```
    document.write('<script src="js/dbIndexed.js">\x3C/script>');
</script>
<script src="js/location.js"></script>   <!-- Geolocation -->
<script src="js/model.js"></script>      <!-- Logik -->
<script src="js/controller.js"></script> <!-- Events -->
<script src="js/view.js"></script>       <!-- View -->
<script src="js/main.js"></script>       <!-- Main -->
```

Anhand dieses Beispiels haben wir deutlich gesehen, dass eine gute Architektur das Lösen von Problemen vereinfacht.

6.2 Die eigene Location ermitteln

Im HTML5-Standard wurde die JavaScript-API um Geolocation erweitert. Dazu dient das Objekt `navigator.geolocation`. Geolocation berechnet die Position aufgrund von GPS-Daten, sofern ein GPS-Gerät zur Verfügung steht, oder etwas ungenauer mithilfe von WiFi- und IP-Auswertungen. Viele Notebooks besitzen heutzutage ebenfalls einen GPS-Chip. Die Ortung muss in der Regel im Browser oder Betriebssystem erlaubt werden.

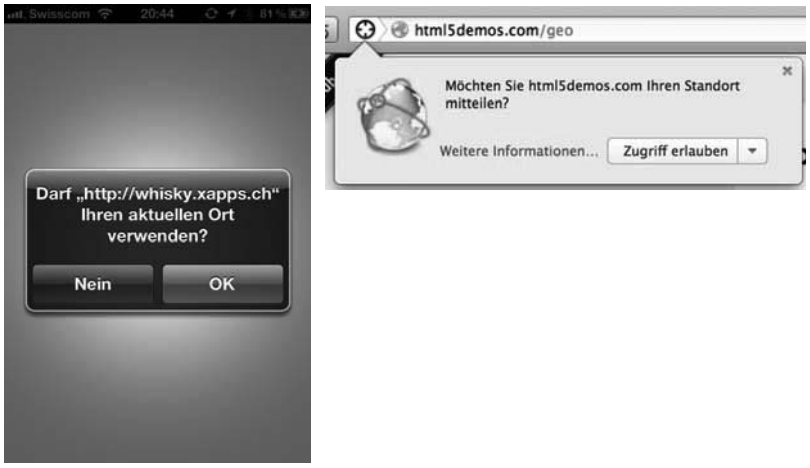

Abb. 6–4
Nachfrage auf dem iPhone und dem Firefox

In unserer Whisky-App soll der Ort der Degustation dargestellt werden:

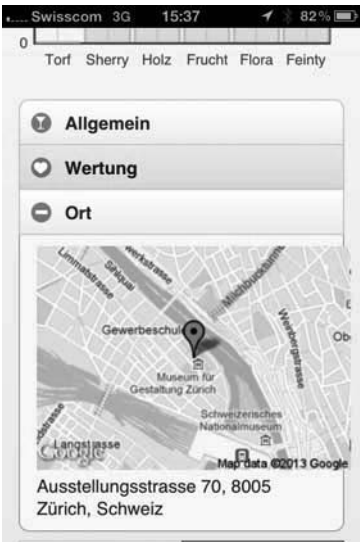

Abb. 6–5
Ort in der Whisky-App

6.2.1 Das Geolocation-Objekt

Mit der Funktion `navigator.geolocation.getCurrentPosition()` kann die aktuelle Position ermittelt werden. War die Ermittlung erfolgreich, befinden sich die Koordinaten im Rückgabe-Objekt. Veränderungen der Position können mit den Funktionen `navigator.geolocation.watchPosition()` und `navigator.geolocation.clearWatch()` beobachtet werden.

Wie es vielleicht zu erwarten war, kapseln wir diese Funktion in einem eigenen Objekt und nennen es `Location`. Dazu legen wir auch eine eigene Datei `location.js` an, fügen sie in der `index.html`-Datei hinzu und instanziieren sie im Main. Da hier für das Testen Konsolen-Ausgaben hilfreich sein können, definieren wir eine Variable `debug` und prüfen vor jeder Ausgabe, ob sie `true` ist. Bei `false` handelt es sich um den produktiven Einsatz und wir machen keine Konsolen-Ausgaben:

```
var WhiskyApp = {
   tastings: new Tastings(),
   controller: new WhiskyAppController(),
   gui: new WhiskyView(),
   tablet: false,
   db: new WhiskyDB(),
   locAPI: new Location(),
   debug: true
}
```

```
// Observer auf Tastings
WhiskyApp.tastings.addObserver(WhiskyApp.gui, "update");
WhiskyApp.tastings.addObserver(WhiskyApp.db, "update");
WhiskyApp.db.readWertungen();

// Observer auf Geolocation
WhiskyApp.locAPI.addObserver(WhiskyApp.gui, "location_update");
```

Nun zu unserem `Location`-Objekt. Da die Positionsermittlung asynchron erfolgt und bei der erstmaligen Ermittlung länger dauern kann, leiten wir das Objekt vom Observer ab. Wenn die Position ermittelt wurde, rufen wir `notify()` auf und aktualisieren auf der View die Ausgabe. Deshalb wurde die `addOvserver()`-Methode im Main positioniert. Wieder haben wir eine elegante Anwendung für unser MVC-Pattern.

Jetzt stellt sich noch die Frage, wann wir die Position ermitteln. Wir dürfen ja die Position einer bestehenden Wertung nicht überschreiben. Ich schlage vor, dass wir jedes Mal, wenn wir eine neue Wertung *anlegen*, die Position ermitteln (`start()`). Dazu merken wir uns die GUID der neuen Wertung. Wenn die Position ermittelt wurde, prüft die View zuerst, ob diese neue Wertung noch aktiv ist oder bereits eine andere dargestellt wird (der Anwender kann ja schneller sein als die Ortung...). Im ersten Fall aktualisieren wir die Anzeige und das Wertungs-Objekt, im zweiten Fall eben nicht.

Sie merken es, diese einfache Funktion bringt viele neue Konzepte und benötigt an verschiedenen Stellen Eingriffe. Aber beginnen wir mit dem Location-Objekt. In der `init()`-Methode definieren wir ein Objekt, das alle wichtigen Werte speichert. In der Datenbank speichern wir ebenfalls dieses Objekt. Als UML-Diagramm ergibt das nachfolgendes Objekt:

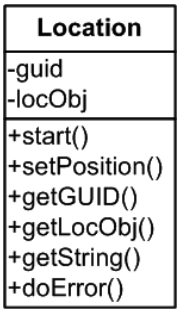

Abb. 6–6
Location-Objekt

Als Programmcode:

Listing 6–8
Location-Objekt

```
/** ********* LOCATION API ********** */
var Location = Observer.extend({
   init : function() {
      this._super();
      this.guid = "";
      this.locObj = {};
      this.locObj.adresse = "";
      this.locObj.pos = "";
      this.locObj.altitude = "";
      this.locObj.state = "";
      this.locObj.string = "";
   },

   /** Startet eine Ermittlung erneut */
   start : function(guid) {
      var that = this;
      that.guid = guid; // Die aktuelle GUID der Wertung

❶     var options = {
         maximumAge : 30000 // max. Alter der Position in ms
      };

      this.locObj.state = "Position ermitteln....";
❷     navigator.geolocation.getCurrentPosition(function(position)
      {
         that.setPosition(position);
      }, function(error) {
         that.doError(error);
      }, options);
   },

   /** Setzt die aktuelle Position (Koordinaten) und
       ermittelt dazu die Adresse */
❸  setPosition : function(position) {
      this.locObj.pos =   position.coords.latitude + "," +
                          position.coords.longitude;
      this.locObj.altitude = position.coords.altitude;
      this.notify();

❹     if ( typeof google != "undefined") {
         var latlng = new google.maps.LatLng(
             position.coords.latitude, position.coords.longitude);
         var geocoder = new google.maps.Geocoder();
         var that = this;

         geocoder.geocode({
            'latLng' : latlng
         }, function(results, status) {
            if (status == google.maps.GeocoderStatus.OK) {
               if (results[0]) {
```

```
                that.locObj.adresse =
                                    results[0].formatted_address;
                that.notify();
             }
          }
       });
   }
},

/** Gibt die GUID der Wertung zurück */
getGUID : function() {
   return this.guid;
},

/** Gibt das LocObj zurück. */
getLocObj : function() {
   this.locObj.string = this.getString();
   return this.locObj;
},

/** Gibt die ermittelte Adresse zurück, wenn vorhanden, sonst
 * die Koordinaten. Und wenn diese auch nicht vorhanden sind,
 * dann den Status. */
getString : function() {
   if (this.locObj.adresse == "") { // keine Adresse
      if (this.locObj.pos == "")
         return this.locObj.state;
      else {
         if (this.altitude != null)
            return Math.round(this.locObj.altitude * 10) / 10
                              + "m";
         else
            return this.locObj.pos;
      }
   } else {
      if (this.altitude != null)
         return this.locObj.adresse + " - " +
              Math.round(this.locObj.altitude * 10) / 10 + "m";
      else
         return this.locObj.adresse;
   }
},

/** Fehler der Ermittlung. */
doError : function(error) {
   switch (error.code) {
      case error.PERMISSION_DENIED:
         this.locObj.state = "Sie haben keine Berechtigung
                              für Geolocation gegeben.";
         break;
```

```
            case error.POSITION_UNAVAILABLE:
               this.locObj.state = "Kann aktuelle Position
                                    nicht finden.";
               break;

            case error.TIMEOUT:
               this.locObj.state = "Time-Out";
               break;

            default:
               this.locObj.state = "Unbekannter Fehler";
               break;
         }

         // Im Debug-Modus ausgeben
         if (WhiskyApp.debug)
            console.log(this.locObj.state);
      }
   });
```

Im Konstruktor prüfen wir, ob Geolocation unterstützt wird. Wenn ja, dann akzeptieren wir auch eine »alte« Position, die nicht älter als 30 Sekunden ist (um den Akku zu schonen). Hier die möglichen Parameter für die Geolocation:

- `timeout` (`long`): maximale Dauer für die Positionsermittlung in Millisekunden
- `maximumAge` (`long`): maximales Alter der letzten Positionsermittlung in Millisekunden. Da eine Positionsermittlung mit dem GPS-Chip Energie benötigt, ist gegebenenfalls eine ältere Position noch akzeptabel.
- `enableHighAccuracy` (`boolean`): Bei `true` wird die bestmögliche Auflösung der Positionsermittlung verwendet, also in der Regel mit dem GPS-Chip. Schneller ist eine einfache Positionsermittlung mithilfe von Wifi und dessen IP-Adresse oder der Position vom Mobilfunknetz (wird je nach Anbieter unterstützt).

Bei ❷ wird mit der Methode

```
navigator.geolocation.getCurrentPosition(function(position){...
}, function(error) {...}, options);
```

die Position ermittelt. Im Erfolgsfall wird die Methode `setPosition()` (1. Parameter) aufgerufen, im Fehlerfall oder einem Timeout die `doError()`-Methode (2. Parameter).

In der `setPosition()`-Methode ❸ werden die Koordinaten ausgelesen und der Observer informiert, dass die Position ermittelt wurde. Das Objekt liefert, sofern die Hardware dies unterstützt, folgende Informationen:

- `position.timestamp`: Zeitpunkt der Ermittlung
- `position.coords.latitude`: geografische Koordinate gemäß WGS84 (World Geodic System), dezimal in Grad
- `position.coords.longitude`: geografische Koordinate gemäß WGS84 (World Geodic System), dezimal in Grad
- `position.coords.accuracy`: Genauigkeit der Koordinaten in Meter
- `position.coords.altitude`: Höhe in Metern. Wenn dies nicht ermittelt werden kann, dann `null`.
- `position.coords.altitudeAccuracy`: Genauigkeit der Höhe in Meter
- `position.coords.speed`: die aktuelle Geschwindigkeit in Meter pro Sekunde. Wenn dies nicht ermittelt werden kann, dann `null`.
- `position.coords.heading`: Grad der Fahrtrichtung vom wahren Norden im Uhrzeigersinn. Wenn dies nicht ermittelt werden kann, dann `null`.

Für den Benutzer unserer App wäre jedoch die aktuelle Adresse interessanter. Anhand von Koordinaten eine Adresse zu ermitteln, wird auch *Reverse Geocoding* genannt. Google (und auch andere) bietet einen solchen Dienst mit der Google-Maps-API an. Damit dies läuft, müssen wir neben unserer JavaScript-Datei `location.js` auch das Google-API in der `index.html`-Datei einbinden:

```
<script src="http://maps.google.com/maps/api/js?sensor=true">
</script>
```

Führen wir also nach dem Zwischenspeichern der Koordinaten die Adressermittlung aus ❹. Wurde die Google-Maps-API erfolgreich geladen, so haben wir das Objekt `google` zur Verfügung. Mit dem Objekt `google.maps.LatLng` erzeugen wir ein Google-Objekt mit den Koordinaten und legen ein neues Objekt `new google.maps.Geocoder()` für die Adressermittlung an. Dieses Objekt bietet die Methode `geocode(Koordinaten, Resultat)` an, der wir die Koordinaten und eine Callback-Funktion übergeben. Darin speichern wir die Adresse und informieren den Observer, dass sich etwas geändert hat.

6.2.2 Anpassungen an der View

In der View müssen wir in den Details der Wertung den Ort darstellen. Der Ort kann ebenfalls mit der Google-Maps-API als Karte angefordert werden. Diesen Aufruf werden wir in der View machen. Auf einer neuen Zeile folgt dann die Adresse, die wir mithilfe eines `div`-Elements als Platzhalter in der HTML-Struktur einfügen:

Listing 6–9
Ort-Definition im HTML

```
<div data-role="collapsible" data-collapsed="true"
     data-collapsed-icon="flag"  data-content-theme="c">
   <h3>Ort</h3>
   <div data-role="ort" id="adr">
      <br />
      <div id="adresse"></div>
   </div>
</div>
```

In der `view.js`-Datei implementieren wir die Methode `location_update()`:

Listing 6–10
Erzeugen der Ortsangabe mit Google Maps

```
/** Geolocation-Update */
location_update : function(scope, data) {
❶    if(scope.getGUID() == WhiskyApp.controller.getGUID()) {
❷    var locObj = scope.getLocObj();
❸    WhiskyApp.controller.setOrt(locObj);
❹    this.setLocation(locObj);
     }
},

setLocation : function(ort) {
❺ $('#adresse').text(ort.string);
❻ $('#adrimg').remove();
❼    if( navigator.onLine )
❽       $('#adr').prepend('<img
           src="https://maps.googleapis.com/maps/api/staticmap?
           center='+ort.pos+'&amp;zoom=14&amp;size=280x200&amp;
           markers='+ort.pos+'&amp;sensor=false" height="200"
           width="280" id="adrimg" />');
}
```

Bei der Zeile ❶ prüfen wir, ob die neue Wertung, zu der die Position ermittelt werden musste, noch aktiv ist. Ansonsten überschreiben wir keine Position. In der Zeile ❷ holen wir unser Positions-Objekt und setzen es bei ❸ im Controller. Weshalb? Nun, wir müssen dieses ja noch speichern. Da der Controller dessen Management übernimmt, müssen wir ihm das Objekt auch zugänglich machen. Umgekehrt muss der Controller auch Positionen von gelesenen Wertungen anzeigen. Deshalb lagern wir die eigentliche Darstellung in eine separate Methode aus ❹. Dort setzen wir die Adresse ❺. In Zeile ❻ löschen wir das Image von Google Maps (wenn bereits vorhanden) und prüfen, ob wir online sind ❼. Dies funktioniert nicht in allen Browsern, aber auf dem Mobile Device ist dies möglich. Wenn wir online sind, fordern wir eine *Static-Map* an ❽. Unsere Position soll in der Mitte sein (`center='+ort.pos+'`), der Zoom-Faktor 14 (`zoom=14`) und die Bildgröße 280 × 200 Pixel (`size=280x200`). Somit passt das Bild perfekt auf ein Smartphone. Nun fehlt noch der Marker. Dieser kann mit dem Parameter `markers` positioniert werden: `markers='+ort.pos+'`. Der Vorteil

dieser *Static-Map* ist, dass wir ein fixes Bild erhalten, das in jedem Browser dargestellt werden kann. Sollte mehr Funktionalität gefordert werden, empfehle ich, einen Link auf die native App des Systems zu erstellen: `http://maps.google.com/maps?q=position`.

Spezielle Links

Die mobilen Browser werten jeweils das `href`-Attribut aus und entscheiden, ob es sich um einen Link im Browser oder einen Link auf eine App handelt. Nebst dem »Google-Link« gibt es weitere spezielle Links:

Telefon: href="tel:0441231212"

SMS: href="sms:0791231212"

Mail: href="mailto:name@domain.ch?subject=Anfrage&body=Hallo"

6.2.3 Anpassungen im Controller

Bei der Darstellung einer Wertung müssen wir auch die Darstellung der Position anstoßen. Dies erfolgt ja in der `refreshWertung()`-Methode:

```
/** Aktualisiert Wertungs-Page
 */
function refreshWertung() {
   // Zuweisungen
   $('#date').val(actWertung.date);
   ...
   $('#kommentar').val(actWertung.kommentar);

   var ort = actWertung.ort;
   if( ort == "" ) {
      $('#adresse').text("");
      $('#adrimg').remove();
   }
   else {
      WhiskyApp.gui.setLocation(ort);
   }
}
```

Wenn es keine Informationen gibt, löschen wir den Text und das Bild. Ansonsten rufen wir in der View die Methode `setLocation()` auf.

Eigentlich wäre es das gewesen – wenn wir die Position nicht speichern wollen. Aber genau das ist ja der Sinn der Sache. Also fügen wir im Controller die öffentlichen Methoden für das Setzen des Ortes ein und lesen die GUID ein:

```
return {
   initialize : function() {
      ...
      getGUID : function() {
         return actWertung.guid;
      },
      setOrt : function(locObj) {
         actWertung.ort = locObj;
      }
   }
}
```

Im Model speichern wir in der Variable `ort` das `locObj`-Objekt, das eine Ansammlung von Variablen ist. Um es in einer relationalen Datenbank korrekt zu speichern, müssen wir es umwandeln in einen Text. Diese Umwandlung kann mithilfe von JSON erfolgen.

JSON (JavaScript Object Notation)

JSON ist ein schlankes Datenaustauschformat, das für Menschen einfach zu lesen und zu schreiben und für Maschinen einfach zu parsen und zu generieren ist. Es basiert auf einer Untermenge der JavaScript-Programmiersprache und ist komplett unabhängig von Programmiersprachen. Es baut im Textformat Namen-Werte-Paare auf, die als eine geordnete Liste von Werten eingefügt werden können. Das folgende Objekt sieht JSON-formatiert so aus:

```
WhiskyAppConfig = {
   debug: true,
   tablet: false
};
{"debug":true,"tablet":false}
```

Der JSON-String kann mit der Methode `JSON.stringify(WhiskyAppConfig);` erzeugt werden und mit der Methode `JSON.parse(string);` geparst und in ein Objekt umgewandelt werden. Tiefe Strukturen sind mit JSON ebenfalls möglich.

Passen wir dazu das Datenbankobjekt an, um die Art der Speicherung gegenüber dem Model transparent zu halten:

Listing 6–11
Speichern und Lesen von Objekten mithilfe von JSON

```
/** Whisky-DB-Zugriff */
var WhiskyDB = Class.extend({
   ...
   /** Alle Wertungen lesen */
   readWertungen : function() {
      var array = new Array();
      this.db.transaction( function(transaction) {
         transaction.executeSql( 'SELECT * FROM wertungen
                                  ORDER BY distillery;',[] ,
         function (transaction, result) {
```

```
            for (var i=0; i < result.rows.length; i++) {
               var row = result.rows.item(i);
               var ort = "";
               // Wertung erzeugen
               try {
                  ort = JSON.parse(row.ort);
               } catch(e) {}
               var wertung = new Wertung(...);
               array.push(wertung);
            }
            // Alle Wertungen dem Model übergeben
            WhiskyApp.tastings.setWertungen(array);
         }, WhiskyApp.db.errorHandler );
      }, WhiskyApp.db.trErrorHandler );
   },

   /** INSERT */
   insertEntry : function(wertung) {
      // Ort in der DB als JSON Objekt
      var ort = "";
      try {
         ort = JSON.stringify(wertung.ort);
      } catch(e) {}

      this.db.transaction( function(transaction) {
         transaction.executeSql(...);
      }, WhiskyApp.db.trErrorHandler );
   },
   ...
});
```

Sollte die Umwandlung zu oder von einem JSON-Objekt nicht funktionieren, so wird eine Exception geworfen. Diese fangen wir auf und lassen dann den Ort leer.

6.2.4 Fortlaufende Beobachtung

Wir haben die einmalige Positionierung kennengelernt. Es gibt auch die Möglichkeit, die Position zu überwachen. Immer wenn der Benutzer die Position geändert hat, ruft der Browser die mitgegebene Callback-Methode auf. Dieser Mechanismus kann mit der Methode

```
var id = navigator.geolocation.watchPosition(function(pos) {
... }
```

gestartet und mit der Methode

```
navigator.geolocation.clearWatch(id);
```

gestoppt werden.

Einer der Entwickler findet auf der Homepage http://www.movable-type.co.uk/scripts/latlong.html die Berechnung der Distanz zwischen zwei Koordinaten. Kombiniert mit den zwei Methoden von oben, schreibt er sich eine kleine Webseite, die ihm fortlaufend die zurückgelegte Distanz berechnet. Klar, dass heute alle Mitarbeiter beim Nach-Hause-Gehen auf das Smartphone schauen werden ... Hier sehen Sie seinen Code:

Listing 6–12
Fortlaufende Distanzberechnung

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <meta name="viewport" content="width=device-width,
                                     initial-scale=1">
      <title>Geolocation</title>
      <link rel="stylesheet" href="jquery.mobile.min.css" />
      <script src="jquery.min.js"></script>
      <script src="jquery.mobile.min.js"></script>
   </head>

   </head>
   <body>
      <div data-role="page">
         <div data-role="header" data-position="fixed" >
            <h1>Distanz</h1>
         </div>
         <div data-role="content">
            <p>
               Start: <span id="startLat"></span>&deg; /
                      <span id="startLon"></span>&deg;
            </p>
            <p>
               Aktuell: <span id="actLat"></span>&deg; /
                        <span id="actLon"></span>&deg;
            </p>
            <p>
               Distanz: <span id="dist"></span>km
            </p>
            <p>
               <button id="stop">Stop</button>
            </p>
         </div>
      </div>
      <script>
         window.onload = function() {
            var startPos;

            if (navigator.geolocation) {
               // Wenn Geolocation unterstützt wird
               // einmalig die Position als Start-Position
               // ermitteln
```

```
            navigator.geolocation.getCurrentPosition(
            function(pos) {
               startPos = pos.coords;
               $('#startLat').html(startPos.latitude);
               $('#startLon').html(startPos.longitude);
            }, function(error) {
               alert("Error code: " + error.code);
            });

            // Nun die Position überwachen und aktualisieren
            var id = navigator.geolocation.watchPosition(
            function(pos) {
               var actPos = pos.coords;
               $('#actLat').html(actPos.latitude);
               $('#actLon').html(actPos.longitude);
               $('#dist').html(
               calculateDistance(startPos.latitude,
                startPos.longitude, actPos.latitude,
                actPos.longitude) );
            });

            $('#stop').click(
            function() {
               navigator.geolocation.clearWatch(id);
            });
         }
      };

      // Distanz berechnen
      // copyright Moveable Type Scripts
      // http://www.movable-type.co.uk/scripts/latlong.html
      // Under Creative Commons License
      function calculateDistance(lat1, lon1, lat2, lon2) {
         var R = 6371; // km
         var dLat = (lat2 - lat1).toRad();
         var dLon = (lon2 - lon1).toRad();
         var a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
                 Math.cos(lat1.toRad()) *
                 Math.cos(lat2.toRad()) * Math.sin(dLon / 2) *
                 Math.sin(dLon / 2);
         var c = 2 * Math.atan2(Math.sqrt(a),
                 Math.sqrt(1 - a));
         var d = R * c;
         return d;
      }

      Number.prototype.toRad = function() {
         return this * Math.PI / 180;
      }
   </script>
</body>
</html>
```

Die Ausgabe sieht so aus:

Abb. 6–7
Distanzberechnung in Aktion

6.2.5 Unterstützung

Die Geolocation-API wird von folgenden Browsern unterstützt:

- Internet Explorer ab Version 9
- Firefox ab 3.5
- Safari ab 5.0
- Mobile Safari ab 3.0
- Chrome ab 5.0
- Android ab 2.0
- Opera 10.6
- Opera Mobile ab 10.1
- Windows Phone 7.0
- Blackberry 6.0

6.3 Diagramme zeichnen

Die sechs Wertungen sollen in einem Diagramm grafisch dargestellt werden. Herr Weber erklärt, dass es in HTML5 einen Standard für Vektorgrafiken (SVG[2], Scalable Vector Graphics) und einen für Pixelzeichnungen (Canvas[3]) gibt. Dazu existieren auf dem Markt zahlreiche Bücher. SVG-Grafiken haben den Vorteil, dass sie skalierbar sind. Sie

2. *http://de.wikipedia.org/wiki/Scalable_Vector_Graphics*
3. *http://de.wikipedia.org/wiki/Canvas_%28HTML-Element%29*