ulm university universität ulm

API Evaluation

Übersicht über API Evaluationstechniken

Michael Barth | 30.01.12 | Seminarvortrag

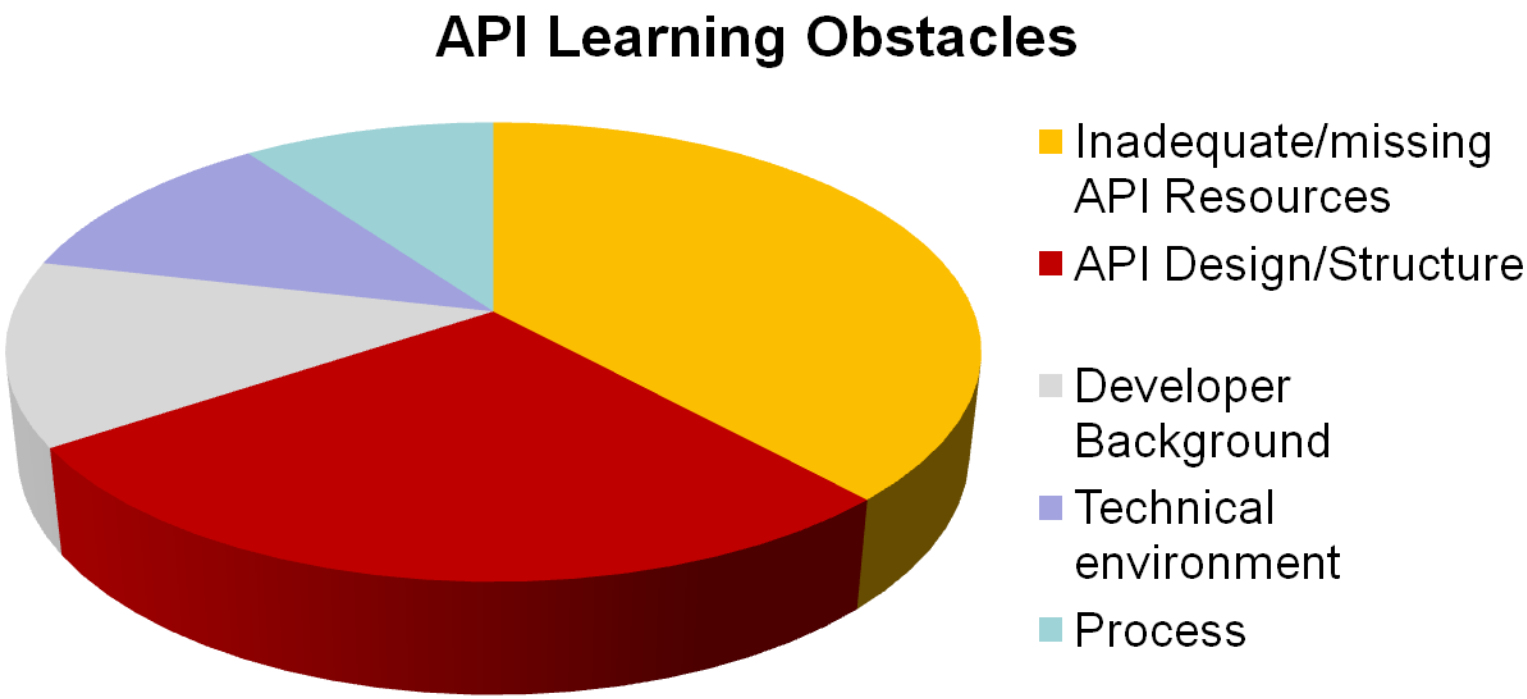
Motivation

- Software wird selten von Grund auf neu geschrieben (Libraries, Frameworks, SDKs, ...)

Vorteile
- Beschleunigung der Entwicklung
- Zunutze machen erprobter und ausgereifter Komponenten
- Entwickler können sich auf Anwendung konzentrieren

API beeinflusst Nutzen von Komponenten maßgeblich:
- inadäquate API => Wrapper Code, Workarounds
- ineffiziente API => Performanceprobleme
- behindert Entwickler => Entwicklungszeit, provoziert Fehler

Umfrageergebnisse



Quelle: *What Makes APIs Hard To Learn? Answers from Developers*
2009, Martin P. Robillard, McGill University

API QUALITÄT

API Qualität

- Problematik:

  Es ist sehr einfach eine schlechte API zu entwerfen,
  aber schwierig, eine gute zu entwerfen

- Wenige gute Gestaltungsmöglichkeiten, aber viele schlechte

- Wird einmal erstellt, aber häufig benutzt
  - Auswirkungen können unverhältnismäßig potenziert werden
  - Spätere Verwendung schwer absehbar (Dauer, Kontext, ...)

- Schwierig zu messen was eine „gute“ API ist

Qualitäten guter APIs – Abstraction Level

- Definiert was verborgen und was enthüllt wird durch API
  - Prinzip des *Information Hiding*
- Tradeoff: Low-level ⟷ High-level

- Sollte zur Aufgabe passen:

	Low-level API	High-level API
Low-level Task	OK	Nicht möglich
High-level Task	Schwierig	OK

High-level – Beispiel

```
car.start();
```

1 Zeile Code

Medium-level – Beispiel

```
car.getEngine().start();
car.getABS().start();
car.getRadio().start();
...
```

3+ Zeilen Code

Low-level – Beispiel

```
Engine eng = car.getEngine();
eng.start();

while (eng.isRunning()) {
    eng.executeIntakeStroke();
    eng.executeCompressionStroke();
    eng.executePowerStroke();
    eng.executeExhaustStroke();
}

ABS abs = car.getABS();
...
```

8+ Zeilen Code

Comprehensibility

„Those who have to use an API must be able to understand it.“

- Verständnis essentiell um API erfolgreich zu nutzen
- Gutes Verständnis verhindert Fehler. Beispiel:

```
car.setBrakes(discBrakes, true);
```

Was genau macht der Code?

Comprehensibility

„Those who have to use an API must be able to understand it.“

- Verständnis essentiell um API erfolgreich zu nutzen
- Gutes Verständnis verhindert Fehler. Beispiel:

```
car.setBrakes(discBrakes, true);
```

  Was genau macht der Code?

- Möglichkeiten: Weist Referenz auf `discBrakes` zu und...
  - Setzt Eigenschaft auf `true`, z.B. `bool hasABS = true`
  - Aktiviert die Scheibenbremsen sofort

Comprehensibility – Beispiel

- Leichter verständlich:

```
car.setBrakes(discBrakes, BrakeFeatures.HasABS);

car.setAndActivateBrakes(discBrakes);
```

- Änderungen verbessern Verständnis, Code selbsterklärend
- Gute Dokumentation hilft hier ebenfalls

Consistency

- Beinflusst wieviel sich von bereits gelernten Teilen folgern lässt
  - Erleichtert und beschleunigt das Lernen und Benutzen
  - Erhöht Bedienkomfort
- Umfasst:
  - Konsistente Namen
  - Gleiche Muster
  - Gleiche Beschreibungen
  - ...
- Beispiel Singleton Pattern. Mögliche Namen für getter Methode: `instance(), getInstance(), getDefault(), ...`

Discoverability

- Das beste Hilfsmittel nützt nichts, wenn man es nicht kennt bzw. wenn die Suche danach länger ist als die Zeitersparnis
- Gute Dokumentation wichtig
  - Problemorientiert
  - Gut navigierbar
- Klassen- und Methodennamen sollten Zweck selbst erklären
  - IDE Auto-Completion
- Ist abhängig von Erwartungen der Person

Discoverability – Beispiel

Fernlicht bei Auto anschalten

Discoverability – Beispiel

Fernlicht bei Auto anschalten

```
car.doMagic();
```

Discoverability – Beispiel

Fernlicht bei Auto anschalten

```
car.doMagic();

car.activateSystem(enum System);
```

Discoverability – Beispiel

Fernlicht bei Auto anschalten

```
car.doMagic();

car.activateSystem(enum System);

car.setLight(enum LightType, bool state);
```

Discoverability – Beispiel

Fernlicht bei Auto anschalten

```
car.doMagic();

car.activateSystem(enum System);

car.setLight(enum LightType, bool state);

car.turnLightOn(enum LightType);
```

Discoverability – Beispiel

Fernlicht bei Auto anschalten

```
car.doMagic();
car.activateSystem(enum System);
car.setLight(enum LightType, bool state);
car.turnLightOn(enum LightType);
car.turnHighBeamLightOn();
```

Learning Barriers

- Mit einer API zu arbeiten ist ein konstanter Lernprozess
  - Bei Erstbenutzung
  - Nach langer Pause
- Trägt direkt zur Benutzbarkeit und Geschwindigkeit bei
- Zeitaufwändig und schwierig zu messen

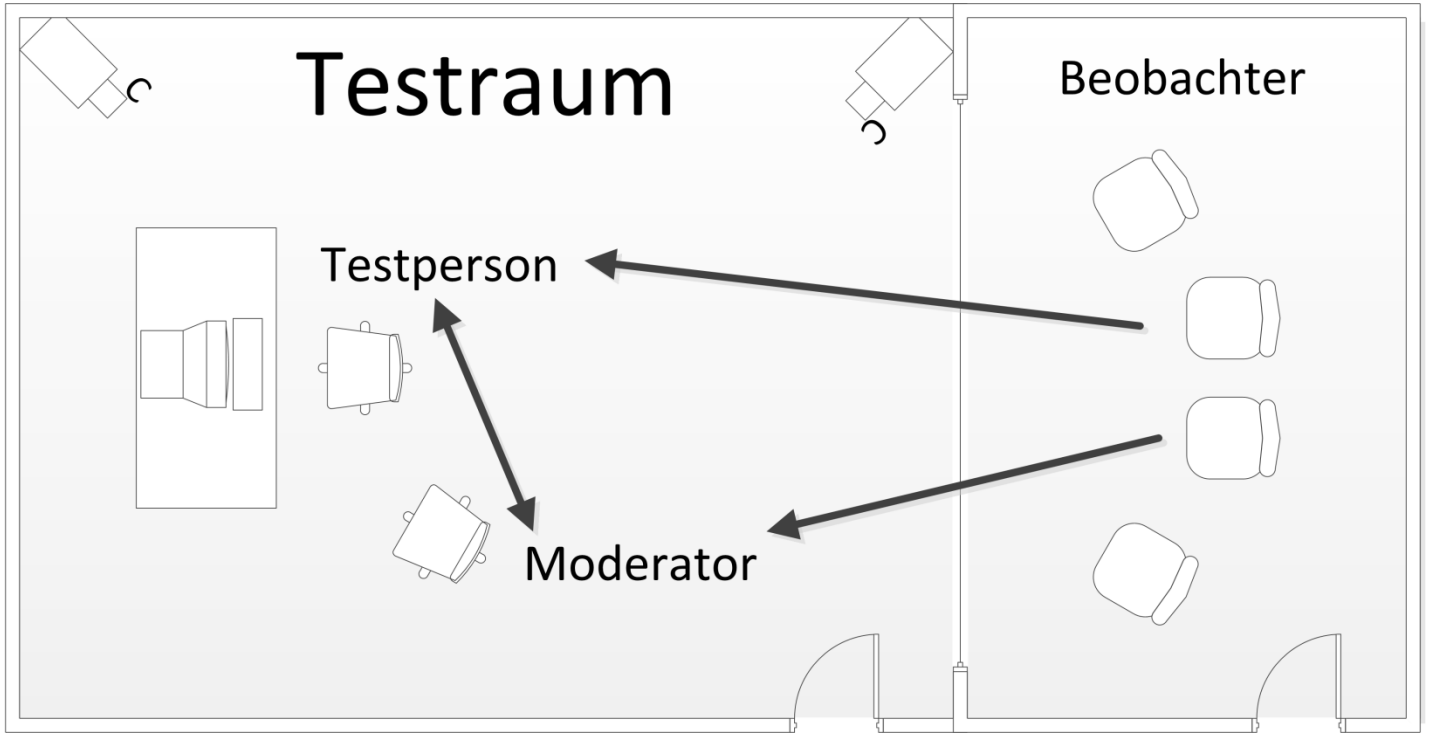
- ...es gibt noch weitere API Qualitäten

API EVALUATION

Besonderheiten bei API Evaluation

- API Qualitäten oft subjektiv und schwer messbar
  - Abhängig von Aufgabe
  - Abhängig von persönlichen Erfahrungen der Benutzer
  - API sollte auf Zielgruppe zugeschnitten werden
- Quality Assurance für API Entwickler, Auswahlhilfe für API Benutzer
- Die meisten Evaluationstechniken setzen auf klassischen Mensch-Computer-Interaktions (MCI) Techniken auf

MCI: Usability Labor



MCI: Think-Aloud Protocol

- Interaktion mit API sehr subtil
- Besonderheiten bei APIs:
  - Mentales Modell des Benutzers
  - Benutzungsfehler nicht unbedingt Mangel an API
    => API lernen ist explorativ, Fehler machen gehört dazu
- Schwierig in Anwendung und Interpretation
  - Hilft aber Problemfelder zu identifizieren
  - Hilft Benutzerverhalten besser zu verstehen

MCI: Cognitive Walkthrough

- Zuerst Aufgabenanalyse: Welche Schritte werden benötigt um Aufgabe zu erfüllen?
  - Festhalten einer Schritt-für-Schritt-Lösung
- Gruppe aus API Entwicklern und Designern trifft sich
  - Lösungen Schritt für Schritt durchzugehen (walk through)
  - Teilnehmer beantworten Fragen zu den Schritten

Vorteile

- Keine Benutzer nötig
- Kann sehr früh eingesetzt werden (sobald API spezifiziert)
- Skaliert gut
- Gutes Kosten-Nutzen-Verhältnis

API EVALUATIONSTECHNIKEN

The Cognitive Dimensions Framework

- *Cognitive Dimensions of Notation* Framework
  - Design Richtlinien für Notationen, UIs und Programmiersprachen
  - Weiterentwickelt um Klassenbibliotheken zu evaluieren
- Fokus auf Abgleich von Benutzeranforderungen mit angebotenen Merkmalen
- 12 Metriken für Nutzer Anforderungen und API Merkmale (Abstraction Level, Learning Style, Consistency, ...)
- Augenmerk auf API-spezifische und umsetzbare Ergebnisse
  - Ergebnisse sollen direkt zu Änderungen an API führen

Anwendung

1. Kernanwendungsszenarios der API bestimmen und in empirischen Studien überprüfen.

2. Aufgaben erstellen anhand der Szenarios.

3. Benutzer in Usability Studie die Aufgaben lösen lassen während Sie auf Video aufgenommen werden.

4. Gesammelte Daten analysieren.
   Suche nach benutzerübergreifenden Mustern und Situationen, wo API Design versagte.

5. Ergebnisse beschreiben mit Hilfe von Fragen für jede Metrik.

Fazit

Vorteile

– Anwendungsorientiert

– Bezieht Lernkurve der API ein

– Metriken bilden gemeinsames Vokabular zum Bewerten und Vergleichen von APIs

– Auch ohne Usability Labor anwendbar (z.B. als Walkthrough)

– Von Microsoft bereits erprobt und erfolgreich angewandt (C#)

Nachteile

– Zeitaufwändig

– Ausgiebiges Training nötig um Methode richtig anzuwenden (Deutung der Ergebnisse)

The API Walkthrough Method

- Fokus auf: Kann Benutzer, basierend auf API Code, akkurates mentales Modell der API entwickeln?
- Verwendung des Think-Aloud Protokolls
- Moderator muss sensibel sein für Frustration des Benutzers
  - Code nicht ausführbar
  - Variablen nicht inspizierbar
- Zwei Materialien nötig:
  - Dokumentierte Use Cases (Primäre Workflows)
  - Grobe Idee/Grundriss der API (Spezifikation)

Anwendung

1. Benutzer wird Ablauf und Zweck der Studie erklärt.
2. Code-Beispiele der Use Cases werden präsentiert.
   - Zufällige Reihenfolge
3. Benutzer soll Code Zeile für Zeile durchgehen und interpretieren.
   - Wichtig, dass Benutzer sich wohl und kompetent fühlt
4. Separate Nachbesprechung mit API Team & Beobachtern, ohne Benutzer.

Fazit

Vorteile

– Sehr früh einsetzbar

– Eignet sich gut zum Finden von
  – Sinnvollen Namen
  – Lücken in Dokumentation
  – Sinnvollen Standardeinstellungen
  – Abstraktionslevel
  – Missverständnissen hevorgerufen durch schlechtes API Design

– Von MathWorks bereits erprobt und erfolgreich angewandt

Nachteile

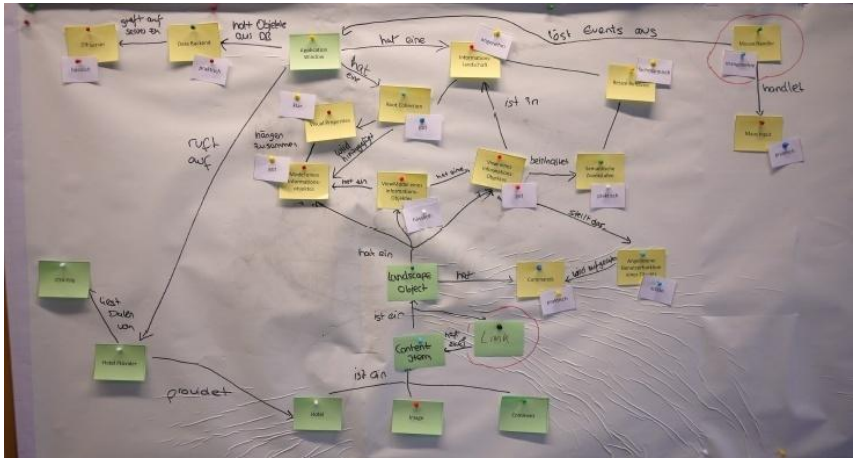
– Angst oder Unwohlsein der Teilnehmer beim Schätzen

– Gefahr das Teilnehmer API missversteht auf eine Art die Moderator unklar ist

– Usability Labor nötig (teuer)

The Concept Maps Method

- Langzeit Studie (über Wochen und Monate)
- Fokus auf Lernhindernisse und Langzeitdaten
- Begrenzte Zeit anderer Evaluationstechniken hat Probleme
  - Relativ simple Aufgaben in Evaluation
  - Wahrnehmung und Lernbarrieren verändern sich über Zeit
- Verwendung eines Artefakts (*Concept Map*) als Gesprächsgrundlage



Anwendung

1. Treffen: Erklärung der Concept Maps Methode

2. Treffen: Erstellung Concept Map

1. Passende Konzepte heraussuchen
2. Anbringen an Concept Map, verbinden mit beschrifteten Linien
3. Review
4. Anbringen der Rating concepts, Einkreisen von Problemfeldern

Alle weiteren Treffen

1. Aktualisieren der Concept Map
2. Erweitern der Concept Map
3. Aktualisieren von Rating concepts und Problemfelder

Alle Treffen werden auf Video aufgezeichnet.

Fazit

Vorteile

– Macht Änderungen über Zeit visuell sichtbar

– Visualisiert Prozesse zwischen API und Software

– Zeichnet Dynamik der API Nutzung auf

– Digitalisierung der Concept Maps leicht möglich
  – Automatische Analyse möglich
  – Diverse Visualisierungen möglich

Nachteile

– Sehr zeitintensiv

– Sehr ressourcenintensiv
  – Viele Personen involviert über langen Zeitraum

– Spät einsetzbar

Metrix - Automatic Evaluation using Complexity Metrics

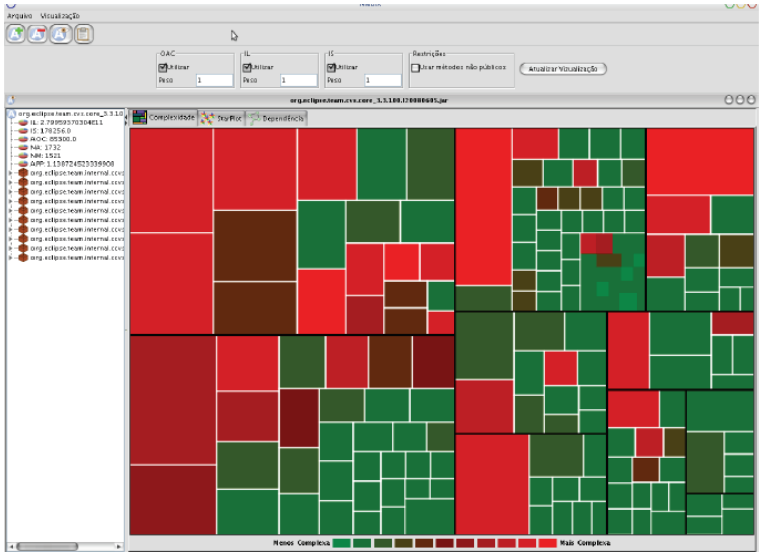
- Nutzung von Softwaremetriken zur Evaluation
  - Komplextitätsmetriken
  - Implementationsdetails unwesentlich für API
- Sofortiges Feedback möglich
  - Integration in IDE denkbar
- Visualisierung der Ergebnisse erleichtert Interpretation
- Problem: Bezugswerte für Bewertung nötig
  - Statistische Daten über durchschn. API Werte als Grundlage

Anwendung

Anwendung analysiert Code völlig automatisch

Visuelle Aufbereitung der Ergebnisse:

- Hierarchische Daten in verschachtelten Rechtecken (z.B. Packages, Klassen, ...)
- Farbe gibt Komplexität an
- Andere Visualisierungen denkbar (StarPlot)



Fazit

Vorteile

– Sofortiges Feedback

– Frühes Feedback (Spezifikation genügt)

– Metriken erlauben Einschätzung und Vergleich von APIs

– Visualisierung erlaubt leichte und schnelle Interpretation

Nachteile

– Komplextitäsmetriken erkennen nicht alle Probleme

– Benutzer werden nicht berücksichtigt

– Aufgabenangemessenheit wird nicht berücksichtigt

Zusammenfassung

- API Qualität kann sich direkt auf die Produktqualität, Wartbarkeit und Entwicklungskosten auswirken
- API Qualität abhängig von vielen Faktoren
  - Aufgabe
  - Benutzern
  - Allgemeinen Designrichtlinien
- API Evaluationsmethoden haben verschiedene Schwerpunkte
  - Keine Methode bietet alles
- API Evaluierung noch eher unbekanntes Thema
  - An API Evaluierungstechniken wird noch geforscht


Fragen?