MSI

Offene Architekturen in Software Ökosystemen

Michael Hassel, Dominik Feininger
Betreuender Dozent: Dr. Jens Knodel

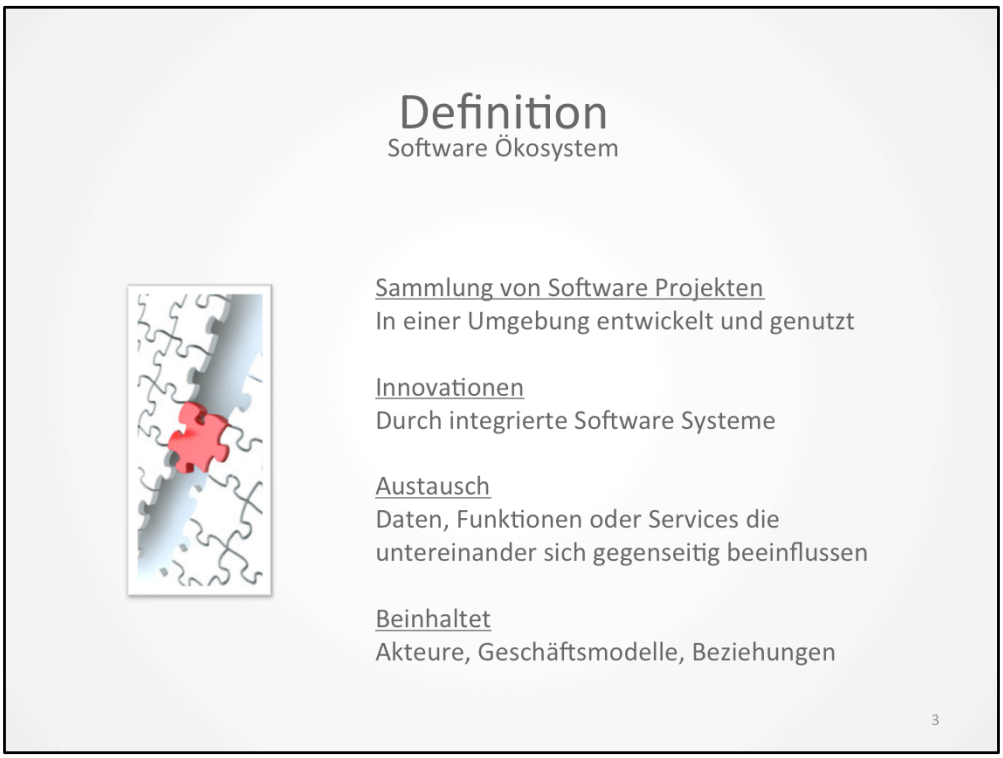
Agenda

Einführung
Definitionen
Beispiele
Zusammenhang

Hauptteil
Architektonische Konzepte
In der Praxis
Anwenungsbeispiele
Plugin-Architektur
Live Coding einer Firefox Erweiterung

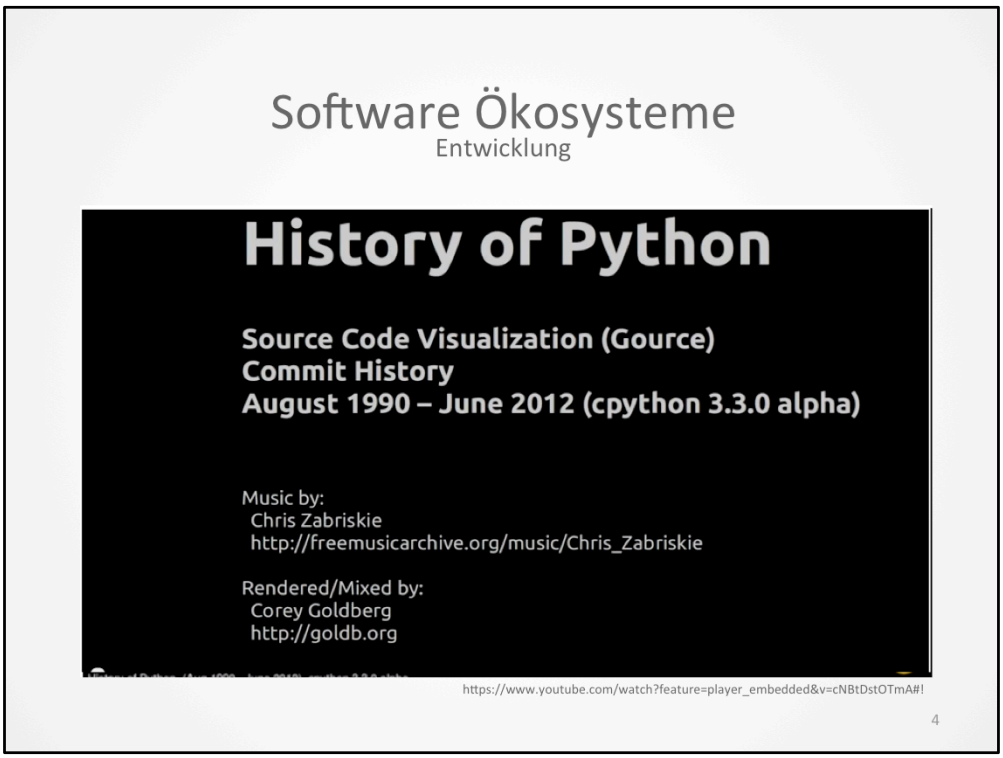
Schluss
Kritische Bewertung
Fazit / Ausblick

2



• Sammlung von Software Projekten die zusammen in einer Umgebung entwickelt und genutzt werden
• Beinhaltet Akteure, Geschäftsmodelle und die Beziehungen zwischen diesen Komponenten
• Eine Sammlung von Geschäftsfunktionen, die als Einheiten auftreten und mit einem Shared Market interagieren
• Besteht aus einer Softwareplattform, interne und externe Entwickler und eine Gemeinschaft von Domain Experten in Services um ein gemeinsames Ziel zu erreichen
• Besteht aus einer Sammlung von Softwarelösungen die die Aktivitäten und Transaktionen der Akteuren ermöglichen, unterstützen und automatisieren in der Organisation und dem Geschäftsumfeld in der sich die Lösungen befinden.

Quelle:[13]

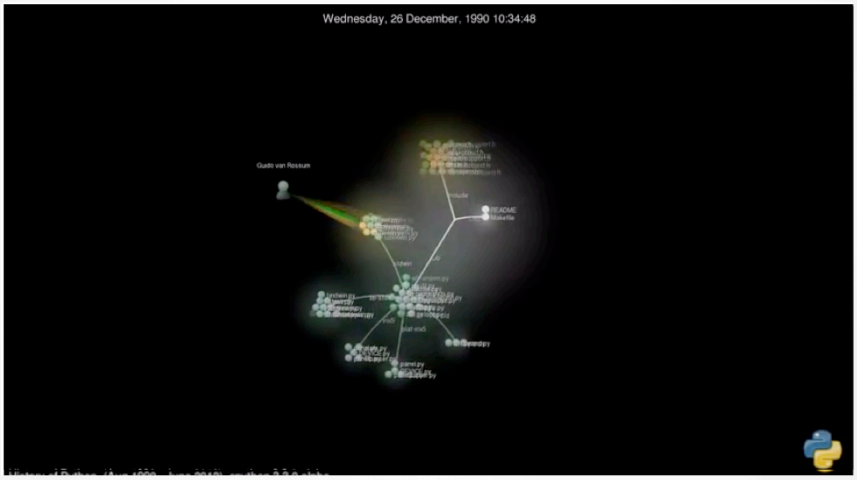


Einführung eines Seco.

Ein stark unterschätzte Eigenschaft eines Secos ist das Wachstum. Hier veranschaulicht durch eine Virtualisierung.

Quelle: [2]

Software Ökosysteme
Entwicklung II



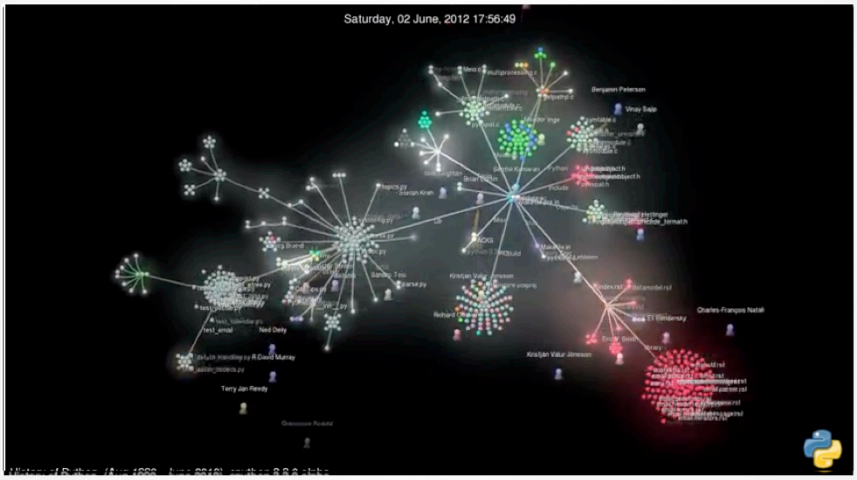
https://www.youtube.com/watch?feature=player_embedded&v=cNBtDstOTmA#!

5

Quelle: [2]

Software Ökosysteme

Entwicklung III



https://www.youtube.com/watch?feature=player_embedded&v=cNBtDstOTmA#!

6

Quelle: [2]

Software Ökosystem

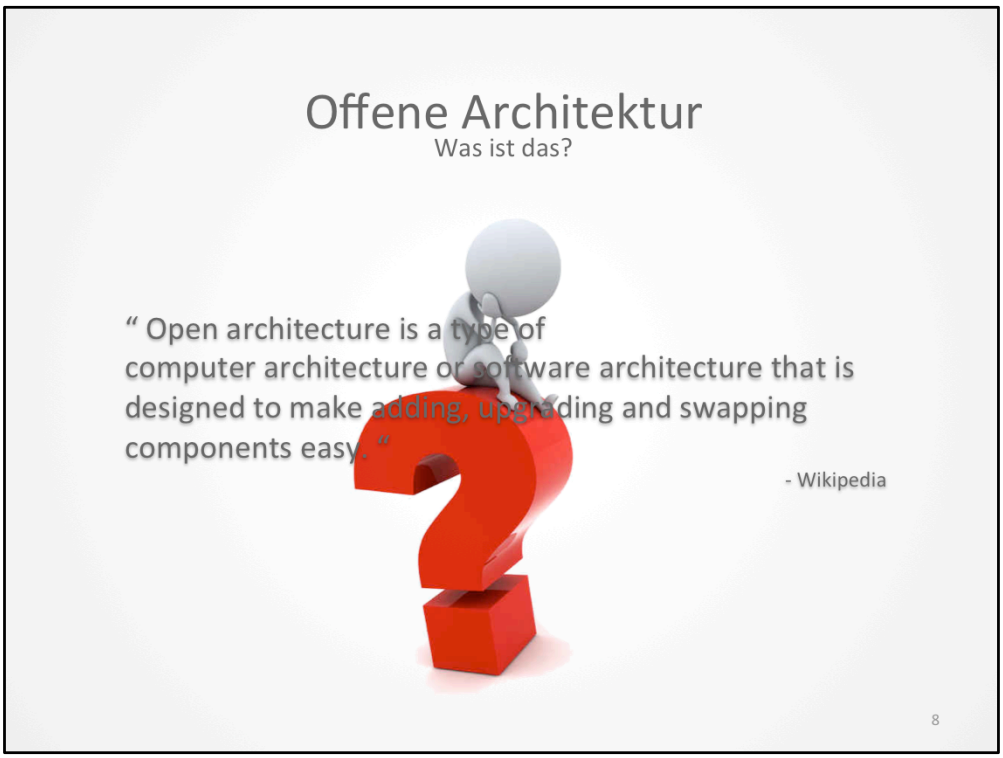
Was ist das?

" A software ecosystem consists of a software platform, a set of internal and external developers and a community of domain experts in service to a community of users that compose relevant solution elements to satisfy their needs. "

- Prof. Dr. Jan Bosch

7

Definition Seco ist wohle jedem klar.



Teilnehmer fragen, wie sie eine offene Architektur definieren würden

Mann muss **zwischen** Offene Architekturen bei **Hardware und Software unterscheiden**. Aber da es um SW-Ökosysteme geht, konzentrieren wir uns auf offene Architekturen bei Software.

Wikipedia Definition zeigen, jedoch reicht uns diese nicht. Auf der anschließenden Folie werden einige für uns wichtige Punkte aufgeführt.

Bei dem grundsätzlichen Design eines Computers kann man sehr viele Unterscheidungen machen, nach der Hardware, Software. Eine sehr wesentliche Entscheidung ist ob eine Architektur geschlossen oder offen ist.

Öffentliche Spezifikationen
- Offizielle Standards
- Private designte Architekturen, deren Spezifikationen veröffentlicht wurden
- Dokumentation sollte/muss vorhanden sein, da sonst keinen Sinn.

Offen:
Jedes Rechnersystem, das diese Prinzipien befolgt, kann in das Netz integriert werden
=> keine Implementierungsvorschrift!
Die Kommunikationssoftware kann in jedem System unterschiedlich implementiert sein, die Protokollschnittstelle muss jedoch eingehalten werden!

Quelle: [14]

Teilnehmer fragen, ob sie Beispiele für offene und geschlossene Systeme kennen
Anschließend unsere Beispiele zeigen, und Teilnehmer entscheiden lassen, zu welcher Kategorie diese gehören

Frage: Warum gibt es überhaupt geschlossene Systeme?
Antwort:
• Kunden abhängig von dem Produkt zu machen
• Produkte die miteinander kommunizieren keine Konkurrenz zu ermöglichen
• Customer 4 Life

Ökosystem Beispiele

Geschlossene Systeme

Apple
iOS
Mac OS

SAP
SAP R/3 und Business Suite

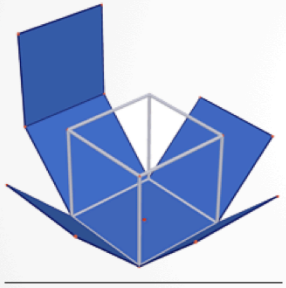
Microsoft
Office
Windows

11

Einige Beispiele

Ökosystem Beispiele

Offene Systeme



Linux
Kernel modules + Abstract APIs

Mozilla Firefox
Plug-ins + Extensions + Themes

Eclipse
Besteht aus OSGi Plugins

Apache
Open Office

12

XML User Interface Language
OSGi Alliance (früher *Open Services Gateway initiative*)

Quelle: [6]

Dieser Artikel soll die historische Entwicklung im PC-Bereich zeigen und einen Ausblick auf die Zukunft geben.

1983 - Apple is so successful in building products with closed architectures when most everyone would agree that open architectures and systems are ultimately the way to go
- ein offenes System teurer in der Produktion gewesen wäre
- Gut dokumentiert

Apple
Der Apple II wurde von Stephen Wozniak als offenes System konzipiert, weil er als Hobbyelektroniker den Rechner so haben wollte, wie er sich einen Rechner wünschte, und nicht wie ein Manager, einen Rechner an dem man möglichst viel verdienen kann, indem man Konkurrenz ausschließt.
-> 8 Erweiterungskartenslots

IBM
Fremdhersteller bauen Erweiterungskarten
Rom-Bios ist geschlossen

1982 - C64
Immerhin war keines der Systeme in der Form geschlossen, als dass die Funktionsweise der Software und Hardware geheim war, sondern es wurde diese offengelegt, nur war es eben nicht ausgelegt, stark erweitert zu werden und die

Apple ist sehr erfolgreich mit geschlossenen Architekturen, allerdings ist sich die meisten Entwickler einig, dass eine offene Architektur eher zum Erfolg führt.

In einer von der EU finanzierten Studie vom Juli 2011 wurden verschiedene Freie-Software-Projekte mit einem Open Governance Index bewertet, Android fiel dabei mit großem Abstand auf den letzten Platz, während Projekte wie MeeGo, Linux oder Eclipse aufgrund ihres offeneren Entwicklungsprozesses punkteten. Android wurde als „the least open mobile open source project“ („das am wenigsten offene Open-Source-Projekt für Mobiltelefone“) bezeichnet.

Der große Vorteil von offenen Architekturen ist, dass jeder Erweiterungen zum System erstellen kann.

Linux hat eine offene Architektur, da der Source Code kostenlos veröffentlicht ist. Im Gegensatz dazu steht DOS, Windows und die Macintosh Architektur und Betriebssystem sind von vorneherein geschlossen.

Quelle:[14]

Ökosystem

Abgrenzung Architekturtypen

Geschlossene Systeme

herstellerspezifischer Produktkonzeptionen

herstellereigenen Spezifikationen und Standards

nicht veröffentlicht und durch Patente oder Lizenzen

Eingeschränkte Interaktionsmöglichkeiten mit "fremden" Systemen

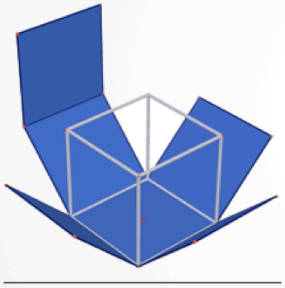
Abhängigkeit des Anwenders vom Hersteller

16

Der oft schwierige Weg, verschiedene geschlossene Systeme in einen großen Systemverbund zu integrieren, führt zu heterogenen Systemen. Für die Übergänge und Berührungsstellen zwischen verschiedenen (geschlossenen) Teilsystemen innerhalb des Gesamtsystemverbundes werden darum oft aufwendige Schnittstellen, Konverter, Umsetzer u.ä. gebraucht, die meist nur einen rudimentären Austausch zwischen den einzelnen Teilsystemen zulassen.

Ökosystem

Abgrenzung Architekturtypen



Offene Systeme

Hinzufügen, upgraden, wechseln von Komponenten vereinfachen

Dokumentierte und freigegebene Schnittstellen

Duplizierbar *

Ausgelegt auf einfache Erweiterbarkeit

Offene Architekturen

Vorteile

Senkt die Gesamtbetriebskosten

Einfache Skalierbarkeit und multinationalen Anwendungen

“Vendor lock-in” wird eliminiert

Einfache Integration

Kostenreduzierung für Käufer

18

Offene Architekturen

Nachteile

Software wird nicht patentiert

Höherer Aufwand für Bereitstellung der Offenheit

Höheres Risiko für Erreichung von Qualitätszielen

19

• Höherer Aufwand für die Bereitstellung der Offenheit und Entwicklung technischer Konzepte dafür
• Höheres Risiko für Erreichung von Qualitätszielen, da Erweiterungen ja die Qualität des Gesamtsystems maßgeblich beeinflussen können

Zusammenhang

Offene Architekturen in Software Ökosystemen

Softwarearchitektur
Eine Softwarearchitektur ist später im Projekt nur mit hohem Aufwand abänderbar

Unterstützung
Offene Architekturen unterstützen Software Ökosystemen z. B. durch öffentliche Schnittstellen

Architekturen bei Ökosystemen
Geben Metastruktur der Systemstruktur vor

20

Bei einfachen Systemen ist Architektur entscheiden für die spätere Qualität

- **Offene Architekturen unterstützen das Erstellen von SW-Ökosystemen damit, dass Sie öffentliche Schnittstellen haben und die Möglichkeit der Erweiterung geben, damit kann durch Zusammenschluss mehrerer Systeme ein Software-Ökosystem entstehen**
- **Architektur gibt Eigenschaften des Ökosystems vor**
- Software Architektur in SW-Ökosystemen gibt die Metastruktur der Struktur des SECO vor
    - Elemente, Eigenschaften der Elemente und die Beziehungen zwischen den Elementen

Quelle: [15]

wichtigen Technologiebereiche und Systemschnittstellen zu fixieren, andererseits darf die Standardisierung auch nicht auf die Spitze getrieben und zum Selbstzweck werden.

neue technologische Trends und deren Stellenwert zu erkennen und rechtzeitig die Standardisierungsarbeit einleiten.

Nicht bestehende Inkompatibilitäten nachträglich überbrücken, sondern sollten die auf einer neuen Technologie basierenden.

Standards sollen Garant für eine langfristig richtige Investitionspolitik der Unternehmen sein und müssen daher einen gewissen Geltungszeitraum haben.

Austauschbarkeit

Durch Unterstützung der Standards und Spezifikationen für Offene Systeme und ihrer Eigenschaft der Interoperabilität lassen sich Teilkomponenten leicht austauschen, um durch neuere, leistungsfähigere oder geeignetere ersetzt zu werden, möglicherweise sogar für den Anwender transparent während des Betriebs.

Skalierbarkeit

Skalierbarkeit bezeichnet die Fähigkeit, Applikationen auf verschiedenen Rechnerleistungsklassen einsetzen zu können und war das Mittel, um im Zuge des Downsizing-Prozesses die vorher oftmals auf Mainframes ablaufenden Anwendungen auf Rechner einer geringeren Leistungsklasse überzuführen. Skalierbarkeit bedeutet also auch Plattformenwechsel ist damit eine Anwendung der Portabilität.viele unterschiedliche Implementierungen ohne gemeinsamen Nenner vor. Die Vereinheitlichung der Systeme an den kritischen Stellen verringert die Integrations- und Ausbildungskosten und schützt getätigte Investitionen in Hardware, Software und User-Know how.

Rightsizing

Aber es ging nicht immer nur darum, Anwendungslösungen auf kleinere Systeme zu portieren, sondern wurde vielmehr auch versucht, der jeweiligen Applikation eine angemessene Ausführungsumgebung bereitszustellen, die am besten geeignet war, den spezifischen Anforderungen der Anwendung gerecht zu werden. Analog zur Skalierbarkeit muß damit auch Rightsizing als eine Form der Portabilität verstanden werden.

Architektonische Konzepte

Standards und Schnittstellen

Investitionsschutz

garantieren den längerfristigen Bestand - sowohl des Standards selbst, wie auch der Hersteller, die ihn unterstützen

beugen der Zersplitterung einer Technologie in viele unterschiedliche Implementierungen ohne gemeinsamen Nenner vor.

Die Vereinheitlichung der Systeme an den kritischen Stellen verringert die Integrationskosten und schützt getätigte Investitionen in Hardware, Software und User-Know-how.

25

Investitionsschutz

Gesicherte Standards und Schnittstellen zwingen die Hersteller, sich daran auszurichten, garantieren den längerfristigen Bestand - sowohl des Standards selbst, wie auch der Hersteller, die ihn unterstützen - und beugen der Zersplitterung einer Technologie in viele unterschiedliche Implementierungen ohne gemeinsamen Nenner vor. Die Vereinheitlichung der Systeme an den kritischen Stellen verringert die Integrations- und Ausbildungskosten und schützt getätigte Investitionen in Hardware, Software und User-Know how.

Offene Architekturen

Protokolle und Standards

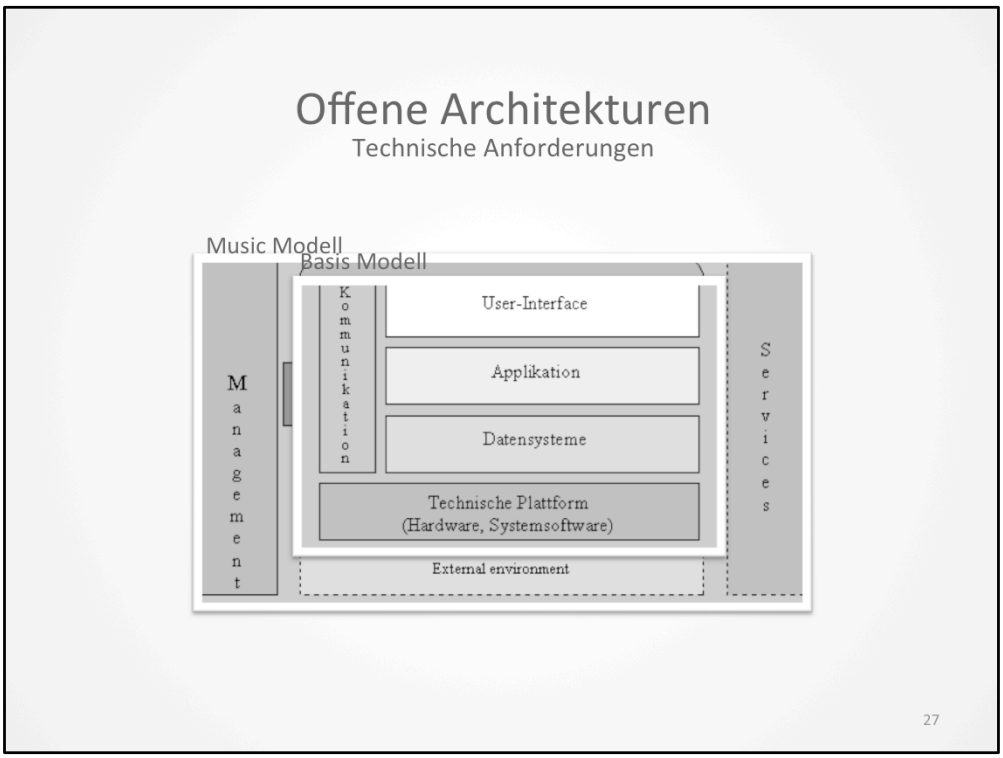
HTTP / HTTPS

SOAP

REST

JSON / XML

26



Music:
eine Entwicklung der CCTA (Central Computer and Telecommunication Agency) Management, User Interface, Systems and Applications, Information and Data und Communication.

Das **Management** schließt Standards für die Systemverwaltung (Ressourcenverwaltung, Monitoring, Systemkonfiguration) ein, die die Netzwerkmanager in die Lage versetzen, auch ausgedehnte heterogene Systeme mit vertretbarem Aufwand noch steuern und kontrollieren zu können.
Der Bereich **User Interface** umfaßt alle Standards für Gestaltung und Einsatz vereinheitlichter Benutzeroberflächen (Style Guides, Software-Ergonomie) auch auf unterschiedlichen Plattformen. Konsistente GUIs mit einheitlichem Look & Feel befähigen den Anwender mit dem Erlernen einer Anwendung auch zum grundsätzlichen Umgang mit anderen Applikationen in bezug auf die Benutzerschnittstelle.
Die Komponente **Systems and Applications** beschreibt die Sammlung der Standards im Bereich der System- und Anwendungssoftware. Dazu gehören Systemschnittstellen, Systemservices und Programmiersprachen. Auf Anwendungsebene zählen dazu etwa APIs, Datenbanksprachen wie SQL oder Graphikstandards wie GKS und PHIGS.
Information and Data faßt Standards zur Repräsentation von Information und Daten

Anwendungsbeispiel

Autosar

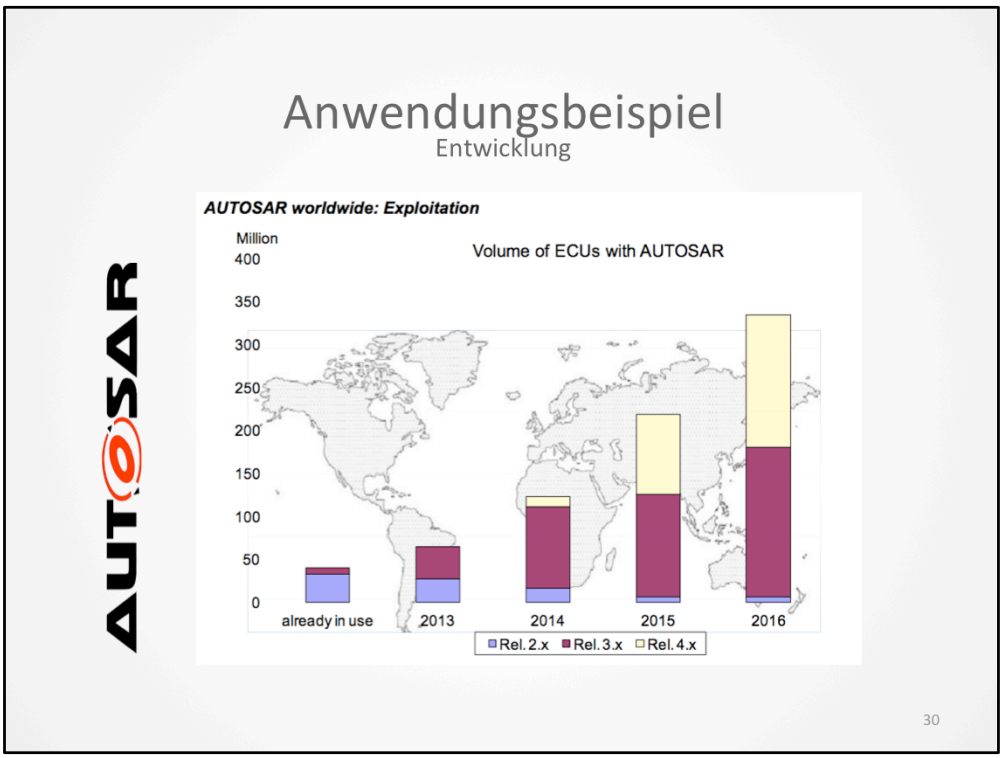
AUTomotive Open System Architecture

Zusammenschluss aus Automobilherstellern, Steuergeräteherstellern und Hersteller von Entwicklungswerkzeugen,

28

Standardisierung domänen-übergreifender Schnittstellen der Anwendungs-Software

Quelle: [5]



AUTOSAR (AUTomotive Open System Architecture)

Quelle: [5]

Anwendungsbeispiel

Ziele

Standardisierung wichtiger Systemfunktionen

Erfüllung zukünftiger Fahrzeuganforderungen bezüglich Verfügbarkeit, Sicherheit und Softwareaktualisierung

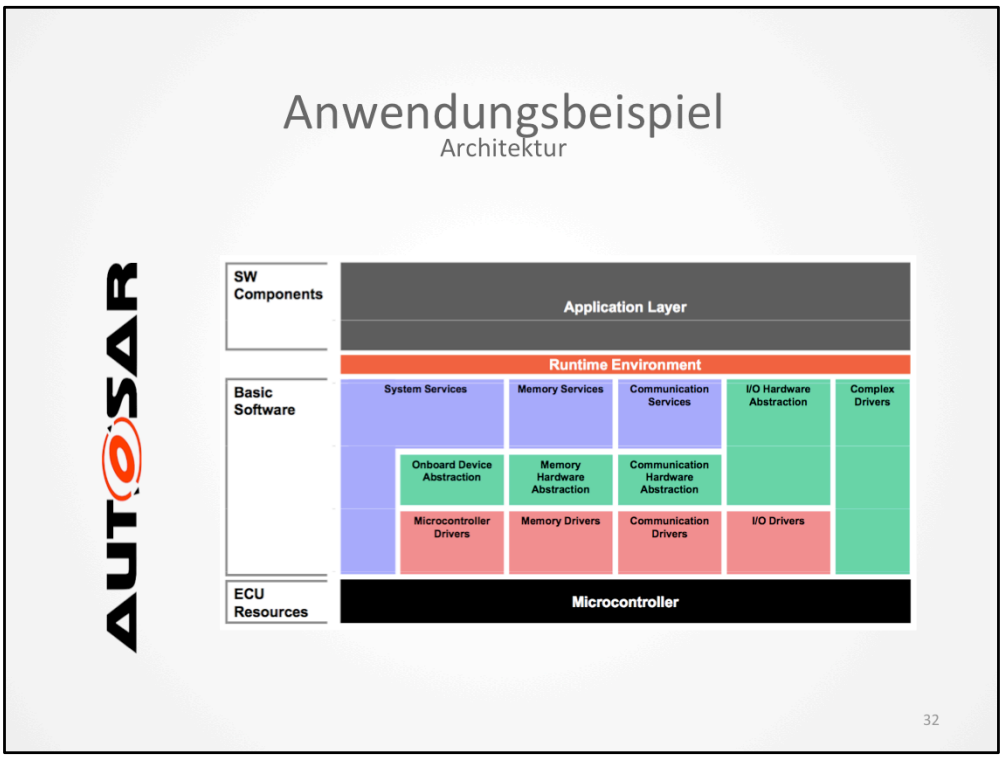
Flexibles Integrieren, Verschieben und Austauschen von Funktionen im Steuergerätenetzwerk

Unterstützung sog. COTS-Software verschiedener Hersteller

Beherrschung der gestiegenen Produkt- und Prozesskomplexität

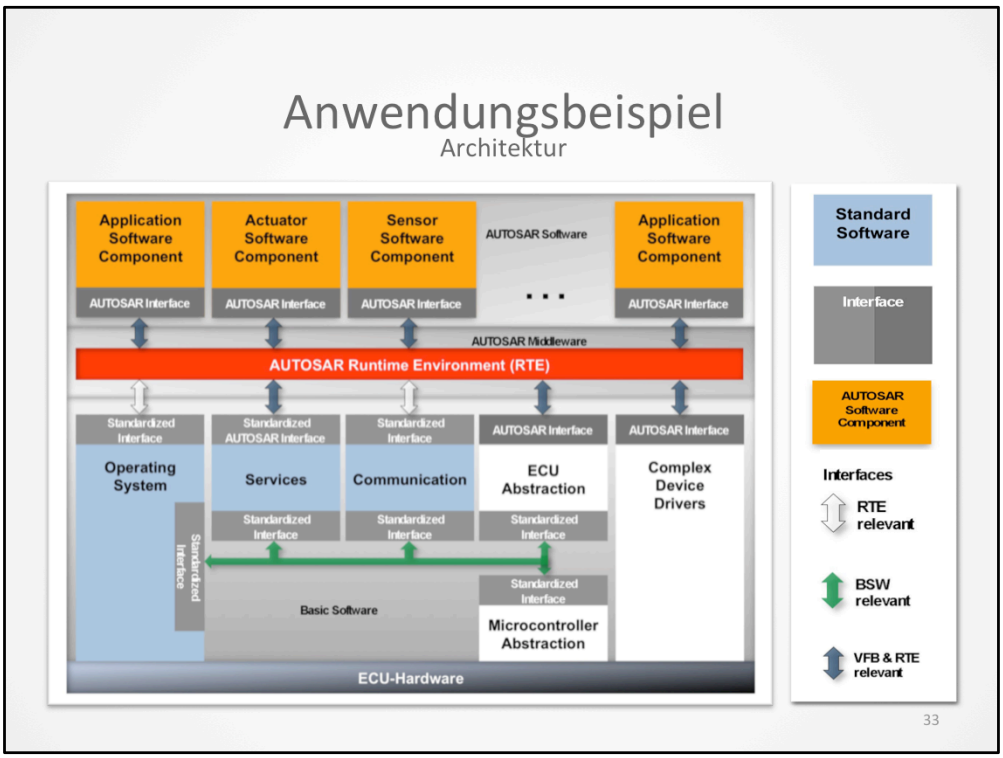
Kostengünstige Skalierbarkeit

Wartbarkeit über den gesamten Produktlebenszyklus



(ECU)Electronic control unit

Quelle: [5]



Modularity

Modularity of automotive software elements enables tailoring of software according to the individual requirements of electronic control units and their tasks.

Scalability

Scalability of functions ensures the adaptability of common software modules to different vehicle platforms to prohibit proliferation of software with similar functionality.

Transferability

Transferability of functions optimizes the use of resources available throughout a vehicle´s electronic architecture.

Re-usability

Re-usability of functions helps to improve product quality and reliability and to reinforce corporate brand image across product lines.

NEXT: 42

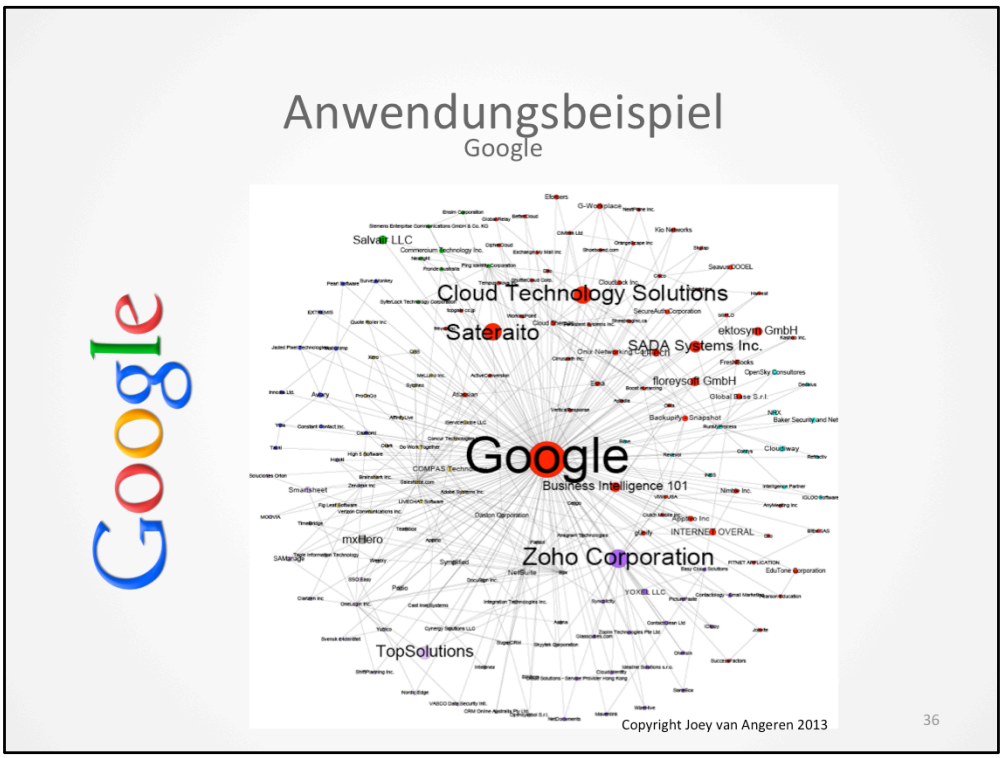
Software Component Description
The operations and data elements that the software component provides and requires.
The requirements which the software component has on the infrastructure.
The resources needed by the software component (memory, CPU-time, etc.).
Information regarding the specific implementation of the software component.

The big 5 benefits
Exchangeability
Reusability
Innovation enabling
Cost reduction
New business

Quelle: [5]

Anwendungsbeispiel

Das Google Ökosystem hat viele frei zugängliche Apis....



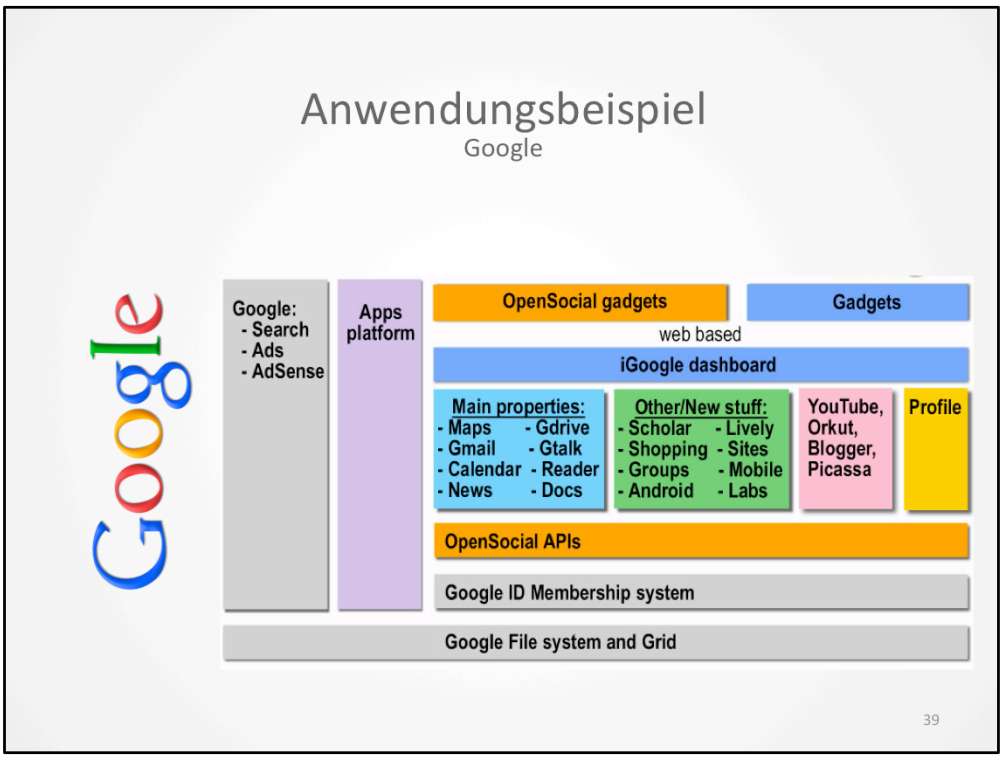
Quelle: [16]

A visualization of the Google Apps ecosystem shows that Google is a strong hub in the Apps ecosystem and few relationships exist between niche players. Google could increase the health of the ecosystem by forging more relationships between partners to improve embeddedness. (Copyright Joey van Angeren 2013).

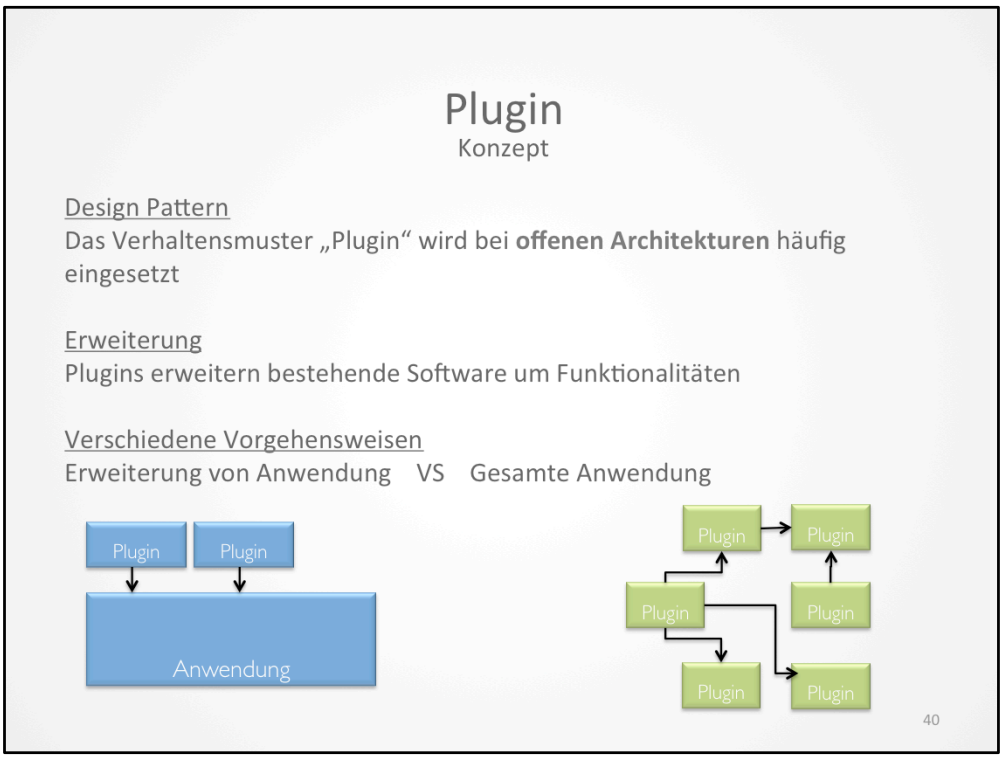
Anwendungsbeispiel
Google

Ist ein Ecosystem aus vielen Anwendungen

Google Code Playground



Ist ein Ecosystem aus vielen Anwendungen



Das **Plugin** gehört zu den Verhaltensmuster.
Es dient der klaren Aufteilung von öffentlichen Teilen der Software (i.d.R. Klassen/ Methoden), die vom Nutzer der Software verwendet werden dürfen und welche die nicht verwendet werden sollen. Mit Hilfe von Plugins lassen sich Programme um Funktionen erweitern. Ein Plugin ist eine selbstständige Software, die eigene Funktionen beinhaltet, Rechnungen und Operationen bereit stellen kann. Dieses kann über eine Schnittstelle, ein Interface, mit dem einem Host-Application verbunden werden gänzlich neue Funktionen zu implementieren.

Quelle: [20]

NEXT:46

Erweiterungen sind lose gekoppelt:
• Es ist nicht erforderlich, dass Erweiterungen vorhanden sind und es ist nicht erforderlich, dass ein Erweiterungspunkt vorhanden ist
Plugins können andere Plugins erweitern:
• Erweiterungen werden registriert
Plugins können andere Plugins benutzen:
• Export- und Import-Schnittstelle
• Abhängigkeiten werden explizit deklariert
• Nutzung eines exportierenden Plugins als eine Bibliothek

Verringerte Speichergröße des Systems:
Z. B. werden nicht alle Tools bei der Browserinstallation benötigt, sondern nur nach Bedarf installiert.

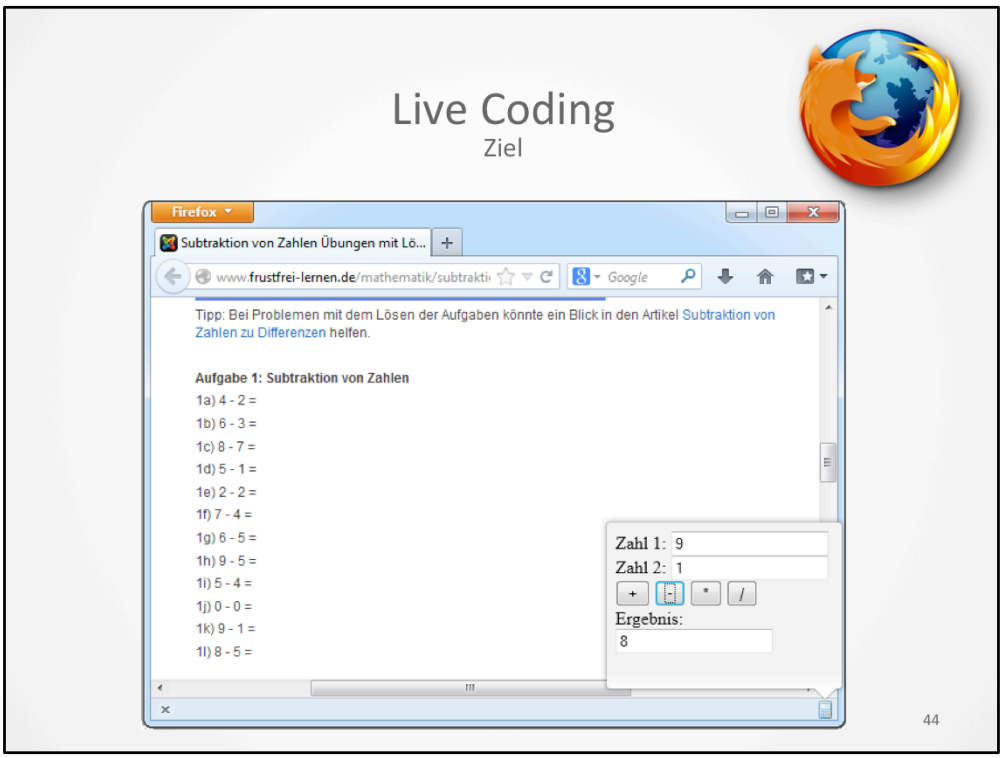
Quelle: [9]

Gefahr der isolierten Entwicklung:
• Parallele Entwicklung ähnlicher Features hat keine direkten negativen Auswirkungen
• Spätere Integration ist problematisch
• Risiko steigt, wenn vorhandene Implementierungen keine (guten) Schnittstellen veröffentlichen: Implementierung wird kopiert, statt Schnittstelle aufwändig an den Bedarf anzupassen

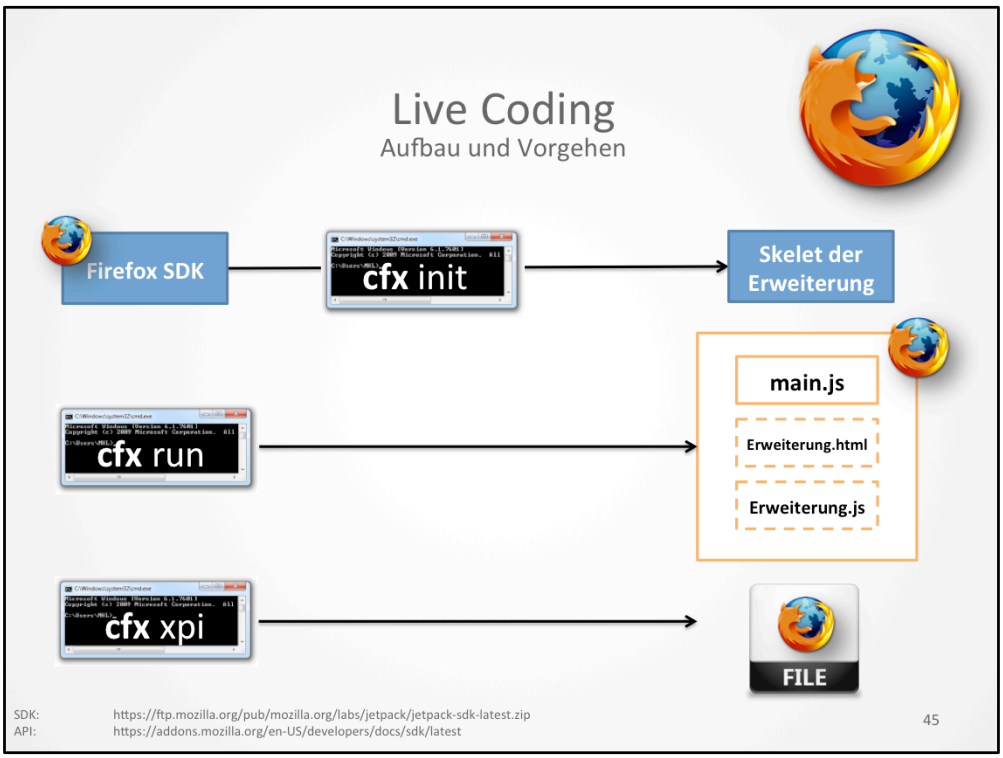
Plugins zwingen zum systematischen Umgang mit Schnittstellen:
• Ohne Veröffentlichung von Schnittstellen ist keine Interoperabilität möglich
• Vorteil: zwingt zum Entwurf von Schnittstellen
• Nachteil: Veröffentlichung von Schnittstellen schränkt die Änderungsmöglichkeiten ein

Lose Kopplung kann zu schwerer verständlichem Code führen:
• Zusammenhänge gehen nicht mehr aus dem Code hervor
• Anforderungen an die Dokumentation sind entsprechend höher
• Debugging ist schwieriger, wenn z. B. Erweiterungen nicht funktionieren

Quelle: [8]



Ziel: Es soll ein kleiner Taschenrechner für den Firefox implementiert werden.



CFX: Ein Kommandozeilenwerkzeug, welches von dem SDK bereitgestellt wird. Um es auszuführen muss Python installiert sein. Es wird unter anderem dazu benutzt, ein Grundgerüst der Erweiterung zu erstellen, die Erweiterung zu Testen und abschließend eine eigene Firefox-Distribution mit der neuen Erweiterung zu deployen.

cfx init: Erstellt eine Skeleton-Struktur für die Erweiterung
cfx run: Führt die Erweiterung in einer Firefox-Instanz aus
cfx xpi: Erstellt eine Firefox-Distribution mit der Erweiterung

Quelle: [21]

Die Main.js Datei im lib-Ordner ist der Einstiegspunkt für die Erweiterung. Vergleichbar mit der main-Methode in Java. Diese wird bei der Installation der Erweiterung ausgeführt und jedes Mal wenn der Browser gestartet wird.
Am Anfang wird eine List von Bibliotheken eingebunden mit dem SDK Befehl REQUIRE():
Die Widget-Bibliothek erstellt das Icon, dass im Browser angezeigt werden soll
Das Data-Objekt von der Bibliothek ist für den Zugriff auf den Data-Ordner verantwortlich. Darin können JS Dateien, Libs, Bilder, Textdateien, HTML Dateien, Stylesheets und weitere Ressourcen liegen.

Durch seinen modularen Aufbau bietet Firefox die Möglichkeit, weitere Funktionen nachzurüsten (z. B. Cookie-Rechtevergabe, Tor-Netzwerk- und Proxy-Server-Anbindungen, Sprachen für die Firefox-Rechtschreibprüfung und zusätzliche Ansichten zum Verändern des Aussehens des Webbrowsers).

Ein Add-on ist ein optionales Modul, welches bestehende Software erweitert. Hierfür werden die vorhandenen Bibliotheken der jeweiligen Anwendung genutzt und um neue Funktionen erweitert. Ein Add-on kann jederzeit entfernt bzw. deinstalliert werden, ohne dabei die Funktionsweise der Hauptanwendung zu beeinträchtigen.
Ein Plug-in hingegen stellt eigene Bibliotheken zur Verfügung, die nicht Teil der Software sind, in die es integriert wird. Mit Hilfe des Plug-ins, und damit des Nutzens fremder Bibliotheken, kann die Software nun neue Funktionen bereitstellen, die nicht Bestandteil des Kerns der Software waren (z. B. die Webbrowser-Plug-ins für PDF, QuickTime, verschiedene Symbolleisten, Pop-up-Blocker usw.).

Seit Firefox 4 gibt es zwei Möglichkeiten, ein Plugin zu schreiben:
• Traditionell durch XUL Erweiterungen. Diese sind mächtiger, aber auch komplizierter und benötigen einen Neustart nach der Installation.
• Limitierte Erweiterungen, die aber keinen Neustart benötigen.

Quelle: [22]

Live Coding

Dateien

HTML-Seite

```
<html>
  <body>
    Zahl 1: </input id="zahl1"></br>
    Zahl 2: </input id="zahl2„></br>
    </input type="button" id="plus"
    value="+"/>
    </input type="button" id="minus"
    value="-"/>
    </input type="button" id="mal"
    value="*"/>
    </input type="button" id="geteilt
    value="/"/>
    </br>
    Ergebnis: </input id="ergebnis">
  </body>
</html>
```

JS-Verarbeitung

```
self.port.on("show", function (arg) {
var zahl1 = document.getElementById('zahl1');
var zahl2 = document.getElementById('zahl2');
var ergebnis = document.getElementById('ergebnis');
var plus = document.getElementById('plus');
var minus = document.getElementById('minus');
var mal = document.getElementById('mal');
var geteilt = document.getElementById('geteilt');

plus.onclick = function(event) {
ergebnis.value = parseFloat(zahl1.value) + parseFloat(zahl2.value);
};
minus.onclick = function(event) {
ergebnis.value = parseFloat(zahl1.value) - parseFloat(zahl2.value);
};
mal.onclick = function(event) {
ergebnis.value = parseFloat(zahl1.value) * parseFloat(zahl2.value);
};
geteilt.onclick = function(event) {
ergebnis.value = parseFloat(zahl1.value) / parseFloat(zahl2.value);
};
});
```

47

Live Coding
Dateien

Main JS des Plugins

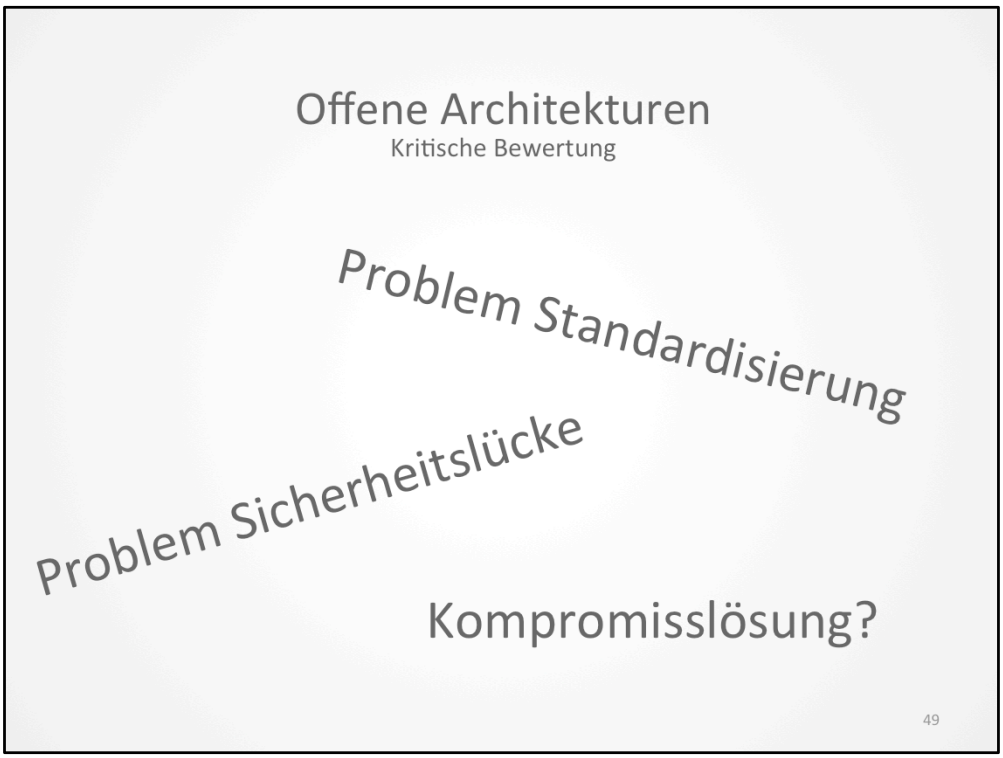
```
var data = require("sdk/self").data;

var calculatorPanel = require("sdk/panel").Panel({
          width: 212,
          height: 150,
          contentURL: data.url("calculatorPage.html"),
          contentScriptFile: data.url("calculate.js")
});

calculatorPanel.on("show", function() {
          calculatorPanel.port.emit("show");
});

require("sdk/widget").Widget({
          label: "Calculator",
          id: "calculator",
          contentURL: data.url ("calculator-icon.ico"),
          panel: calculatorPanel
});
```

48



Probleme Offener Architekturen
So sehr die Vorteile und positiven Wirkungen Offener Architekturen auch überzeugen, ganz ohne potentielle Gefahren und Schwierigkeiten ist der Übergang zu ihnen nicht.

Problem der Standardisierung
Offene Architekturen beruhen auf Standards und können erst durch ein umfassendes, konsistentes Standardisierungswerk ihre Vorzüge entfalten. Das Unix-Beispiel zeigt aber, dass durch diese Bewegung dennoch eine Vielfalt an UNIX-Derivaten gibt, die ihre eigene Standards rausgebracht haben.
Der Wettbewerb zwischen den verschiedenen Systemen fördert ohne Zweifel den technologischen Fortschritt und garantiert die ständige technische Erneuerung und Weiterentwicklung der Systeme, die Vereinheitlichung aber bleibt dabei auf der Strecke.

Problem einer Sicherheitslücke
Eine andere Problematik, die sich direkt aus der Natur der Offenen Architekturen ergibt und unmittelbar der Standardisierung und Vereinheitlichung der Systeme entspringt, ist die schwierigere Gewährleistung der Systemsicherheit. Je offener das System, je bekannter und berechenbarer Systemkomponenten, Systemverhalten und verwendete Sicherheitsmechanismen sind, desto mehr Angriffsfläche bieten solche

Offene Architekturen

Fazit

Ruf
Kann durch schlechte Erweiterungen beschädigt werden

Zukunftspotenzial
Auch bei geschlossene Systemen vorhanden

Zusammenfassend
Offene Architekturen unterstützen SW-Ökosysteme

Abhängig von dem Ziel was verfolgt wird, ist es förderlich/hinderlich eine offene Architekturen zu haben

50

NEXT: 59

Ruf kann beschädigt werden:
System mit schlechten Erweiterungen von Drittanbietern kann einen schlechten Ruf auf das eigentliche System werfen (obwohl das eigentliche System nichts für die schlechten Erweiterungen kann).

Zukunftspotenzial:
Grafik

Offene Architekturen unterstützen das Erstellen von SW-Ökosystemen in dem Sie öffentliche Schnittstellen bereitstellen und die Möglichkeit der Erweiterung geben. Damit kann durch Zusammenschluss mehrerer Systeme ein Software-Ökosystem entstehen.

Quellen I

[1] Peyman Oreizy – Open Architecture Software: A Flexible Approach to Decentralized Software Evolution – 2000

[2] Jan Bosch – Architecture Challenges for Software Ecosystems – 2010

[3] Konstantinos Manikas, Klaus Marius Hansen - Software Ecosystems, A Semantic Literature Review – 02/2012

[4] RS2 Technologies White Paper - Open Architecture in Access Control Systems- 09/2008

[5] Autosar - http://www.autosar.org/

[6] Roy T. Fielding, Ph.D. Chief Scientist, Day Software - O'Reilly OSCON Open Architecture

[7] Vorlesung Rechnernetze - Prof. Dr. H. König TU Cottbus

[8] http://agis-www.informatik.uni-hamburg.de/fileadmin/swt/SeminarAusgewaehlteThemen-SoSe2008/2009-01-13-plug-in-architekturen.pdf

[9] https://vsr.informatik.tu-chemnitz.de/proseminare/www01/doku/plugins/plugins/vorteile.html

Quellen II

[10] Clouds brauchen offene Standards - http://www.computerwoche.de/a/clouds-brauchen-offene-standards,1233071 - 25.11.2011

[11] Der Einsatz Offener Systeme in der Praxis - Technologien, Standards, Probleme - https://www2.hs-augsburg.de/informatik/abschlussarbeiten/langfassungen/stempfle-stork-1996/OpenSystems

[12] Tobias Genge, Matthias Kurzer - Offene vs. geschlossene Systeme – Risikoanalyse am Beispiel Android und iPhone OS - FOM Hamburg

[13] Prof. Dr. Peter Knauber, Dr. Jens Knodel - Seminar im Master (MSI) - http://www.informatik.hs-mannheim.de/~knauber/MSc-MSI-13/Einf%FChrung.pdf

[14] Definition Open Architecture - http://www.webopedia.com/TERM/O/open_architecture.html

[15] Thomas A. Alspaugh, Hazeline U. Asuncion, Walt Scacchi - The Challenge of Heterogeneously Licensed Systems in Open Architecture Software Ecosystems - http://www.ics.uci.edu/~wscacchi/Papers/New/SEAMBNSI-AlspaughAsuncionScacchi.pdf

[16] Slinger Jansen - http://elgarblog.wordpress.com/2013/05/22/on-the-necessity-of-software-ecosystem-analysis-can-any-software-company-do-without-by-slinger-jansen/ - May 22, 2013

Quellen III

[17] https://code.google.com/apis/ajax/playground/

[18] Bernd Leitenberger - Computergeschichte(n): Die ersten Jahre des PC – 2012

[19] History of Python - Gource - development visualization https://www.youtube.com/watch?feature=player_embedded&v=cNBtDstOTmA#!

[20] Software Ecosystems A Sytematic Literature Review - Konstantinos Manikas and Klaus Marius Hansen 02/2012

[21] https://addons.mozilla.org/en-US/developers/docs/sdk/latest/dev-guide/cfx-tool.html

[22] https://developer.mozilla.org/en-US/docs/Building_an_Extension

Vielen Dank

Offene Architekturen in Software Ökosystemen

Michael Hassel, Dominik Feininger