LE06: Management der Anwendung

Moritz Schüll Simon Müller Frederic Forster
Georg Moosbrugger Philipp Winder

29.05.2018

Definition. *Anwendung/Application: Software-System mit Domänen-Bezug im Unternehmen; dies können generische Dinge wie Mail-Software sein, aber auch Komplexes wie z.B. Produktionssteuerungen.*

1 Sollte sich jeder für Open Source entscheiden?

- Wenn alle bei Closed Source Unternehmen kaufen, führt dies zu einer Monopolstellung, dies führt dann zu weniger Innovation und höheren Kosten → Alternativen am Markt hochhalten (OpenSource Business Alliance)
- Lizenzmodelle haben einen großen ökonomischen Einfluss auf eine Organisation (Beispiel TUM online)

2 Management des Anwendungszyklus

Remark. *Funktionalität: Implementierte Requirements die aufgerufen werden können und einen Nutzen erfüllen.*

2.1 Funktionalitätsbedarf und -angebot bei Standardsoftware

Anforderungen an Software ändern sich, d.h. es gibt manchmal mehr Anforderungen als Funktionalität und umgekehrt (→ dynamisch)

- A → die Software umfasst zu viele Funktionen
- B → die Funktionalität der Software reicht nicht mehr aus, da sich die Anforderungen ändern
- C → die Funktionalität ist höher als die Anforderungen
- D → die Anforderungen und die Funktionalität stimmen überein

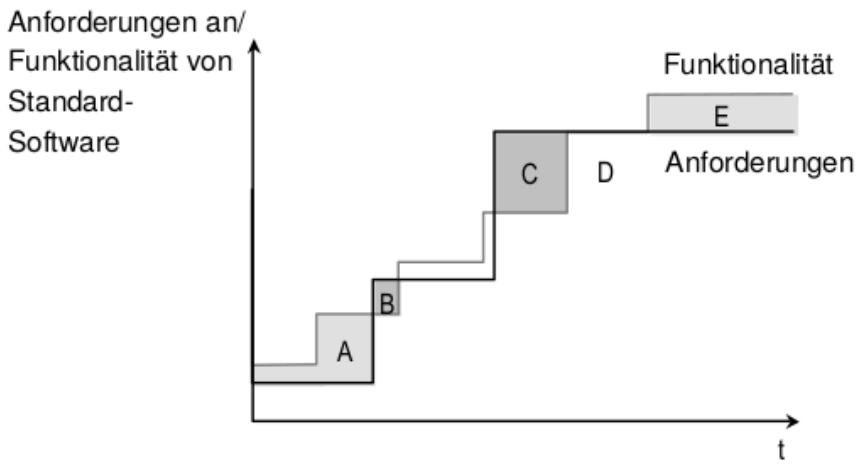

Abbildung 1: Funktionalitätsbedarf und -angebot bei Standardsoftware

2.2 Anpassung von Standardsoftware

Ist ein laufender Vergleich, wie sich die eigenen Funktionalitätsbedürfnisse im Vergleich zum -angebot der Standardsoftware entwickeln.

2.3 Einführung von Software

Modellorientierte Einführung von Standardsoftware (→ Referenzmodell), wobei es sich um ein Auswahlproblem mit Anpassung handelt.

2.4 Anforderungen an Software

- Funktionalität
- Parametrisierbarkeit: nicht nur Anpassung der Oberfläche/UI → Anpassung der Datenbasis
- Kompatibilität zur installierten Basis ist keine Aussage über die Güte der vorhandenen Basis, sondern dient lediglich dazu die Komplexität in der IT niedrig zu halten
- Benutzerfreundlichkeit
  - Aufgabenangemessenheit
  - Selbstbeschreibungsfähigkeit
  - Steuerbarkeit
  - Erwartungskonformität: System verhält sich so, wie der Benutzer es erwarten würde
  - Fehlerrobustheit

- Effizienz
- Sicherheit
- Anschaffung: wie lange dauert es, bis ich die Software kriege (Vertragsverhandlungen können länger dauern, nicht immer nur ein einfacher Download)
  - Kaufpreis: Rabatte, etc.
  - Wartung: Hohe jährliche Kosten
- Anbieter: kann der Anbieter Service-Requests bearbeiten? Wie lange wird es den Anbieter vermutlich noch geben?
  - Nutzungsdauer von Software im Unternehmen ca. 10-15 Jahre, da für die Einführung neuer Software hohe Investitionen getätigt werden müssen (insbesondere für Schulungen der Mitarbeiter). → eine neue Software zu kaufen ist i.d.R. nicht das Problem, sondern der organisatorische Aufwand zur Einführung.

Remark. *Open Source-Teil übersprungen von incl. Folie 12 bis excl. Folie 15*

2.5 Alternativen der Software-Bereitstellung

- Purchase
  - Buy
  - Rent - ist teurer, da Risiko über Verwendung mit Anbieter geteilt wird
- Make
  - Fremdentwicklung
  - Eigenentwicklung
  - Anwender (End-/Power-/User-Computing) - sich seine Anwendungen z.B. in Excel selbst zusammenbauen zu können → sehr individuell & bringt Problematik ausreichend guter Dokumentation mit sich

2.6 Software-Lizenzmodelle

Wie soll die für die Software zu bezahlende Summe ermittelt werden? Ständiges Hin- und Her → unterschiedliche Modelltypen und primäre Bezugsgrößen

- **Verschenken:** kostenlose Abgabe der Software & anschließend kostenpflichtige Beratung und Support
- **Anzahl der Nutzer:** Wer sind die definierten Nutzer? Was passiert, wenn mehrere Menschen dieselbe Nutzer-Identität in der Software verwenden? (einen Übernutzer (z.B. Online-Zugang) der von mehreren verwendet wird)

- **Personalbestand oder Herstellungskosten der verkauften Produkten**
- **Dauer der Nutzung:** 365 Tage für x Euro
- **Ausmaß der Nutzung der genutzten Infrastruktur:** z.B. pro Prozessor Kern
- Beispiel SAP: Verschiedene Szenarien für verschiedene Lizenzmodelle
  - Gemeinsamer Zugriff mehrerer User (z.B. Krankenstation mit 2 Rechnern und 15 KrankenpflegerInnen) → Pro-Device-Lizenz
  - Alternativ: Smartphone für alle Mitarbeiter auf der Station → Pro-Device-Lizenz vermutlich nicht mehr optimal

Kriterien für die Bewertung von Lizenzmodellen

- Geringe Initialkosten → Subskriptions-Lizenzmodell
- Testphase → userbezogene Lizenzierung
- Gemeinsamer Zugriff mehrerer User → Pro-Device-Lizenzen
- Flexible Infrastruktur → infrastrukturbezogenen Software-Lizenzmodell
- Regionale oder branchenspezifische Besonderheiten → wertbasiertes Lizenzmodell
- Application Service Provider oder Cloud Provider → Sercice-Provider-Lizenzmodelle

2.6.1 onlineTED-Frage

Sie sind der CIO einer großen Marketingfirma. Bei bestimmten Marketingkampagnen benötigen Ihre Mitarbeiter eine spezielle Standardsoftware zur Videobearbeitung. Welches Lizenzmodell bevorzugen sie?

A) Nutzerbezogene Modelle

B) Wertbezogene Modelle

C) Zeitbezogene Modelle

D) Infrastrukturbezogene Modelle

→ Keine richtige Antwort, je nach Auslegung ist ein anderes Lizenzmodell besser.
Es gibt zwar die Möglichkeit das optimale Lizenzmodell zu errechnen, aber dies ändert sich über die Zeit.

2.6.2 Lebenszyklus-Modell

- Systemkosten: in Entwicklung hoch und nehmen anschließend ab. Auf 0 zu bringen schwierig/unmöglich (Beispiel: 30-jährige Softwarehaltbarkeit)
- Systemnutzen: benötig einige Zeit bis Nutzen entsteht und Entwicklungskosten anfangen gedeckt zu werden. → Ziel: Vorlaufzeit zu verringern → profitieren durch kürzere Entwicklungszeit
- Generelles Problem: Software wird bei Wartung laufend erneuert, wohingegen bei Wartung von Hardware lediglich der Ursprungszustand wiederhergestellt wird. → Das Herausnehmen/Ersetzen von Hardware ist deutlich komplizierter und kostenintensiver als von Software

Remark. *Nach Folie 23 war fertig, weiter mit Gastvortrag von .msg systems.*

3 Gastvortrag: Msg Systems AG

Open Source Software im Unternehmenseinsatz von Dipl.-Inf. Univ. Ralf S. Engelschall (Leiter msg Applied Technology Research). Seit 25 Jahren im Open Source Bereich persönlich tätig (Mitbegründer der Apache Foundation)
Betriebliche Software kann heute ohne (massivem) Einsatz von Open Source-Lösungen kaum mehr wirtschaftlich entwickelt werden.

Von Open Source spricht man dann, wenn die Software konform zu den folgenden Kriterien ist:

- Free redistribution
- Source code availability
- Allowance of derived work
- No discrimination against persons or groups
- No discrimination against fields of endeavor
- Distribution of License
- License must not be specific to a product
- License must not restrict other software
- License must be technology-neutral

3.1 10 Mythen von Open Source

1. **Open Source Software wird von einer Community entwickelt:** Stimmt nicht, nur wenn man unter Community einen oder wenige Entwickler sieht. Die, die entwickeln sind sehr wenige, viele quatschen mit, reporten Bugs etc..

2. **Open Source Software wird von Nerds erstellt:** Sind oft professionelle Software Engineers, die enthusiastisch eine technologische Leadership (beste Technologie etc.) erlangen wollen.

3. **Open Source Software einzusetzen ist trivial:** Wenn man denkt, dass man sich nur um den Download kümmern muss. Support/Wartung oder ähnliches zu beziehen ist schwierig. Aufwand der Integration ist groß.

4. **Open Source Software kostet nichts:** (Üblicherweise) in der Anschaffung, aber dafür eventuell mehr in der Wartung! → muss selbst Wartung vornehmen

5. **Auf IT Konferenzen trifft man echte Open Source Entwickler:** Aber nur, falls sie ausnahmsweise einen Vortrag über ihre eigene Software halten sollten. Dort sind nur User und Politiker, aber nicht die Personen die hinter der Software sind.Es wird zwischen drei verschiedenen Persönlichkeitsgruppen rundum Open Source Software unterschieden: Es gibt Software Sharing (Dogmatism, Social Equity, Politics), Software Hacking (Fundamentalism, Art, Hacking), Software Engineering (Pragmatism, Business, Engineering). Nur Software Hacker entwickeln Software (sind wenige).

6. **In der Software Industrie kann man virale GPL-Lizensierte Software nicht nutzen:** Nur falls mit Nutzen das Embedding in kommerzielle Produkte gemeint ist. Gegenbeispiel: Linux Kernel, welcher unter GPL 3.0 lizenziert ist. Verwenden und Einbauen ist ein großer Unterschied.

7. **Es gibt gute und schlechte Open Source Lizenzen:** Nein, es gibt nur gute/schlechte Einsatzmöglichkeiten → License Compliance Cheking Meta-Model
   Welche Lizenz soll gewählt werden? Strong Copyleft (e.g. GPL), Weak Copyleft(e.g. MPL, LGPL), No Copyleft(e.g. MIT, Apache) - diese Frage ist nicht trivial.

8. **Man kann mit Open Source Software Geld verdienen:** Falls man ein Venture Capital basiertes Geschäftsmodell verwendet, aber im klassisch deutschen Modell des Verkaufens nicht. Services für Open Source anzubieten ist oft nicht sehr rentabel.

9. **Open Source Entwickler werden oft von der Industrie bezahlt:** Für andere Tätigkeiten und erhalten das ok nebenbei auch Open Source Entwicklung zu betreiben.

10. **Jeder kann bei Open Source Projekten mitentwickeln:** Als Freiberufler, in der abgegrenzten Freizeit oder wenn die Schöpfungshöhe nicht sehr hoch ist - sonst nicht! → Klausel im Arbeitsvertrag zur rechtlichen Absicherung.