Architektonische Herausforderungen in Software-Ecosystemen

Michael Strotz

Friedrich-Alexander-Universität Erlangen-Nürnberg

22. November 2012

1. Perspektive auf das Software Ecosystem (SECO)
2. Erstellung und Weiterentwicklung der SECO-Architektur
   - Plattformanalyse
   - Öffnung der Plattformarchitektur vorbereiten
   - Transparenz und Modularität ausbalancieren
     - Transparenz und Modularität
     - Schnittstelleninstablität
     - Schnittstellenkomplexität
     - Aktivitätsbewusstsein
     - Schnittstellentransluzenz
   - Weiterentwicklung der Architektur
3. Zusammenfassung

Perspektive auf das SECO

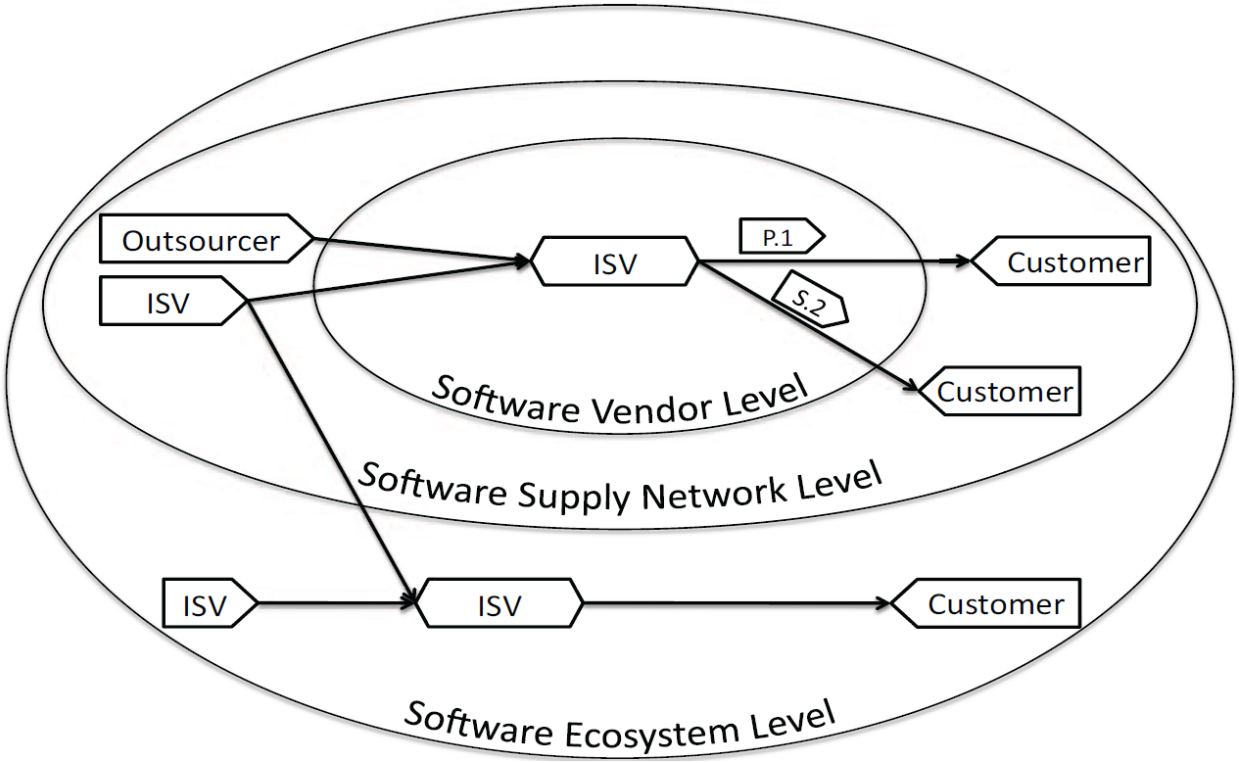


Abbildung : „Software Ecosystem Perspectives“ [5]

- Organisation ←
- Software-Liefernetz (SSN) ←
- Software-Ecosystem (SECO)
- Architektur ←
- geschäftliche Dimension
- soziale Dimension

1. Perspektive auf das Software Ecosystem (SECO)

2. Erstellung und Weiterentwicklung der SECO-Architektur
   - Plattformanalyse
   - Öffnung der Plattformarchitektur vorbereiten
   - Transparenz und Modularität ausbalancieren
     - Transparenz und Modularität
     - Schnittstelleninstablität
     - Schnittstellenkomplexität
     - Aktivitätsbewusstsein
     - Schnittstellentransluzenz
   - Weiterentwicklung der Architektur

3. Zusammenfassung

Erstellung und Weiterentwicklung der SECO-Architektur

- Umbau einer Plattform zum SECO
- Umgang mit den Herausforderungen eines SECO
- Erstellung der Architektur in drei Schritten (nach [2])
  - Plattformanalyse
  - Öffnung der Plattformarchitektur vorbereiten
  - Transparenz und Modularität ausbalancieren
- Weiterentwicklung der Plattform

Plattformanalyse

- Rollen identifizieren
  - Hubs
    - ★ Keystones
    - ★ Dominators
  - Niche Players
    - ★ Influencer
    - ★ Hedger
    - ★ Disciple
- Vitalität analysieren
  - Produktivität
  - Stabilität
  - Nischenschöpfung

Öffnung der Plattformarchitektur vorbereiten

- entspricht Design-Phase
- Vorgehen:
  - Abstraktionsebenen festlegen
  - Integrationsstufen festlegen
  - Rechte und Pflichten definieren
- Herausforderungen:
  - Integration (fremder Komponenten)
  - Sicherheit und Verlässlichkeit

Abstraktionsebenen festlegen

- Module und Abstraktionsebenen identifizieren
- Beispiel:

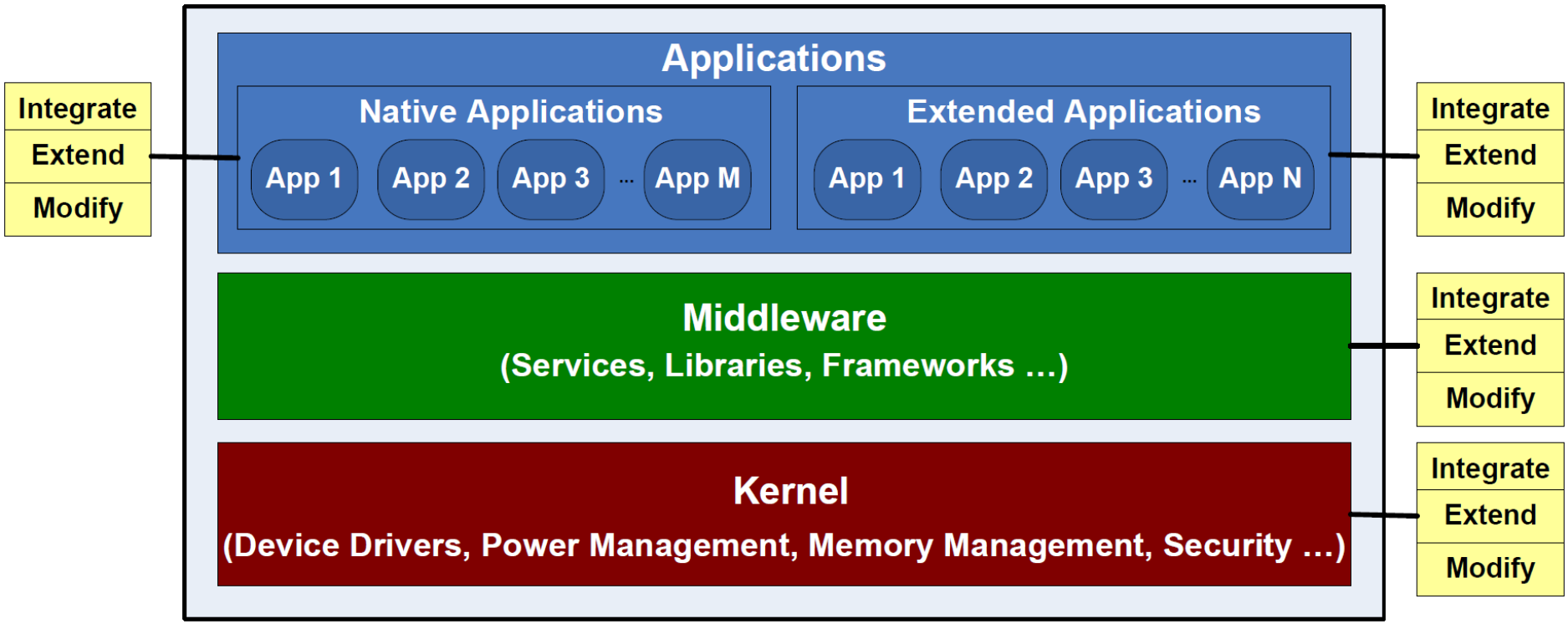


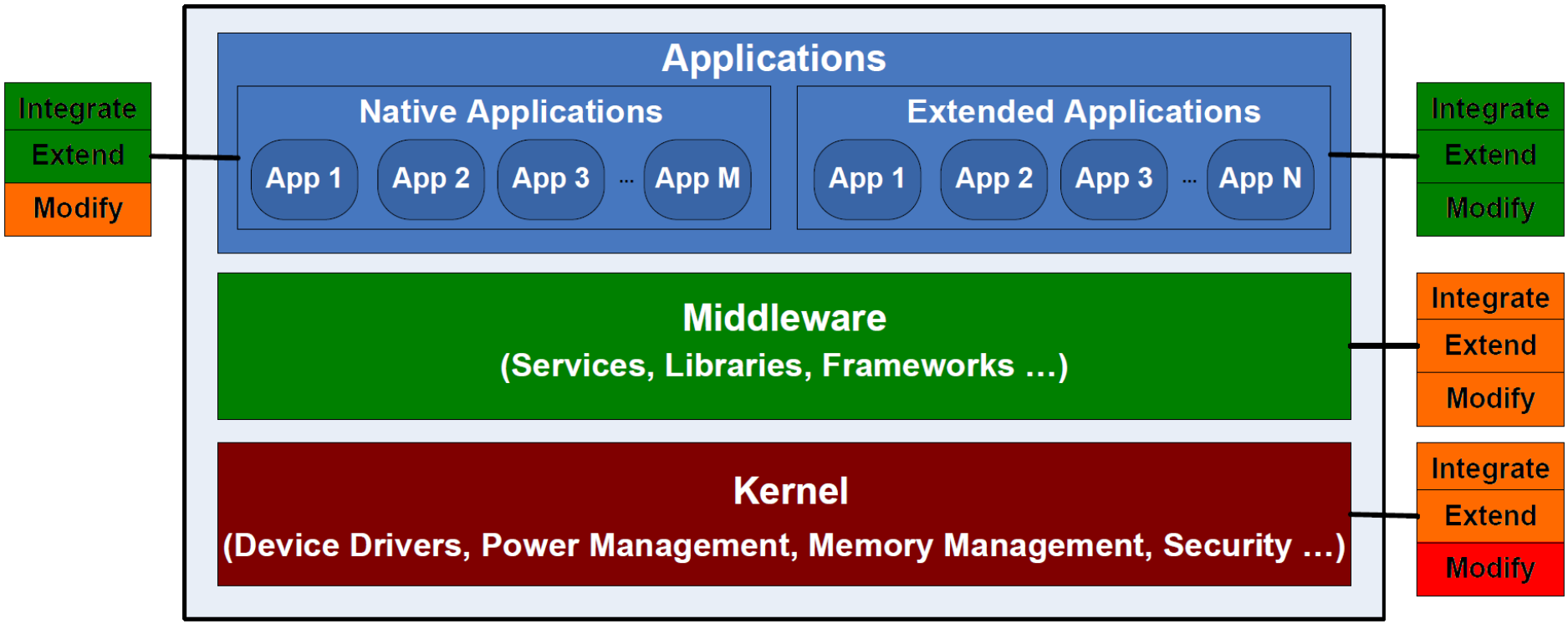
Abbildung : „Architectural Openness Model for Mobile Software Platforms“ [3]

Integrationsstufen festlegen

- Zugriffsmöglichkeiten auf verschiedene Ebenen explizit festlegen
- Integrationsstufen (Beispiel):
  - „integrate“: Kann vorhandene Komponenten der Ebene aufrufen
  - „extend“: Kann sich in Komponenten der Ebene integrieren
  - „modify“: Kann Komponenten der Ebene ersetzen oder modifizieren

Rechte und Pflichten definieren

- Aktoren durch Rechte und Pflichen steuern
- Zertifizierungsprozesse definieren
- Abhängig von:
  - Ebene
  - Integrationsstufe
- Beispiel:



1. Perspektive auf das Software Ecosystem (SECO)

2. Erstellung und Weiterentwicklung der SECO-Architektur
   - Plattformanalyse
   - Öffnung der Plattformarchitektur vorbereiten
   - Transparenz und Modularität ausbalancieren
     - Transparenz und Modularität
     - Schnittstelleninstablität
     - Schnittstellenkomplexität
     - Aktivitätsbewusstsein
     - Schnittstellentransluzenz
   - Weiterentwicklung der Architektur

3. Zusammenfassung

Transparenz und Modularität

- Transparenz
  - erlaubt Entwicklern die Plattform besser zu verstehen
  - kann externe Entwickler überfordern
  - beschränkt durch Sicherheit und Schutz geistigen Eigentums
- Modularität
  - erleichtert Arbeitsteilung
  - schlechte Unterstützung unerwarteter querschneidender Belange
  - Entwicklern bleiben nützliche Informationen verborgen

$\Rightarrow$ Transparenz und Kapselung abgewägen

Schnittstelleninstablität

- unerwartete Änderungen senken die Vitalität des SECO
- Besipiel [6]:
  - „The sample code, in the tutorial, gives me too many errors. So I am not sure, if I build the code properly or some thing is wrong with the tutorial.“
  - „Read a lot of source code. Clang is a moving target. If you are writing tools based on clang, then you need to recognize that clang is adding and fixing features daily [...] read a lot of code!“

Schnittstelleninstablität

$\Rightarrow$ Wahrscheinlichkeit einer Schnittstellenänderung kommunizieren
  - möglichst frühzeitig
  - Entwickler können sich vorbereiten
- Bewertung der Wahrscheinlichkeit:
  - Erfahrung der Architekten
  - performance-kritische Komponenten
  - Projekthistorie

Schnittstellenkomplexität

- komplexe Schnittstellen...
  - verhindern eine Standardisierung
  - schrecken neue Entwickler ab
  - verursachen Benutzungsfehler
- Beispiel:

```
int futex(int *uaddr, int op, int val,
          const struct timespec *timeout, int *uaddr2, int val3);
      vs
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Schnittstellenkomplexität

- Komplexität kommunizieren
- auf alternative Schnittstellen verweisen
- Bewertung der Komplexität:
  - Signatur (Typen, Anzahl der Parameter)
  - Contraints (Pre-/Post-Conditions)

Aktivitätsbewusstsein („Activity Awareness“ [4])

- interne Designentscheidungen einer Komponente können andere Komponenten beinflussen
  - $\Rightarrow$ wichtige Designentscheidungen kommunizieren
  - $\Rightarrow$ Evolution der Komponente dokumentieren
- Beispiele:
  - Contraints (Pre-/Post-Conditions)
  - verwendete Ressourcen
  - verwendete Kommunikationsmechanismen
  - evtl. Algorithmen/Komplexität

Schnittstellentransluzenz („Interface Translucence“ [4])

- vereint Vorteile von Transparenz und Modularität
- Transparenz bestimmter Schlüsselinformationen
  - Schnittstelleninstabilität („Uncertainty“)
  - Schnittstellenkomplexität („Complexity“)
  - Aktivitätsbewusstsein („Activity Awareness“)

```
/*
* Foo: performs some operation
* @param p1
* @uncertainty URI_to_computation
* 'some assessment of the uncertainty of the foo() API'
* @complexity URI_to_computation
* 'some assessment of the complexity of the foo() API'
* @labormetric URI_to_computation
* 'some assessment of the development activity in the
* implementation of the foo() API'
*/
Void foo(int p1) {
      // implementation of the API
}
```

Abbildung : „javadoc-style implementation of Interface Translucence“ [4]

1. Perspektive auf das Software Ecosystem (SECO)
2. Erstellung und Weiterentwicklung der SECO-Architektur
   - Plattformanalyse
   - Öffnung der Plattformarchitektur vorbereiten
   - Transparenz und Modularität ausbalancieren
     - Transparenz und Modularität
     - Schnittstelleninstablität
     - Schnittstellenkomplexität
     - Aktivitätsbewusstsein
     - Schnittstellentransluzenz
   - Weiterentwicklung der Architektur
3. Zusammenfassung

Weiterentwicklung der Plattform

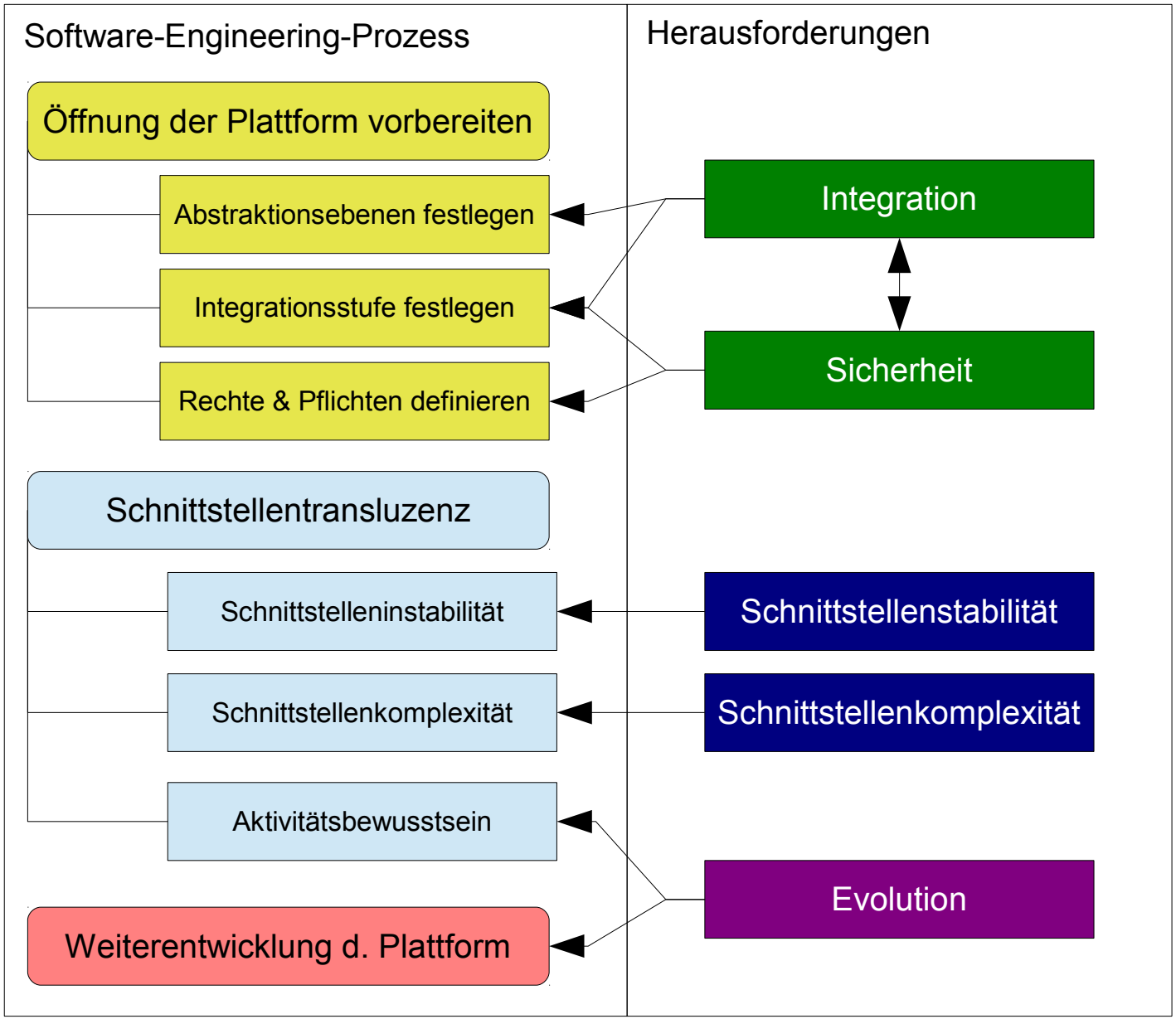
- neue Funktionalität einbinden
  - Anwendungsbereich der Plattform ausweiten
  - neue Schnittstellen frühzeigtig bekannt geben
- Plattform entschlacken
  - gesteigerte Komplexität senkt Produktivität
  - Anwendungsbereich eingrenzen
  - Veralteten Code entfernen
- Schritte zur Erstellung der Architektur wiederholen
- Schnittstellen zu Industriestandards entwickeln

1 Perspektive auf das Software Ecosystem (SECO)

2 Erstellung und Weiterentwicklung der SECO-Architektur
- Plattformanalyse
- Öffnung der Plattformarchitektur vorbereiten
- Transparenz und Modularität ausbalancieren
  - Transparenz und Modularität
  - Schnittstelleninstablität
  - Schnittstellenkomplexität
  - Aktivitätsbewusstsein
  - Schnittstellentransluzenz
- Weiterentwicklung der Architektur

3 Zusammenfassung

Zusammenfassung




J. Bosch, „Architecture challenges for software ecosystems“ in Proceedings of the Fourth European Conference on Software Architecture Companion Volume - ECSA '10, 2010, p. 93


R. Pereira, C. Maria, and L. Werner, „A Proposal for Software Ecosystems Engineering“ in Proceedings of the Workshop on Software Ecosystems 2011, 2011, pp. 40-51.


M. Anvaari and S. Jansen, „Evaluating Architectural Openness in Mobile Software Platforms“ in Proceedings of the Fourth European Conference on Software Architecture Companion Volume - ECSA '10, 2010, p. 85.


M. Cataldo and J. D. Herbsleb, „Architecting in Software Ecosystems: Interface Translucence as an Enabler for Scalable Collaboration“ in Proceedings of the Fourth European Conference on Software Architecture Companion Volume - ECSA '10, 2010, p. 65.


S. Jansen, A. Finkelstein, and S. Brinkkemper, „A sense of community: A research agenda for software ecosystems“ in 2009 31st

International Conference on Software Engineering - Companion Volume, 2009, pp. 187-190.

http://stackoverflow.com/questions/5130695/how-to-make-use-of-clangs-ast (Abgerufen: 21.11.2012-19:32, Erstellt: 27.02.2011)