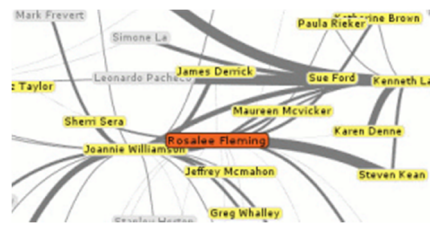


Abbildung 21: Beispiel der verschiedenen Kantenanpassungen in Prefuse (Prefuse, 2015)

## Graph-Eigenschaften:

Ergebnis: Verschiedene Graph-Eigenschaften werden nur bedingt unterstützt.

Kommentar: Durch Gruppierung der Elemente können unterschiedliche Regeln für die visuellen Eigenschaften angewendet werden. Manche unterstützen direktes Mapping von eingelesenen Informationen in den Objekten auf eine angelegte Skala, sodass als Beispiel Farben von dem Kantengewicht abhängig sind oder auch für unterschiedliche Knoten andere Symbole verwendet werden. Da dies aber nicht für alle Eigenschaften gilt, muss für den entsprechenden Fall erst nachgeschaut werden.

## Layout-Algorithmen:

Ergebnis: Die Algorithmen sind umfangreich und anpassbar.

Kommentar: Die Bibliothek liefert mehrere Layout-Algorithmen, die meist auf Kräften basieren, welche für die Verteilung zuständig sind. Trotz diverser einstellbarer Parameter existiert zusätzlich noch einen Simulator für die Einwirkung diverser Kräfte, welcher iterativ eingestellt werden kann.

## Stabilität:

Ergebnis: Prefuse läuft stabil, gibt aber immer wieder Warnungen aus. (+ +, Details siehe Tabelle 2)

Kommentar: Unabhängig von der Größe unserer Graphen zeigten sich keine Abstürze oder Fehler während des Betriebs. Verändert man jedoch zu schnell Einstellungen, welche durchgeführt werden sollen oder verwirft das Fenster während Animationen, so werden entsprechend Warnungen ausgegeben. Zusätzlich werden Informationen zu manchen Arbeitsschritten direkt ausgegeben. Fehler oder Abstürze entstehen nicht.

## Anmerkungen und Sonstiges:



Abbildung 22: Knotenformen in Prefuse

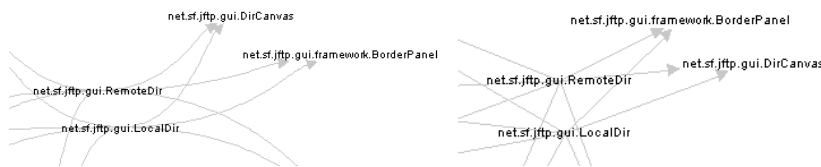


Abbildung 23: Kantenarten in Prefuse

## b) Bewertung

Bei den von uns getesteten Java-Bibliotheken zeigte sich vor allem, dass beide Bibliotheken unterschiedliche Zielgruppen anvisieren.

Während JGprahX, auch unter JGraph6 oder mxGraph zu finden, besonders viel Aufmerksamkeit auf den visuellen Aspekt gelegt hat überzeugt Prefuse hingegen mit vielen Möglichkeiten bezüglich des Renderns, des Layouts und den Animationen. JGraphX legte augenscheinlich besonders viel Wert auf die umfangreichen Anpassungsmöglichkeiten für jedes Objekt in der Visualisierung selbst, sowie die Interaktion zwischen den Objekten untereinander. So können Texte direkt ohne das eigentliche Objekt zu verändern, verschoben, umgeschrieben, hinzugefügt oder auch gelöscht werden. Zudem besitzen auch die Kanten unterschiedliche Möglichkeiten für Anpassungen. Neben den üblichen geraden Linien, welche zwei Knoten miteinander verbinden, kann man diese auch separat alle ohne Verknüpfung zu Knoten in die Visualisierung integrieren. Des Weiteren können die Kanten beliebig verformt und mit einem aus einer Liste an vordefinierten Symbolen an deren Anfang oder Ende versehen werden. So kann eine gerade Linie so angepasst werden, dass sie in unterschiedliche Teilbereiche zerlegt und diese Teilbereiche unabhängig voneinander verformt werden können um danach beispielsweise eine wellenförmige Linie zu erhalten. Im Vergleich hierzu bietet Prefuse wiederum die Möglichkeit durch zusätzliche Objekte die existierenden zu erweitern und so zu verbessern. Das Hauptaugenmerk liegt jedoch bei dieser Bibliothek verstärkt auf den Interaktionen und den darauf basierenden Anpassungen. So kann als Beispiel die Visualisierung durch eine Suche in einem Textfeld automatisch neu eingefärbt oder auch angeordnet werden. Durch die unterschiedlichen Möglichkeiten für Aktionen wie Farben, Layout oder Design, die wahlweise auf den eingelesenen Daten basieren können, kann die komplette Visualisierung nach einer Interaktion komplett anders aussehen und andere Objekte in den Vordergrund schieben.

Die Performanz von beiden Bibliotheken ist sehr gut, lediglich bei kleineren Graphen geht JGraphX in Führung und benötigt deutlich weniger Zeit. Mit größer werdenden Graphen pendeln beide Bibliotheken sich auf eine sehr ähnliche Performanz ein. Hierbei ist zu beachten, dass die verschiedenen Interaktionen, die den Graph verändern ebenfalls anbieten den Graph mittels Live-Transformation in die neue Form zu überführen, unabhängig davon, ob das Live-Rendering für den Graphen aktiviert ist oder nicht.

Bezüglich Layouts liefern JGraphX sowie Prefuse beide unterschiedliche bereits implementierte Algorithmen zur Auswahl. Die von uns getesteten Kräfte-basierten sind performant und anpassbar. Je nach Verwendungszweck und Anwendungsbereich ist eine dieser zwei Bibliotheken besser als die Andere. Beide erledigen die von ihnen angebotenen Funktionen jedoch ähnlich gut.

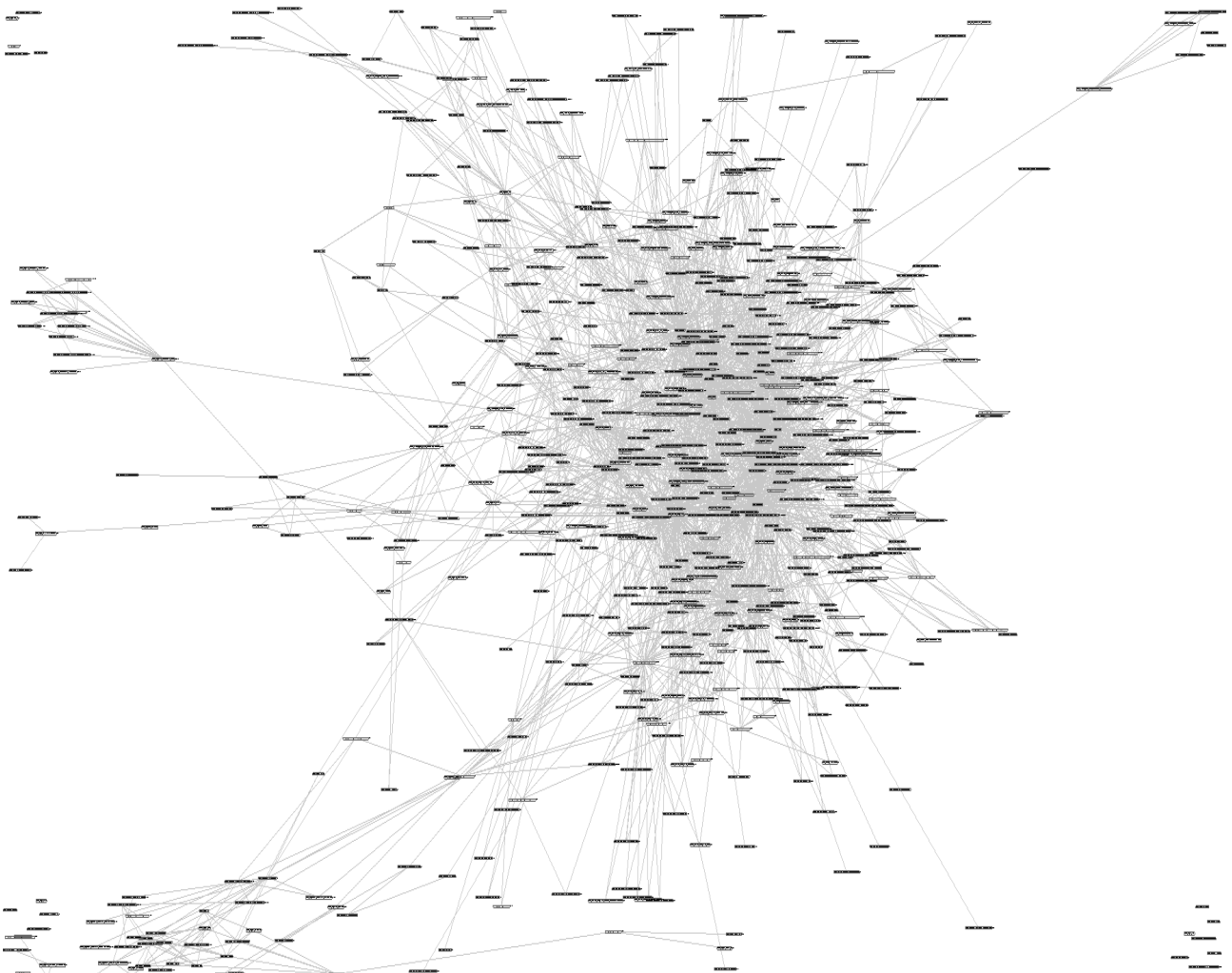


Abbildung 24: Datensatz Wicket in Prefuse mit FR-Algorithmus

## 5. Bewertung der JavaScript-Bibliotheken

### a) Einzelbewertungen

#### CytoScapeJS

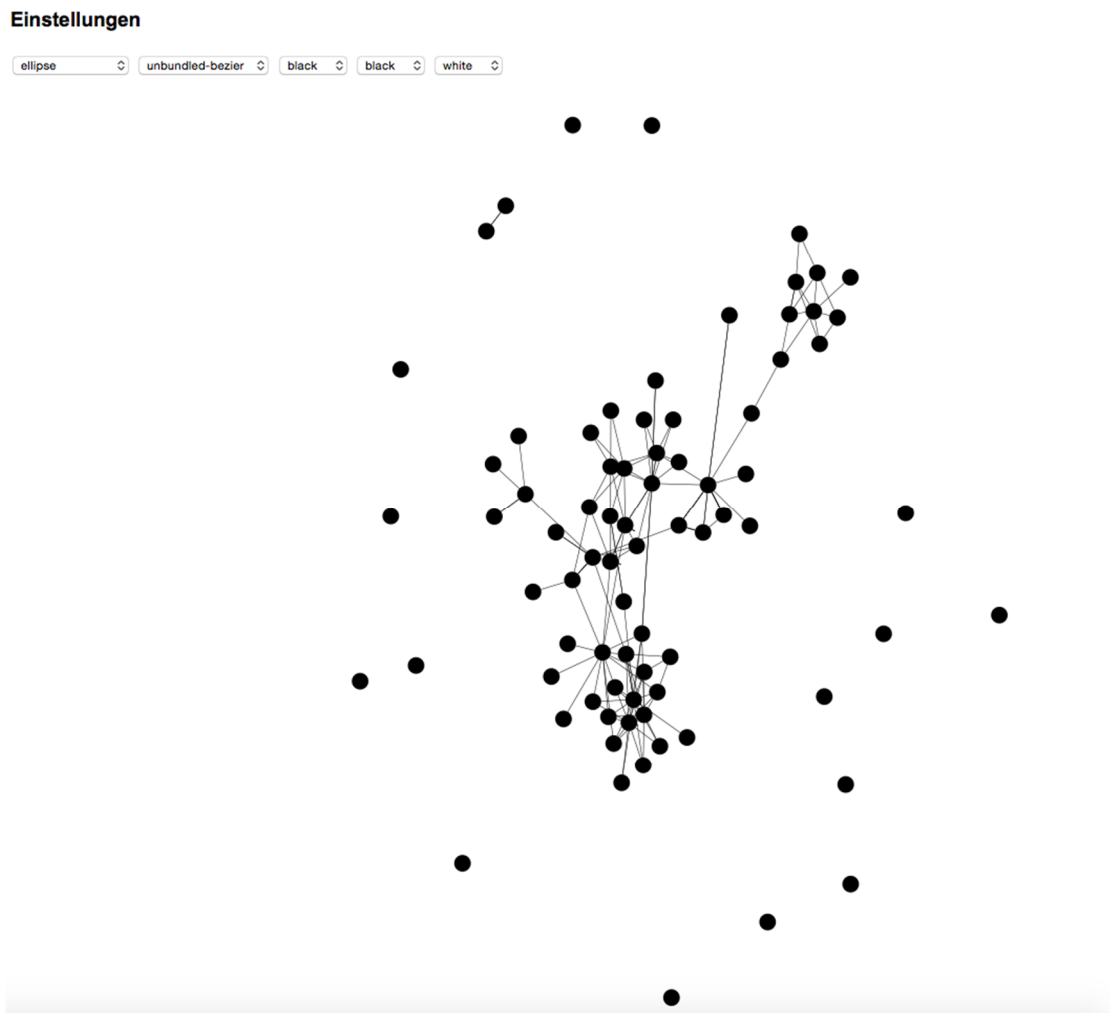


Abbildung 25: Datensatz jFTP mit CytoScapeJS

**Getestet mit:** Firefox Portable – Version 34.05

#### **Kurzbeschreibung:**

CytoScapeJS ist eine JavaScript Bibliothek, welche auch als Java-Bibliothek existiert. In diesem Fall wurde jedoch ausschließlich die JavaScript-Variante evaluiert. Um mit CytoScapeJS einen Graphen dazustellen, kann man die hierfür nötigen Daten im JSON-Format zur Verfügung stellen, muss hierbei jedoch das Einlesen der Daten manuell durchführen. Die Einbindung der Bibliothek und das Erstellen eines ersten Graphen mit ihr sind verhältnismäßig einfach und gelingen auch recht schnell. Dabei ist die Online-Dokumentation von großem Nutzen, welche auch zum Teil kleinere Codebeispiele enthält. Die Resultate für kleine Graphen sind recht gut, wobei die Bibliothek mit zunehmender Größe sehr starke Probleme bekommt. Grund hierfür scheint unter Anderem der nicht ganz ausgereifte Force-Directed Layout-Algorithmus zu sein, welcher bei sehr großen Graphen sehr unbrauchbare Ergebnisse erzeugt.

### Art und Anzahl der Importformate:

Ergebnis: CytoScapeJS unterstützt nur den Import von JSON (manuelle Implementierung des Einlesens). (X)

Kommentar: Die Bibliothek verfügt über keinerlei Import-Funktionalität. Sie ist allerdings in der Lage einen Graphen aus einem JSON-Objekt zu erstellen. Hierbei muss der Import der JSON-Daten allerdings manuell erstellt werden.

### Einrichtung und Einbindung:

Ergebnis: Sehr schnelle, einfache und unkomplizierte Einbindung. (3h)

Kommentar: Das Einbinden der Bibliothek ist recht einfach und unkompliziert. Sie ist sehr leicht verständlich und mithilfe der online Dokumentation kann man sich sehr gut in CytoScapeJS einarbeiten. Da nahezu jede Funktion oder Eigenschaft mithilfe eines Beispiels dokumentiert ist, benötigt man nicht sehr viel Zeit um den ersten Graphen anzuzeigen.

### Render-Performanz:

Ergebnis: Die Render-Performanz ist dürftig. (-)

Kommentar: Die Animation des Live-Renderings ruckelt bei größer werdenden Graphen immer stärker, was dann auch nicht zur Übersicht über die visualisierten Daten beiträgt.

### Übersichtlichkeit des Graphen:

Ergebnis: Sehr schlechte Übersicht über den Inhalt des (Standard-)Graphen trotz Layout-Algorithmus.

Kommentar: Die initialen Einstellungen eines Graphen sind sehr grob. Dieser Umstand lässt sich jedoch durch den sehr nützlichen nativen Zoom etwas mindern. So entstehen bei kleineren bis mäßigen Datensätzen Graphen, wie sie in den folgenden Abbildungen zu sehen sind. Bei etwas größeren Graphen sinkt die Übersichtlichkeit erheblich. Auch die Anwendung von verschiedenen Layout-Algorithmen bringt kaum Besserung, da diese bei sehr großen Graphen zum Teil sehr sinnlose Ergebnisse liefern (vgl. Abbildung 28). Grund dafür sind die zu großen Elemente. Sowohl die Knoten als auch deren Beschriftung ist initial recht groß und führt daher schnell zu Unübersichtlichkeit. Durch individuelle Einstellungen und dem Entfernen der Beschriftung kann dies allerdings verbessert und angepasst werden. Die Einstellungen müssen aber für die verschiedenen Datensätze unter Umständen separat optimiert werden.

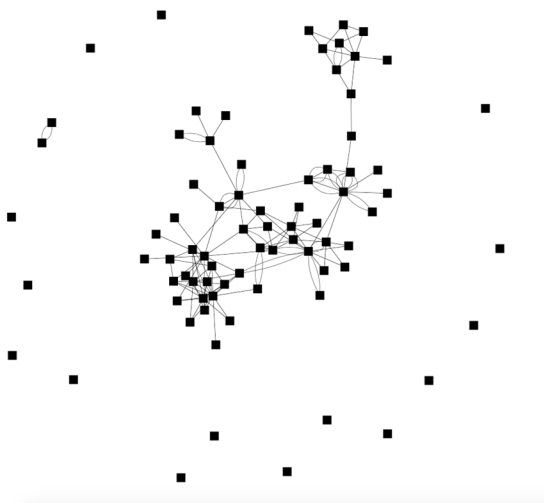


Abbildung 26: Datensatz jFTP mit CytoScapeJS

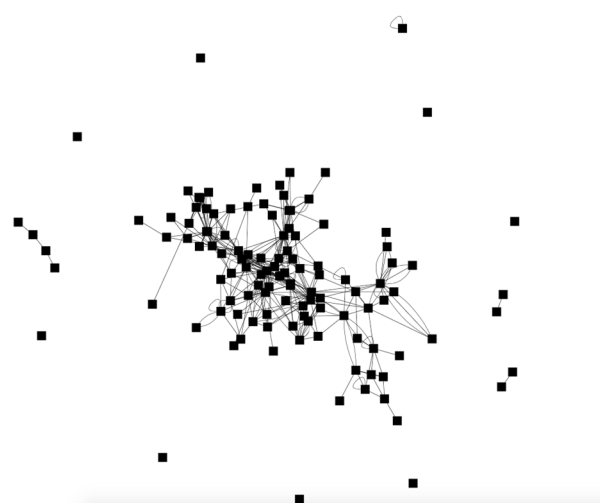


Abbildung 27: Datensatz junit mit CytoScapeJS





Abbildung 28: Datensatz Wicket mit CytoScapeJS

### Interaktion:

Ergebnis: Zoomen und verschieben einzelner Knoten (und Kanten) möglich. (+)

Kommentar: CytoScapeJS ermöglicht es einzelne Knoten zu verschieben. Hierbei werden alle Kanten entsprechen mit verschoben und der Knoten bleibt an der Stelle, zu welcher er bewegt wurde. Diese Eigenschaft geht auch bei der Anwendung eines Layout-Algorithmus nicht verloren. Abgesehen vom Verschieben von Knoten ist es auch möglich zu zoomen. Die Reichweite des Zooms ist sehr groß.

### Interaktions-Performanz:

Ergebnis: Flüssig bei kleinen Graphen, mangelhaft bei größeren. (+)

Kommentar: So lange man nur mit kleinen Graphen arbeitet ist sowohl der Zoom als auch das Verschieben des Graphen performant und flüssig. Werden die Graphen allerdings größer so zeigt sich, dass die Performanz doch stark abnimmt. So reagierte CytoScapeJS im Test bei unserem größten Graphen (~3700 Elemente) doch mit sichtbarer Verzögerung.

### Hauptspeicherverbrauch:

Ergebnis: Etwa **22 MB** mit Spitzen bis zu 65MB.

Kommentar: Nach dem Laden des Graphen, während des Live-Renderings benötigt CytoScapeJS etwa 65MB wobei dies auf durchschnittlich 22 MB zurück geht, sobald das Live-Rendering beendet ist.

### CPU-Last:

Ergebnis: Für das Rendern und bei Interaktionen wird die CPU vollständig ausgelastet. (**100%**)

### Dokumentation:

Ergebnis: Ausführliche und gute Dokumentation. (✓)

Kommentar: CytoScapeJS stellt eine Online-Dokumentation zur Verfügung. Diese Dokumentation ist ausführlich und bietet für fast alle Einstellungsmöglichkeiten Code-Beispiele. Diese Beispiele helfen sehr beim Verständnis und erklären die Einstellungen sehr anschaulich. Die Dokumentation gibt auch für alle Elemente ihre zugehörigen Attribute an. Dies erleichtert das Ändern der Optik des Graphen und seiner Elemente. Die Dokumentation macht einen gepflegten Eindruck und macht es sehr einfach die Einstellungen des Graphen zu verändern.

### **Dokumentationsverfügbarkeit und -qualität:**

Ergebnis: Die Verfügbarkeit ist sehr hoch (Online-Dokumentation), qualitativ gute Dokumentation. (++, Details siehe Tabelle 2)

Kommentar: Die Online-Dokumentation von CytoScapeJS beschreibt zwar nahezu alle Einstellungen, welche man vornehmen kann (zum Großteil auch mit Codebeispiel), es fehlen jedoch essentielle Dinge wie zum Beispiel das Rendern eines Graphen aus einem JSON-Objekt. Diese Möglichkeit ist in der Dokumentation nicht auf Anhieb zu finden und muss durch eine Suche im Internet überbrückt werden. Für die Anpassung des Graphen selbst ist die Dokumentation sehr hilfreich.

### **Code Dokumentation:**

Ergebnis: Sehr wenige, aber hilfreiche Kommentare im Code. (--, Details siehe Tabelle 2)

Kommentar: Im Code der Bibliothek befinden sich einige wenige Kommentare. Diese Kommentare beschränken sich hauptsächlich darauf einzelne Funktionen und Fallunterscheidungen zu erläutern. Es gibt zudem auch ein paar „ToDo“-Einträge des Entwicklers, welche nahelegen, dass die Entwicklung noch nicht vollständig abgeschlossen ist. Die wenigen Kommentare sind sehr hilfreich beim Verständnis bestimmter Codeabschnitte.

### **Live-Rendering:**

Ergebnis: Live-Rendering wird standardmäßig genutzt. (--, Details siehe Tabelle 2)

Kommentar: Das Live-Rendering des Graphen ist bei CytoScapeJS integriert, aber nicht wirklich performant. Das Live-Rendering von CytoScapeJS besteht darin in gewissen Intervallen den aktuellen Stand des Layout-Algorithmus statt einer flüssigen Animation anzuzeigen.

### **Visuelle Besonderheiten:**

Ergebnis: Viele verschiedene Formen von Knoten und Kanten.

Kommentar: Die Bibliothek unterstützt verschiedene Formen für Knoten. Hierzu gehören Rechtecke (mit oder ohne abgerundeten Ecken), Ellipsen, Dreiecke, Fünf-, Sechs- Sieben-, Achtecke und Sterne. Die Bibliothek ermöglicht auch das Darstellen von gerichteten oder nicht gerichteten Kanten. Für gerichtete Kanten gibt es die Möglichkeit verschiedene Formen auszuwählen. Es gibt Dreiecke, Quadrate, Kreise, Diamanten und T-Formen. Hierbei können diese Formen an 4 verschiedenen Positionen angezeigt werden. Abgesehen vom Anfang und Ende der Kante können die Formen noch an zwei Positionen dazwischen gesetzt und angezeigt werden.

### **Graph-Eigenschaften:**

Ergebnis: Unterstützung von Graph-Eigenschaften gibt es seitens CytoScapeJS nicht.

Kommentar: Die Bibliothek ermöglicht es nicht, den Graphen auf bestimmte Eigenschaften zu untersuchen oder gar diese nach Möglichkeit bei der Visualisierung darzustellen bzw. zu beachten.

### Layout-Algorithmen:

Ergebnis: Verschiedene force-directed Varianten, Random-Layout, Zentriertes oder Kreisförmiges Layout.

Kommentar: Die Bibliothek stellt verschiedene Möglichkeiten für das Layout eines Graphen zur Verfügung. Abgesehen von einer zufälligen Anordnung der Knoten gibt es die Möglichkeit die Knoten kreisförmig anzuordnen. Auch eine rasterförmige Anordnung ist möglich. Eine weitere Darstellungsmöglichkeit ist eine zentrierte Anordnung. Hierbei muss man eine Metrik definieren, mithilfe welcher bestimmt wird, welche Knoten zentrierter angezeigt werden als andere. Die Bibliothek ermöglicht es auch, die Knoten hierarchisch anzuordnen.

Für azyklische Graphen und Bäume gibt es ein Layout, welches die Knoten unter Berücksichtigung dieser Kriterien anordnet. Es gibt auch einige force-directed Layout-Varianten, welche auf individuellen Implementierungen einiger Entwicklern basieren.

### Stabilität:

Ergebnis: Okay für nicht allzu große Graphen. (+, Details siehe Tabelle 2)

Kommentar: Wenn die Graphen eine bestimmte Größe erreichen wurde im Test immer wieder nachgefragt, ob ein Skript der Bibliothek angehalten oder darauf gewartet werden soll. Außerdem ist der Ressourcen-Verbrauch teilweise so hoch, dass Firefox eine Fehlermeldung ausgegeben hat, dass die Test-Website Firefox ausbremst und ob die Skripte auf der Seite deaktiviert werden sollen.

### Anmerkungen und Sonstiges:

Die möglichen Knotenformen für CytoScapeJS sehen wie folgt aus:



Abbildung 29: Mögliche Knotenformen von CytoScapeJS

# D3JS

## Einstellungen

circle line black black white

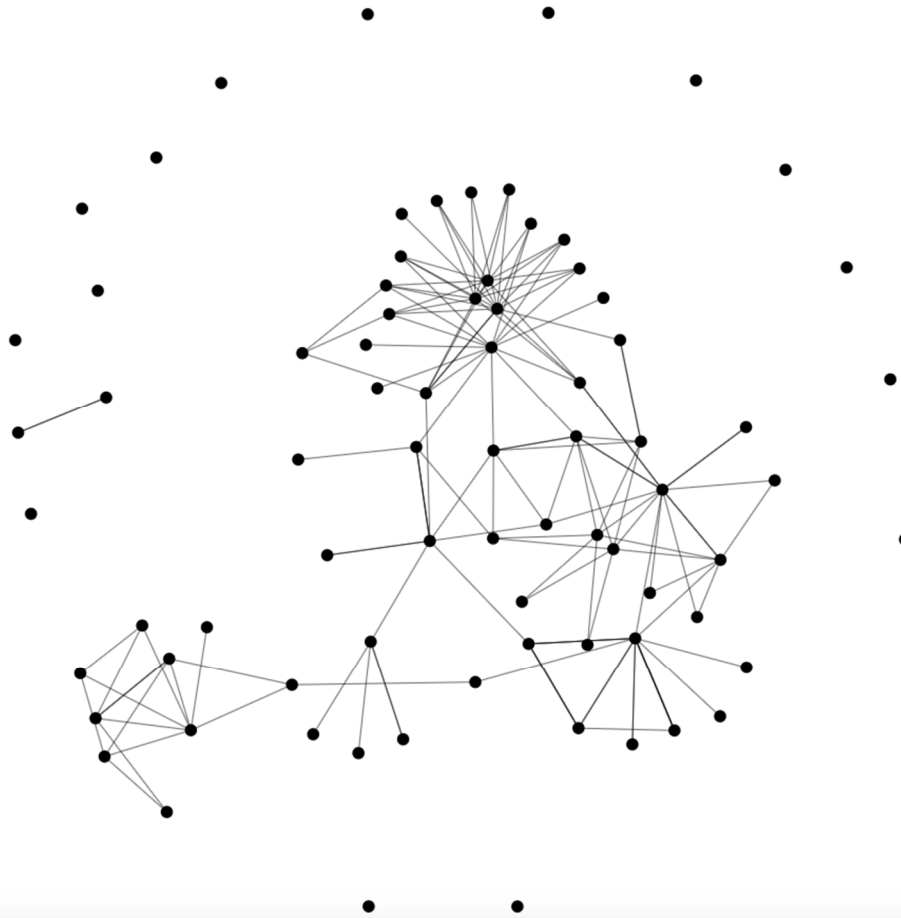


Abbildung 30: Datensatz jFTP mit D3JS

**Getestet mit:** Firefox Portable – Version 34.05

### Kurzbeschreibung:

D3.js ist eine der populärsten JavaScript Visualisierungsbibliotheken. Das Einarbeiten in diese Bibliothek gestaltet sich sehr einfach. Grund hierfür ist allem voran die sehr komfortable Importierungsfunktionalität, welche neben dem gängigen JSON-Format auch das XML-, CSV-, HTML- und TSV-Format unterstützt. Die einzige Einschränkung hierbei ist, dass Namen wie zum Beispiel der Knotenbezeichner fest vorgeschrieben sind. Aufgrund der wirklich einfachen Importierung von Graphdaten dauert es auch nicht lange, bis der erste Graph angezeigt werden kann. Dabei stellt man fest, dass der Force-Directed Layout-Algorithmus von D3.js einen sehr ausgereiften Eindruck macht und selbst bei sehr großen Datensätzen einen recht übersichtlichen Graphen erstellt. Einzig das Verschwinden von Knoten (und Kanten) aus dem Fenster muss hier negativ angemerkt werden. Es kann nämlich passieren, dass Knoten aus dem für die Anzeige zur Verfügung stehenden Fenster verschwinden und somit unerreichbar werden. Da keine native Scrollfunktionalität gegeben ist, bleiben diese zunächst verschwunden.

### Art und Anzahl der Importformate:

Ergebnis: Der Import von JSON, XML, CSV, HTML, TSV ist problemlos möglich. (✓)

Kommentar: Die Bibliothek D3.js stellt verschiedene Funktionen für das importieren von Graph-Daten zur Verfügung. Hierzu gehören Funktionen zum Auslesen und importieren von JSON-, Text-, XML-, CSV-, HTML- und TSV-Dateien. Hierbei müssen die Daten aber bestimmte Kriterien (wie z.B. Namen der Knotenbezeichner) besitzen, welche von der Bibliothek erkannt und eingelesen werden können. Bei erfolgreichem Parsen entsteht automatisch ein Graph mit welchem D3.js arbeiten kann.

### Einrichtung und Einbindung:

Ergebnis: Sehr einfache Implementierung und auch Importieren von Daten. (1h)

Kommentar: Die Einbindung der Bibliothek und das darstellen des ersten Graphen ist sehr einfach. Grund dafür sind unter anderem zahlreiche Beispiel-Implementierungen und eine sehr einfach gehaltene Import-Funktionalität. Die Bibliothek erstellt einen force-directed Graphen, welcher mit Initialwerten erstellt wird. Es ist durchaus möglich einzelne Werte zu überschreiben oder dynamisch über eigene Funktionen berechnen zu lassen.

### Render-Performanz:

Ergebnis: Sehr effizientes und flüssiges Rendering des Graphen. (++)

Kommentar: Die Dauer des Einlesens und Renderns des ersten Graphen geschieht sehr schnell. Zwischen dem Start des Imports- und Rednervorgangs und der Anzeige des Graphen vergehen in der Regel beim kleinsten Graphen etwa 50ms und beim größten Graphen etwa 200ms. Bei dieser ersten Anzeige des Graphen werden alle Knoten und Kanten an einer initial berechneten Position angezeigt. Direkt im Anschluss daran kann man dem Graphen dabei zusehen, wie er sich anhand der Kräfte verändert und die Knoten sich entsprechend an ihre Positionen bewegen. Dieser Vorgang dauert je nach Größe des Graphen etwas, verläuft aber weitgehend flüssig.

### Übersichtlichkeit des Graphen:

Ergebnis: Bei sehr großen Graphen müssen Einstellungen des Graphen angepasst werden. Kleine Graphen werden sehr übersichtlich dargestellt.

Kommentar: Die Übersichtlichkeit des Graphen ist stark abhängig von der Größe des Graphen. Verhältnismäßig kleinere Datensätze werden sehr gut und übersichtlich dargestellt, wie man den folgenden Abbildungen entnehmen kann. Bei sehr großen Graphen wird der Graph recht schnell unübersichtlich, da die Initialwerte des force-directed Graphen bei zentralen Knotenpunkten diese nicht stark genug „isolieren“. Dies lässt sich aber durch Anpassen der Werte verbessern. Bei sehr großen Graphen besteht die Gefahr, dass bei der Anwendung des Layout-Algorithmus einige Knoten aus dem Fenster verschwinden d.h. nicht mehr sichtbar und somit unerreichbar sind.

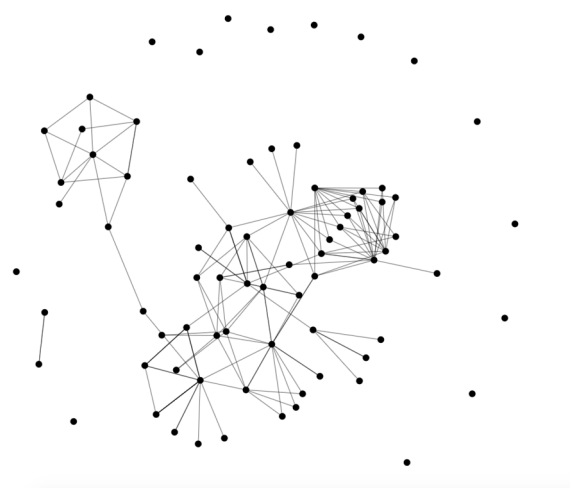


Abbildung 31: Datensatz jFTP mit D3JS

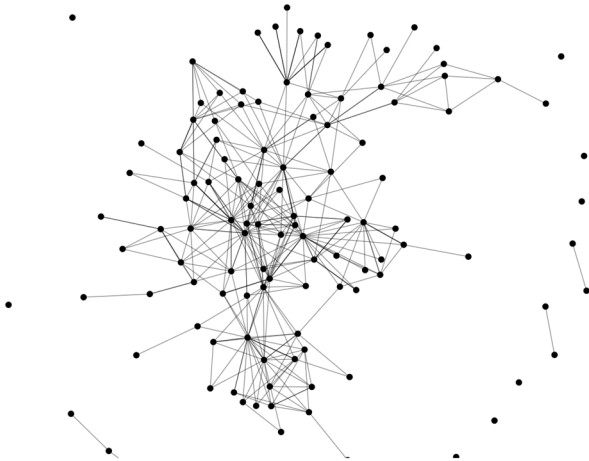


Abbildung 32: Datensatz jUnit mit D3JS

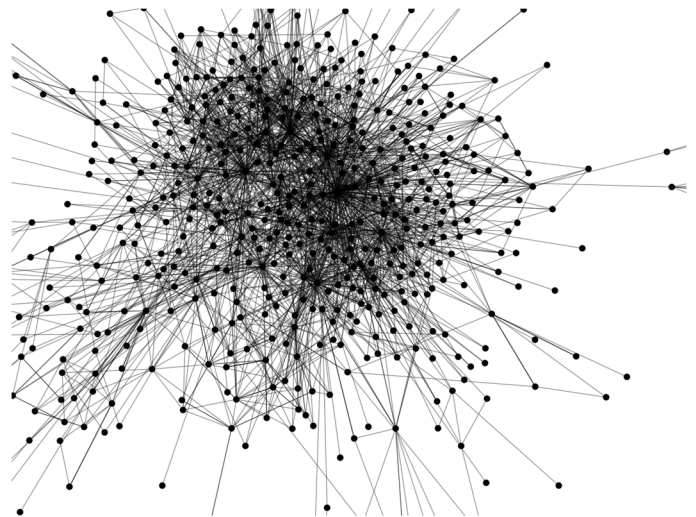


Abbildung 33: Datensatz Wicket mit D3JS

### Interaktion:

Ergebnis: Die Ansicht kann verschoben werden, was allerdings nicht beibehalten wird. (+)

Kommentar: Die Bibliothek ermöglicht das Umherziehen einzelner Knoten, wobei diese sich ihren Kräften entsprechend nach loslassen wieder entsprechend anordnen. Diese Funktion lässt sich durch eine Attribut an- und abschalten. Dies gilt allerdings nicht für den Umstand, dass Änderungen nicht beibehalten werden.

### Interaktions-Performanz:

Ergebnis: Gute Performanz bei kleinen Graphen, sichtbare Lags bei größeren Graphen. (+ +)

Kommentar: Die Performanz beim Drag&Drop einzelner Knoten ist sehr gut. Der Knoten und seine entsprechenden Kanten verändern sich dem gezogenen Knoten entsprechend. Selbst adjazente Knoten werden dem gezogenen Knoten entsprechend „hinterher“ gezogen. Dies geschieht sehr flüssig allerdings sind bei größeren Graphen doch deutlich Lags zu sehen.

### Hauptspeicherverbrauch:

Ergebnis: Etwa **15 MB** pro Graph.

Kommentar: Nach ca. 30 Sekunden sorgt die Garbage Collection dafür, dass die nicht mehr benötigten Ressourcen freigegeben werden.

### CPU-Last:

Ergebnis: Für Interaktionen und beim Rendern wird die CPU zu **100%** ausgelastet.

### Dokumentation:

Ergebnis: Sehr gute und ausführliche Dokumentation. (✓)

Kommentar: Die Dokumentation dieser Bibliothek ist sehr gut und ausführlich. Es existiert eine online Dokumentation, welche über jeden Browser auf <http://d3js.org> abgerufen werden kann. Hier existieren auch zahlreiche Code- und Implementierungsbeispiele, welche eine Einbindung sehr stark erleichtern. In dieser Dokumentation finden sich auch einige Attribute, welche individuell angepasst werden können. Daher ist die Dokumentation auch sehr hilfreich bei der optischen Anpassung der Graphen.

### **Dokumentationsverfügbarkeit und -qualität:**

Ergebnis: Verfügbarkeit sehr hoch (Online-Dokumentation), qualitativ sehr gute Dokumentation. (+ +, Details siehe Tabelle 2)

Kommentar: Die Online-Dokumentation von D3js hat eine sehr hohe Verfügbarkeit und enthält alle nötigen Informationen um Daten einzulesen und Graphen anzuzeigen. Die Dokumentation reicht als alleiniges Hilfsmittel bei der Nutzung der Bibliothek.

### **Code Dokumentation:**

Ergebnis: Unübersichtlicher und nicht dokumentierter Code. (✗)

Kommentar: Der Code der Bibliothek ist etwas unübersichtlich und weder dokumentiert noch kommentiert. Das Zurechtfinden im Code ist recht mühsam und es ist ohne Dokumentation und Kommentierung des Codes sehr zeitaufwändig die Funktionsweise einzelner Elemente anhand des Codes zu verstehen.

### **Live-Rendering:**

Ergebnis: Sehr schnelles und flüssiges Live-Rendering, welches sich auch beim Ziehen von Knoten nicht verschlechtert. (+, Details siehe Tabelle 2)

Kommentar: Sowohl das initiale Anzeigen als auch Änderungen im Graphen (z.B. durch Drag&Drop) werden live-gerendert und dargestellt. Dies geschieht sehr flüssig und wird mit zunehmender Größe des Graphen nicht wesentlich langsamer. Auch die auftretenden Lags sind zwar sichtbar aber keinesfalls gravierend.

### **Visuelle Besonderheiten:**

Ergebnis: Einige verschiedene Darstellungen von Knoten möglich.

Kommentar: Die Knoten des Graphen lassen sich durch verschiedene Objekte darstellen. Abgesehen von traditionellen Kreisen für einen Knoten lassen sie sich auch als Diamant, Dreieck, Quadrat und Kreuz darstellen.

### **Graph-Eigenschaften:**

Ergebnis: Unterstützung von Graph-Eigenschaften gibt es seitens D3js nicht.

Kommentar: Die Bibliothek ermöglicht es nicht, den Graphen auf bestimmte Eigenschaften zu untersuchen oder gar diese nach Möglichkeit bei der Visualisierung darzustellen bzw. zu beachten.

### **Layout-Algorithmen:**

Ergebnis: Knoten verschwinden nach Anwendung eines Layout-Algorithmus teilweise aus der Anzeigefläche.

Kommentar: D3js bietet einen Force-Directed Layout-Algorithmus, welcher auf den Graphen angewendet werden kann. Dieser Algorithmus ist unter Verwendung von „Verlet integration“ implementiert. Die Anwendung des Algorithmus auf den Graphen geschieht zunächst unter vollständiger Umsetzung der Gewichte bei der Positionierung der Knoten.

Mit fortschreitender Zeit werden die Gewichte immer weniger stark umgesetzt, was dazu führt, dass der Algorithmus letztendlich terminiert und das Layout fest steht. Dieses Prinzip wird auch „simulated annealing“ genannt. Das Layout des Graphen orientiert sich jedoch leider nicht an der Größe des Fensters in welchem der Graph angezeigt wird. Dies führt unter Umständen dazu, dass einige Knoten und Kanten aus dem Fenster verschwinden und nicht mehr erreichbar sind.

Abgesehen von einem force-directed Layout ist D3JS auch in der Lage ein hierarchisches Layout ebenso wie ein Histogramm-ähnliches Layout zu erstellen. Hierbei kann das hierarchische Layout neben einer klassischen hierarchischen Darstellung auch in einer kreisförmigen Hierarchie erfolgen. Auch TreeMaps, Baum-, Stack- und Kuchenförmige Layouts werden von D3JS ermöglicht. Zudem kann man Knotenbäume partitionieren lassen.

**Stabilität:**

Ergebnis: D3JS weist eine sehr hohe Stabilität auf. (++, Details siehe Tabelle 2)

Kommentar: D3JS ist relativ stabil und wirft keine Fehlermeldungen. Auch konnte kein Absturz von Firefox (Version 34.05 - Portable) provoziert werden.

**Anmerkungen und Sonstiges:**

Einige Knoten und Kanten werden außerhalb des Fensters platziert. In sieht man recht deutlich, dass das Fenster für den Graphen unterhalb der Einstellungen beginnt und dort einige Kanten plötzlich abgeschnitten sind. Diese Kanten führen zu Knoten, welche sich außerhalb des Fensters befinden.



Abbildung 34: Problem - Die Knoten und Kanten werden außerhalb des Fensters platziert

Potentiell lassen sich mit D3JS folgende Formen als Knoten verwenden:

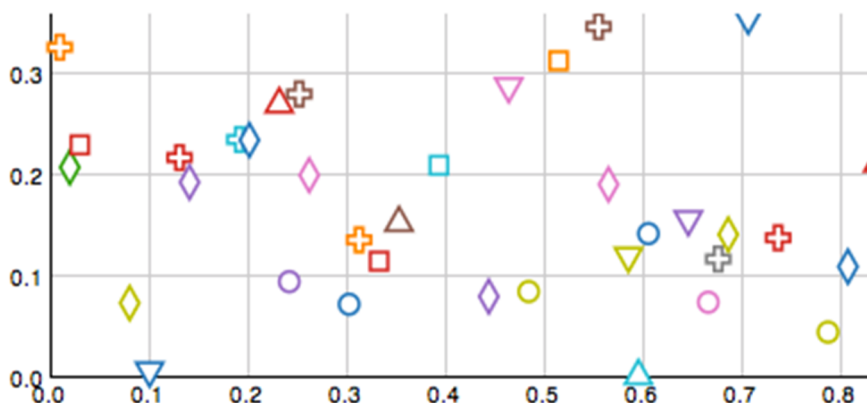


Abbildung 35: Knotenformen von D3JS



# SigmaJS

## Einstellungen

circle line black black white

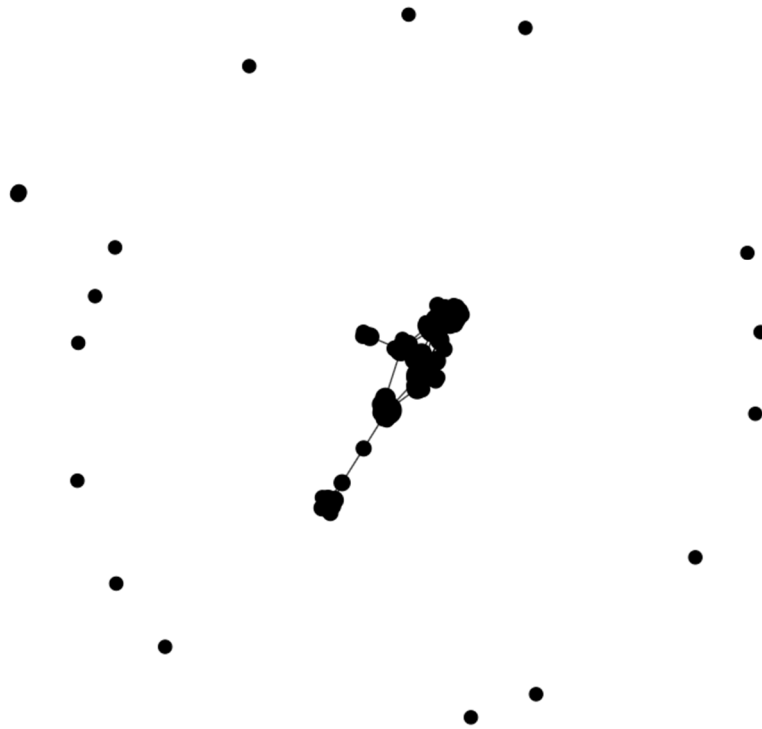


Abbildung 36: Datensatz jFTP mit SigmaJS

**Getestet mit:** Firefox Portable – Version 34.05

### **Kurzbeschreibung:**

SigmaJS ist die dritte zu evaluierende JavaScript Bibliothek. Auch diese Bibliothek bietet eine recht komfortable Importierungsfunktion für JSON- und GEXF-Daten an. Man muss lediglich den Dateipfad angeben und schon werden die Daten eingelesen. Nach dem einlesen ist es auch kein Problem mehr den ersten Graphen anzuzeigen, denn auch diese Einrichtung ist sehr schnell vollzogen. Hierbei gilt es aber zu beachten, dass man jedem einzelnen Knoten initiale Positionen zuweisen muss, da andernfalls keine Knoten angezeigt werden. Hat man dann den ersten Graph angezeigt, stellt man sehr schnell fest, dass der Force-Directed Layout-Algorithmus sehr verschwenderisch mit dem Platz umgeht. So werden große Knotenbündel sehr zentral zu einem „Haufen“ gerendert während außerhalb mit großem Abstand zum Zentrum einzelne Knoten platziert werden. Dieser Umstand scheint leider nicht abhängig von der Größe des Datensatzes zu sein und schränkt die Übersichtlichkeit des Graphen sehr stark ein.

### Art und Anzahl der Importformate:

Ergebnis: SigmaJS kann JSON, GEXF importieren. (✓)

Kommentar: SigmaJS stellt Parser für 2 Formate zur Verfügung. Einer dieser Parser ermöglicht das Importieren von JSON-Dateien während der andere den Import von GEXF-Dateien ermöglicht. Beim Import selbst genügt es den Pfad zu der jeweiligen Datei anzugeben. Man muss die eingelesenen Graphdaten zwar nicht manuell einlesen, man muss den einzelnen Knoten allerdings ihre Koordinaten, Größe und Farbe zuordnen, da ansonsten kein Graph angezeigt wird.

### Einrichtung und Einbindung:

Ergebnis: Sehr schnelle und einfache Einbindung. (1,5h)

Kommentar: Die Einbindung von SigmaJS ist sehr einfach. Auf der Projektseite finden sich eine Beispiel-Implementierung und einige Einführungen, mit deren Hilfe man sehr schnell einen ersten Graphen angezeigt bekommt. Hierfür muss auch nicht viel Code geschrieben werden, denn es wird zunächst der Parser (JSON oder GEXF) aufgerufen und diesem eine entsprechende Datei zugeschoben woraus ein Sigma-Graph entsteht. Der Graph wird allerdings so angezeigt wie er Initial gerendert wird. Für das Initiale Rendern muss man bestimmte Attribute manuell setzen (wie z.B. Position, Größe und Farbe der einzelnen Knoten). Setzt man diese Attribute nicht, wird leider statt zufälliger oder standardisierter Attribute einfach nichts angezeigt.

### Render-Performanz:

Ergebnis: Die Performanz ist ganz okay aber spürbar ruckelig. (+)

Kommentar: Mit größer werdenden Graphen nehmen beim Live-Rendering auch die sichtbaren Ruckler zu – allerdings in einem akzeptablen Maß.

### Übersichtlichkeit des Graphen:

Ergebnis: Stark eingeschränkte Übersichtlichkeit, teilweise starke Platzverschwendung.

Kommentar: Die Bibliothek stellt einen (optionalen) Layout-Algorithmus zur Verfügung. Wie in Abbildung 37 zu erkennen ist, macht der Algorithmus einen sehr unreifen Eindruck und hilft leider nur sehr wenig die Übersichtlichkeit des Graphen zu verbessern. Grund dafür ist, dass sich viele Knoten aufgrund ihres „Grades“ zu einem Bündel formen und wieder andere Knoten, welche z.B. keine Kante zu dem Bündel haben sehr weit davon entfernt platziert werden. Hierbei entsteht ein sehr großer Freiraum, welcher leider nicht genutzt wird. Dieser Umstand macht es nahezu unmöglich etwas in größeren Graphen erkennen oder unterscheiden zu können, da die entstehenden Bündel auch durch zoomen nicht übersichtlicher werden.

Der Algorithmus verhält sich leider auch bei anderen Datensätzen nicht besser und ordnet auch hier zu viele Knoten auf zu engem Raum an. Bei sehr großen Graphen verschlimmert sich dieser Umstand noch mehr. Bei solchen Graphen macht es der Algorithmus sehr schwer die Graphen als groß zu erkennen. Die aus großen Datensätzen entstehenden Graphen unterscheiden sich fast überhaupt nicht von sehr kleinen Datensätzen. Dieser Umstand kann in Abbildung 38 betrachtet werden. Verzichtet man hingegen auf die Nutzung des Algorithmus, so muss man die Position jedes Knotens explizit angeben. Dies kann natürlich randomisiert (oder mit einem eigenen Algorithmus) geschehen, bedarf jedoch einer eigenen Implementierung.

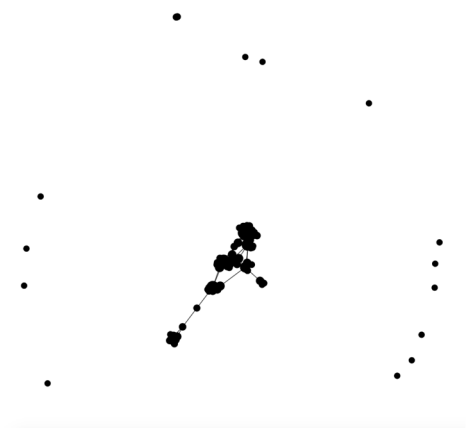


Abbildung 37: Datensatz jFTP mit SigmaJS



Abbildung 39: Datensatz jUnit mit SigmaJS



Abbildung 38: Datensatz Wicket mit SigmaJS

### Interaktion:

**Ergebnis:** Zoomen, Verschieben der Ansicht und auch Knoten sowie einzelne Highlighting-Features sind vorhanden. (+)

**Kommentar:** Abgesehen von „Zoomen“, Verschieben der Sicht auf den Graphen und einzelnen Highlighting-Features gibt es die Möglichkeit, einzelne Knoten des Graphen zu verschieben. Diese Funktion wird durch ein separat einzubindendes Plugin ermöglicht. Hierbei können einzelne Knoten (und ihre anliegenden Kanten) verschoben werden. Bei der Nutzung des Layout-Algorithmus bleiben die Knoten allerdings nicht an der Stelle zu welcher sie gezogen wurden, sondern werden nach dem Loslassen der Maustaste wieder an ihren ursprünglichen Platz zurück geschoben.

### Interaktions-Performanz:

**Ergebnis:** Gut bei kleinen, ruckelig bei größeren Graphen. (+)

**Kommentar:** Bei einem kleinen Graphen ist die Performanz absolut zufriedenstellend. Sowohl Zoom als auch das Verschieben des Graphen sind völlig flüssig möglich. Wird der Graph aber größer so treten durchaus Lags bei der Interaktion mit dem Graphen auf. So ergeben sich bei unserem größten Graphen mit 3679 Elementen Verzögerungen von etwa 300ms - 500ms.

### Hauptspeicherverbrauch:

**Ergebnis:** Etwa **57 MB** pro Graph.

**Kommentar:** Bei dieser Bibliothek werden die nicht mehr benötigten Ressourcen nach einiger Wartezeit wieder freigegeben, sodass sich der Speicherverbrauch bei etwa 57MB einpendelt.

### CPU-Last:

**Ergebnis:** **100%** für Rendern und Interaktionen

### Dokumentation:

**Ergebnis:** Kurze, informative und hilfreiche Dokumentation. (✓)

**Kommentar:** Die Dokumentation für SigmaJS ist online Verfügbar. Die Dokumentation ist nach einer Art Wiki aufgebaut und beinhaltet die wichtigsten Informationen um mit der Bibliothek zu arbeiten. Dieses Wiki stellt unter anderem eine Auflistung und Beschreibung aller Attribute und Einstellungen zur Verfügung. Des Weiteren werden in diesem Wiki auch kurz die wichtigsten Informationen für die erste Nutzung und das Rendering der Bibliothek zusammengefasst.

### **Dokumentationsverfügbarkeit und -qualität:**

Ergebnis: Sehr hohe Verfügbarkeit (Online-Dokumentation), qualitativ sehr gute Dokumentation. (++, Details siehe Tabelle 2)

Kommentar: Die Online-Dokumentation von SigmaJS hat eine sehr hohe Verfügbarkeit und enthält alle nötigen Informationen um Daten einzulesen und Graphen anzuzeigen. Mit Hilfe der Online-Dokumentation ist man sehr schnell in der Lage mit der Bibliothek zu arbeiten und erste Ergebnisse zu erzielen, da die Dokumentation viele Erklärungen und Beispiele enthält.

### **Code Dokumentation:**

Ergebnis: Keinerlei Code-Dokumentation. (X)

Kommentar: Der Code der Bibliothek ist unglücklicherweise nicht dokumentiert und ist keinerlei Hilfe beim Verständnis der einzelnen Funktionen.

### **Live-Rendering:**

Ergebnis: SigmaJS bietet die Möglichkeit des Live-Renderings. (–, Details siehe Tabelle 2)

Kommentar: Das Live-Rendering beschränkt sich ohne eigene Implementierungen für Algorithmen oder Interaktionen auf die Anwendung des einzigen Algorithmus (ForceAtlas2) der Bibliothek. Die Knoten werden initial gesetzt und anschließend mithilfe des Algorithmus neu positioniert. Diesem (recht schnellen) Prozess kann man live zu sehen. Allerdings fällt auf, dass das Live-Rendering, wie auch die Interaktions-Performanz, bei größeren Graphen anfängt zu ruckeln.

### **Visuelle Besonderheiten:**

Ergebnis: Einige verschiedene Darstellungsmöglichkeiten für Knoten (und Kanten).

Kommentar: SigmaJS ermöglicht das Darstellen von gerichteten Kanten. Auch einige unterschiedliche Knotendarstellungen werden ermöglicht. Zu diesen Darstellungen gehören kreis-, diamant-, quadrat-, polygon-, stern-, und kreuzförmige Knoten. Zudem gibt es die Möglichkeit die Knoten wie „Pacman“ aussehen zu lassen. Auch bei der Kantendarstellung bietet SigmaJS ein paar individuelle Möglichkeiten an. So kann man zum Beispiel sowohl ungerichtete wie auch gerichtete Kanten anzeigen lassen. Zusätzlich hierzu steht auch eine gestrichelte oder gepunktete Kantenlinie zur Verfügung. SigmaJS bietet außerdem an, dass eine Kante in Form von zwei parallelen Linien angezeigt wird.

### **Graph-Eigenschaften:**

Ergebnis: Kein Umgang mit bestimmten Graph-Eigenschaften vorhanden.

Kommentar: Die Bibliothek stellt nativ keine besonderen Graph-Eigenschaften oder den Umgang mit solchen zur Verfügung.

### **Layout-Algorithmen:**

Ergebnis: Es wird nur der Algorithmus „ForceAtlas2“ angeboten.

Kommentar: Der einzige Algorithmus, welchen die Bibliothek zur Verfügung stellt ist der kräftebasierte ForceAtlas2 Algorithmus. Die Verwendung dessen muss aber explizit angegeben werden.

**Stabilität:**

Ergebnis: SigmaJS bietet eine gute Stabilität. (++, Details siehe Tabelle 2)

Kommentar: SigmaJS ist relativ stabil und wirft keine Fehlermeldungen. Auch konnte kein Absturz von Firefox (Version 34.05 - Portable) provoziert werden.

**Anmerkungen und Sonstiges:**

SigmaJS stellt die folgenden Formen als Knoten zur Verfügung. Dabei ist zu beachten, dass das „Pacman“-Symbol hier nur aus Gründen der besseren Sichtbarkeit auf einem grünen Hintergrund dargestellt wird.



Abbildung 40: Mögliche Knotenformen von SigmaJS

# yFiles HTML

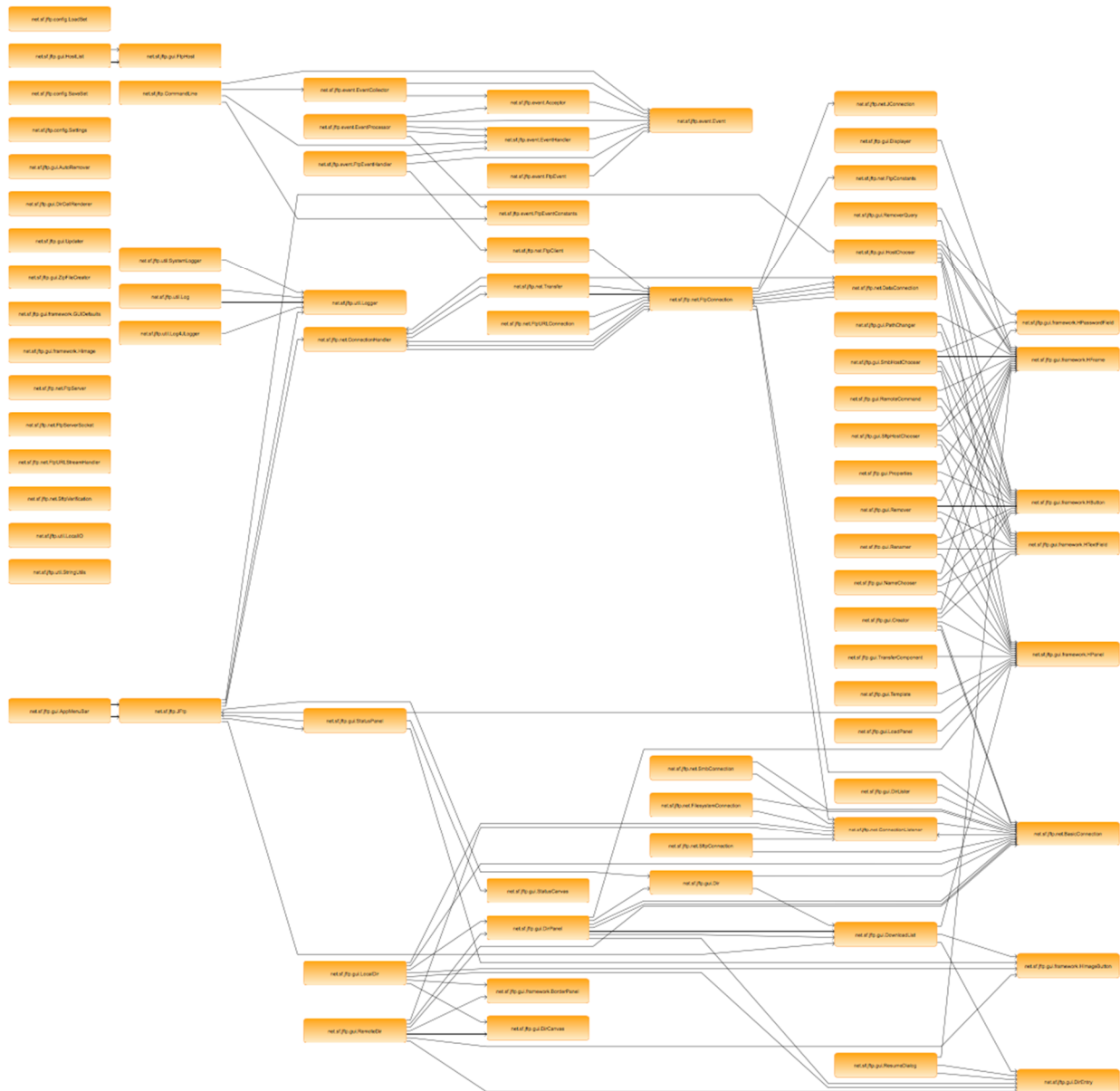


Abbildung 41: Datensatz jFTP mit yFiles HTML

**Getestet mit:** Firefox Portable – Version 34.05

## Kurzbeschreibung:

yFiles HTML ist eine kommerzielle JavaScript Bibliothek, für welche eine Lizenz zum Testen erworben werden konnte. Die Bibliothek wird mit sehr vielen Beispiel-Implementierungen und einer Dokumentation geliefert. Doch trotz der vielen Beispiele und der mitgelieferten Dokumentation war eine Einarbeitung in die Bibliothek sehr mühsam und nahezu unmöglich. Grund dafür war, dass die Beispiel-Implementierungen sehr stark abhängig von der umgebenden Ordnerstruktur sind und diese auch sehr viele komplexe (eigene) Abhängigkeiten mit sich bringen. Dies macht es unmöglich eigene Daten in die Graphen zu importieren oder gar die Bibliothek selbst zu implementieren.

Hierbei hilft leider auch die Dokumentation nicht, da diese zwar alle einzelnen Eigenschaften beschreibt, aber nicht wie diese zusammen verwendet werden können und müssen. Aus diesem Grund war es hier mit viel Aufwand lediglich möglich einen Graphen mit hierarchischem Layout zu konstruieren. Dieser Graph selbst macht einen sehr schönen und übersichtlichen Eindruck und der Layout-Algorithmus scheint auch mit größeren Datensätzen keine Probleme zu haben. Aufgrund der Tatsache, dass nur ein hierarchisches Layout erzielt werden konnte, war eine genauere Evaluation der Bibliothek nicht möglich.

### **Wichtig:**

Da mit dieser Bibliothek nur ein hierarchisches Layout erzielt werden konnte, beziehen sich die Evaluation des erstellten Graphen und die Bewertung der einzelnen Eigenschaften auch nur auf dieses Layout, wodurch auch ein paar Bewertungspunkte wegfallen.

### **Art und Anzahl der Importformate:**

Ergebnis: yFiles HTML kann nur JSON importieren (manuelle Implementierung des Einlesens).

Kommentar: Die Bibliothek verfügt über keinerlei Import-Funktionalität. Sie ist allerdings in der Lage einen Graphen aus einem JSON-Objekt zu erstellen. Hierbei muss der Import der JSON-Daten allerdings manuell erstellt werden. Positiv aufgefallen ist hierbei, dass die Graphdaten in diesem JSON-Objekt keine bestimmten Vorgaben bezüglich der Bezeichner erfüllen müssen. Es ist möglich der Bibliothek mitzuteilen, wie die entsprechenden Bezeichner heißen bzw. wo sie in dem Objekt zu finden sind.

### **Einrichtung und Einbindung:**

Ergebnis: Sehr komplizierte und langwierige Implementierung.

Kommentar: Das Einrichten ist sehr kompliziert und dauert sehr lange. Leider war es in der vorgegeben Zeit nicht möglich ein force-directed Layout mit dieser Bibliothek und den Testdaten zu erstellen. Daher ist auch ein direkter Vergleich mit den anderen JavaScript Bibliotheken nicht möglich. Für das Verständnis der Bibliothek existieren viele Beschreibungen aller einzelnen Komponenten aber keine über die Abhängigkeiten oder Zusammenarbeit zwischen diesen. Daher wurde versucht eigene Testdaten in eines der bereits existierenden Beispiele einzubringen. Dies gelang allerdings nur mit dem Beispielprojekt für das hierarchische Layout und nicht mit den Beispielprojekten der anderen Layouts. Die Beispiele verwenden bestimmte Abhängigkeiten (z.B. require.js) und relative Pfade, welche es nur mit sehr hohem Aufwand möglich machen eigene Dateien einzulesen. Auch der Bedarf der Validierung der Lizenz ist sehr hinderlich bei der Einbindung in eigene Applikationen, da die Validierung selbst auch von bestimmten Strukturen im Projekt abhängt, welche in eigenen Applikationen nicht existieren.

### **Übersichtlichkeit des Graphen:**

Ergebnis: Der Graph ist sehr übersichtlich, es gibt kaum Visual Clutter.

Kommentar: Das hierarchische Layout ist sehr übersichtlich und leidet kaum unter Kantenüberschneidungen. Die Knoten und ihre Kanten werden wohl-platziert angezeigt und das Layout zeigt hierarchische Strukturen sehr gut und zuverlässig an.

### **Interaktion:**

Ergebnis: Es besteht die Möglichkeit des Zoomens und des Verschiebens der Ansicht.

Kommentar: Das Beispiel ermöglichte das Zoomen im Graph sowie das verschieben dessen Ansicht. Es ist auch möglich, manuell neue Kanten im Graphen hinzuzufügen. Leider ist es unklar, ob diese Interaktionen in jedem Graphen zur Verfügung stehen oder ob diese explizit implementiert worden sind.

### **Dokumentation:**

Ergebnis: Ausführliche Dokumentation einzelner Elemente, nicht ausreichend für die Einarbeitung in die Erstellung erster Graphen.

Kommentar: Im Lieferumfang der Bibliothek befindet sich viel ausführliche Dokumentation über einzelne Elemente der yFiles Bibliothek. Doch mithilfe dieser Dokumentation ist es fast unmöglich die Bibliothek in eine eigene Applikation zu integrieren oder den Graph mit eigenen Testdaten zu füllen. Grund dafür ist, dass die Abhängigkeiten der einzelnen Elemente recht schlecht erklärt sind und mühsam zusammen gesucht werden müssen. Es gibt leider keine Einführung in einen ersten „simplen“ Graphen, welchen man dann zum Beispiel nach und nach erweitern könnte. Aus diesem Grund ist die Dokumentation trotz ihrer Ausführlichkeit kaum zu gebrauchen um sich mit der Bibliothek vertraut zu machen.

### **Dokumentationsverfügbarkeit und -qualität:**

Ergebnis: Die Verfügbarkeit ist sehr hoch, die Qualität jedoch unzureichend.

Kommentar: Die Dokumentation von yFiles befindet sich im Lieferumfang der Bibliothek und hat daher eine sehr hohe Verfügbarkeit, da diese nicht von einer Internetverbindung oder ähnlichem abhängt. Sie enthält zwar sehr ausführliche Informationen über einzelne Elemente der Bibliothek, erklärt jedoch nicht deren Zusammenhang und Abhängigkeiten wodurch sie für eine erste Implementierung eines Graphen nicht ausreicht.

### **Code Dokumentation:**

Ergebnis: Es befinden sich nahezu keine Kommentare im Code.

Kommentar: Der Code der Bibliothek selbst ist leider so gut wie gar nicht kommentiert. Grund dafür ist unter anderem, dass viele der Script-Dateien „minimiert“ vorliegen, wodurch überflüssige (Leer-)Zeichen jeglicher Art entfernt werden. Daher lassen sich auch aus dem Code keine Rückschlüsse auf die Funktionalität der Bibliothek ziehen.

### **Visuelle Besonderheiten:**

Ergebnis: Eigene Templates für Knoten können erstellt werden.

Kommentar: Die Bibliothek stellt einige Templates für Knoten zur Verfügung. Leider ist unklar, ob diese Templates automatisch in Abhängigkeit des Layouts ausgewählt werden oder ob der Entwickler hier selbst Einfluss ausüben kann.

### **Graph-Eigenschaften:**

Ergebnis: Das hierarchische Layout achtet darauf, den Graphen so planar wie möglich zu Erstellen.

Kommentar: Die Bibliothek rendert mit unseren Testdaten einen sehr übersichtlichen und planaren Graphen (sofern die Daten einen solchen Graphen zulassen). Selbst wenn ein rein-planarer Graph auf Grund der Testdaten nicht möglich ist, werden die Kantenüberschneidungen auf einem Minimum gehalten.

### **Layout-Algorithmen:**

Ergebnis: Familienbaum-, hierarchisches-, organic-, orthogonales-, partielles-, planares-, radiales-, Baumlayout

Kommentar: Die Bibliothek unterstützt viele verschiedene Layouts. Darunter befindet sich zum Beispiel ein (Familien-)Baumlayout oder auch eine hierarchische Darstellung der Daten. Auch ein radiales oder planares Layout ist möglich. Neben diesen Varianten kann man sich zudem für ein orthogonales, partielles oder Organiclayout entscheiden. Leider ist in der vorgegebenen Zeit nur die Verwendung eines hierarchischen Layouts möglich gewesen.



## Anmerkungen und Sonstiges:

`net.sf.jftp.config.LoadSet`

Abbildung 42: Default-Darstellung eines Knotens in den Beispielen von yFiles HTML & yFiles WPF

### b) Bewertung

Der Einstieg in die JavaScript Bibliotheken war durchschnittlich leicht und man benötigte bei den meisten nur sehr wenig Zeit bis man den ersten Graphen anzeigen konnte. Grund dafür ist, dass sie alle mit dem bekannten JSON-Format umgehen, Graph-Daten hieraus auslesen und verwenden können. Allerdings gibt es beim Einlesen der Daten Unterschiede. So muss man zum Beispiel bei CytoScapeJS und yFiles das Einlesen der Datei selbst implementieren, da hierfür keine native Funktion existiert. Während CytoScapeJS und yFiles die Graph-Daten lediglich aus einem JSON-Objekt akquirieren können, ist es bei SigmaJS und D3JS möglich, diese aus anderen Formaten zu importieren. SigmaJS ermöglicht zum Beispiel auch das Importieren aus dem GEXF-Format. D3JS hingegen besitzt noch ein etwas breiteres Spektrum und ermöglicht das Einlesen von Daten aus sechs verschiedenen Formaten (JSON, XML, CSV, HTML, TSV, TXT). Abgesehen von yFiles waren die Einrichtung der Bibliotheken sowie das Anzeigen eines ersten Graphen sehr schnell und einfach. Hierbei ist die Online-Dokumentation der Bibliotheken D3JS, CytoScapeJS und SigmaJS sehr hilfreich und kann jederzeit und für nahezu jedes Problem konsultiert werden. Die Dokumentation dieser drei Bibliotheken geben für fast alle Einstellungsmöglichkeiten eine Erklärung und zum Teil auch ein Codebeispiel an. Hierdurch wird die Individualisierung des Graphen sehr stark erleichtert sowie die Produktivität gefördert. Auch wenn die Quelltexte aller getesteten Bibliotheken nicht ausreichend kommentiert sind, kann man sich mithilfe der Online-Dokumentationen sehr gut in diese einarbeiten. Im Falle von yFiles muss man leider sagen, dass die Dokumentation, welche mit der Bibliothek selbst mitgeliefert wird, zwar ausführlich die einzelnen Elemente beschreibt, jedoch deren Abhängigkeiten nicht hinreichend erklärt oder nennt. Dies macht die Einarbeitung sehr schwer und zeitaufwendig und man verliert recht schnell die Lust daran mit dieser Bibliothek zu arbeiten. Zusätzlich zu der mitgelieferten Dokumentation gibt es auch einige ausführbare Graph-Implementierungen.



Abbildung 43: Datensatz jFTP in D3JS, CytoScapeJS und SigmaJS

Leider helfen auch diese nicht beim Verständnis und diese sind aufgrund sehr vieler Abhängigkeiten nur schwer zu erweitern oder gar mit eigenen Daten zu speisen. Aufgrund der mangelnden Dokumentation war es auch nicht möglich einen Graphen aus den eigenen Daten mit einem Layout zu erzeugen, welchen man mit den anderen Bibliotheken vergleichen hätte können. Aus diesem Grund bleibt eine Bewertung dieser Bibliothek im Folgenden leider aus.

Die drei Bibliotheken D3JS, CytoScapeJS und SigmaJS stellen jeweils mindestens einen force-directed Layout-Algorithmus zur Verfügung. In den folgenden Bildern wurde derselbe Datensatz von jeder der drei Bibliotheken mit Hilfe des force-directed Layouts visualisiert.

Beim Testen dieser Algorithmen mit verschiedenen (größeren) Datensätzen wurde hierbei festgestellt, dass der Algorithmus von SigmaJS mit zunehmender Größe des Graphen sehr schnell Probleme bekommt und der Graph sehr unübersichtlich wird. Dabei fällt sehr stark ins Auge, dass der Algorithmus sehr verschwenderisch mit dem Platz umzugehen scheint, denn es entstehen sehr viele und sehr große Freiräume, in welchen sich weder Knoten noch Kanten befinden. Das Anpassen der Größe einzelner Elemente schafft hier leider keine Abhilfe und man hat nahezu keinerlei Einfluss auf den Algorithmus und kann diesen somit auch nicht optimieren um den Platz besser auszunutzen. CytoScapeJS hingegen nutzt den Platz bei kleinen Datensätzen zwar etwas besser aus, hat jedoch ebenfalls mit großen Graphen sehr starke Probleme.

Am übersichtlichsten gestaltet jedoch der Layout Algorithmus von D3JS den Graphen. Selbst bei sehr großen Graphen bleibt die Übersichtlichkeit durchschnittlich gut und man kann diese natürlich zusätzlich noch verbessern indem man bestimmte Einstellungen anpasst.

Leider orientiert sich der Algorithmus nicht an der ihm zur Verfügung stehenden Größe des Fensters und so passiert es recht häufig, dass einige Knoten und Kanten aus dem Fenster verschwinden, was in der Abbildung der Visualisierung des Wicket-Graphen zu sehen ist.

Da D3JS in den Standardeinstellungen keine Möglichkeit des Scrollings oder anderen Interaktionen besitzt, ist es leider auch nicht möglich diese Knoten wieder in das Sichtfeld zu bringen. Abgesehen von diesem Problem geht D3JS sehr viel flüssiger und besser mit größeren Graphen um als die anderen Bibliotheken dies tun. Dennoch kommt auch hier recht schnell Unübersichtlichkeit zustande, welche sich durch das Ändern der Initialwerte mindestens genauso schnell wieder reduzieren lässt.

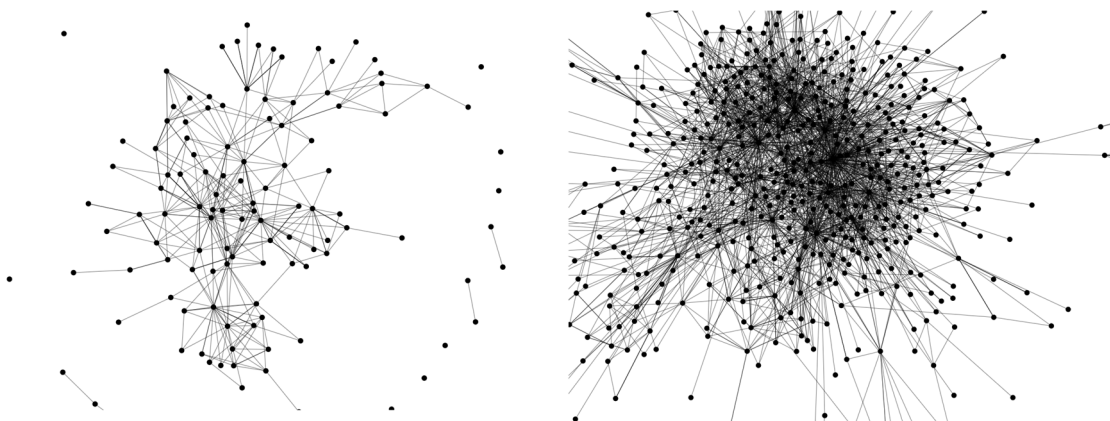


Abbildung 44: Datensatz jUnit und Wicket in D3JS

Auch wenn sich D3JS in Sachen Performanz am besten schlägt, muss man bei den nativen Interaktionsmöglichkeiten kleine Abstriche machen. Denn die einzige Interaktion, welche D3JS nativ unterstützt ist Drag&Drop von Knoten. Hierbei werden die Knotenpositionen jedoch nicht beibehalten, sondern die Knoten kehren an ihren ursprünglichen Platz zurück. Selbiges gilt auch für SigmaJS: Auch hier werden die Knotenpositionen nicht beibehalten. Die einzige Bibliothek, welche das persistente Verschieben von Knoten erlaubt ist CytoScapeJS. Hier kann man die Knoten beliebig verschieben und platzieren und diese Änderung bleibt erhalten. Zusätzlich zum Drag&Drop wird von SigmaJS und CytoScapeJS auch eine Zoomfunktion angeboten. Diese machen das Navigieren im Graphen zum Teil etwas leichter.

Im Großen und Ganzen ist D3JS die rentabelste Bibliothek der Kategorie JavaScript. Trotz der Tatsache, dass manche Knoten außerhalb des Fensters platziert werden könnten schneidet sie in der Summe am besten ab. Grund hierfür ist der sehr leichte Einstieg, die große Importierungsvielfalt und die hohe Performanz. Auch der Layout Algorithmus von D3JS macht den ausgereiftesten Eindruck. Die Bibliothek ist sehr einfach zu verwenden und bietet viel Funktionalität für wenig Aufwand.

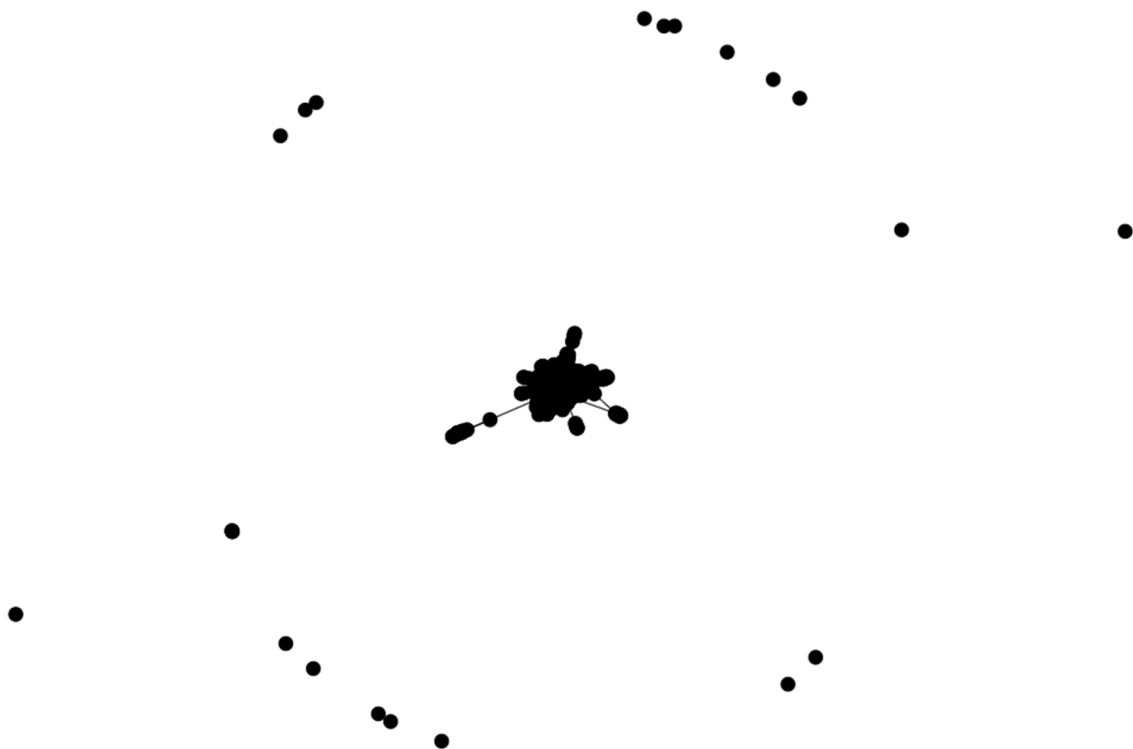


Abbildung 45: Datensatz Wicket mit SigmaJS

## 6. Bewertung der C#-Bibliotheken

### a) Einzelbewertungen

#### GraphX

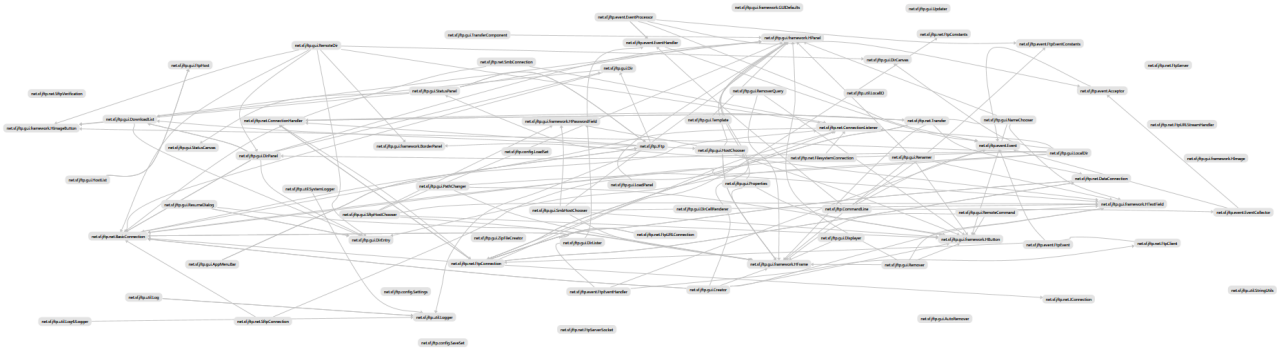


Abbildung 46 : Datensatz jFTP mit GraphX



#### Kurzbeschreibung:

GraphX ist eine freie Bibliothek die von einem russischen Entwickler, der sich „Panther“ nennt, entwickelt. Dabei setzt GraphX aber vollständig auf QuickGraph auf und erweitert diese Bibliothek um Funktionalität. Es wird aber auch Code aus den Bibliotheken Graph# und NodeXL genutzt. Der Entwickler von GraphX hat auch eine sehr brauchbare Anleitung mit einigen Code-Beispielen geschrieben, die zeigt, wie die Bibliothek eingebunden wird. Auch zu ein paar erweiterten Themen gibt es Anleitungen, die aber nicht mehr ganz so ausführlich sind.

Obwohl man erkennen kann, dass sich der Entwickler viel Mühe gegeben hat hier eine vernünftige Bibliothek für die Graph-Visualisierung zu entwickeln fallen die Testresultate für GraphX leider nicht gut aus. Die Bibliothek hinkt ihren Konkurrenten leider sowohl in Sachen Performanz als auch in Sachen Layout hinterher. Doch sie stellt aktuell noch die vermutlich beste freie Variante dar. mal abgesehen von NodeXL, das leider nicht getestet werden konnte (vgl. Abschnitt „Gefundene Bibliotheken und Begründung für die Auswahl“).

#### Art und Anzahl der Importformate:

Ergebnis: Die Bibliothek bietet keine Möglichkeit des Importes von Dateien. (✗)

Kommentar: Es sind keine fertigen Funktionen zum Laden von Standard-Formaten integriert. Allerdings ist die notwendige Logik vorhanden um eigene Importfunktionen in GraphX zu integrieren.

#### Einrichtung und Einbindung:

Ergebnis: Für die initiale Einbindung mussten etwa **2,5 Stunden** aufgebracht werden.

Kommentar: Da die Dokumentation einen beim Einbinden unterstützt gelingt es relativ schnell einen Graphen anzuzeigen. Bei der Nutzung von GraphX mit WPF ist allerdings zu beachten, dass GraphX nicht WPF konform ist und somit der Graph nicht mittels DataBinding gesetzt werden kann.

## Render-Performanz:

Ergebnis: Um einen kleinen Graphen (78 Knoten, 144 Kanten) zu laden und zu rendern benötigt die Bibliothek etwa eine 700ms, für einen größeren Graphen (622 Knoten, 3057 Kanten) hingegen 46 Sekunden. (--)

Kommentar: Kleine Graphen werden in einer durchaus guten Zeit dargestellt, wohingegen für größere Graphen die Performanz massiv einbricht. Dabei scheint primär die Anzahl der Knoten, die angezeigt werden müssen, entscheidend zu sein wobei auch eine deutliche höhere Zahl an Kanten zu einem spürbaren Performanzverlust führt.

## Übersichtlichkeit des Graphen:

Ergebnis: Es werden immer alle Elemente angezeigt. Somit ist die Übersichtlichkeit bei großen Graphen nicht wirklich gegeben.

Kommentar: Auch wenn verschiedene Layout-Algorithmen mit verschiedensten Einstellungen zur Verfügung stehen, scheinen alle ab einer gewissen Größe des Graphen überfordert zu sein. Während jUnit (Abbildung 48) noch relativ gut erkennbar ist hat man mit Wicket (Abbildung 49) definitiv ein Problem, das bei höherer Zoomstufe noch deutlicher wird, wie Abbildung 51 zeigt. Es besteht zwar die Möglichkeit eigene Algorithmen einzubinden. Allerdings ist hierbei nicht vorgesehen irgendwelche Elemente auszublenden. Somit dürften große Graphen hier immer ein Problem darstellen.

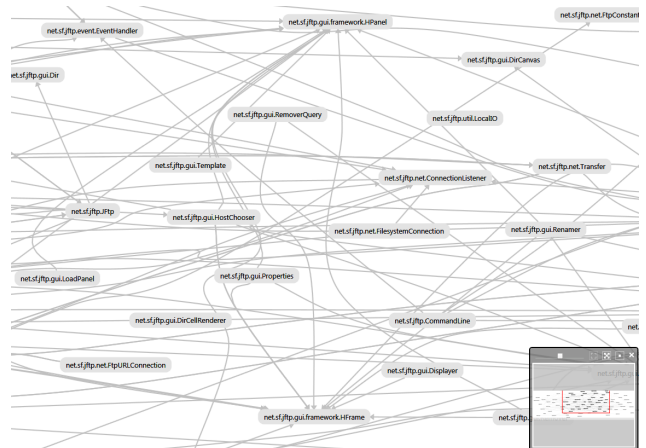


Abbildung 47: Datensatz jFTP mit GraphX

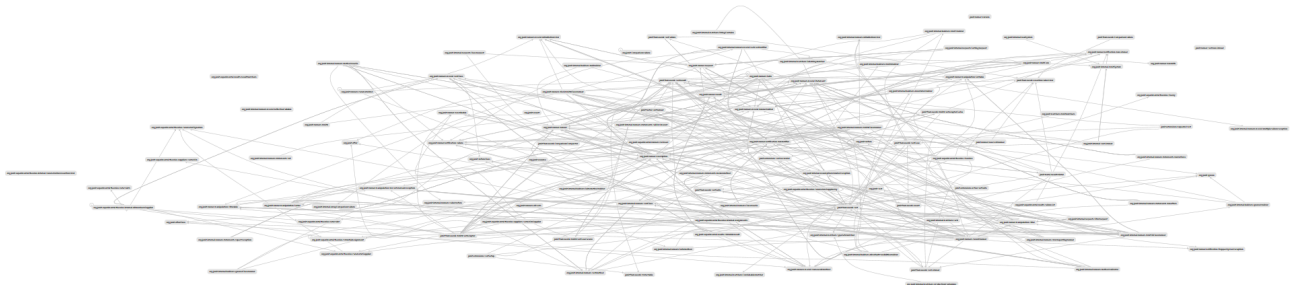


Abbildung 48: Datensatz jUnit mit GraphX

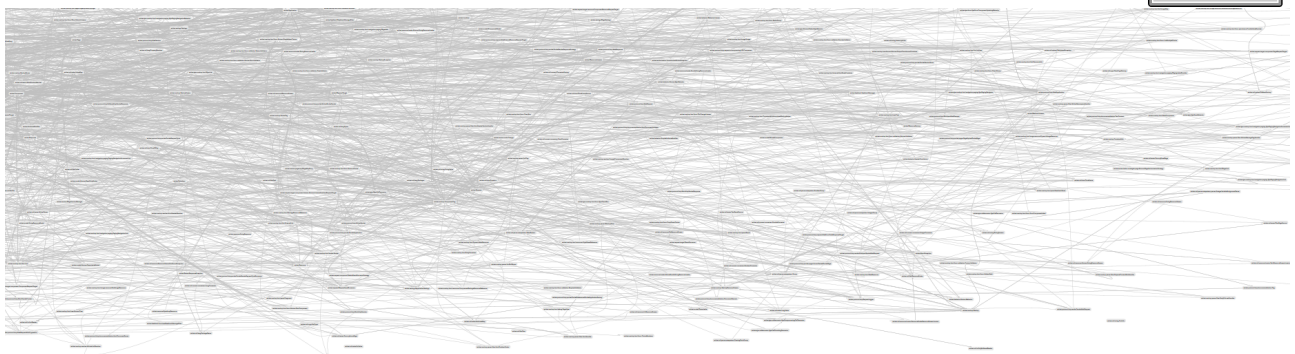


Abbildung 49: Datensatz Wicket mit GraphX

### Interaktion:

Ergebnis: Unterstützt werden der Zoom und das Verschieben der Ansicht. (+)

Kommentar: Der Graph wird, sofern man die entsprechende Funktion aufruft, immer zentral im Zeichnungsbereich angezeigt, wobei keine Elemente außerhalb des sichtbaren Bereichs angezeigt werden. Außerdem lassen sich mit einer kleinen ToolBox mit integrierter Minimap (vgl. Abbildung 50) die Zoomstufe und der Anzeigebereich verändern. Diese Interaktionen können aber auch mittels Mausrad bzw. Drag des Graphen durchgeführt werden.

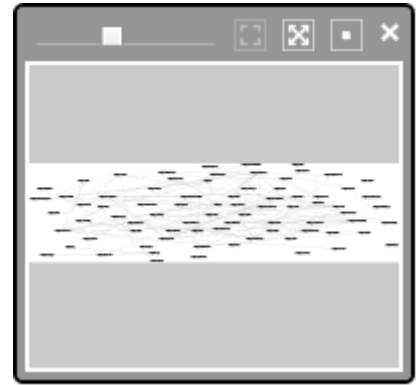


Abbildung 50: Datensatz jFTP in der Minimap der ToolBox

### Interaktions-Performanz:

Ergebnis: Die Interaktions-Performanz ist mangelhaft. (--)

Kommentar: Bei kleinen Graphen werden keine bzw. wenn, dann kaum Lags wahrgenommen, wohingegen bei größeren Graphen Wartezeiten von mehreren Sekunden entstehen. Dabei passiert eben nach der Interaktion eine gute Zeit lang nichts und irgendwann wird der Graph dann neu gezeichnet. In GraphX gibt es keine direkten Einstellungsmöglichkeiten um mehr Performanz zu erhalten. Allerdings geht der Entwickler in seiner Dokumentation auf das Problem ein und schildert, wie man noch etwas Performanz herausholen kann, indem unter anderem die Anzeigequalität vermindert wird.

### Hauptspeicherverbrauch:

Ergebnis: Etwa **130 MB** für Graphen mit den meisten Elementen.

Kommentar: Beim Test wurde der Graph mehrfach nacheinander gerendert, um an Durchschnittswerte zu gelangen. Dabei ist aufgefallen, dass die Anwendung immer mehr RAM verbraucht hat. Hier scheint es ein Speicherleck zu geben.

### CPU-Last:

Ergebnis: Die CPU wird teilweise **vollständig (100%)** ausgelastet. Die GPU wird nur von der WPF selbst genutzt.

### Dokumentation:

Ergebnis: Die Dokumentation ist auf der Website des Entwicklers zu finden und vollständig in Englisch verfasst. Dazu sind viele Code-Beispiele integriert. Daneben steht außerdem ein Forum für weitergehende Fragen zur Verfügung. (✓)

Kommentar: Die Dokumentation wurde in gut verständlichen Englisch verfasst und sinnvoll mit Codebeispielen versehen. Außerdem wurde die Dokumentation in verschiedene Bereiche untergliedert, sodass man schneller die Erklärung findet, die man benötigt.

### Dokumentationsverfügbarkeit und -qualität:

Ergebnis: Die Dokumentation ist auf der Website des Entwicklers zu finden und ist aktuell. (+ +, Details siehe Tabelle 2)

Kommentar: Dadurch, dass die Dokumentation als Website vorliegt ist sie von überall erreichbar. Außerdem kann sie so sehr einfach aktuell gehalten werden.

### Code Dokumentation:

Ergebnis: Keine Codekommentare vorhanden. (+ +, Details siehe Tabelle 2)

### Live-Rendering:

Ergebnis: Die Bibliothek unterstützt kein Live-Rendering. (✘)

Kommentar: Nachdem der Graph verändert wurde muss dieser neu gerendert werden. Dabei wird erst das Layout berechnet und dann erst der fertige Graph angezeigt.

### Visuelle Besonderheiten:

Ergebnis: Highlighting und Anpassung der Knoten ist möglich.

Kommentar: Das Highlighting von Knoten und Kanten beim MouseOver ist möglich. Ebenso kann das Aussehen der Knoten angepasst werden. Im Standard-Layout passen sich die Knoten und Kanten aber nicht an.

### Graph-Eigenschaften:

Ergebnis: Gerichtete Kanten werden erkennbar als solche dargestellt. Schlingen werden ebenfalls dargestellt. Die Kantengewichte sind aber nicht erkennbar.

Kommentar: Bei den „gerichteten“ Schlingen wird der Richtungspfeil allerdings nicht korrekt auf der Kante gerendert. Ansonsten ist der Informationsgehalt des Graphen recht gering, da er nur die Knoten mit ihrem Bezeichner und die gerichteten Kanten anzeigt. Es wäre aber auch möglich mehr Informationen zu visualisieren, indem man die Knoten noch einfärbt oder andere Anpassungen vornimmt.

### Layout-Algorithmen:

Ergebnis: Fruchtermann-Reingold, Kamada-Kawai, ISOM, LinLog, Simple Tree, Simple Circle, Sugiyama und Compound Graph Layout.

Kommentar: Auch wenn einige grundlegende Layout-Algorithmen implementiert sind ist deren Nutzung doch Aufwändig. Möchte man z.B. das Layout wechseln, so muss hierfür der komplette Code angepasst und die layoutspezifischen Eigenschaften angegeben werden. Zudem fällt auf, dass die Layout-Algorithmen weder sonderlich performant sind noch sonderlich intelligent arbeiten. Meist sind die Ergebnisse nicht wirklich befriedigend, wenn man nicht an den Einstellungen herum spielt um die Einstellungen zu finden, die den gewünschten Graphen einigermaßen vernünftig darstellen.

### Stabilität:

Ergebnis: Die Bibliothek läuft relativ stabil, nutzt aber viel CPU-Leistung für das Rendering. (+ +, Details siehe Tabelle 2)

Kommentar: Mehrfach gleiche Kanten werden problemfrei gerendert. Auch sind keine Abstürze vorgekommen. Das Rendern eines großen Graphen dauert zwar lange und benötigt viel CPU-Leistung aber auch hier konnte kein Absturz verursacht werden.



Abbildung 51: Datensatz Wicket in GraphX (mit Zoom)

# yFiles WPF

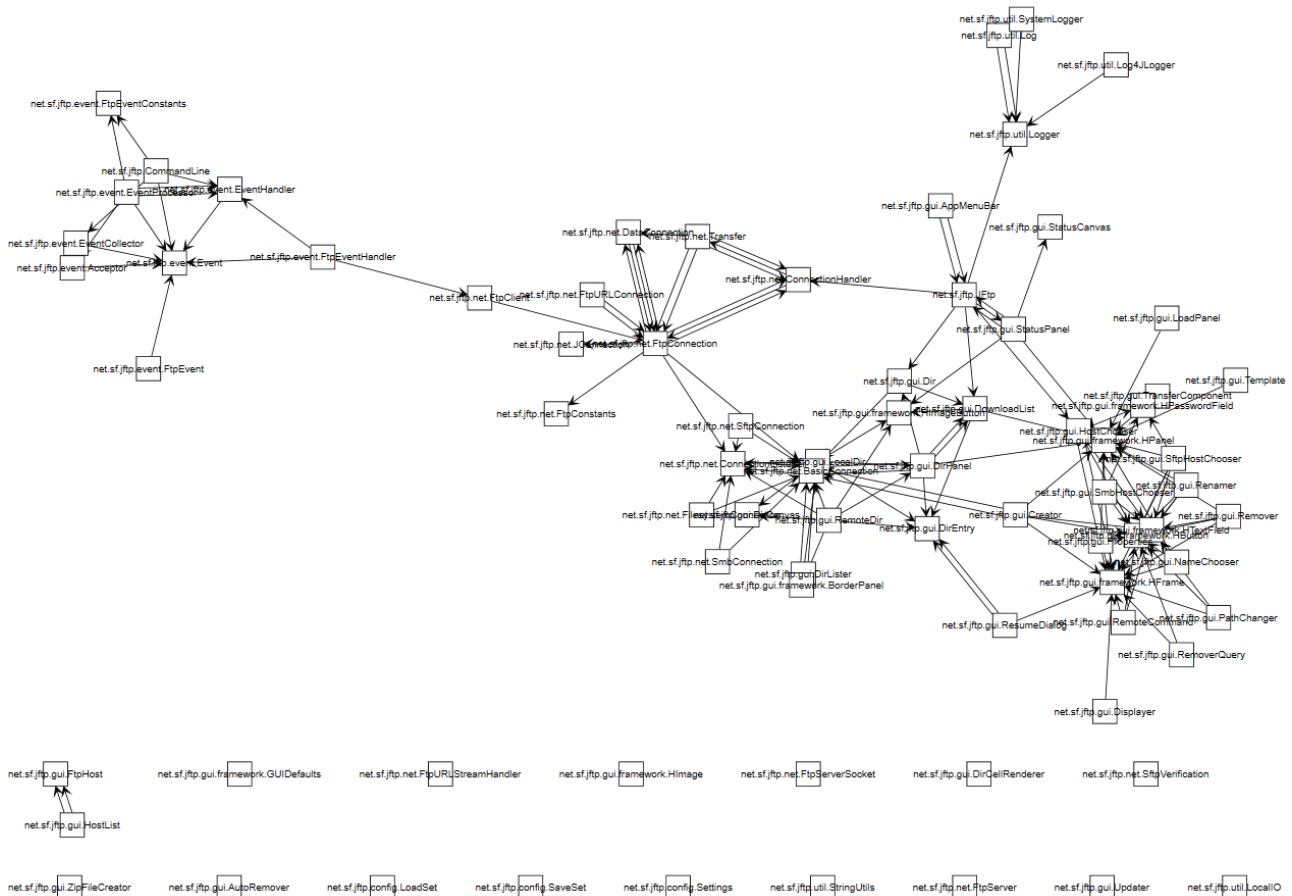


Abbildung 52: Datensatz jFTP in yFiles WPF

## Kurzbeschreibung:

yFiles WPF ist eine Variante der Graphvisualisierungsbibliothek von yWorks. Neben yFiles WPF wird auch yFiles.NET (für WinForms), yFiles Java und yFiles HTML angeboten. Allerdings ist diese Bibliothek nicht frei verfügbar. Im Rahmen dieser Fachstudie wurde uns aber eine Evaluationskopie zur Verfügung gestellt um auch ein kommerzielles Produkt bewerten zu können und so einen Einblick zu erlauben, wie gut ein kommerzielles Produkt neben den freien Bibliotheken abschneidet.

Dadurch, dass yFiles WPF ein kommerzielles Produkt ist, verwundert es auch nicht, dass eine sehr ausführliche und verständliche Dokumentation vorhanden ist, die verschiedene Anwendungsszenarien abdeckt und erklärt, wie man diese mit der Bibliothek umsetzen kann. Neben dieser Dokumentation werden mit der Bibliothek auch sehr viele Beispielprojekte mit geliefert, die noch einmal zeigen sollen, was in der Dokumentation schriftlich erklärt wird.

Sollten dann mal alle Stricke reißen hat man durch den Kauf der Bibliothek auch in einen guten Support investiert, der von offensichtlich fachkundigen Mitarbeitern geleistet wird. So trat im Test ein seltsamer Fehler auf, wo nicht nachvollziehbar war, wie er zustande kommt. Letztlich konnte das Problem (Bug im VisualTree-Live-Debugger) mit Hilfe des Supportes gelöst werden.

## Art und Anzahl der Importformate:

**Ergebnis:** Es gibt eine Importfunktionalität für das GraphML-Format. (✓)

**Kommentar:** Als einziges Standard-Format wird von yFiles WPF das GraphML-Format unterstützt.



## Einrichtung und Einbindung:

Ergebnis: Die initiale Einrichtung benötigte nur etwa **eine Stunde**.

Kommentar: Aufgrund einer guten Dokumentation und vieler Beispiele war die Einbindung der Bibliothek sehr komfortabel.

## Render-Performanz:

Ergebnis: Um einen kleinen Graphen (78 Knoten, 144 Kanten) zu laden und zu rendern benötigt die Bibliothek etwa 197ms, für einen größeren Graphen (622 Knoten, 3057 Kanten) hingegen ca. 4 Sekunden. (+ +)

Kommentar: Bei dieser Bibliothek fällt zum einen auf, dass die Performanz relativ gut erhalten bleibt, selbst wenn die Graphen größer werden. Außerdem fällt auf, dass die Performanz primär davon abzuhängen scheint, wie viele Knoten im Graph enthalten sind. Nimmt man 2 Graphen mit ähnlich vielen Knoten und Kanten wird der Graph mit weniger Knoten schneller gerendert.

## Übersichtlichkeit des Graphen

Ergebnis: Im einfachsten Modus werden alle Elemente angezeigt. Dies führt bei größeren Graphen zwangsläufig zu Visual Clutter. Allerdings gibt es Möglichkeiten bestimmte Knoten und Kanten auszublenden oder Knoten manuell zu verschieben.

Kommentar: Auch bei Graphen, die sehr dicht sind schaffen es die Layout-Algorithmen ein paar Grundstrukturen herauszuarbeiten. Durch die Möglichkeit den Graph dann manuell zu verändern oder auch nur Teilbereiche mit einem anderen Layout-Algorithmus neu anzuordnen, kann man mit etwas Arbeit auch mehr aus dem Graphen herauslesen. Die Bibliothek bietet dafür relativ viele Möglichkeiten.

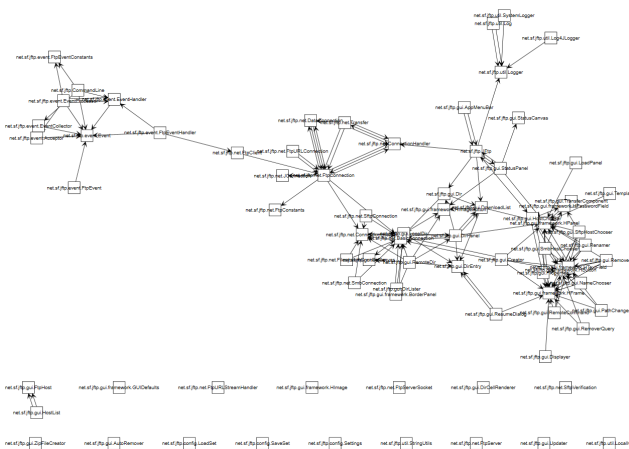


Abbildung 53: Datensatz jFTP in yFiles WPF

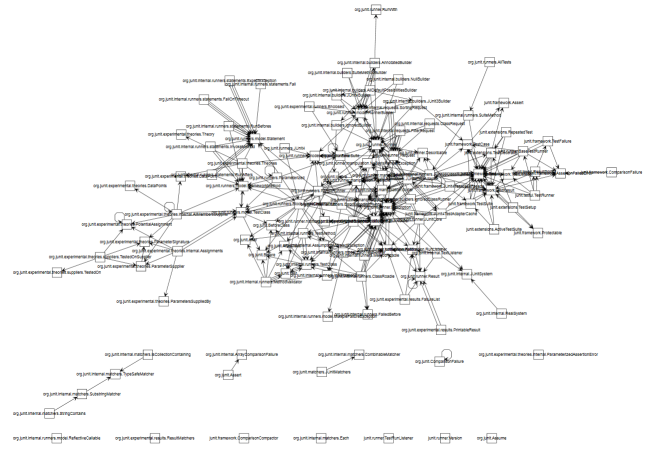


Abbildung 55: Datensatz Wicket in yFiles WPF

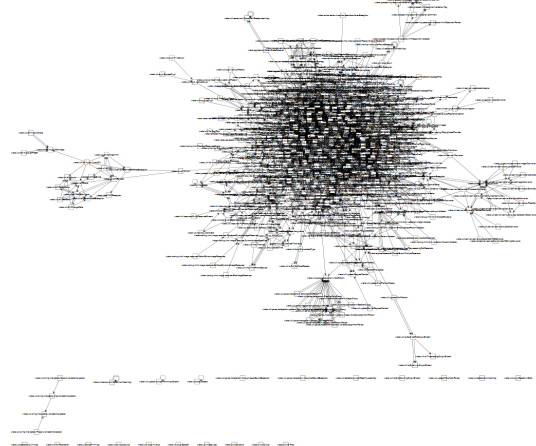


Abbildung 54: Datensatz Wicket in yFiles WPF

### **Interaktion:**

Ergebnis: Unterstützt werden Zoom, Verschieben der Ansicht und Ziehen von Knoten. (++)

Kommentar: Der Graph wird, sofern man die entsprechende Funktion aufruft, immer zentral im Zeichnungsbereich angezeigt, wobei keine Elemente außerhalb des sichtbaren Bereichs angezeigt werden. Zudem lassen sich weitere Interaktionen und zugehörige Schaltflächen mit den beigefügten Beispielen recht einfach integrieren.

### **Interaktions-Performanz:**

Ergebnis: Durchgehend gut, mit kleinen Einschränkungen. (++)

Kommentar: Bei kleinen Graphen sind eigentlich keine Lags vorhanden, wobei bei größeren Graphen ganz leichte Lags auftreten können. Wirklich deutliche Lags konnten nicht hervorgerufen werden, wenn man nicht gerade den kompletten Graphen hin und her zieht.

### **Hauptspeicherverbrauch:**

Ergebnis: Etwa **50 MB** für Graphen mit den meisten Elementen.

Kommentar: Bei dieser Bibliothek fällt auf, dass der Speicherverbrauch relativ stabil bleibt. Sobald man mit dem Graphen interagiert wird zwar auch deutlich mehr Speicher benötigt. Aber dieser wird auch wieder frei, sofern man die Garbage Collection anwirft.

### **CPU-Last:**

Ergebnis: Die CPU wird für das Layouting **vollständig (100%)** ausgelastet. Die GPU wird nur von der WPF selbst genutzt.

### **Dokumentation:**

Ergebnis: Als Dokumentation steht online eine Knowledge Base als auch eine API-Dokumentation zur Verfügung. Zudem wird auch bei der Installation der Bibliothek angeboten die Dokumentationen als auch eine große Zahl an Beispielprojekten lokal zu installieren. (✓)

Kommentar: Die Dokumentationen sind recht ausführlich und verständlich geschrieben und lassen kaum Raum für Fragen offen. Sollten aber doch Fragen unbeantwortet bleiben ist es möglich sich an den Support zu wenden, wo einem gerne weitergeholfen wird.

### **Dokumentationsverfügbarkeit und -qualität:**

Ergebnis: Die Dokumentation ist auf der Website der Entwickler zu finden und kann bei Bedarf aber auch lokal mitinstalliert werden. (++, Details siehe Tabelle 2)

Kommentar: Da die Dokumentation sowohl online als auch offline verfügbar ist, kann eigentlich zu jedem Zeitpunkt auf eine Version der Dokumentation zugegriffen werden.

### **Code Dokumentation:**

Ergebnis: In der Bibliothek selbst sind keine Codekommentare vorhanden. In den Programmierbeispielen wurde dafür sehr ausführlich kommentiert. (++, Details siehe Tabelle 2)

Kommentar: Für den Code werden die XML-Kommentare nicht mit ausgeliefert.

### **Live-Rendering:**

Ergebnis: Eine Art Live-Rendering ist vorhanden. (++, Details siehe Tabelle 2)

Kommentar: Diese Bibliothek bietet die Möglichkeit eine Art Live-Rendering durchzuführen. Dabei kann der Funktion, die den Layout-Algorithmus startet eine „morph duration“ mitgegeben werden. Dies sollte aber nicht mit allzu großen Graphen gemacht werden, da hier die Animation nicht mehr wirklich flüssig läuft.

### **Visuelle Besonderheiten:**

Ergebnis: Highlighting und Anpassung der Knoten und Kanten ist möglich.

Kommentar: Man kann nicht nur ein einfaches Highlighting einbauen sondern auch das Aussehen von Knoten und Kanten nahezu beliebig verändern. So kann man z.B. auch unterschiedliche Knotensymbole für verschiedene Knoten anzeigen lassen und so ein Computernetzwerk mit Servern und Routern graphisch anzeigen lassen. Ebenso kann man alle Kanten animieren um auch den Traffic in einem solchem Netzwerk anzeigen zu können. Es können also auch mit großer Wahrscheinlichkeit komplett neue Ideen in die Visualisierung des Graphen integriert werden.

### **Graph-Eigenschaften:**

Ergebnis: Sowohl gerichtete Kanten als auch Schlingen werden dargestellt. Auch Kantengewichte können angezeigt werden.

Kommentar: Die Standard-Einstellung der Bibliothek wirkt etwas unübersichtlich, da alle Elemente sehr einfach dargestellt werden. Wenn man aber ein wenig Zeit investiert, kann man alle Elemente nach Belieben anpassen, um so für mehr Übersichtlichkeit zu sorgen.

### **Layout-Algorithmen:**

Ergebnis: (Smart-) Organic, Circular, (Directed) Orthogonal, (Incremental-) Hierarchic, Radial und Single Cycle Layouter.

Kommentar: Zu den oben genannten gibt es auch noch einige Layouter für Graphen, die eine Baumstruktur aufweisen. Außerdem ist es möglich eigene Layouter zu integrieren oder die bestehenden abzuwandeln. Die Bibliothek wurde auch explizit dafür ausgelegt, dass man schnell und mit so wenig Aufwand wie möglich eigene Layouter integrieren kann. Die genannten Layouter nutzen alle in irgendeiner Weise auch ein kräftebasiertes Modell.

### **Stabilität:**

Ergebnis: Die Bibliothek läuft stabil, nutzt aber viel CPU-Leistung für das Rendering.

(+ +, Details siehe Tabelle 2)

Kommentar: Mehrfach gleiche Kanten werden problemfrei gerendert. Auch sind keine Abstürze vorgekommen. Das Rendern benötigt viel CPU-Leistung, aber dafür sind die Render-Zeiten wirklich sehr kurz, bedenkt man die Zahl der Elemente, die einigermaßen Sinnvoll platziert werden müssen. Zudem konnte bei dieser Bibliothek kein Absturz herbeigeführt werden.

Bei der Nutzung von C# 6 und VisualStudio 2015 CTP 6 ist allerdings ein Problem mit dem VisuelTree-Debugger aufgetreten. Dies konnte nur behoben werden, indem das Live-Debugging des VisualTrees abgeschaltet wurde. Dieser Fehler wird allerdings durch einen Bug im VisualTree-Debugger von Microsoft hervorgerufen.

## b) Bewertung

Bei den C#-Bibliotheken war das Ziel sie mit Microsofts neuem GUI-Framework WPF zu testen. So standen Anfangs drei Bibliotheken zur Auswahl, die getestet werden sollten: GraphX, NodeXL und die kommerzielle Bibliothek yFiles WPF, für die uns eine Testlizenz gewährt wurde, um die Bibliothek im Rahmen dieser Fachstudie zu evaluieren.

Dabei ist aber NodeXL leider ausgeschieden, da hierfür keine funktionierende Dokumentation gefunden werden konnte. NodeXL liefert seine Dokumentation in einer Windows-Hilfedatei (CHM-Datei) aus. Diese Datei ist leider beschädigt und konnte nicht geöffnet werden. Eine andere Dokumentation ist nicht verfügbar.

Es blieben also GraphX und yFiles WPF, wobei GraphX im kompletten Test mit Abstand am schlechtesten abgeschnitten hat. Das kann daran liegen, dass GraphX auch nicht für WPF entwickelt wurde. Darauf lässt die Tatsache schließen, dass die Bibliothek nicht dem, für die WPF typischen und auch wichtigen MVVM-Patttern folgt. Dennoch ließ sich die Bibliothek problemlos einbinden, zumal hierfür auch eine sehr ausführliche Dokumentation vorhanden ist.

Wie bereits erwähnt hat die Bibliothek GraphX am schlechtesten abgeschnitten. Das liegt daran, dass sie für das Layout des größten Graphen (Wicket) fast eine Minute benötigt, das Layout aber auch nicht wirklich gut ist. Für die Tests wurde GraphX mit dem CompundFDP Algorithmus und dem Überlappungsalgorithmus FSA genutzt. Dabei mussten aber auch die Algorithmus-Paramater auch manuell gesetzt werden, da hier offenbar keine oder zumindest keine sinnvollen Standardwerte gesetzt wurden. Ein weiteres Manko von GraphX ist, dass die Interaktionen wie Zoom oder Drag nicht wirklich performant laufen. Sie scheinen auch relativ schlecht zu skalieren, sodass bei mäßig großen Graphen schon deutliche ruckler bei der Interaktion mit dem Graphen wahrgenommen werden können, was das Arbeiten mit dem Graphen deutlich erschwert. Dennoch kann GraphX durchaus eine Option gegenüber einer kommerziellen Lösung darstellen. Vor allem dann, wenn man nur mit kleinen Graphen arbeiten muss oder kein Budget für eine kommerzielle Bibliothek hat.

Weniger überraschend ist, dass yFiles WPF sich gegenüber GraphX behaupten konnte. Die Bibliothek hat sehr ausgereifte Layoutalgorithmen, die auch schon mit sinnvollen Paramatern initialisiert werden, sodass man als Entwickler auch einfach nur den Algorithmus anwenden kann und schon sehr brauchbare Ergebnisse erhält. Ebenfalls schön ist, dass yFiles WPF für den größten Graphen nicht mehr als zwei Sekunden für Layout und Rendering benötigt. Das ist zwar noch langsamer als bei den anderen Bibliotheken liegt aber vermutlich an dem Overhead den WPF mit sich bringt. Die Entwickler geben aber auch Hinweise, wie man noch etwas mehr Performanz aus der Bibliothek herausholen kann, indem man bestimmte WPF-Funktionalitäten abschaltet.

yFiles WPF bietet zudem eine sehr ausführliche Dokumentation und eine Vielzahl an Beispielprojekten, die dabei helfen verschiedene Anwendungsszenarien zu ermöglichen. Sei es, dass bestimmte Interaktionen ermöglicht, oder dass eigene eingebaut werden sollen. Prinzipiell ist jede Funktion der Bibliothek sowohl in der Dokumentation als auch in einem Beispielprojekt erklärt. Sollten dennoch Fragen bleiben bietet yFiles einen sehr fachkundigen und schnellen Support an, der auch die letzten verbliebenen Wünsche erfüllen können sollte.

Betrachtet man die beiden Bibliotheken etwas genauer fällt auch auf, dass GraphX in sich nicht wirklich konsistent sondern eher wirr wirkt. Dahingegen wurde bei yFiles WPF sehr großen Wert darauf gelegt, dass ein Entwickler sich leicht damit tut, die Bibliothek zu verstehen. Dabei ist die Ordnung und Konsistenz in der Bibliothek eine große Unterstützung. Wer also das Budget hat um sich eine kommerzielle Bibliothek einzukaufen ist mit yFiles WPF definitiv gut bedient.

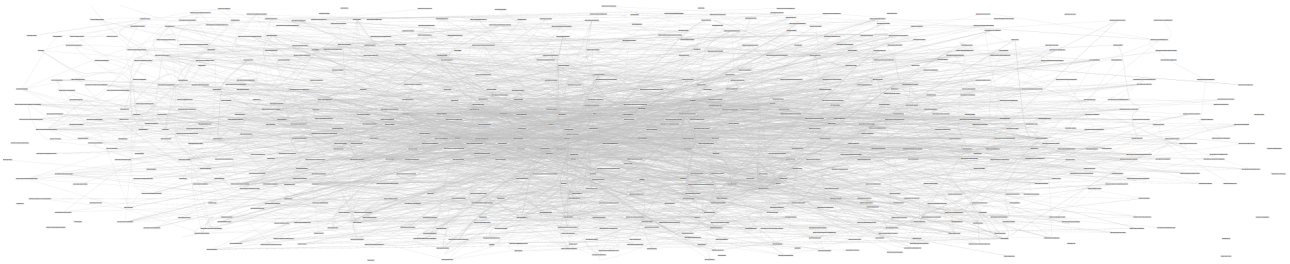


Abbildung 57: Datensatz Wicket in GraphX

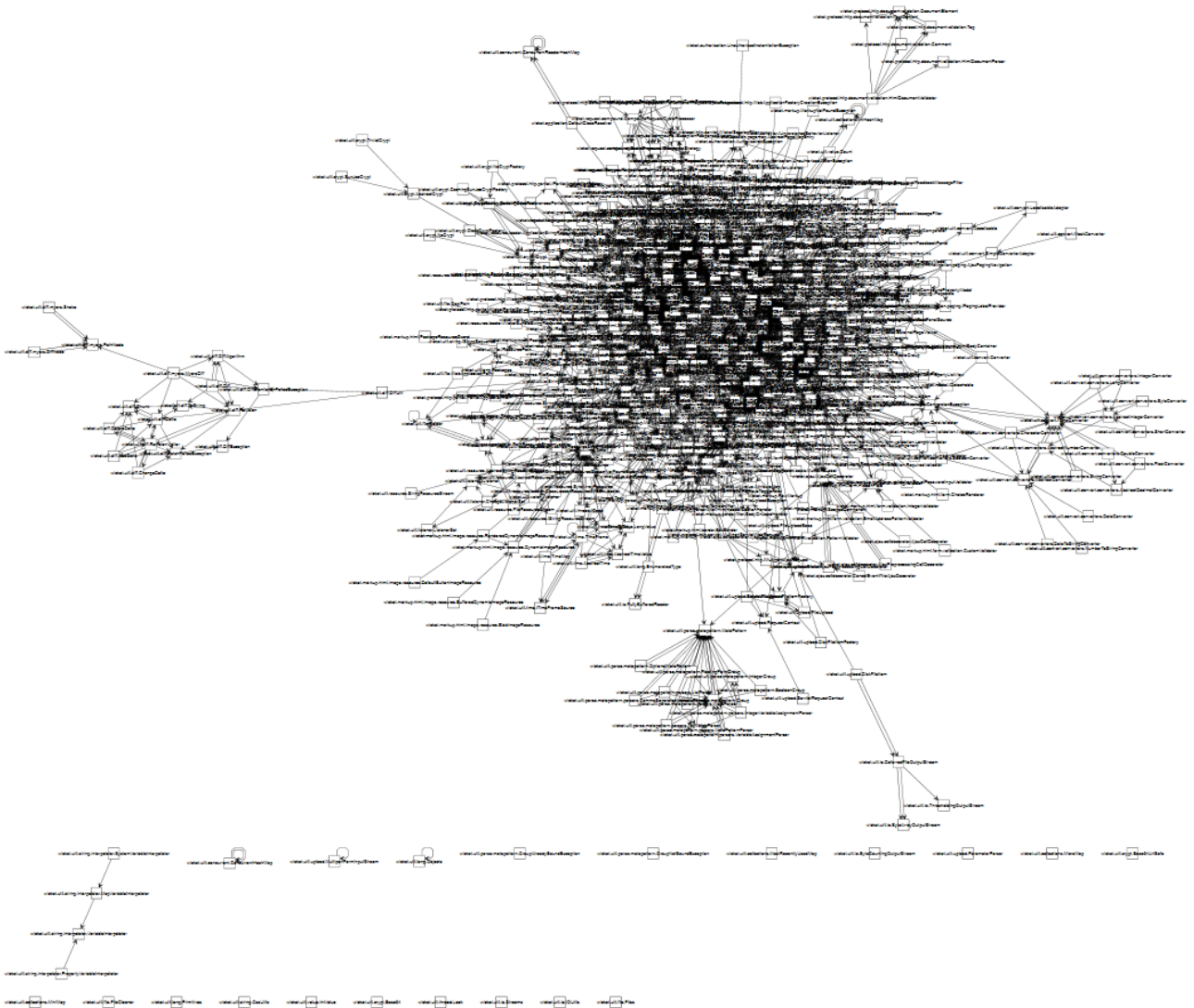


Abbildung 56: Datensatz Wicket in yFiles WPF

Wie in Abbildung 57 und Abbildung 56 zu sehen ist, gelingt es yFiles WPF deutlich besser in die Randknoten herauszuarbeiten, während bei GraphX der gesamte Graph eigentlich nur ein einziger grauer Ball ist, in dem man keine wirkliche Struktur erkennen kann.

## 7. Gesamtbewertung

Bibliotheken Bewertungskriterien	JGraphX	Prefuse	CytoScapels	D3JS	SigmaJS	GraphX	yFiles WPF
	<b>Einrichtungszeit</b>	1h	3h	3h	1h	1,5h	2,5h
<b>Import-Funktion</b>	✓	✓	✗	✓	✓	✗	✓
Anzahl Import-Formate	1	3	0	5	2	0	1
<b>Render-Performance</b>	++	++	-	++	+	--	++
kleinster Graph	12ms	792ms	118ms	19ms	72ms	649ms	193ms
größter Graph	2026ms	1719ms	1144ms	117ms	876ms	55201ms	3955ms
<b>native Interaktionen</b>	✓	✓	✓	✓	✓	✓	✓
Umfang der Interaktionen	++	+	+	+	+	+	++
Interaktions-Performanz	++	++	+	++	+	--	++
<b>Speicher-Verbrauch</b>	75 MB	50 MB	22MB	15MB	57 MB	130 MB	50 MB
<b>CPU-Last</b>	100%	100%	100%	100%	100%	100%	100%
GPU Unterstützung	✗	✗	✗	✗	✗	✗	✗
<b>Dokumentation</b>	✓	✓	✓	✓	✓	✓	✓
Vollständigkeit	++	-	+	++	+	++	+
Verständlichkeit	++	+	++	++	++	++	++
Verfügbarkeit	++	++	++	++	++	++	++
Benutzbarkeit	++	-	++	++	++	++	++
Qualität	++	-	++	++	++	++	+
<b>Code-Dokumentation</b>	✓	✓	✓	✗	✗	✓	✓
Vollständigkeit	++	+	-			+	++
Verständlichkeit	++	++	--			++	++
Benutzbarkeit	++	++	--			++	++
<b>Beispiele</b>	✓	✓	✓	✓	✓	+	++
Funktionalität	++	++	+	+	0	+	++
Verständlichkeit	++	++	++	++	++	+	++
Erklärungen / Kommentierung	++	++				++	++
Anleitungen	++	-	--	--	--	+	++
<b>Live-Rendering</b>	✗	✓	✓	✓	✓	✗	✓
Performanz		++	--	++	+		++
de- / aktivierbar		✓	✗	✗	✗		✓
Benutzbarkeit		+	-	+	-		+
<b>Stabilität</b>	++	++	+	++	++	++	++
Zuverlässigkeit	++	++	+	++	+	++	++
Abstürze	++	++	--	++	++	++	++
Fehlermeldungen	++	+	-	++	++	++	+
Warnungen	++	+	+	+	+	++	++
Auswirkung auf das System	++	++	+	++	-	++	++

Tabelle 2: Bewertungen der einzelnen Bibliotheken als Gesamtübersicht.

Da die Bibliotheken auf verschiedenen Programmiersprachen und Technologien basieren ist ein direkter Vergleich der Bibliotheken nicht oder nur sehr eingeschränkt möglich.

Dennoch möchten wir im Folgenden ein paar allgemeine Themengebiete noch einmal für alle Bibliotheken im Gesamten beleuchten, um zumindest ein gutes Gesamtbild der getesteten Bibliotheken zu vermitteln.

## a) Support

Da die meisten Bibliotheken freie OpenSource-Projekte sind, wird von den Entwicklern auch kein Support zu den Bibliotheken gegeben. Hier stellen die C#-Bibliotheken **yFiles WPF**, als kommerzielles Produkt, als auch **GraphX** die Ausnahme dar. Während man bei **GraphX** den Entwickler über ein Kontaktformular erreichen kann, bekommt man von yFiles direkt die Support-E-Mail-Adresse, wo man auch innerhalb kürzester Zeit eine durchaus sehr brauchbare Antwort erhält.

## b) Community

Je nach Bibliothek existiert eine mehr oder minder aktive und große Community, die sich gegenseitig mit Rat und Tat unterstützt und auch untereinander Support für die jeweilige Bibliothek anbieten.

Während die Community der Java-Bibliothek **Prefuse** sehr stark am Abnehmen und mittlerweile schon als inaktiv bezeichnet werden kann ist für die C# (WPF)-Bibliothek **GraphX** keine wirkliche Community vorhanden. Die JavaScript-Bibliotheken können zumindest eine rudimentäre GitHub-Community mit 97 – 1460 Followern (97 bei **CytoscapeJS**, 254 bei **SigmaJS** und 1460 bei **D3JS**) vorweisen.

Die Java-Bibliothek **JGraphX** hingegen hat eine sehr aktive Community und der von den Entwicklern sogar ein Diskussionsforum zur Verfügung gestellt wird. Allerdings vertreiben die Entwickler von **JGraphX** auch eine stark erweiterte und kommerzielle JavaScript-Version der Bibliothek „mxGraph“. Darum beziehen sich die Entwickler in ihren Dokumenten auch ausschließlich auf ihre kommerzielle Bibliothek, wobei vieles davon aber auch für die freie Version übernommen werden kann.

## c) Einarbeitungsaufwand

Die meisten Bibliotheken ließen sich sehr einfach und schnell einbinden. Hierbei fällt auf, dass eine gute Dokumentation dabei natürlich einen entscheidenden Faktor spielt. Aber nicht weniger spielt die Komplexität der Bibliothek eine Rolle.

So war das Einbinden von **JGraphX** (Java), **CytoScapeJS** (JavaScript), **F** (JavaScript), **SigmaJS** (JavaScript) und **GraphX** (C#) doch mehr oder minder einfach.

Bei **yFiles WPF** (C#) war die Einbindung etwas aufwändiger, da hier noch Probleme mit der Lizenz auftraten, die zunächst behoben werden mussten. Mittlerweile ist das aber auch kein Problem mehr, da hier der Hersteller nachgebessert hat und die Lizenzdatei nun direkt mitinstalliert wird.

Lediglich die Java-Bibliothek **Prefuse** war deutlich aufwändiger einzubinden. Zunächst musste ein Archiv mit den Quelldaten heruntergeladen werden. Mittels einer kurzen Anleitung wird erklärt, wie diese Quelldaten dann mittels eines (vorher entsprechend anzupassenden) Skriptes zu einer nutzbaren JAR-Datei kompiliert werden kann.

Eine weitere Hürde ist die Komplexität der Bibliothek, der die nicht wirklich vollständige Dokumentation auch nicht so ganz gerecht wird. Zum einen müssen nämlich einige Schritte beachtet und ausgeführt werden, ehe die Daten angezeigt werden können, zum anderen müssen aber auch diverse Werte gesetzt werden, die leider nicht mit Standard-Werten versehen wurden. So lange hier aber keine Werte gesetzt werden, wird auch der Graph nicht angezeigt.

## d) Performanz

Bei der Performanz muss zwingend unterscheiden werden ob es um die Renderperformanz oder um die Performanz der Interaktionen geht, da sich die Bibliotheken hier teilweise sehr stark unterscheiden. Eine Bibliothek, die vielleicht noch gut im Rendern ist kann dafür bei den Interaktionen so inperformant sein, dass man die Interaktionen quasi nicht wirklich benutzen kann.

### Rendering:

Was das Rendering angeht war die C#-Bibliothek **GraphX** die langsamste und lieferte schon bei etwas größeren Graphen kein wirklich zufriedenstellendes Ergebnis mehr. Für unseren größten Graphen mit 3679 Elementen musste man im Schnitt sogar eine Minute warten, bis der Graph gerendert war.

Alle anderen Bibliotheken hingegen waren beim Rendering doch zumindest akzeptabel.

Es fällt aber auf, dass die C# (WPF) Bibliotheken, wenngleich **yFiles WPF** auch bedeutend schneller als **GraphX** arbeitet, am längsten brauchen um den Graph zu visualisieren.

Ein wenig besser als **yFiles WPF** sind die Java-Bibliotheken die durchschnittlich 2 Sekunden für den größten Graphen benötigen. Die beste Renderperformanz mit durchschnittlich 700ms bieten die JavaScript-Bibliotheken, die von uns im FireFox getestet wurden.

### Interaktion:

Nachdem **GraphX** (C#) beim Rendering schon schlecht abgeschlossen hat ist es wenig verwunderlich, dass es auch bei der Interaktionsperformanz die Bedürfnisse nicht wirklich erfüllen kann. Leider genauso wenig zu gebrauchen ist die Bibliothek **CytoScapeJS**.

Erfreulicherweise sind die restlichen Bibliotheken alle durchaus brauchbar, was ihre Interaktionsperformanz angeht. Darum ist das folgende Ranking auch teilweise eher subjektiv als dass es mit objektiven Daten untermauert werden könnte.

Dennoch gut erkennbar ist, dass **SigmaJS** noch die schlechteste Performanz die Interaktionen betreffend aufweist, da hier doch noch einige stärkere Ruckler wahrgenommen werden können. Sichtbar besser sind hingegen **JGraphX**, **D3JS** und **yFiles WPF** die alle doch sehr brauchbare Ergebnisse mit kaum wahrnehmbaren Rucklern liefern, sofern man nicht den kompletten Graphen hin- und her zieht.

Diese Interaktion verarbeitet nämlich ausschließlich Prefuse ruckelfrei, weswegen es in diesem Ranking ganz klar am besten abschließt.



## e) Einlesen von (Standard-)Formaten

Während die JavaScript-Bibliothek **CytoScapeJS** und die C#-Bibliothek GraphX keinerlei Importfunktionalität mitbringen fällt auf, dass die JavaScript-Bibliotheken alle unter anderem mit einem jeweils leicht unterschiedlichen JSON-Format arbeiten können, wohingegen die Java- und C#-Bibliotheken durchgehend das einheitliche XML-Format GraphML unterstützen. Darüber hinaus kann die JavaScript-Bibliothek **SigmaJS** noch GEXF-Format verarbeiten, während die Konkurrenzbibliothek D3JS zu JSON noch die Formate CSV, HTML und TSV unterstützt, die aber alle einer bestimmten Dokumentstruktur genügen müssen.

Bei den Java-Bibliotheken überrascht **Prefuse** mit der größten Vielfalt, da es zu GraphML auch noch die Formate TreeML und CSV unterstützt. Dabei müssen auch diese Dokumente eine bestimmte Struktur aufweisen, sodass **Prefuse** damit arbeiten kann.

Somit ist also **D3JS** mit insgesamt fünf Importformaten am vielfältigsten.

## f) Vielfalt (Knoten- / Kantendarstellung)

Das Ändern der Darstellung von Knoten und Kanten unterstützen im Prinzip alle Bibliotheken. Bei **D3JS** ist aufgefallen, dass es im Test nur die Kreis-Form für die Darstellung der Knoten unterstützt hat, obwohl laut Dokumentation verschiedene Knotenformen unterstützen soll. Wenn wir allerdings eine andere Form ausgewählt haben wurde aber nichts mehr angezeigt. Die Kantenfarben lassen sich allerdings aber genauso zuverlässig wie die Knotenfarben ändern. Ähnlich wie in **D3JS** sind auch in den beiden anderen JavaScript-Bibliotheken **CytoScapeJS** und **SigmaJS** verschiedene Knotenformen vordefiniert, mit dem Unterscheid, dass diese Bibliotheken die Knoten auch korrekt darstellen können. Die Kanten können ebenfalls auch nur farblich verändert werden, wobei **SigmaJS** auch unterschiedliche Linienarten unterstützt, während dafür bei **CytoScapeJS** das Symbol für gerichtete Kanten geändert werden kann.

Deutlich offener sind die Java- und C#-Bibliotheken die im Grunde alle ein komplett eigenes Design für die Knoten zulassen. Auch die Kanten können bei diesen Bibliotheken verändert werden. Nur stellt hierbei **GraphX** eine etwas größere Herausforderung dar, da der Entwickler zwar Bilder veröffentlicht hat, in denen er zeigt, dass es funktioniert. Auch existiert eine Kurzanleitung wie man Knoten und Kanten verändern können soll. Aber diese ist leider alles andere als hilfreich, weswegen man hier doch etwas basteln muss.

**Prefuse** hingegen ist bei der Anpassung der Kanten etwas eingeschränkt, da man hier nur aus den vordefinierten Formaten wählen kann, die dafür aber ausreichend umfangreich sind. Für **yFiles WPF** gibt es ein Beispiel, in welchem sogar animierte Kanten genutzt werden.

## g) Layout (als Ranking)

Die C#-Bibliothek **GraphX** bietet einige der bekanntesten Algorithmen für das Layouting an, setzt aber keine vernünftigen Standard-Werte und scheint auch sehr naiv zu arbeiten, sodass hier die mitunter am schlechtesten lesbaren Graphvisualisierungen entstehen. Allerdings kann man sicherlich mit den richtigen Parametern eine einigermaßen vernünftige Visualisierung erzeugen.



Abbildung 58: Graph „Wicket“ mit 3679 Elementen gerendert durch CytoScapeJS

Wenig besser ist auch die JavaScript-Bibliothek **CytoScapeJS**, die ebenfalls sehr schnell starkes Visual Clutter erzeugt. Außerdem scheint der Algorithmus mit größeren Graphen überfordert zu sein. In unseren Testfällen hat er zumindest immer wieder sehr zweifelhaft Visualisierungen geliefert (vgl. Abbildung).

Einigermaßen vernünftige Ergebnisse liefern die JavaScript-Bibliothek **SigmaJS** und die Java-Bibliothek **Prefuse**. Dabei stellen beide den Graphen recht ähnlich dar. Die Einzelknoten werden mit sehr viel Platzverlust weit entfernt vom Hauptgraphen gerendert und überlappen sich aber dabei teilweise. Der Hauptgraph hingegen ist aber sehr unübersichtlich, da die Knoten dazu neigen sich eher zueinander hin statt voneinander weg zu bewegen. Dies führt aufgrund der hohen Anzahl von Knoten und Kanten unweigerlich zu Visual Clutter.

Die Java-Bibliothek **JGraphX** arbeitet mit einem sehr ähnlichen Prinzip, nur dass es ihr gelingt, in den Randbereichen des Hauptgraphen die Cluster und die Knoten mit wenig Verbindungen zum Hauptgraphen herauszuarbeiten und diese übersichtlich darzustellen. Zwar überlappen sich auch hier teilweise die Knoten aufgrund der langen Klassennamen was aber durch die geringe Anzahl von Knoten in diesen Clustern nicht allzu gravierend ist.

Ein Manko von **JGraphX** ist, dass die ganzen Einzelknoten auf einem Fleck in einer Ecke des Anzeigebereiches überlappend angezeigt werden. Das liegt daran, dass die Knoten initial alle am Punkt 0/0 in die Visualisierung gesetzt werden. Da aber auf die Einzelknoten, da sie ja keine Kanten haben, keine Kräfte wirken, bleiben diese alle an dieser Position und überdecken sich gegenseitig.

Hier haben sich die Entwickler von der C#-Bibliothek **yFiles WPF** etwas einfallen lassen und ihren Layout-Algorithmus so verbessert, dass alle Knoten zunächst in einem Gittermuster in den Anzeigebereich gesetzt werden um dann erst das force-directed Layout auszuführen. Dies führt dazu, dass **yFiles WPF** wie auch schon **JGraphX** im Hauptgraphen in den Randbereichen die Cluster und Satelliten herausarbeiten kann – aber im Gegensatz zu **JGraphX** auch die Einzelknoten sauber und übersichtlich, zudem ohne großen Platzverlust dargestellt werden.

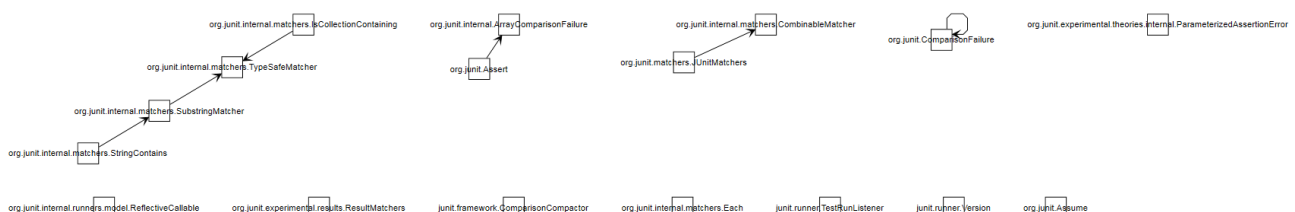


Abbildung 59: Einzelknoten in yFiles WPF

Am besten scheint der Layout-Algorithmus der JavaScript-Bibliothek **D3JS** zu arbeiten. hier werden alle Knoten und Kanten übersichtlich dargestellt. Außerdem werden die Knoten soweit entzerrt, dass man selbst in unserem größten Graphen noch erkennen kann, welche Kante wohin führt – selbst im Kern des Hauptgraphen. Dabei ist aber auch entscheidend, dass **D3JS** die Knotenbezeichnungen ausblendet und diese nur in einem ToolTip angezeigt werden, sobald man mit der Maus über einen Knoten führt. Durch das Weglassen dieser Information hat man natürlich deutlich weniger Probleme bei der Anordnung der Knoten, weswegen die Visualisierung insgesamt auch auf den ersten Blick sehr übersichtlich aussieht. Aber beim Arbeiten mit dem Graphen kann dieses Verhalten doch sehr hinderlich sein. Ein großer Nachteil von **D3JS** ist, dass die Knoten und Kanten auch aus dem vordefinierten Anzeigebereich verschwinden und somit auch nicht mehr erreichbar sind, da bei **D3JS** der Graph nicht komplett verschoben werden kann.

Somit sind **JGraphX**, **yFiles WPF** und **D3JS** auf einem Niveau was die Qualität der resultierenden Visualisierung in Bezug auf das Layout angeht.

## h) Algorithmen

Bis auf die JavaScript-Bibliothek **SigmaJS** stellen alle Bibliotheken mehrere verschiedene Layout-Algorithmen zur Verfügung. Zumeist sind das aber eigene oder angepasste Layout-Algorithmen, wobei aber teilweise auch die Basisalgorithmen wie Fruchtermann-Reingold oder Kamada-Kawai zur Verfügung stehen.

Außerdem werden von den Bibliotheken auch spezielle Algorithmen für Graphen, die eine Baumstruktur aufweisen, zur Verfügung gestellt, damit genau diese Baumstruktur in der Visualisierung wiedergefunden werden kann.

Die C#-Bibliothek **GraphX** enthält nur Basisalgorithmen, für die aber keine brauchbaren Standardwerte gesetzt wurden, weswegen hier die Ergebnisse auch nicht so recht überzeugen können. Außerdem sind, wie bereits unter Performanz beschrieben, die Algorithmen nicht wirklich performant.

Mit nur einem einzigen Algorithmus, dem sogenannten „forceAtlas2“, ist die JavaScript-Bibliothek **SigmaJS** auch sehr eingeschränkt in ihren Auswahlmöglichkeiten. Auch ist es nicht möglich der Bibliothek einen eigenen Layout-Algorithmus mitzugeben.

**CytoScapeJS** (JavaScript) hat schon mehr Auswahl, was den Layout-Algorithmus angeht, unterstützt aber im Gegensatz zu den folgenden Algorithmen keine besonderen Graphstrukturen wie zum Beispiel eine Baumstruktur. Außerdem sind die Algorithmen nicht wirklich ausgereift, wie man dem Bild im Abschnitt „Layout“ entnehmen kann.

Die Java-Bibliotheken **JGraphX** und Prefuse sowie die JavaScript-Bibliothek **D3JS** und die C#-Bibliothek **yFiles WPF** unterstützt eine ganze Reihe von verschiedenen Layout-Algorithmen, wobei auch diverse angepasste Algorithmen für verschiedene Baumstrukturen enthalten sind. Ebenso werden hierarchische Layouts sowie ein paar verschiedene andere Layouts unterstützt, die dann aber je nach Bibliothek variieren.

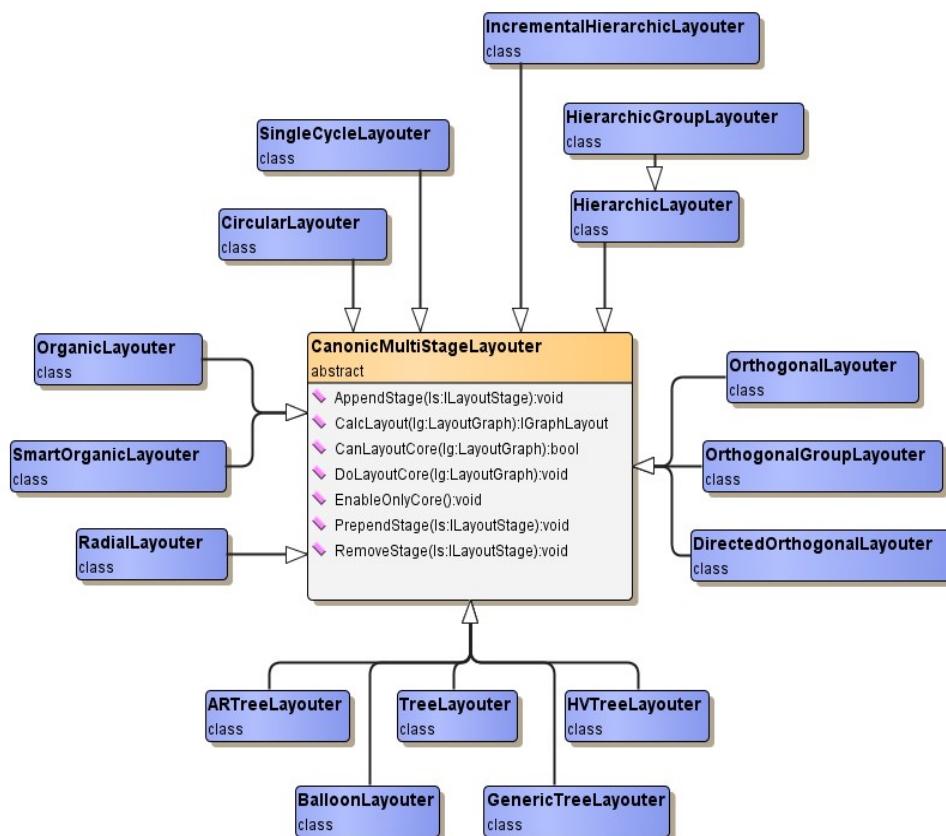


Abbildung 60: Übersicht der Layout-Algorithmen von yFiles WPF (yWorks, 2015)

## i) Vielfalt (nicht-)/kommerziell

Für Microsofts Programmiersprache C# existieren durchaus viele Bibliotheken zur Visualisierung von Graphen. Sucht man allerdings nach kostenlosen Versionen schränkt sich die Zahl der Möglichkeiten sehr stark ein. In unserem Fall blieben dabei drei Bibliotheken übrig: **NodeXL**, **Graph#** und **GraphX** (was die Weiterentwicklung von **Graph#** darstellt). Da **GraphX** relativ schnell an seine Grenzen kommt ist man bei C# schon fast genötigt für viel Geld auf eine kommerzielle Bibliothek auszuweichen.

Deutlich einfacher hat man es hier mit Java und JavaScript:

Bei Java gibt es einige stabile und gut funktionierende, frei verfügbare Bibliotheken zur Graphvisualisierung. Doch auch an kommerziellen Produkten, die dann erweiterte Möglichkeiten und auch Support bieten mangelt es für Java nicht.

JavaScript bietet vermutlich die größte Vielfalt an frei verfügbaren Bibliotheken. Da aber viele OpenSource-Projekte im JavaScript-Bereich existieren neigen viele Entwickler dazu aus anderen Bibliotheken Code zu übernehmen. Ebenfalls gibt es viele Projekte, die im Grunde nur einen Ableger („Fork“) einer Bibliothek von einer anderen Entwicklergruppe darstellen.

Wirklich eigenständige und von Grund auf entwickelte Bibliotheken gibt es in JavaScript nicht allzu viele, weswegen die restlichen Bibliotheken zumeist eine Art Flickenteppich aus Codeschnipseln anderer Bibliotheken darstellen. Das mag der Funktionalität vielleicht keinen Abbruch tun, vielleicht auch nicht der Performanz. Die Verständlichkeit des Codes leidet aber definitiv stark unter dieser Vorgehensweise. Einen weiteren Nachteil von JavaScript kann auch das relativ geringe Angebot an kommerziellen Bibliotheken darstellen.

## 8. Quellen

*CytoScapeJS*. (07. 06 2015). Von <http://js.cytoscape.org> abgerufen  
D3JS. (07. 06 2015). Von <http://www.d3js.org> abgerufen  
JGraphX. (19. 05 2015). Von mxGraph Handbuch:  
[https://igraph.github.io/mxgraph/docs/manual\\_javavis.html](https://igraph.github.io/mxgraph/docs/manual_javavis.html) abgerufen  
NodeXL. (07. 06 2015). Von <http://nodexl.codeplex.com/> abgerufen  
PantheR. (07. 06 2015). Von <http://www.panthernet.ru/en/projects-en/graphx-en> abgerufen  
Prefuse. (31. 05 2015). *Prefuse Homepage*. Von <http://www.prefuse.org/gallery> abgerufen  
SigmaJS. (06. 07 2015). Von <http://www.sigmajs.org> abgerufen  
*SourceForge*. (07. 06 2015). Von Prefuse: <http://sourceforge.net/projects/prefuse/> abgerufen  
*Wikipedia*. (02. 06 2015). Von Graph (Graphentheorie):  
[http://de.wikipedia.org/wiki/Graph\\_%28Graphentheorie%29](http://de.wikipedia.org/wiki/Graph_%28Graphentheorie%29) abgerufen  
yWorks. (31. 05 2015). *yWorks Developers Guide*. Von  
[http://docs.yworks.com/yfileswpf/developers-guide/major\\_layouters.html](http://docs.yworks.com/yfileswpf/developers-guide/major_layouters.html) abgerufen

Abbildung 1: ungerichtetes Knoten-Kanten-Diagramm .....	5
Abbildung 2: Adjazenzmatrix .....	5
Abbildung 3: Adjazenzliste.....	5
Abbildung 4: Liste der gefundenen Tools und Bibliotheken.....	11
Abbildung 5: Datensatz jFTP mit JGraphX.....	12
Abbildung 6: Datensatz jFTP in JGraphX .....	13
Abbildung 7: Datensatz Wicket mit JGraphX .....	14
Abbildung 8: Datensatz JUnit mit JGraphX .....	14
Abbildung 9: Visuelle Einstellungsmöglichkeiten (JGraphX, 2015) .....	14
Abbildung 10: Einstellungsmöglichkeiten für die Spitze einer Linie (JGraphX, 2015).....	15
Abbildung 11: Mögliche Symbole für gerichtete Kanten in JGraphX .....	16
Abbildung 12: Knotenformen von JGraphX.....	16
Abbildung 13: Kantenarten von JGraphX .....	16
Abbildung 14: Datensatz jFTP mit Prefuse .....	17
Abbildung 15: Überlappung auch bei wenigen Objekten .....	18
Abbildung 16: Datensatz JUnit mit Prefuse.....	18
Abbildung 17: Datensatz jFTP in Prefuse.....	18
Abbildung 18: Datensatz Wicket mit Prefuse.....	18
Abbildung 19: Automatisch an die Textgröße angepasste Knoten in Prefuse.....	20
Abbildung 20: Beispiel für visuelle Besonderheiten in Prefuse (Prefuse, 2015).....	20
Abbildung 21: Beispiel der verschiedenen Kantenanpassungen in Prefuse (Prefuse, 2015).....	20
Abbildung 22: Knotenformen in Prefuse .....	21
Abbildung 23: Kantenarten in Prefuse.....	21
Abbildung 24: Datensatz Wicket in Prefuse mit FR-Algorithmus .....	22
Abbildung 25: Datensatz jFTP mit CytoScapeJS .....	23
Abbildung 26: Datensatz jFTP mit CytoScapeJS .....	24
Abbildung 27: Datensatz jUnit mit CytoScapeJS .....	24
Abbildung 28: Datensatz Wicket mit CytoScapeJS.....	25
Abbildung 29: Mögliche Knotenformen von CytoScapeJS.....	27

Abbildung 30: Datensatz jFTP mit D3JS.....	28
Abbildung 31: Datensatz jFTP mit D3JS.....	29
Abbildung 32: Datensatz jUnit mit D3JS .....	30
Abbildung 33: Datensatz Wicket mit D3JS .....	30
Abbildung 34: Problem - Die Knoten und Kanten werden außerhalb des Fensters platziert.....	32
Abbildung 35: Knotenformen von D3JS .....	32
Abbildung 36: Datensatz jFTP mit SigmaJS .....	33
Abbildung 37: Datensatz jFTP mit SigmaJS .....	34
Abbildung 38: Datensatz Wicket mit SigmaJS.....	35
Abbildung 39: Datensatz jUnit mit SigmaJS .....	35
Abbildung 40: Mögliche Knotenformen von SigmaJS.....	37
Abbildung 41: Datensatz jFTP mit yFiles HTML .....	38
Abbildung 42: Default-Darstellung eines Knotens in den Beispielen von yFiles HTML & yFiles WPF .....	41
Abbildung 43: Datensatz jFTP in D3JS, CytoScapeJS und SigmaJS.....	41
Abbildung 44: Datensatz jUnit und Wicket in D3JS .....	42
Abbildung 45: Datensatz Wicket mit SigmaJS.....	43
Abbildung 46 : Datensatz jFTP mit GraphX .....	44
Abbildung 47: Datensatz jFTP mit GraphX .....	45
Abbildung 48: Datensatz jUnit mit GraphX .....	45
Abbildung 49: Datensatz Wicket mit GraphX.....	45
Abbildung 50: Datensatz jFTP in der Minimap der ToolBox .....	46
Abbildung 51: Datensatz Wicket in GraphX (mit Zoom).....	47
Abbildung 52: Datensatz jFTP in yFiles WPF.....	48
Abbildung 53: Datensatz jFTP in yFiles WPF.....	49
Abbildung 54: Datensatz Wicket in yFiles WPF .....	49
Abbildung 55: Datensatz Wicket in yFiles WPF .....	49
Abbildung 56: Datensatz Wicket in yFiles WPF .....	53
Abbildung 57: Datensatz Wicket in GraphX .....	53
Abbildung 58: Graph „Wicket“ mit 3679 Elementen gerendert durch CytoScapeJS.....	57
Abbildung 59: Einzelknoten in yFiles WPF .....	58
Abbildung 60: Übersicht der Layout-Algorithmen von yFiles WPF (yWorks, 2015) .....	59

Wir versichern, diese Arbeit selbstständig verfasst zu haben. Wir haben keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Wir haben diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.