Information Visualization

hgkz iad
Semester 4
Jahr SS 2007
Modul Information Visualization
Dozent Prof. Jürgen Späth, Prof. Dr. Harald Gall, Sandro Boccuzzo
Gruppe Christoph Schmid, Jeremy Stucki, Ramun Liesch

Inhalt

Aufgabenstellung

Visualisierung und Interaktion von dynamischen Daten auf screenbasierenden Medien. Die Entwicklung einer visuellen Informationsdarstellung von Software steht dabei im Vordergrund.

«Program visualization is the visualization of the actual program code or data structures in either static or dynamic form.»

Price, Baecker und Small

Definition

information (uncountable) (Short form: info)

A collection of related data.
Knowledge about a topic.
Data that have been processed into a format that is understandable by its intended audience.
A service provided by telephone which provides listed telephone numbers of a subscriber: see 411.
answer to a question, minimal part of information is bit (2), which is answer yes or no.

visualization (plural visualizations)

the act of visualizing, or something visualized
the visual representation of data

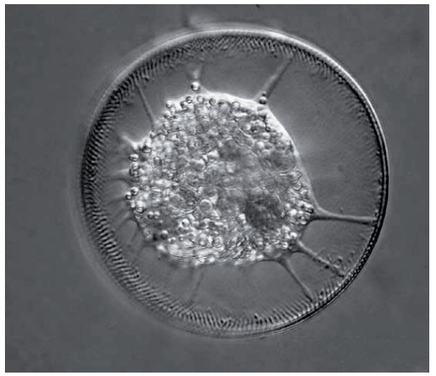
Quelle: http://en.wiktionary.org/wiki/

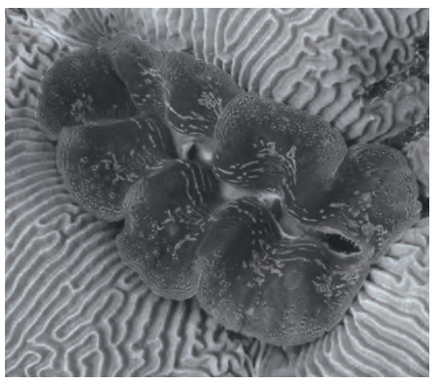
Recherche

Um eine grosse Vielfalt an Inspirationen zu generieren, wurde während der Recherche in verschiedenen Themenbereichen nach potentiellem Material gesucht.

Inspirationsfelder

Natur
Architektur
Typografie
generatives Design



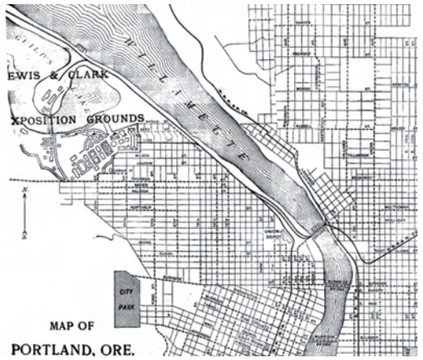


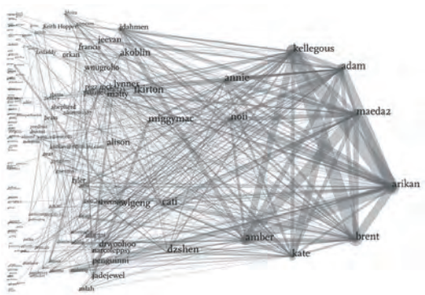
Natur
Software ist dynamisch. Sie wächst und verändert sich. Es liegt deshalb nahe, Inspirationen in organischen Systemen zu suchen.

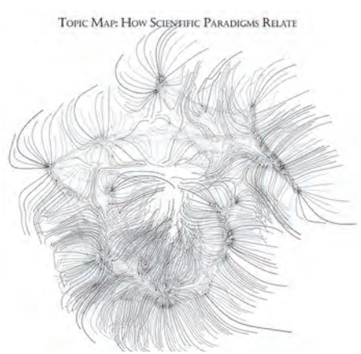
Architektur
Um Daten kognitiv lesbar zu machen, können sie topografisch angeordnet werden. Die Architektur gibt uns einige Beispiele in Raumordnung und Darstellung.

Typografie
Information lässt sich auf jegliche Art typografisch darstellen. Durch die Gestaltung von typografischen Informationsmodellen können visuell interessante Strukturen entstehen, welche die Aussage als gestalterisches Element nutzen.

generatives Design
Strukturen, die sich nach vorgegebenen Regeln, Algorithmen entwickeln.







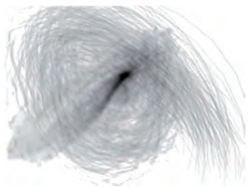
Analyse

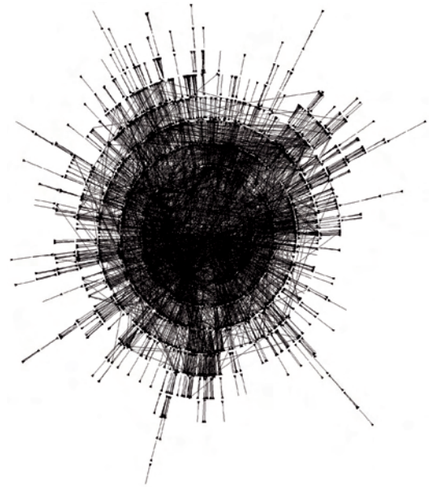
Um die nutzbaren Eigenschaften und Werte der gesammelten Recherche besser auswerten zu können, wird das Bildmaterial mit folgenden visuellen Parametern analysiert.

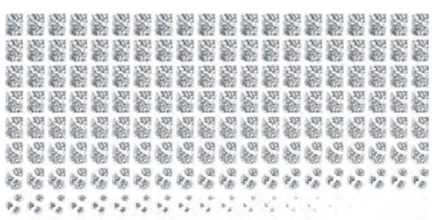
Parameter

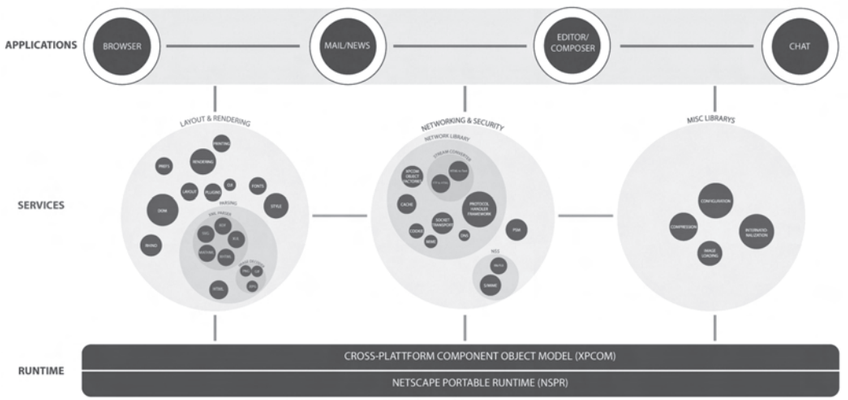
hell – dunkel
Zentrum – Peripherie
gross – klein
statisch – dynamisch
Netzwerk – Hierarchie
achromatisch – chromatisch
erkennbare Muster – unerkennbare Muster

Gleichzeitig wird der Aufbau einer Software analysiert. Als Datenmodell wird in diesem Fall Mozillla Firefox verwendet. Dabei geht es vor allem um die Erkenntnisgewinnung über den modularen Aufbau von Software und das Ausmass der verwendeten Metriken.









1. Runtime Layer

1.1 Netscape Portable Runtime (NSPR):
Provides a common runtime library for the variety of platforms on which Mozilla runs. It serves as the base on top of which all other components are built. It provides APIs for, threading, synchronization, I/O (file, memory and network), network addressing, time, memory management, process management, data structures, dynamic library.
1.2 Cross platform Component Object Model (XPCOM):
Just above NSPR is the Cross platform Component Object Model (XPCOM) component. This is an object services framework, which allows objects that define some basic interfaces for reference counting and query for supported interfaces to be used in other languages or libraries.

2. Service Layer

2.1.1 Gecko
Gecko (formerly ngLayout) is the Mozilla component responsible for actually turning data files into graphic displays. At its core is a modular layout engine that is fed by a collection of content parsing modules and in turn drives a rendering system that actually paints pixels on the screen. The same layout engine does double-duty as the underpinnings for XPFE, which draws the UI components for application packages. The document parsing, layout & rendering pipeline is thereby interwoven with the event-driven UI.

2.1.2. Document parsing
Gecko includes a native XML parser that is the basis for several modules that handle specific XML formats like XHTML, MathML, SVG, RDF and XUL. In addition to this, there is a separate HTML parser that is capable of handling non-standard mark-up.
2.1.3. Image library

Images are loaded by Libimg2 (AKA libpr0n), which can decode PNG, GIF and JPEG formats. It has support for full 24-bit alpha channels.
2.1.4. Style system

The style system is not limited to CSS and can be extended to handle other style sheet languages as well. There has been some work toward adding support for XSL.
2.1.5. Interactivity
After parsing has taken place, the resulting structured content is made available for inspection and manipulation in JavaScript via the Document Object Model API. Rhino, the JavaScript engine, uses Interface Definition Language bindings to access the DOM.

2.1.6. Plug-ins
The layout engine treats all plug-ins the same way. Essentially, they are allocated a canvas on which to draw. Sitting atop the plug-in system is the Open Java Interface, which allows the user to choose between JVM implementations.

2.1.7. Rendering
The cross-platform rendering engine handles both composition & transformations. It is also responsible for drawing widgets using the native graphics tool kit of the OS.

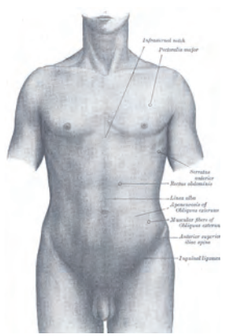
2.1.8. UI
The UI is written in XUL that is parsed into an RDF tree structure in memory. This is then fed into the DOM for the chrome (everything surrounding the document display area).

Softwarearchitektur als biologisches System

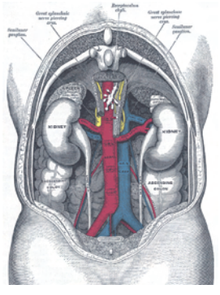
System Mensch
Wir gehen vom Menschen als System aus. Das System enthält Subsysteme (Programme) wie etwa das «Blutreinigungsprogramm» Leber.
Die Leber ist aus verschiedensten Geweben (Modulen) zusammengesetzt, deren Zusammenarbeit essentiell ist: einige produzieren benötigte Stoffe, andere sorgen für die Strukturierung, manche kontrollieren die Aktivität. Diese Gewebe kommunizieren jedoch nur über genau festgelegte Schnittstellen (Interfaces) miteinander. Es können aber auch andere Subsysteme wie der Magen der Leber mitteilen, mehr Gallensaft zu produzieren (API).
Die einzelnen Gewebe bestehen aus Zellen der jeweils gleichen Art, etwa Muskelzellen. Diese Zellen können als Instanzen der Klasse «Muskelzelle» angesehen werden.

Zelle als Instanz einer Klasse
Eine Zelle kann als Instanz einer Klasse angeschaut werden. In der DNA (Source-Code-Beschreibung der Klasse) ist festgehalten, wie eine Instanz dieser Klasse auszusehen hat. Wird ein Objekt basierend auf dieser Klasse instanziiert, erhält es sowohl allgemeine, für alle Instanzen dieser Klasse gültige Klassenvariablen, aber auch Instanzvariablen, die nur für dieses Objekt gelten.

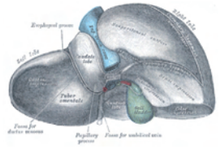
Kommunikation
Die innerhalb des Systems am Beispiel der Nervenzellen.



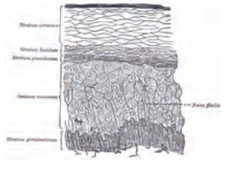
System



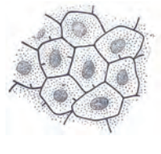
System Bestandteile



Subsystem



Modul



Instanz einer Klasse

Zeichenentwicklung I

Abb. 01
Durch die Anordnung der Objekte auf verschiedenen Radien wird eine Teilung in Gruppen sichtbar. Die Verbindungen der Objekte zeigen Abhängigkeiten und die Hierarchie auf. Der Kreis birgt den Vorteil, dass von allen Objekten zueinander eine visuelle Verbindung möglich ist.
Diversen Parametern können Metriken zugeteilt werden. Zum Beispiel die Höhe eines Objektes vom Zentrum aus. Jede Abweichung von der guten Form (in diesem Fall der Kreis) zieht Aufmerksamkeit auf sich und kann als Warnung o.ä. interpretiert werden.

Abb. 02
Gleiches Element, das in jeder Zoomstufe neu visualisiert wird. Bei jedem zoom out wird das Zeichen neu generalisiert und verliert dadurch an Individualität.

Abb. 01

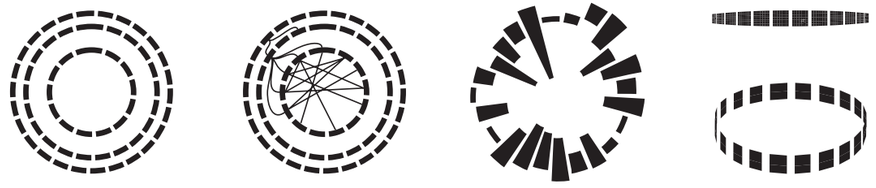


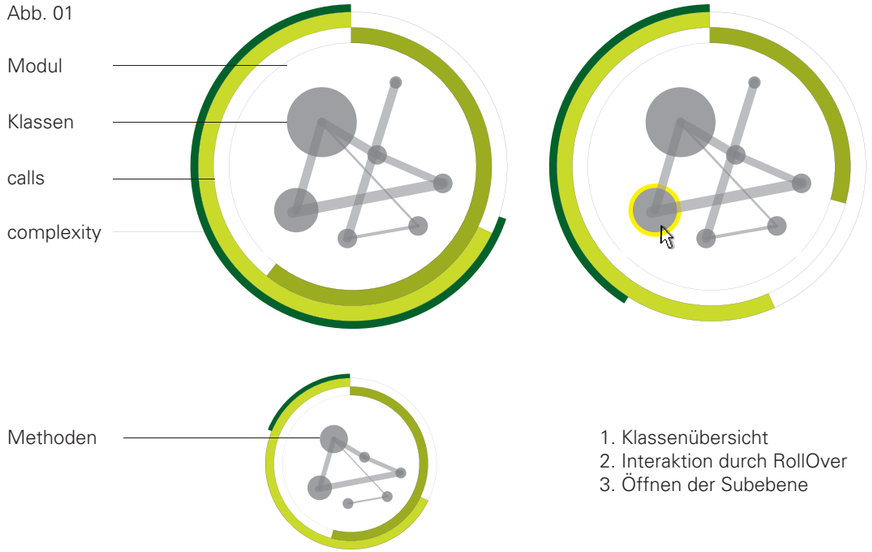
Abb. 02

Generalisierung

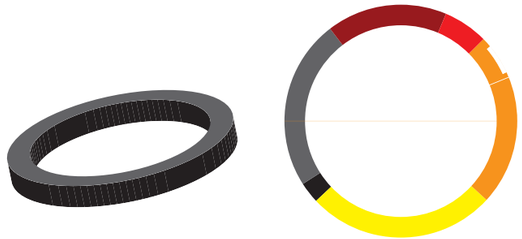
Modellentwurf A

Für die Datenvisualisierung werden verschiedene Ansätze für ein mögliches Datenmodell gesucht. Die Kreisdarstellung (Abb. 01) zeichnet sich aus durch ihre Einfachheit und Geschlossenheit. Das Modell wird in einen Aussenbereich (Container) und Innenbereich (Inhalt) unterteilt, wobei letzterer durch verschiedene Metriken angeordnet wird. Durch die Selbstähnlichkeit lässt sich die Darstellung beliebig tief visualisieren.
Durch die Erweiterung in die Dreidimensionalität lassen sich zusätzliche Metriken wie zum Beispiel Versionen oder Entwickler einbinden. Interessant wird der Aspekt der Dreidimensionalität als visueller Effekt von Perspektivenwechsel (Abb. 02).
Abb. 03 zeigt einen möglichen Ansatz für die Zoomstufendarstellung. Wichtig dabei ist, in jeder Stufe den Kontext in vereinfachter Form darzustellen.

Abb. 01



1. Klassenübersicht
2. Interaktion durch RollOver
3. Öffnen der Subebene



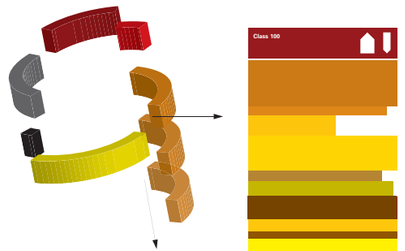
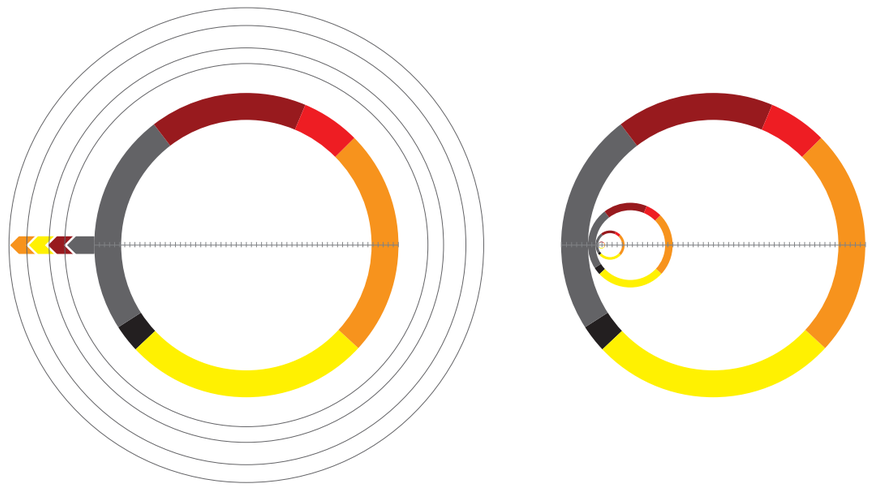


Abb. 02

Abb. 03

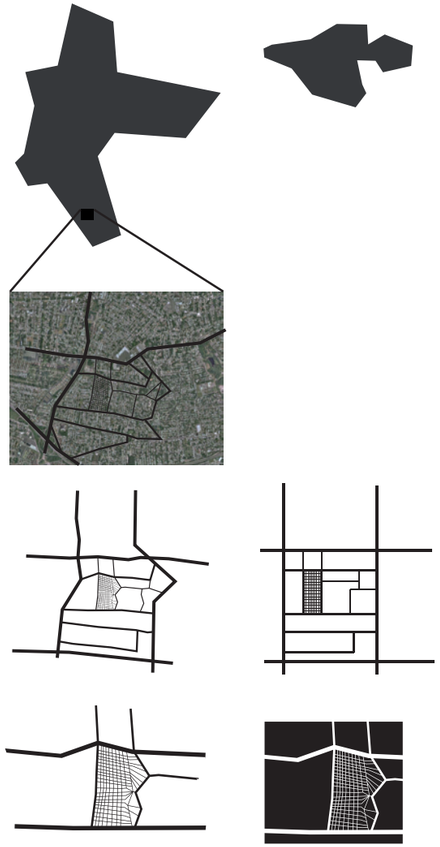


Modellentwurf B

In einer Landkarte findet man eine interessante Analogie zur Software. Die Erde bildet als ganzes ein System. Kontinente, Inseln sind Subsysteme. Interessant zu beobachten ist, das dieses ganze System, jedes seiner Teile und ebenfalls eine Hierarchie visuell kommuniziert wird. Ohne Vorkenntnisse können Teile und Hierarchiestufen problemlos zugeordnet werden.

Die Kommunikation innerhalb des Systems findet in Form von Gütertransport und -austausch, Geldfluss, Ressourcenverteilung (Rohstoffe und Energieträger), Menschenströme ect. Eine problematische Region in der Lebensgrundlagen wie etwa die Wasserversorgung nicht vorhanden sind, könnten in der Software z.B. fehleranfällige Module darstellen. Völkerwanderungen und Entstehung von Populationen können die Entwicklung einer Software aufzeigen.

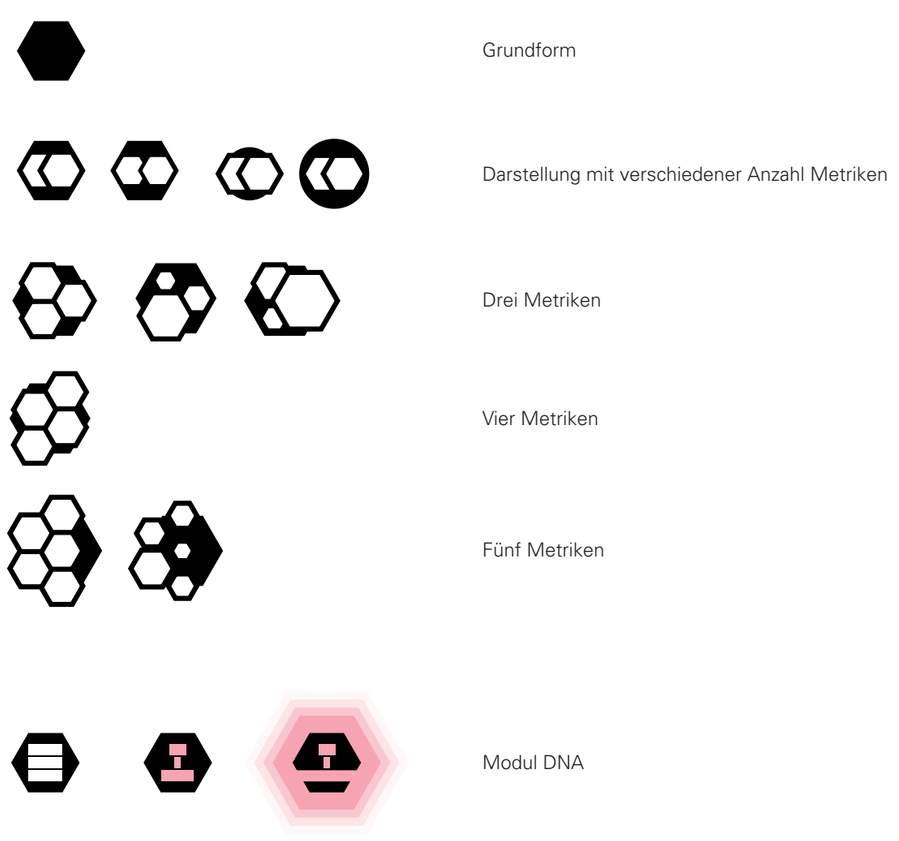
Am Beispiel einer Stadt (Subsystem) soll gezeigt werden wie Teile eines Ganzen erkennbar sind, und Abweichungen von der guten Form als Störung und Fehlerquelle erkennbar werden.

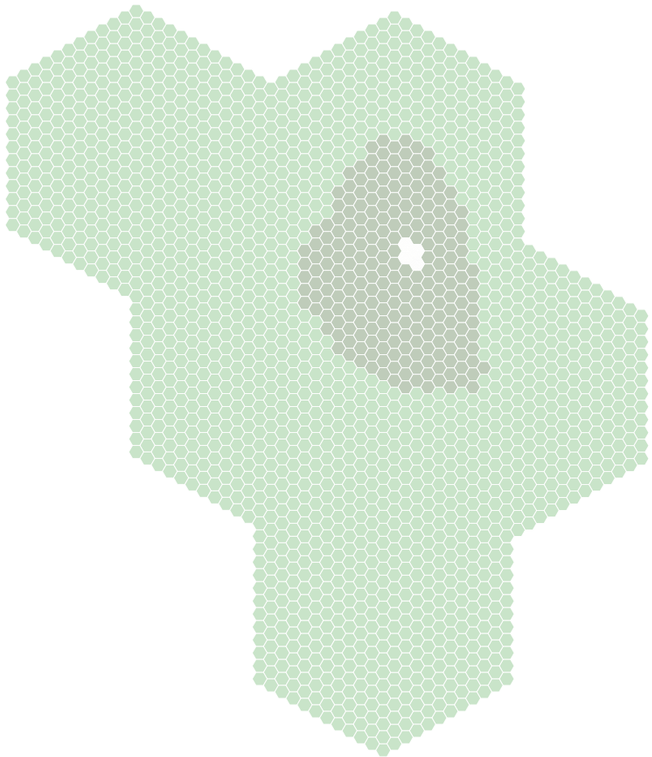


Modellentwurf C

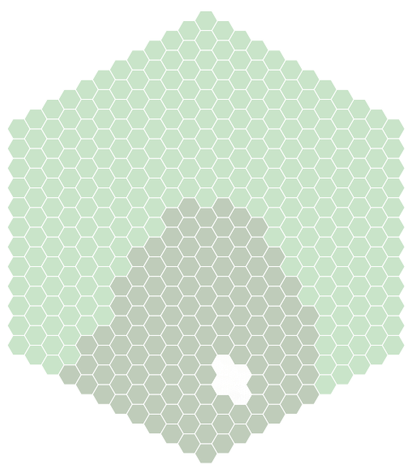
Zeichen
Die einzelnen Elemente des Systems werden durch Sechsecke dargestellt. Die jeweiligen Metriken werden durch weisse Sechsecke repräsentiert und gleichmässig im Element angeordnet. Elemente, bei denen alle Metrikwerte im optimalen Bereich liegen, erscheinen als ausgeglichen und gleichmässig (sie haben eine „gute Form“). Wenn einzelne oder mehrere Werte abweichen, wirkt das Element unausgeglichen und macht so auf sich aufmerksam.

Karte (Seite 21)
Ein System soll selbstähnlich sein, das heisst auf allen Ebenen aus ähnlichen oder den selben Repräsentanten aufgebaut werden. Durch hineinzoomen kann so in tiefere Ebenen vorgedrungen werden, die sich jedoch in der Leseart nicht von den übergeordneten unterscheiden. Elemente, die auf sich aufmerksam machen wollen, färben umliegende und übergeordnete Elemente ein, so dass auch auf höheren Zoomstufen Auffälligkeiten erkannt werden können.





Kartensystem



Zeichenentwicklung II

Für die Darstellung von zwei Metriken eignet sich vorzugsweise eine Form, die aus zwei Grössen besteht. Die Wahl fällt deshalb auf das Rechteck. Damit lässt sich in diesem Fall in der Y-Achse die Komplexität und in der X-Achse Anzahl lines of code (LOC) zeigen. Durch die Modifikation des Zeichens (Abb. 01) können schnell Anomalien erkannt und die daraus resultierenden Schlussfolgerungen gezogen werden.

Durch die lineare Anordnung der Zeichen ergeben sich individuelle Superzeichen, die wiederum Auskunft über die Menge liefern. Die Sortierung der Zeichen erfolgt nach Alphabet, da diese Darstellung im Softwarebereich oft verwendet wird und deshalb intuitiv lesbar ist. Andere Sortierungen wie zum Beispiel nach Grösse oder Wichtigkeit könnten via Benutzer manuell gezeigt werden.

Abb. 01

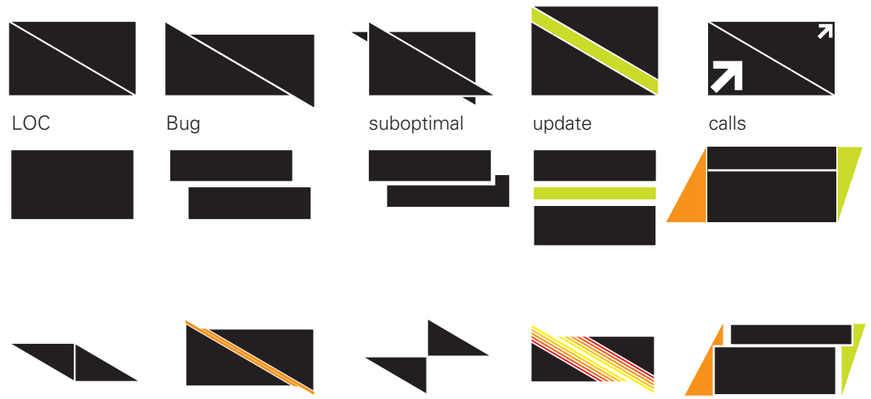
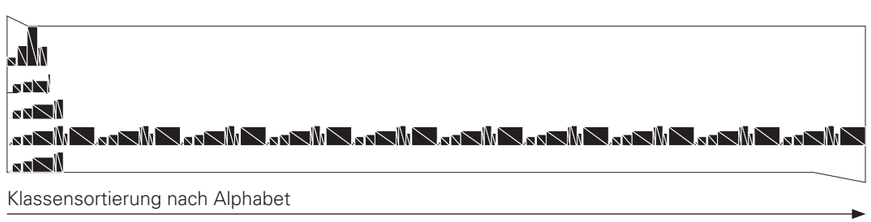


Abb. 02



Zeichenentwicklung III

Das in Phase II entwickelte Zeichensystem bringt eine grosse Anzahl an verschiedenen Zeichen, diese sind jedoch bei starkem verkleinern nicht mehr korrekt lesbar. Die Zeichen werden deshalb weiter abstrahiert um die Lesbarkeit besser zu gewährleisten, ohne jedoch auf die wesentlichen Metriken zu verzichten (Abb. 01).
Die rechteckige Zeichenfom wird beibehalten, für die Auszeichnungen vom Fehlern und calls neue Darstellungsformen gesucht (Abb. 02).

Abb. 01

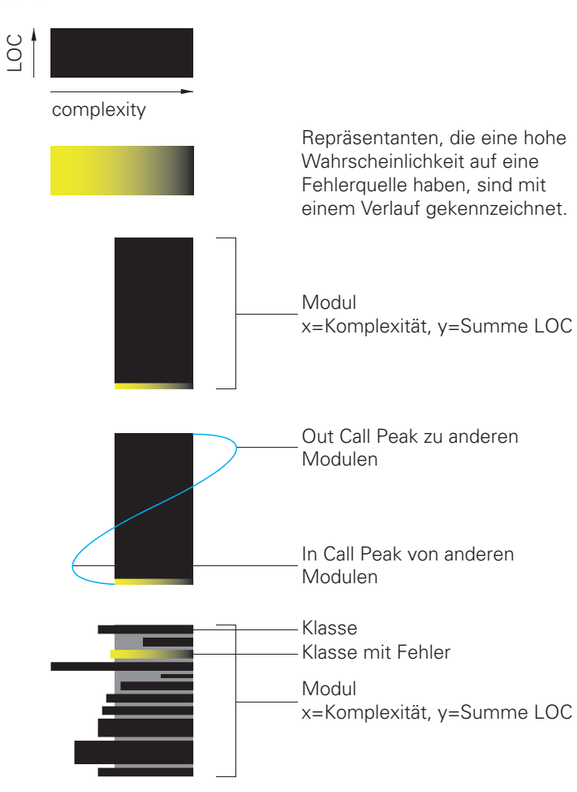
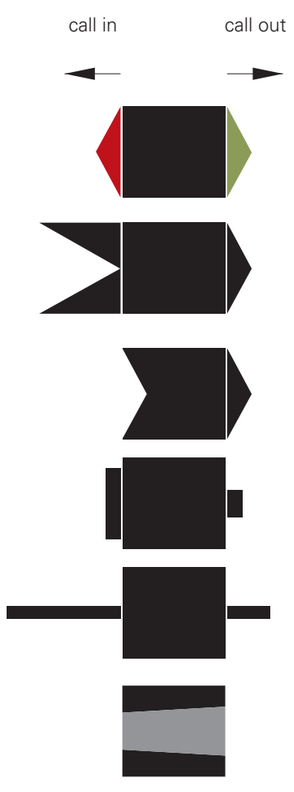


Abb. 02

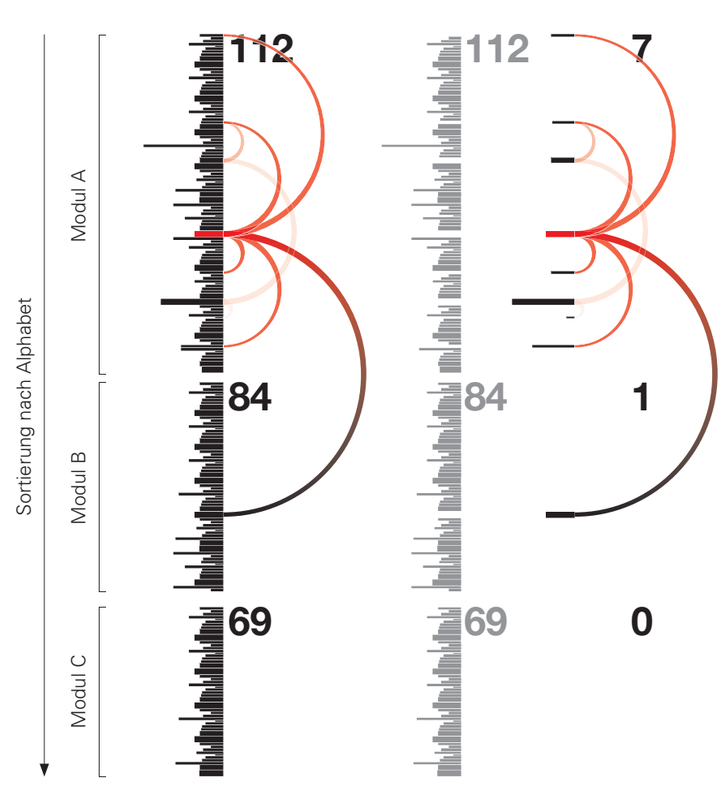


Modellentwurf D

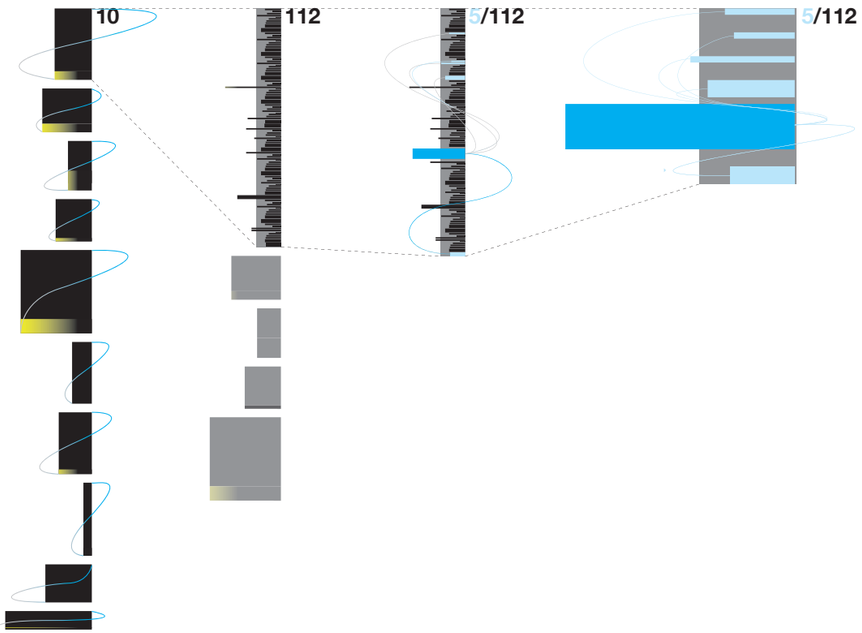
Gemäss den Grundsätzen der Zeichenentwicklung III wird ein entsprechendes Modell entwickelt. Die einzelnen Zeichen repräsentieren Klassen, einzelne Teilbereiche Module. Dank der linearen Anordnung (Abb. 01) können die Repräsentanten besser verglichen werden, die Kommunikation lässt sich einfach durch Linien visualisieren.
Als Inspiration dient das Modell für die New York Times von Ben Fry http://benfry.com/linking
Die Stärke der Linie kennzeichnet die Frequenz der einzelnen calls. Verbindungen zweiter und dritter Instanz werden transparent dargestellt. Dadurch lassen sich Abhängigkeiten und Aufrufe anderer Klassen aufzeichnen.
Durch ein RollOver separieren sich die verknüpften Klassen, die nicht tangierten Klassen werden transparent und treten in den Hintergrund (Abb. 02).

Abb. 01

Abb. 02

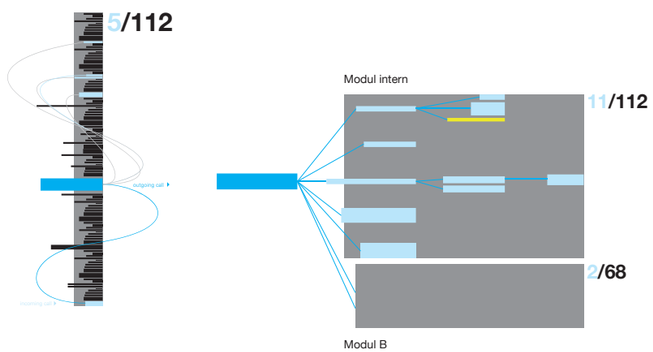


Modellentwurf E



Modulübersicht Modulinhalt RollOver Klasse Klassenseparation

Darstellung linear Darstellung hierarchisch



Hauptmodell

Bei Modell E ist es schwierig, die Aussage der calls genau miteinander zu vergleichen, da der Ursprung der calls je nach Komplexität der Klassen unterschiedlich ausfällt.
Die Repräsentanten müssen deshalb alle gleich hoch sein, das heisst, die y Metrik (Komplexität) fällt weg und wird durch die Farbigkeit des Repräsentanten dargestellt.
Die Farbigkeit zeigt sich in vier Abstufungen eines Farbwerts. Je intensiver der Farbwert, desto komplexer das Modul.
Um den Informationsgehalt über die calls zu erweitern, werden die Verbindungslinien (call lines) mit einer zusätzlichen Metrik versehen. Eine Farbigkeit gibt an, ob es sich um einen incoming call (Rot) oder einen outgoing call (cyan) handelt. Der Ausschlag der Linienkurve visualisiert die Frequenz, mit der ein Repräsentant ein anderer aufruft, resp. aufgerufen wird. Je höher die Frequenz, desto grösser der Ausschlag.
Um die Summe der calls darzustellen, werden am Zeichen beidseitig Ausschlagbalken angehängt. Dadurch ist auf einen Blick ersichtlich, welche Zeichen viele calls generieren und welche viele calls erhalten auch dann, wenn die call lines zur Übersichtlichkeit ausgeblendet werden.

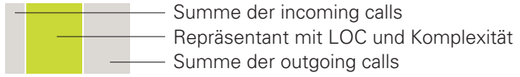
level of complexity

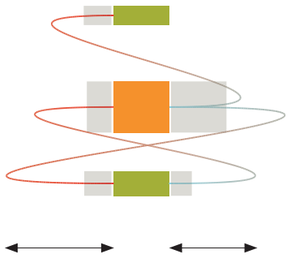
probability of bugs

aktives Modul

calls

48 Anzahl Repräsentanten





verknüpftes Modul

aktives Modul

verknüpftes Modul

Ausschlag der calls

Modellinteraktion

Abb. 01
Modell Ausganglage

Abb. 02
Rollover Modul: Anzeige detaillierte Textinformationen und Kommunikation zu anderen Modulen

Abb. 03
Click Modul: Separation der direkt Verknüpften Module. Anordnung nach Anzahl der Calls.

Abb. 04
Click separiertes Modul: Anzeige der enthaltenen Klassen.

Abb. 05
Click in Modul enthaltene Klasse: Anzeige der mit dieser Klasse verknüpften Klassen, Anordnung nach Anzahl Calls.

Abb. 06
Rollover Klasse: Hervorheben des Modul welches die Klasse beinhaltet und der Klassen welche sich ebenfalls in diesem Modul befinden.

Abb. 01

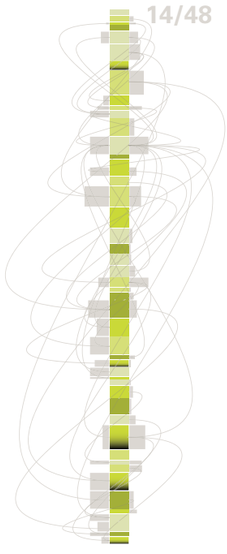


Abb. 02

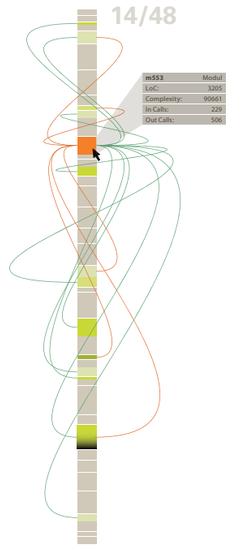


Abb. 03

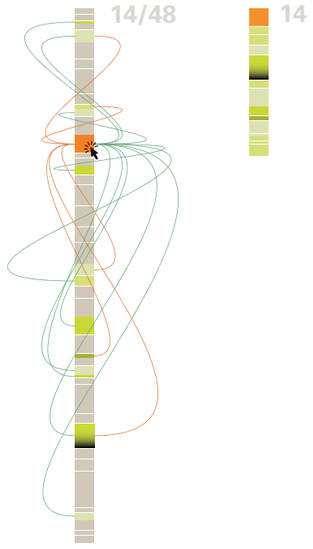


Abb. 04

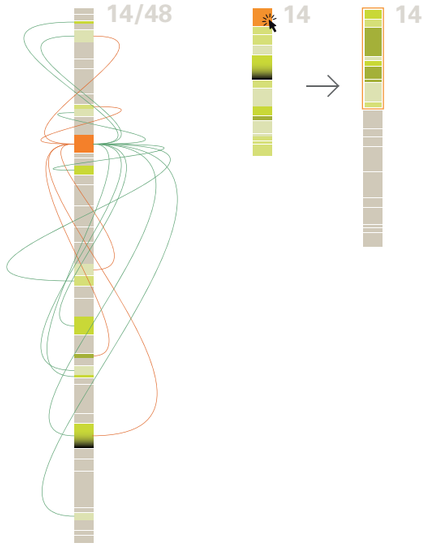


Abb. 05

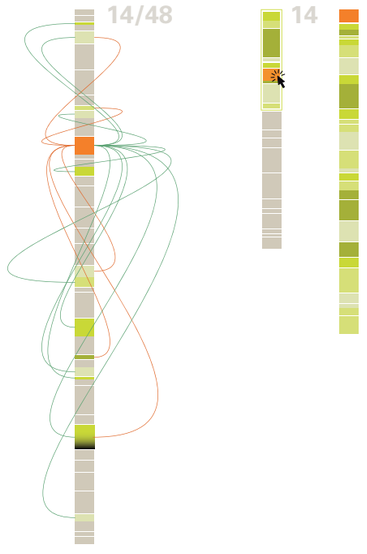
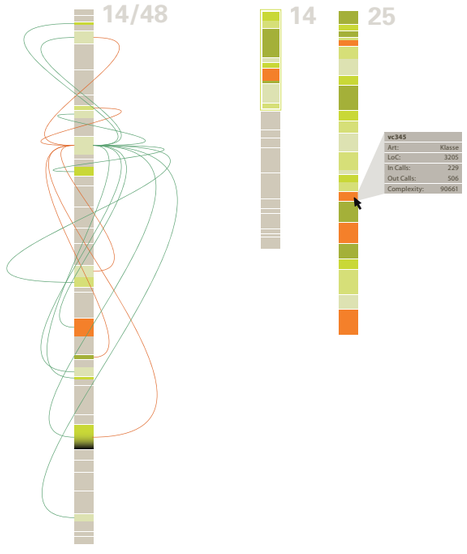


Abb. 06

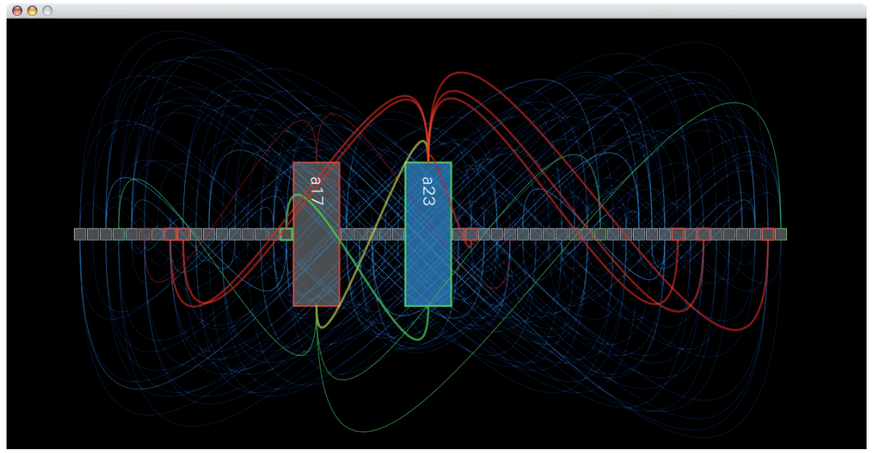


Umsetzung

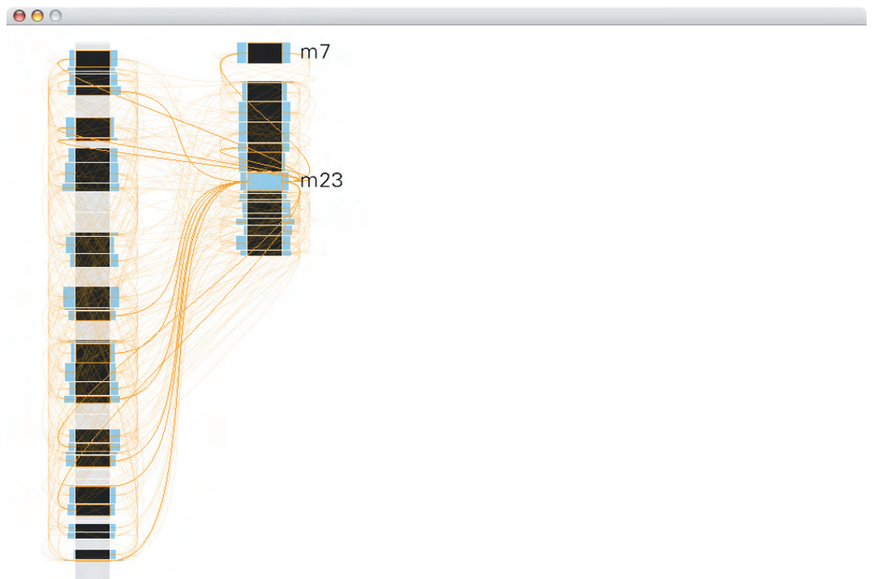
Processing:

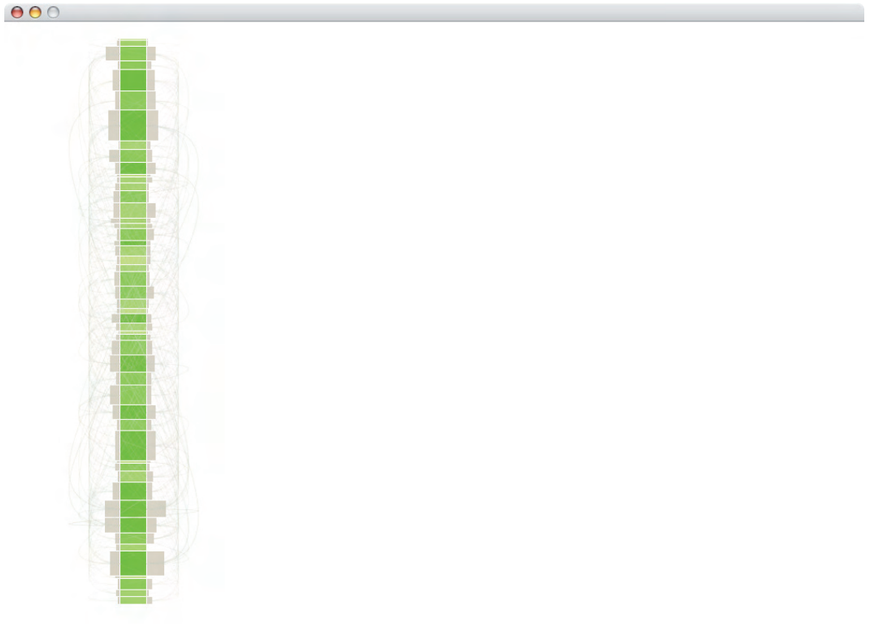
Der Entwurfsprozess und die Umsetzung des interaktiven Prototyps wurde mit Processing realisiert. Processing ist eine Open-Source-Programmiersprache und Entwicklungsumgebung für die Programmierung von Bild, Animation und Sound. Es basiert auf Java, was unter anderem den Vorteil hat, dass für verschiedene Betriebssysteme Applikationen und Java-Applets für das Internet exportiert werden können. Ferner ermöglicht Processing über diverse Libraries auch das Verarbeiten von Realdaten. Dies ist im aktuellen Stadium noch nicht implementiert, wäre aber in einer allfälligen Weiterentwicklung realisierbar.

http://www.processing.org



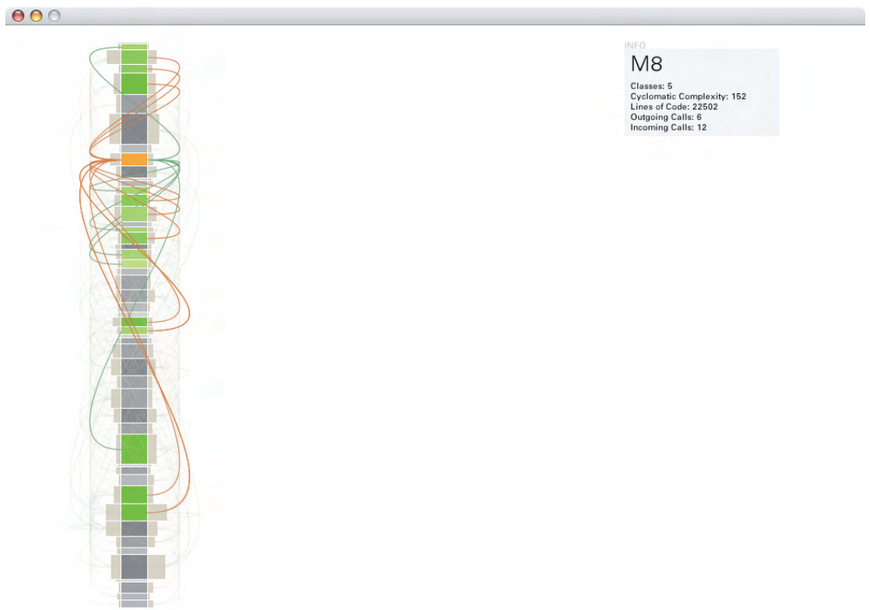
Frühe Prototypen

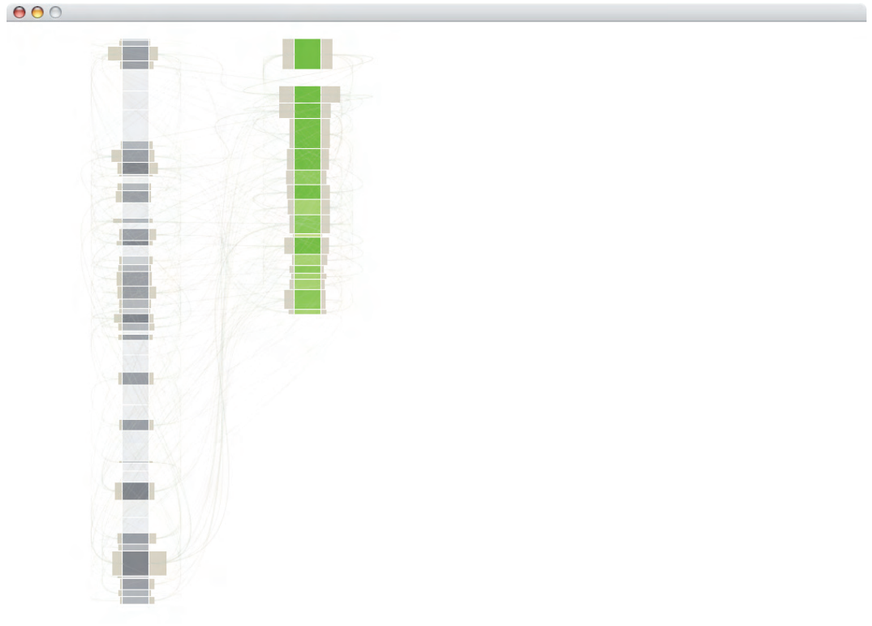




Gesamtsystem

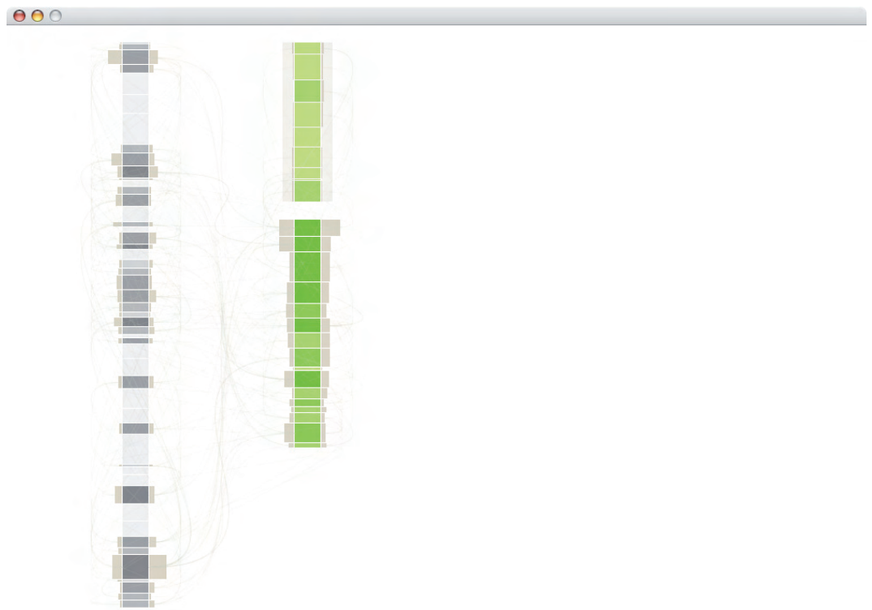
Rollover: Anzeige der calls vom und zum aktiven Modul, der verbundenen Module, Info des Moduls.

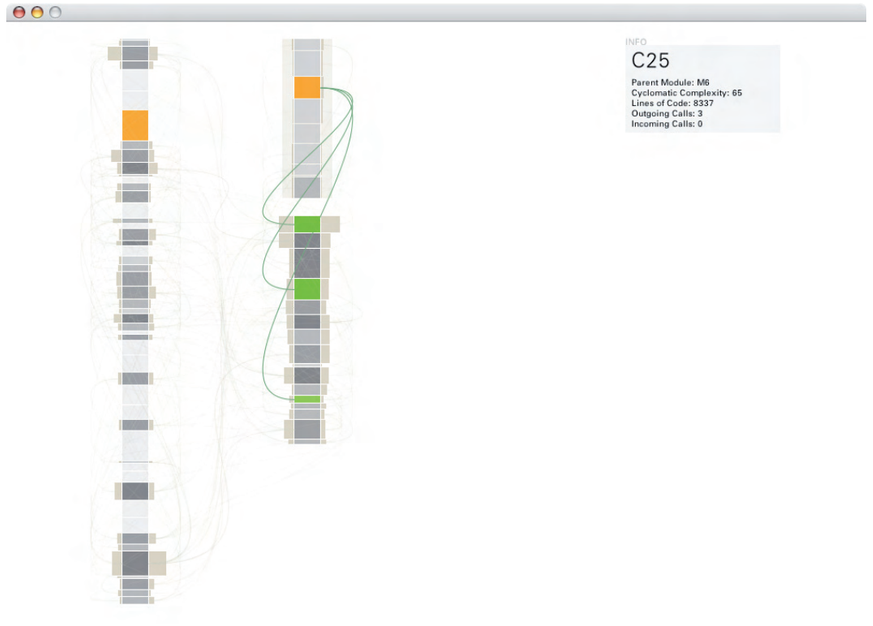




Auswahl eines Moduls: Modul und seine verbundenen Module (sortiert nach Verbindungsstärke) kommen auf eine neue Ebene.

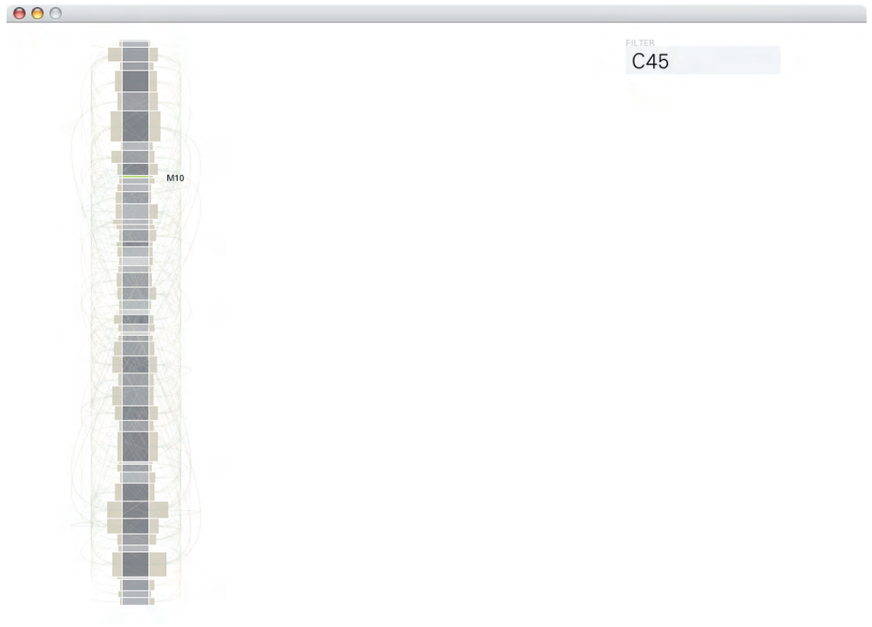
Aufklappen des aktiven Moduls: Klassen des Moduls werden angezeigt.





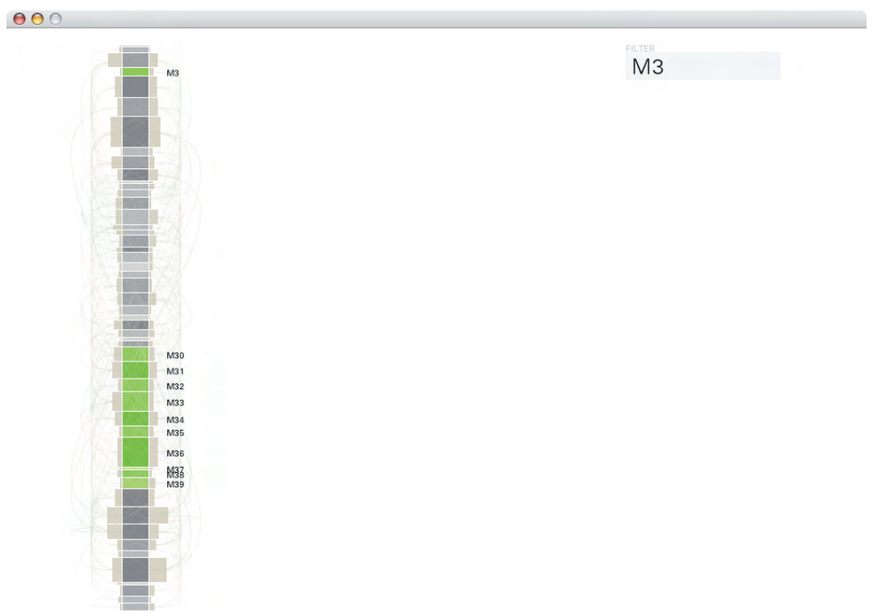
Rollover bei der Klasse analog Modul.

Auswahl einer Klasse: Klasse und seine verbundenen Klassen (sortiert nach Verbindungsstärke) kommen auf eine neue Ebene.



Bei der Suche nach Klassenname werden übergeordnete Module angezeigt.

Filtern nach Modulname.



Konklusion

Software wird immer komplexer. Lösungen, um Softwaresysteme sichtbar und erfahrbar zu machen sind noch rar. Bisherige Ansätze beschränken sich weitestgehend darauf eine statische Struktur eines Systems aufzuzeigen. Sie sind dem besseren Verständnis nicht unbedingt zuträglich, da sie lediglich die Komplexität eines solchen Systems aufzeigen. Zusammenhänge lassen sich höchstens erahnen.

Bei den Datenmengen und der Komplexität einer Softwarearchitektur reicht die statische Darstellung in Form einer Übersicht nicht mehr aus. Eine Visualisierung muss dynamisch sein, um Veränderungen in dem System abbilden zu können. Sie muss interaktiv sein und den Betrachter zum Benutzer machen. Er muss für ihn irrelevante Sachverhalte rausfiltern können und zielgerichtet auf wichtige Daten und Zusammenhänge schnell zugreifen können, ohne den Kontext des Gesamtsystems zu verlieren.

Bein unserer fünfwöchigen Arbeit ist ein Prototyp entstanden, der auf diesen Prinzipien aufbaut. Der Benutzer navigiert zielgerichtet durch verschiedene Ebenen des Systems und hat dabei immer Zusammenhänge, den Gesamtkontext und vorhergegangene Schritte im Überblick. Relevante Informationen sind immer schnell zugänglich und einfach erfassbar.

In einer nächsten Phase müsste der Prototyp mit verschiedenen Datensätzen und mit einer Gruppe von Anwendern getestet werden. Diverse Aspekte von Softwaresystemen, die bis jetzt noch nicht berücksichtigt wurden (z.B. Versionierung) müssten integriert und implementiert werden.

Quellenangaben

http://en.wiktionary.org/wiki/
http://www.levitated.net/
http://www.benfry.com/disarticulate/
http://www.ilog.com/products/jviews/demos/index.cfm
http://graphexploration.cond.org/look.html#screenshots.html
http://ivtk.sourceforge.net/
http://www.dcc.uchile.cl/~rbaeza/cursos/visual/bl/index.html
http://www.processing.org

Projektblog

http://iad.projects.hgkz.ch/infovis07/

Software Visualization II

zhdk iad
Semester 5
Jahr WS 2007
Modul Applied Interaction I
Dozent Prof. Jürgen Späth, Prof. Dr. Harald Gall, Dipl. Ing. Beat Fluri, Sandro Boccuzzo
Gruppe Christoph Schmid, Jeremy Stucki, Ramun Liesch, Reto Stalder

Inhalt

Aufgabenstellung

In Zusammenarbeit mit Informatikstudenten der Universität Zürich werden Visualisierungsansätze aus dem Kurs «Dynamic Data» evaluiert und mit Realdaten realisiert.
Die Erarbeitung von Lösungen in der Gruppe steht im Vordergrund.

«The purpose of computing is insight, not numbers»

Richard Hamming (1962)

Definition

Softwarevisualisierung

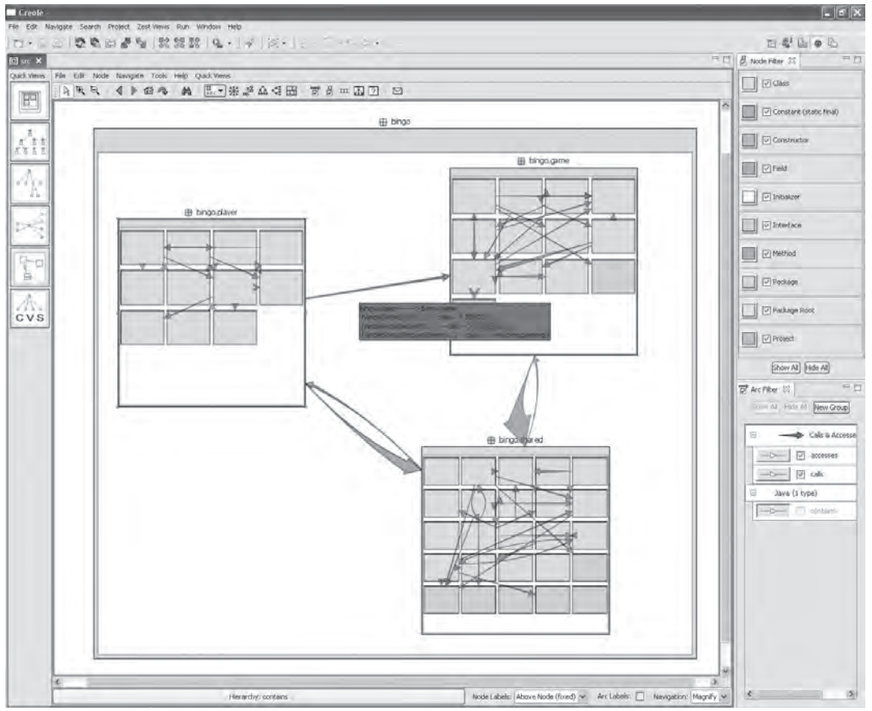
Softwarevisualisierung beschäftigt sich mit der statischen oder animierten Darstellung von 2D oder 3D Informationen über Softwaresysteme.
Die Informationen können dabei die Struktur, die Grösse, den historischer Verlauf oder das dynamische Verhalten des Systems beschreiben. Typischerweise werden dabei Informationen mittels Softwaremetriken zur Visualisierung verwendet. Visualisierung ist von Natur aus keine Methode zur Software-Qualitätssicherung kann aber dazu verwendet werden manuell Anomalien (z. B. Zyklen) aufzuspüren oder erkannte Defekte mit in die Visualisierung zu integrieren. Dieser Prozess wird auch visuelles Data Mining genannt. Die Ziele der Softwarevisualisierung beinhalten das Verstehen von Softwaresystemen (z. B. Aufbau und Morphologie) und Algorithmen (z. B. die Animation von Suchalgorithmen) sowie der Analyse von Softwaresystemen zur Entdeckung von Anomalien (z. B. durch Darstellung von Klassen mit (zu) hoher Kopplung).

Quelle: http://de.wikipedia.org, Stand 17. Oktober 2007

Recherche

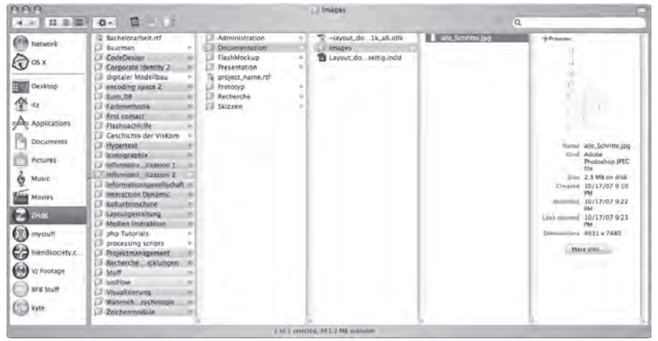
Creole

Die Applikation Creole versucht in ähnlicher Art und Weise, Software und deren komplexen Vorgänge zu visualisieren. Dabei kann von der obersten Packageansicht bis zur Initialisierung einer Variable praktisch jeder Teil betrachtet und direkt im Quellcode modifiziert werden. Durch die verschiedenen Darstellungsformen können relativ viele Parameter in Kombinationen aufgezeigt werden.
Die Umsetzung ist jedoch äusserst komplex und unübersichtlich geraten, sodass das Auslesen einzelner Parameter fast unmöglich wird.



Apple Finder

Der Finder ist ein zentraler Bestandteil der grafischen Benutzeroberfläche des Mac OS-Betriebssystems und stellt die Schnittstelle zwischen Benutzer und System dar.
Finder bietet eine grafische Sicht des Dateisystems und der genutzten Geräte.
Der Finder ist für den Aspekt der Skalierbarkeit sowie der Datenorganisation ein spannendes Vergleichsobjekt.
Ähnliche Applikation sind Path Finder, Macintosh Explorer oder Windows Explorer.



Analyse

Potential und Defizite des bestehenden Prototypen

Im Gespräch mit den Informatikstudenten hat sich relativ schnell die Brauchbarkeit des bestehenden Prototypen gezeigt, jedoch auch einige fehlende Aspekte, die während der Entwicklung im letzten Semester nicht berücksichtigt wurden.
Die vier wichtigsten Punkte wurden in der Analyse definiert und als Konzeptgrundlage für die Weiterentwicklung festgelegt.

A Skalierbarkeit / Packagestruktur
B Farbkonzept
C Vererbungsansicht
D Generalisierung weiterer Metriken

Potentielle Elemente aus der ersten Phase wurden in den Überarbeitungsprozess miteinbezogen und an die neuen Konzeptpunkte angepasst.

Konzeption

Konzept der Skalierbarkeit

Ein wesentlicher Kritikpunkt des bestehenden Prototypen war die limitierte Darstellung von nur drei Hierarchiestufen. Die Anbindung an eine reale Datenbank hat jedoch gezeigt, das eine Verschachtelungstiefe von zehn oder mehr Stufen durchaus realistisch ist. Dadurch ergeben sich drei Probleme:

1. Das gesamte Modell muss skalierbar sein
2. Durch die Verschachtelung muss die Objekttypisierung und deren Ausprägung geregelt werden
3. Das Farbkonzept muss entsprechend angepasst werden

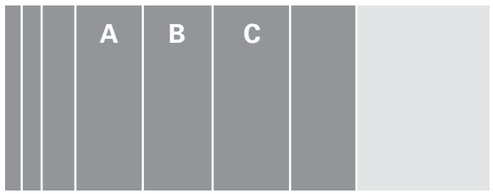
Als Lösungsansatz sehen wir eine horizontale Skalierung der Objekte. Da bisher der X-Wert keiner Metrik zugewiesen ist, kann die Breite theoretisch beliebig verändert werden, ohne die restlichen Information zu beeinflussen. Einen ähnlichen Ansatz verfolgt das Dock im Apple`s Betriebssystem OSX. Das durch den Benutzer ausgewählte Programmicon skaliert sich in der Grösse. Wichtige Elemente werden also durch Animation vergrössert und so zum Fokus.

OSX Dock mit Fokusierung der ausgewählten Programmicons

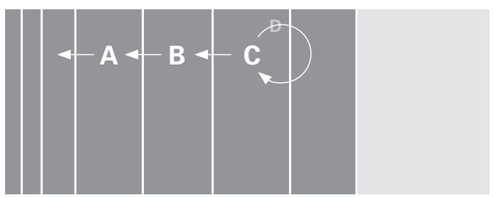
Konzept der Mitte

Im Zusammenhang mit der Skalierung stellt sich die Frage nach der Darstellung des aktuellen Bereichs. Dieser wird immer anderselben Stelle dargestellt. Dies verhindert verwirrende Sprünge mit dem Auge und erlaubt eine effiziente Navigation durch das System.
Der Inhalt eines Packages wird also an der Stelle visualisiert, wo das aktuelle Package positioniert ist. Das übergeordnete Element verschiebt sich um eine Stufe nach links. Um so weiter weg vom Fokus, desto grösser die Skalierung der Elemente.

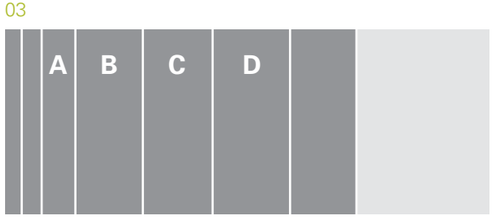
01



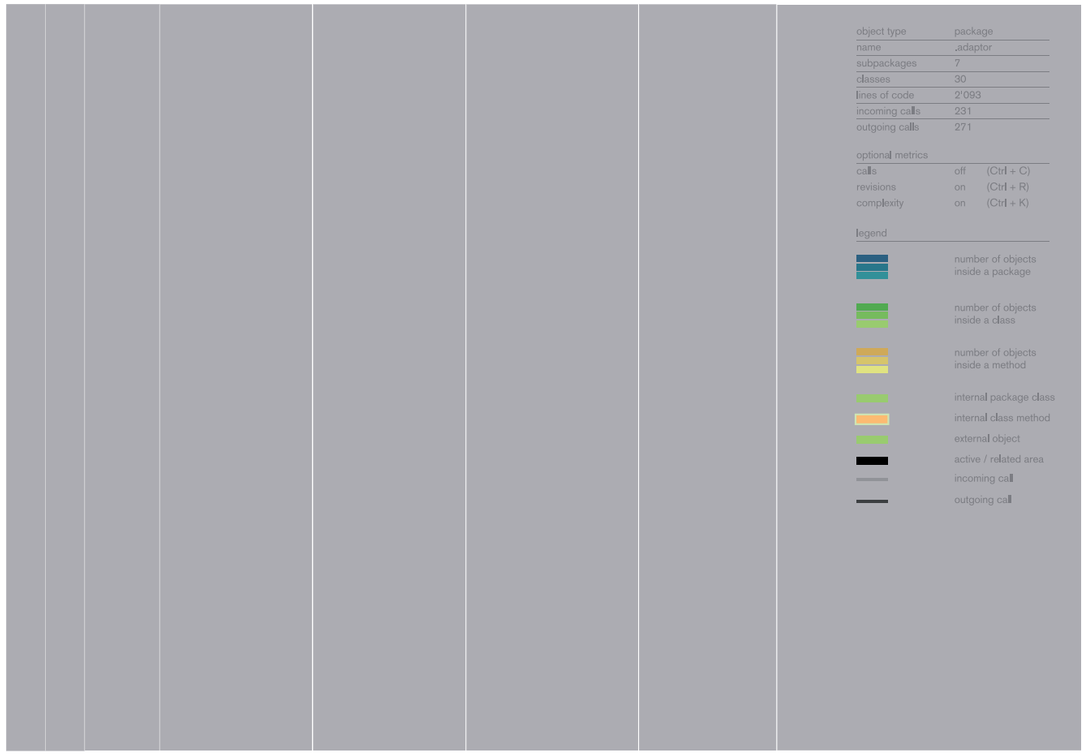
02



03



Darstellung des Skalierungsrasters



Konzept der Farbe

Helle Objekte werden im Vergleich zu dunklen Objekten als grösser empfunden.
Dunkle Objekte wirken wie eine Verdichtung oder Überlagerung von helleren Objekten. Sie scheinen mit Inhalt gefüllt zu sein.

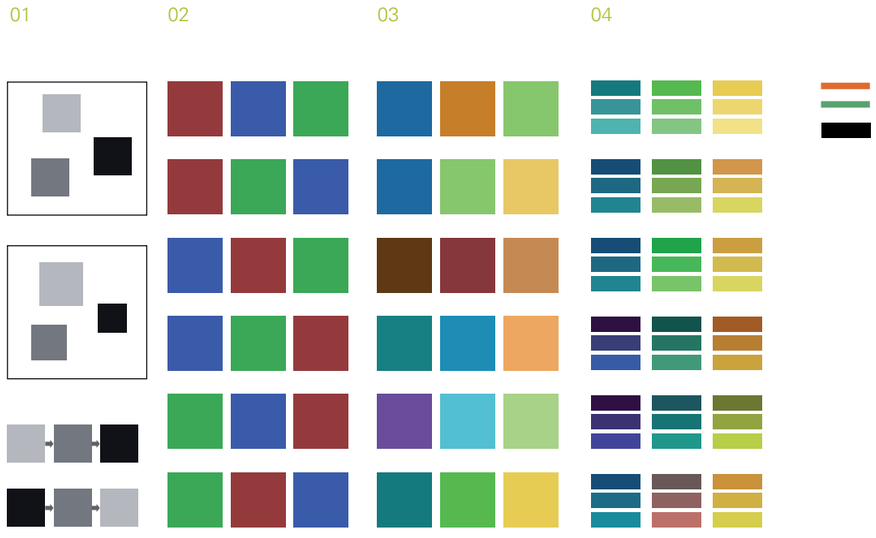
Hierarchiestufen dunkel zu hell. Als schlecht funktionierendes Gegenbeispiel der Hierarchieverlauf von hell zu dunkel.

Gleicher Versuch wie (2), aber mit Farben gleicher Helligkeit und gleicher Sättigung.

Weiterführend zu (3) der Versuch, die drei Hierarchiestufen mit diversen Farben gleicher Helligkeit und gleicher Sättigung darzustellen.

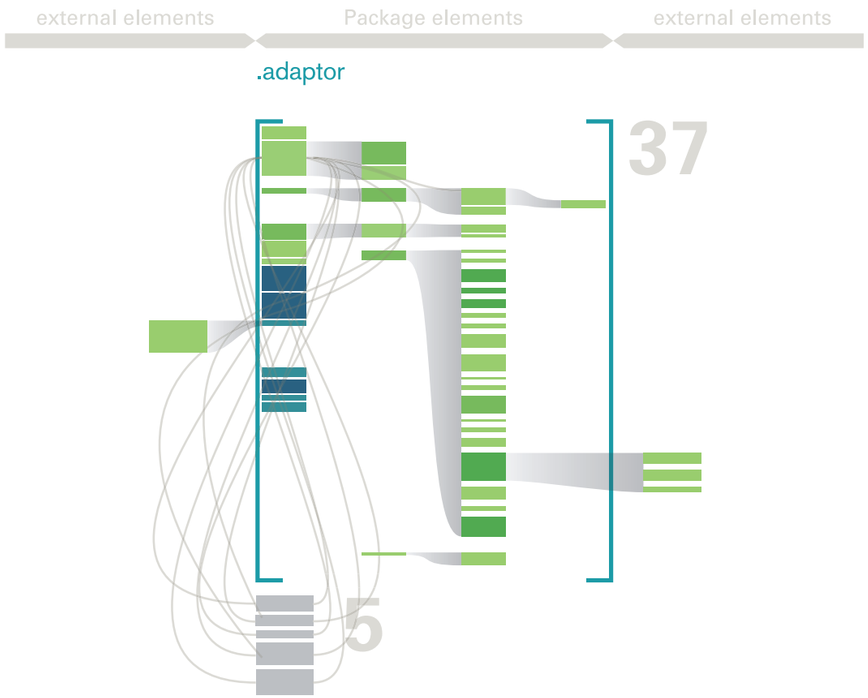
Erarbeitung des endgültigen Farbkonzepts (4). Das Farbkonzept soll auf diversen Screens funktionieren, weshalb kräftigere Farben und stärkere Kontraste nötig sind, gleichzeitig soll eine gewisse Gesamtharmonie spürbar sein.

Das endgültige Farbkonzept beinhaltet 13 Farben:
Package mit jeweils 3 Abstufungen, welche die Inhaltsmenge visualisieren.
Klasse mit jeweils 3 Abstufungen, Methode mit jeweils 3 Abstufungen, Aktives Element, Inaktives Element, Incoming Calls, Outgoing Calls, Hintergrund, Schrift.



Konzept der Vererbung

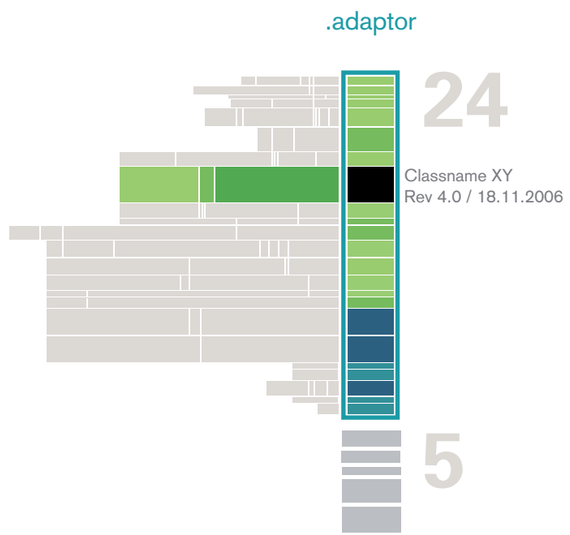
Ein einfacher Ansichtswechsel erlaubt die zentrale Metrik der Vererbung darzustellen. Die Vererbung wird in der aktuellen Ansicht dargestellt, ohne die vorhergehende Ansicht zu verlieren. Alle Metriken sind immer noch sichtbar und werden mit der Metrik der Vererbung in Bezug gesetzt.



Generalisierung weiterer Metriken

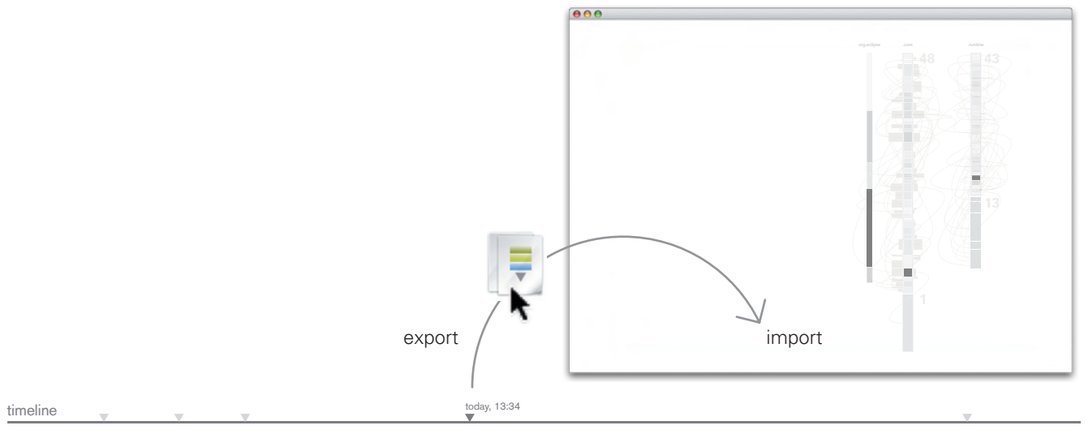
Eine integrierte Ansicht erlaubt es, weitere, nicht zentrale Metriken darzustellen. Diese werden in einem zweidimensionalen Balkendiagramm visualisiert, welche einen leichten Vergleich der Werte erlaubt. Diese allgemeine Darstellungsart erlaubt eine beliebige Anzahl weiterer Metriken anzuzeigen.

Alter und Revisionen
Der Balken wird zur Zeitachse. Die Unterteilungen stellen die Revisionen dar, welche zusätzlich für jede Revision die lines of code anzeigt. Somit lässt sich durch eine visuell leicht fassbare Darstellung die ganze Verlaufsgeschichte eines Objektes zeigen.



Konzept des Informationsaustausches

Der User kann bestimmte angezeigte Konstellationen im Softwaresystem speichern. Diese werden auf einer Zeitleiste archiviert und können jederzeit wieder aufgerufen werden. Zusätzlich können diese Snapshots exportiert werden und ein anderer User kann dieselben in der eigenen Applikation darstellen. Die Zeitleiste bietet einen hilfreichen Überblick über vergangene Ansichten.



Imformationsbereich

Im ersten Block werden genauere Informationen zum momentan ausgewählten Element dargestellt. Die numerischen Werte sind eine essenzieller Mehrwert und ergänzen die visuellen Parameter des Modells.

object type	package
name	.adaptor
subpackages	7
classes	30
lines of code	2'093
incoming calls	231
outgoing calls	271

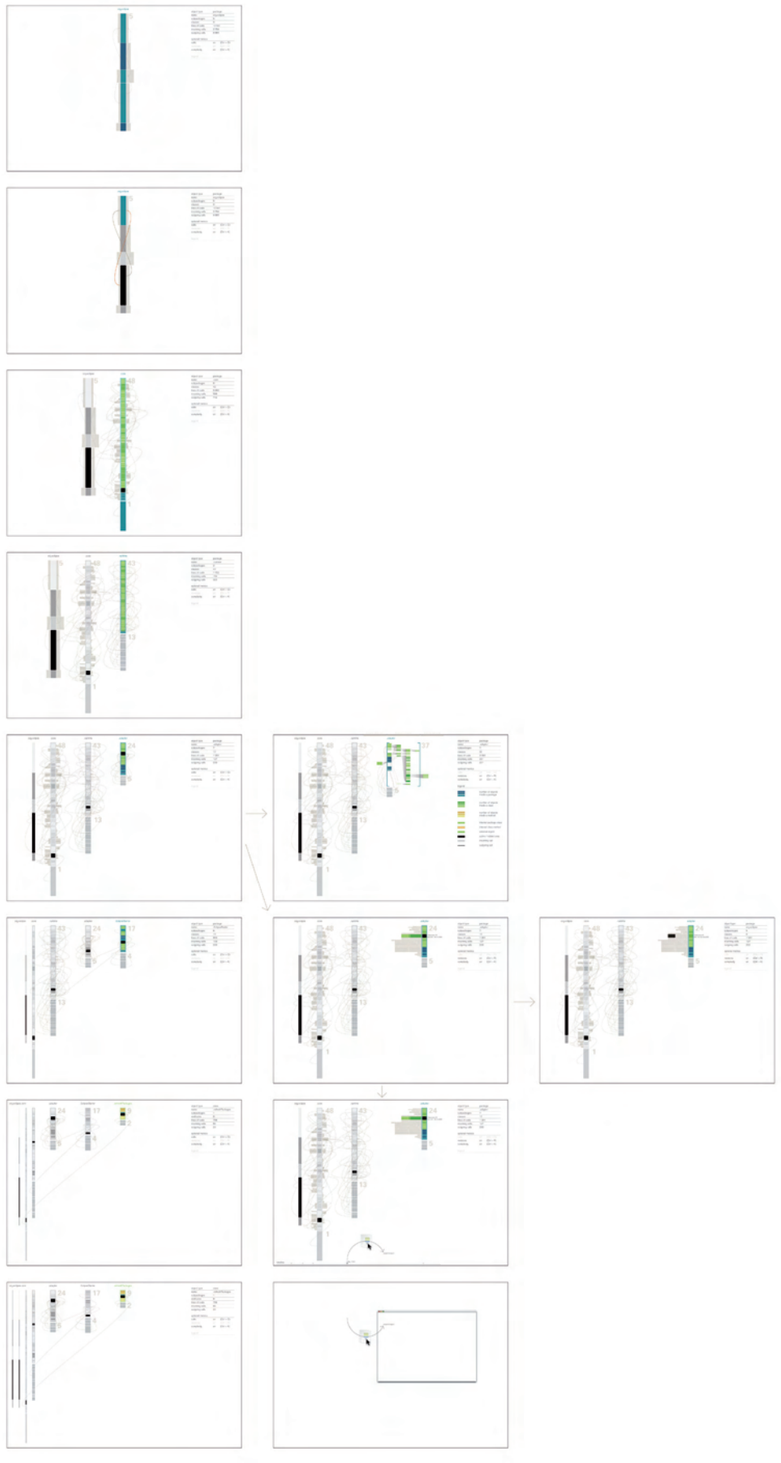
Im zweiten Block können optionale Parameter ein- und ausgeschaltet werden. Als Defaultwert sind die calls aktiv, revisions und complexity inaktiv.

optional metrics		
calls	off	(Ctrl + C)
revisions	on	(Ctrl + R)
complexity	on	(Ctrl + K)

Im dritten Block die Erklärung des Farbschlüssels.

legend

- number of objects inside a package
- number of objects inside a class
- number of objects inside a method
- internal package class
- internal class method
- external object
- active / related area
- incoming call
- outgoing call
- 37 number of elements



Modellinteraktion

01

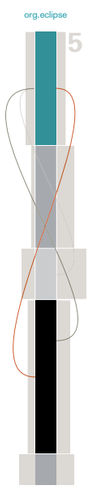
Übersicht der Packages

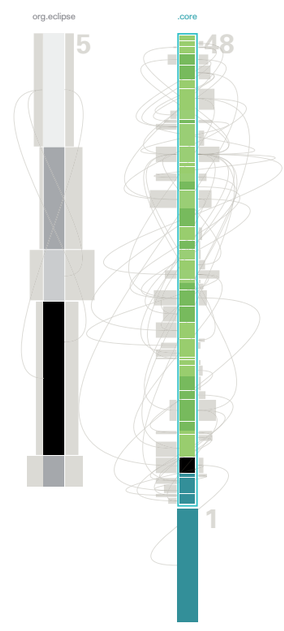
02

RollOver und Relationsdarstellung

03

Selektion und Inhaltsdarstellung. Verschiebung des Packages nach links





04

Auswahl und Darstellung
eines weiteren Packages

05

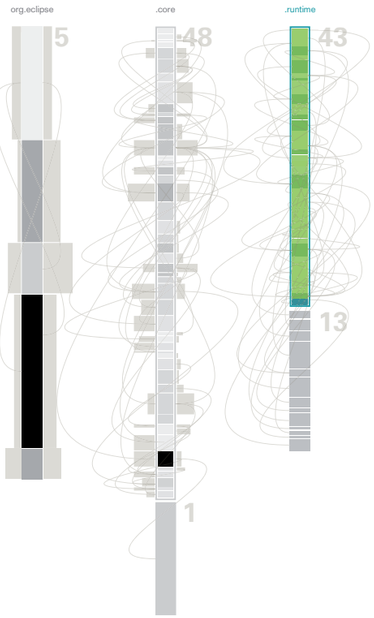
Auswahl und Darstellung eines
weiteren Packages

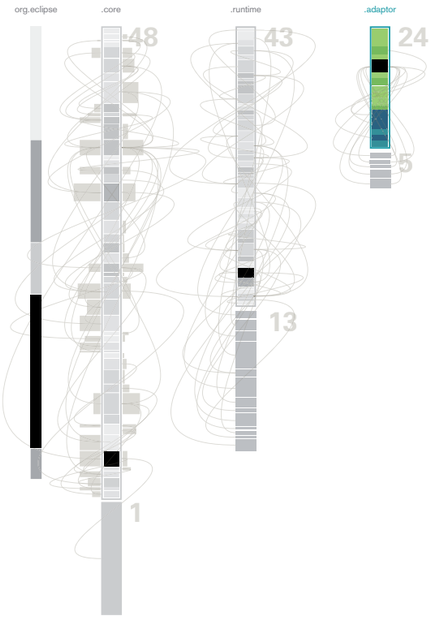
06

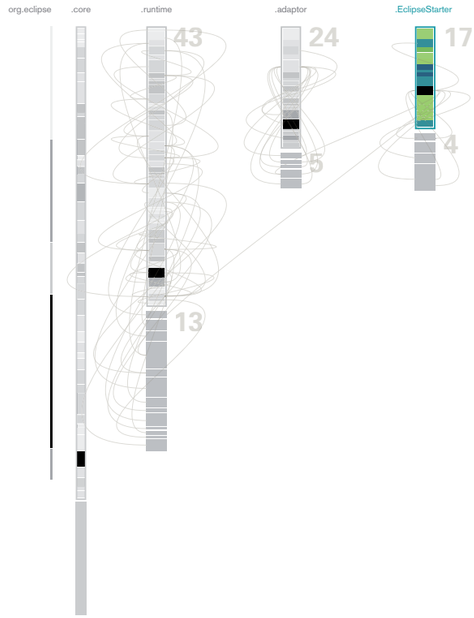
Auswahl und Darstellung
eines weiteren Packages

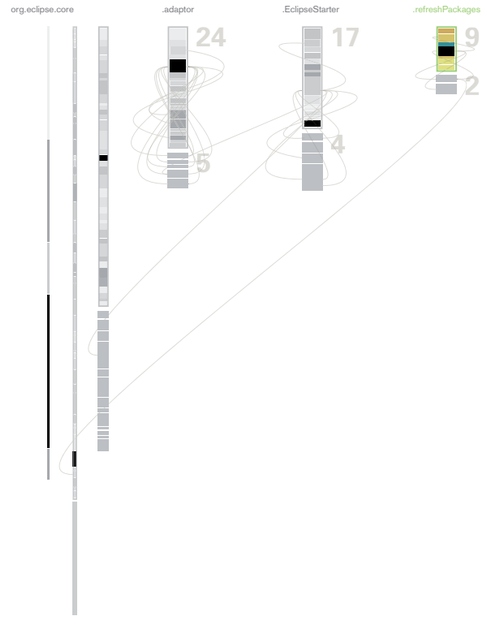
07

Auswahl und Darstellung einer
Klasse









08

Ausgangslage:
Darstellung eines Packages
in der 4. Verschachtelungs-
tiefe

09

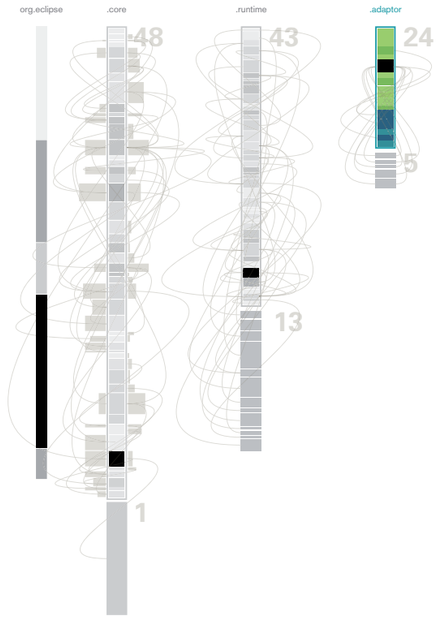
Durch Interaktion durch den
Benutzer wird für das entspre-
chende Objekt eine Verer-
bungsdarstellung generiert.

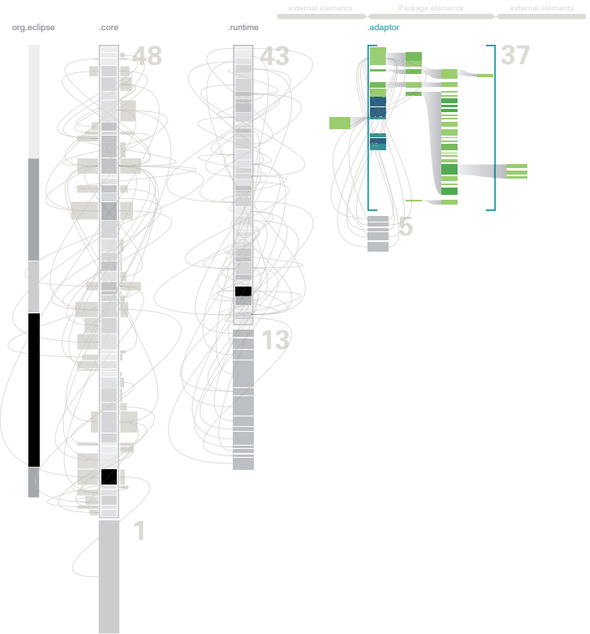
10

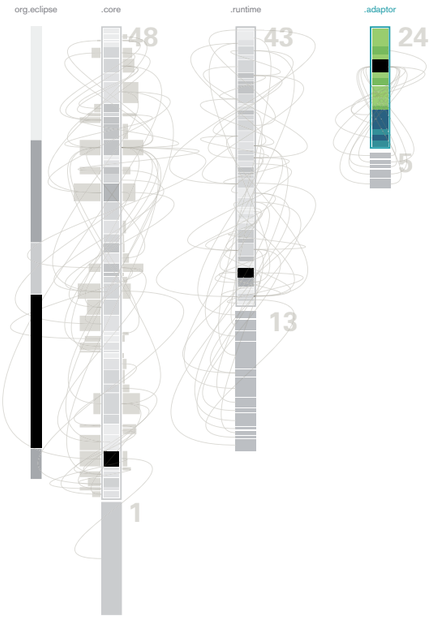
Ausgangslage:
Darstellung eines Packages
in der 4. Verschachtelungs-
tiefe

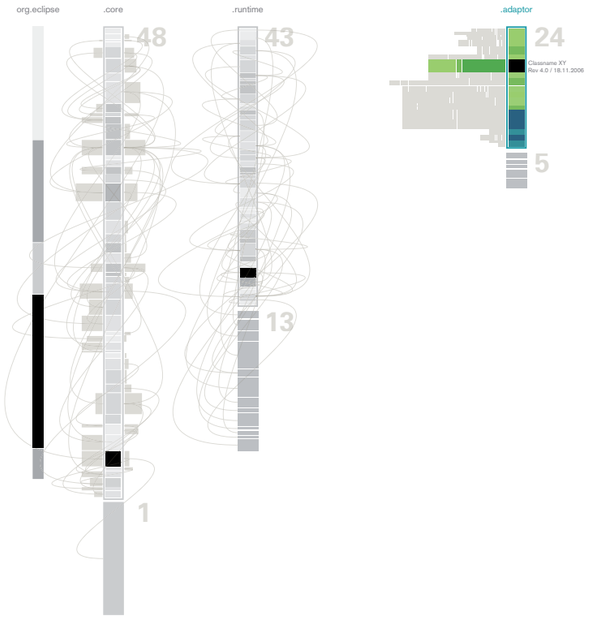
11

Durch Interaktion durch den
Benutzer wird für das entspre-
chende Objekt eine Revisions-
darstellung generiert.









12

Durch RollOver der internen Klasse können genauere Information über die Revisionen und Alter eingesehen werden.

13

Durch RollOver der Revisionen können genauere Information über deren Entwicklung eingesehen werden.

Prototyp

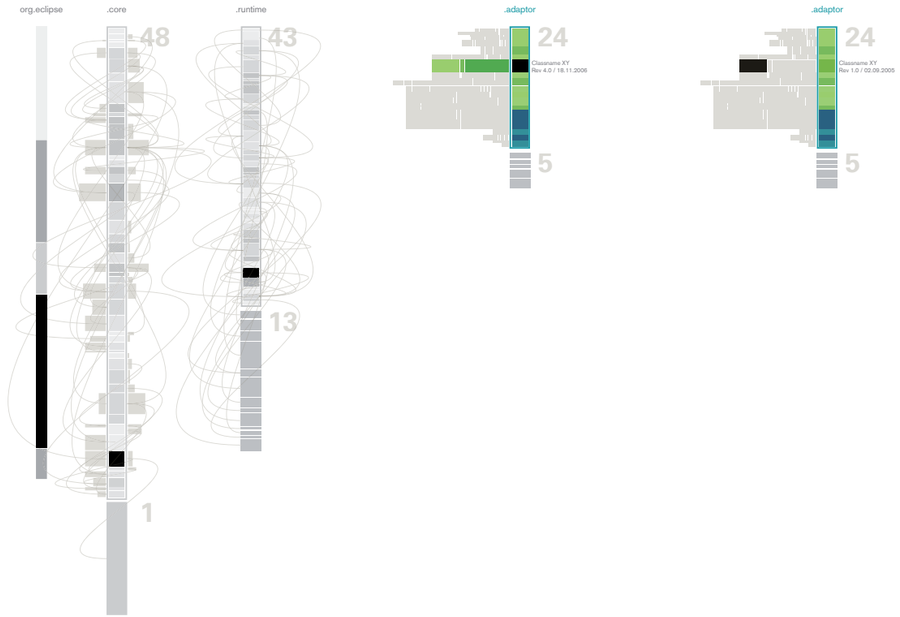
Processing

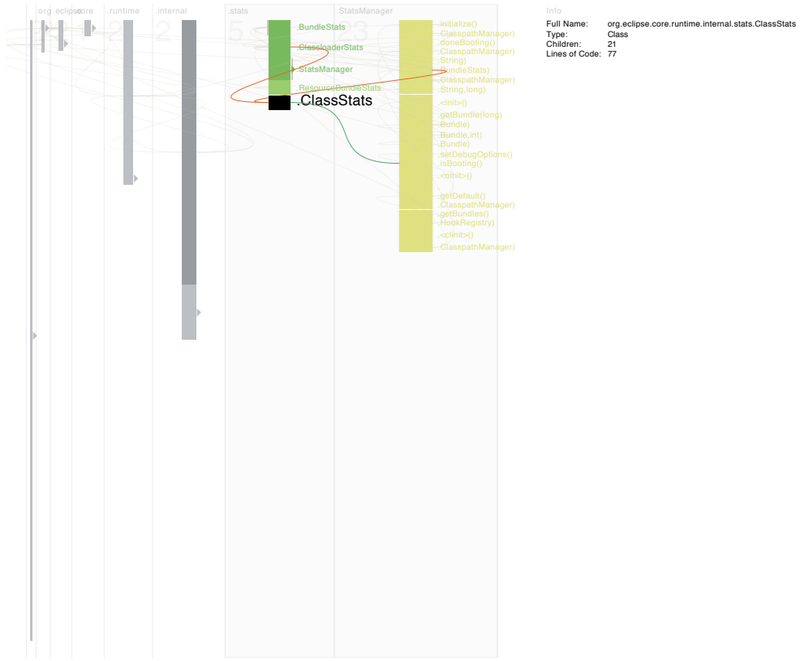
Die Umsetzung des Prototypen wurde in Processing programmiert. Auf den Seiten 27 bis 30 werden Screenshots bei der Benutzung mit der Applikation gezeigt.

Seite 27
Darstellung in der achten Hierarchiestufe. Es werden zwei verschiedene RollOver Status mit den entsprechenden Calls visualisiert.

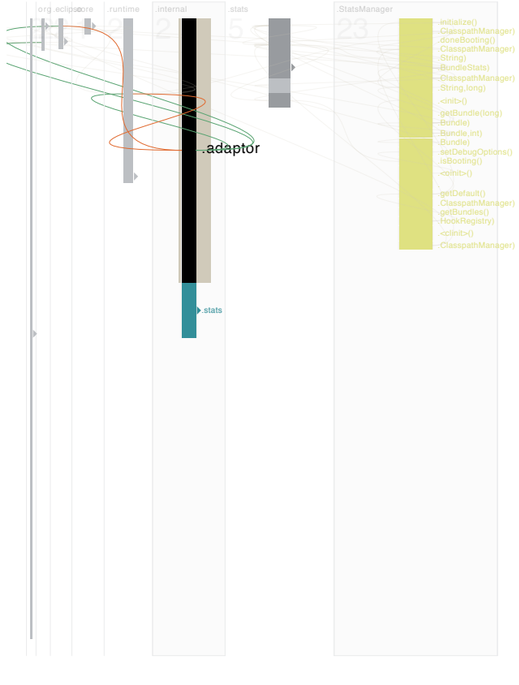
Seite 28 und 29
RollOver über die .PluginParser Klasse. Aufgerufene Elemente werden durch die Calls angezeigt.

Seite 30
RollOver über das .OSGI Package, welches von vielen Klassen aufgerufen wird.

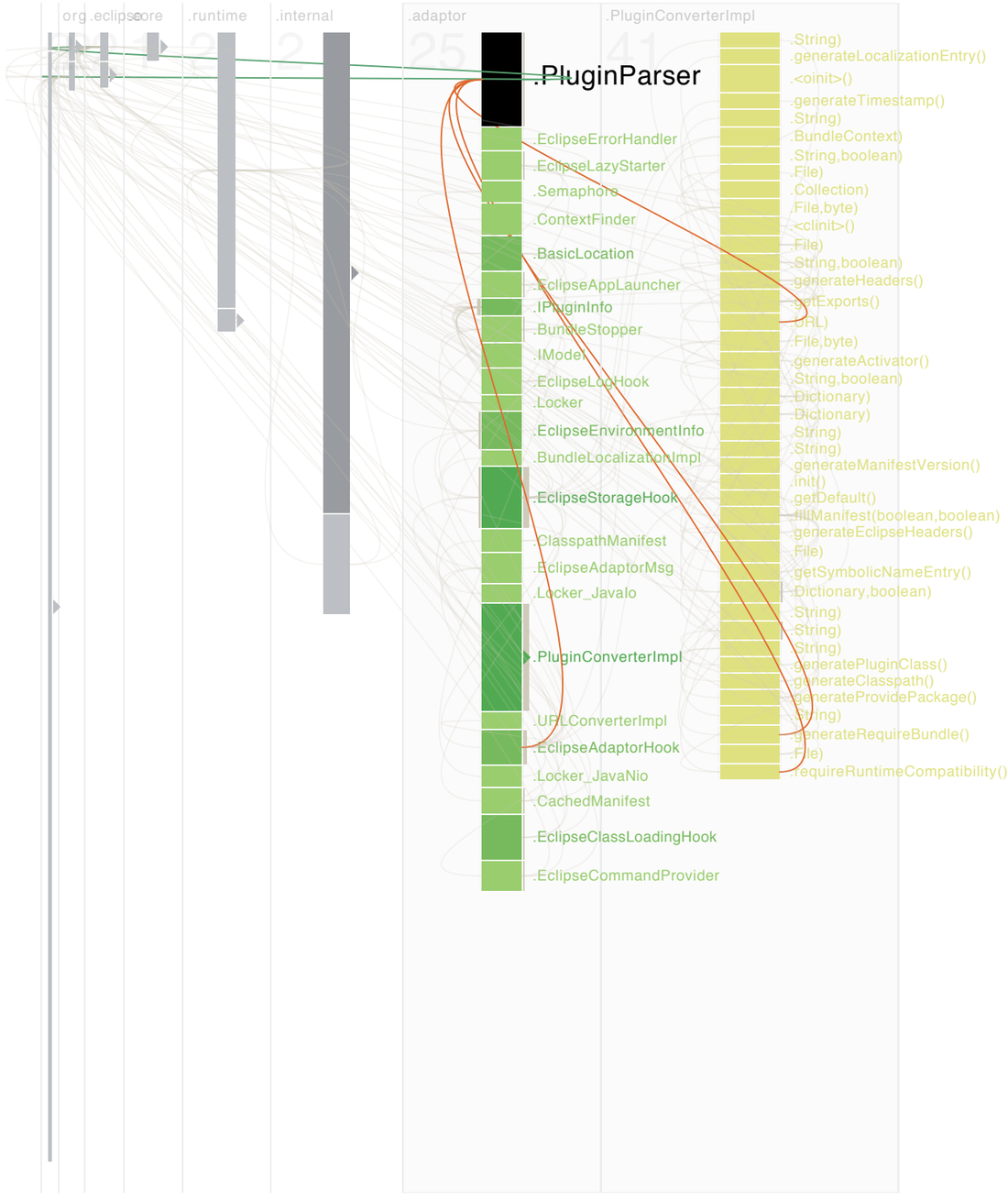




Info
Full Name: org.eclipse.core.runtime.internal.stats.ClassStats
Type: Class
Children: 21
Lines of Code: 77

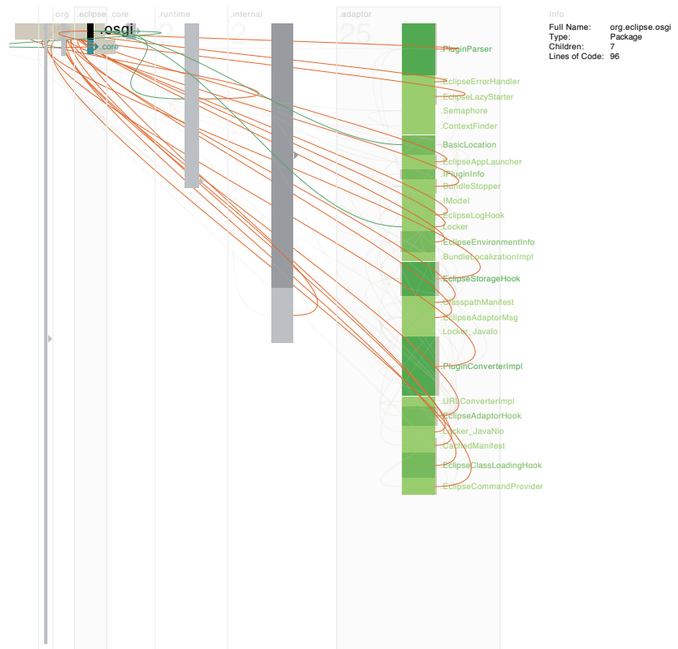


Info
Full Name: org.eclipse.core.runtime.internal.adaptor
Type: Package
Children: 25
Lines of Code: 3136



Info

Full Name: org.eclipse.core.runtime.internal.adaptor.PluginParser
Type: Class
Children: 32
Lines of Code: 538



Konklusion

Im Folgemodul «Software Visualization 2« konnte auf den bereits existierenden Prototypen und den daraus resultierenden Erfahrungen aus dem ersten Teil aufgebaut werden.
Dank der Zusammenarbeit mit Informatikstudenten der Universität Zürich ist ein sehr wichtiger Input in der Gestaltung einer Softwarevisualisierung dazugekommen. Die zum Teil komplett andere Sicht eines Entwicklers auf Problemstellungen haben zu neuen, interessanten Ansätzen in der Entwicklung unseres Prototypen geführt.
Die Anbindung an eine reale Datenbank hat unserer bisheriges Konzept in ihrer Benutzbarkeit bestätigt, sodass wir am bestehenden Konzept möglichst wenig ändern wollten. Die dazugekommenen Änderungen beziehen sich in erster Linie auf die Darstellung von beliebig vielen Verschachtelungstiefen und die damit verbundene Skalierung der Applikation.

In einer nächsten Phase wäre die Umsetzung der Snapshots interessant. Das bisher nur als Konzept existierende Feature erlaubt es, interessante Situationen innerhalb des Systems geografisch festzuhalten. Diese «Bookmarks» können mit anderen Entwicklern ausgetauscht werden, was eine Kommunikation immens erleichtert.

Quellenangaben

http://www.thechiselgroup.org/creole
http://acg.media.mit.edu/people/fry/revisionist/
http://tecfa.unige.ch/perso/yvan/PathFinder/index.htm
http://kuler.adobe.com
http://www.processing.org

Projektblog

http://iad.projects.hgkz.ch/infovis07