Software Engineering II (IB)

Softwareevolution

Prof. Dr. Oliver Braun

Fakultät für Informatik und Mathematik
Hochschule München

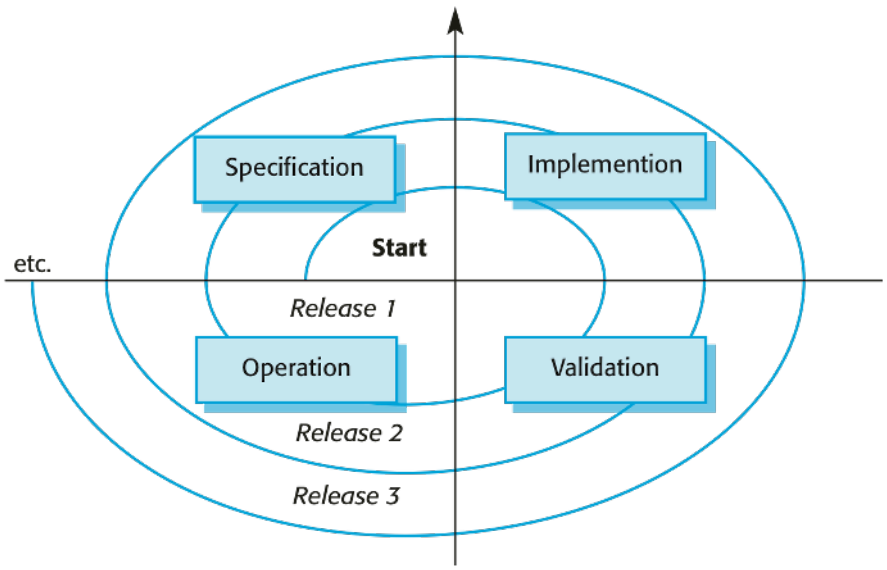
Letzte Änderung: 07.04.2019 10:40

Inhaltsverzeichnis

Softwareveränderungen

- Veränderungen der Software sind unvermeidbar
  - neue Anforderungen
  - Geschäftsumgebung verändert sich
  - Fehler müssen behoben werden
  - neue Hardware wird in das System eingebracht
  - die Performanz oder Zuverlässigkeit muss erhöht werden
- Veränderungen an bestehender Software sind ein großes Problem

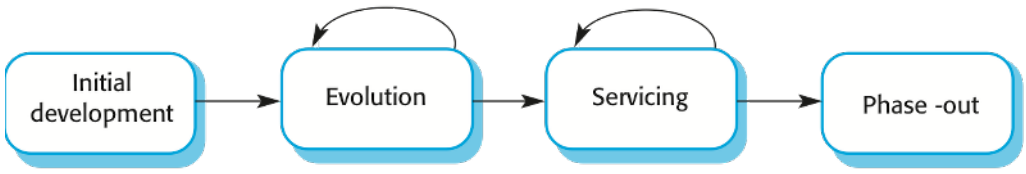
Spiralmodell von Entwicklung und Evolution



Spiralmodell (Quelle: I. Sommerville: Software Engineering)

Evolution und Instandhaltung

Alternativer Ansatz der, sobald die Umsetzung neuer Anforderungen immer weniger kosteneffizient ist, von der Evolution zur Instandhaltung übergeht:

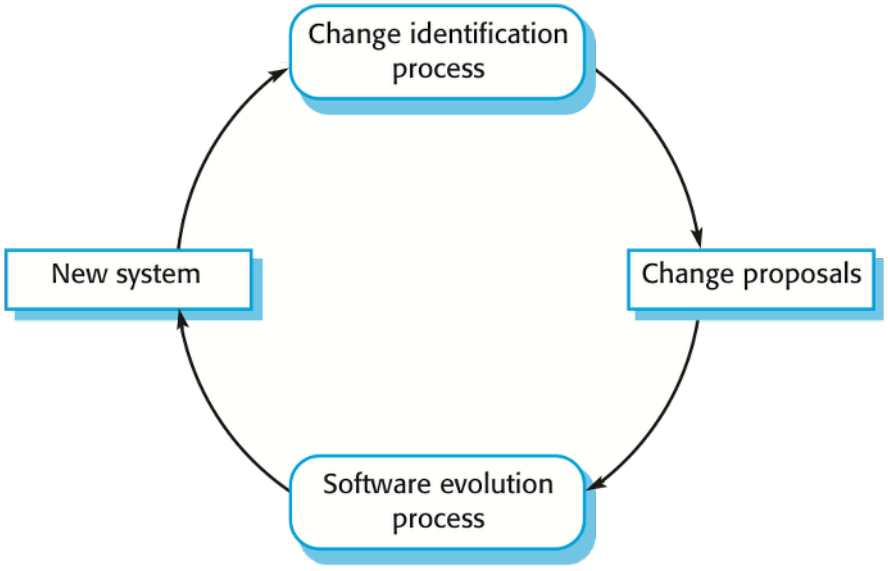


Evolution und Instandhaltung (Quelle: I. Sommerville: Software Engineering)

Evolutionsprozesse

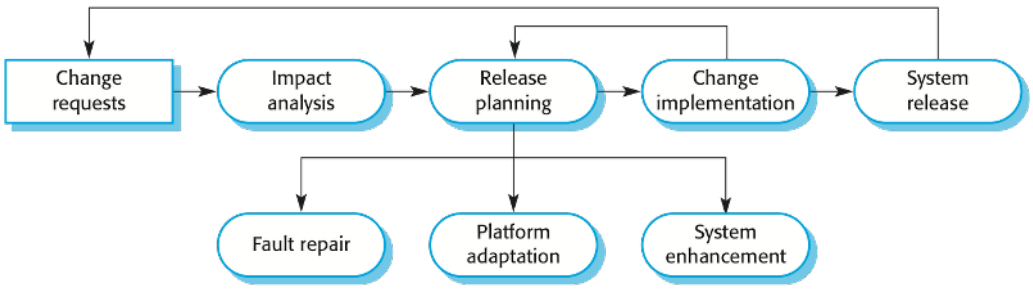
- abhängig von
  - Art der Software
  - Entwicklungsprozess
  - Fähigkeiten der beteiligten Personen
- Vorschläge für Systemänderungen sind der Hauptantrieb für die Weiterentwicklung
- setzen sich über den gesamten Lebensdauer eines Systems fort

Identifizieren von Änderungen und Evolutionsprozesse



Identifizieren von Änderungen und Evolutionsprozesse (Quelle: I. Sommerville: Software Engineering)

Softwareevolutionsprozess



Der Prozess der Softwareevolution (Quelle: I. Sommerville: Software Engineering)

Agile Methoden und Evolution

- Entwicklung kann direkt in Evolution übergehen
- automatisiertes Testen ist dabei sehr sinnvoll einsetzbar
- Änderungen werden einfach als zusätzliche User Stories bzw. als neue Anforderungen in das Product Backlog aufgenommen

Dynamik der Programmevolution

- Untersuchung von Systemänderungen
- nach einigen empirischen Studien sind die eine Reihe von Regeln, *Lehmans Gesetze*, entstanden
- diese sind wahrscheinlich gültig für alle große betriebliche Softwaresysteme

Lehmans Gesetze

Kontinuierliche Veränderung In einer realen Umgebung muss sich ein System unvermeidlich verändern. Sonst verliert es nach und nach an Nutzen.

Zunehmende Komplexität Struktur wird mit Veränderungen immer komplexer. Zusätzliche Ressourcen für Erhalt und Vereinfachung der Struktur.

Evolution umfangreicher Programme Programmevolution ist selbstregelnder Prozess. Systemeigenschaften wie Größe, Anzahl gemeldeter Fehler, … bleiben vergleichbar groß.

Organisatorische Struktur Über die Lebensdauer bleibt Entwicklungsrate annähernd konstant.

Lehmans Gesetze — Fortsetzung

Bewahrung der Vertrautheit Umfang der vorgenommenen Änderungen für jede neue Version nahezu konstant.

Kontinuierliches Wachstum Funktionsumfang muss kontinuierlich wachsen für Kundenzufriedenheit.

Abnehmende Qualität Die Qualität wird abnehmen, bis sie modifiziert wird, um Veränderungen in der Betriebsumgebung widerzuspiegeln.

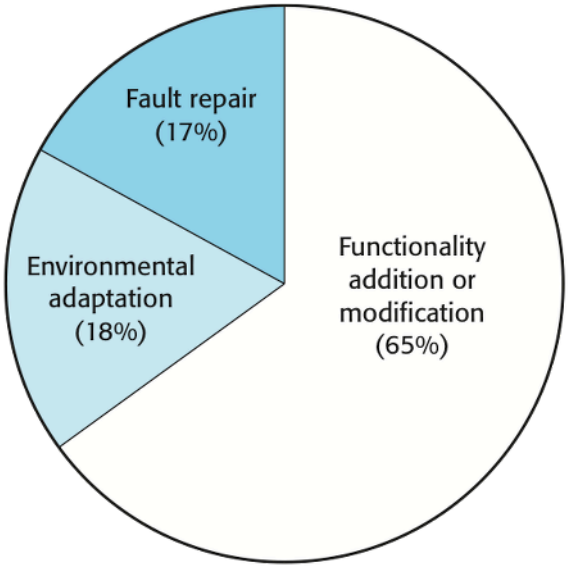
Umgang mit Rückmeldungen Müssen systematisch behandelt werden um messbare Produktverbesserung zu erreichen.

Softwarewartung

- Fehlerbehebung
  - z.B. Programmierfehler, Fehler im Entwurf, Fehler in Anforderungen
- Anpassung an eine Umgebung
  - z.B. neue Hardware, anderes Betriebssystem, …
- Hinzufügen von Funktionalität
  - wenn sich Systemanforderungen auf Grund von organisations- oder geschäftsbedingten Verändnerungen wandeln

Verteilung des Wartungsaufwands

Wartungskosten i.d.R. ein Vielfaches der Entwicklungskosten.



ungefähre Aufteilung der Wartungskosten (Quelle: I. Sommerville: Software Engineering)

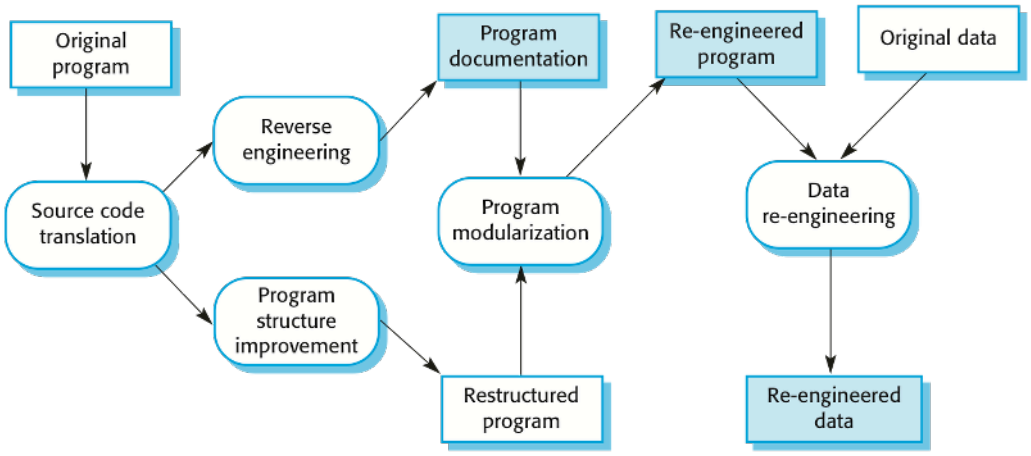
Kostenfaktoren Wartung

- Stabilität des Teams
- Schlechte Entwicklungspraxis
  - Wartungsvertrag oft unabhängig von Entwicklungsvertrag
- Mitarbeiterfähigkeiten
- Alter und Struktur des Programms

Software-Reengineering

- Restrukturieren oder neu schreiben von Teilen des Systems
- sinnvoll, wenn einige, aber nicht alle, Subsysteme regelmäßig gewartet werden müssen
- Vorteile
  - verringertes Risiko gegenüber einer Neuentwicklung
  - geringere Kosten gegenüber einer Neuentwicklung

Der Reengineering-Prozess



Reengineering-Prozess (Quelle: I. Sommerville: Software Engineering)

Aktivitäten im Reengineering-Prozess

1. Übersetzung des Quellcodes
   - mit Tool in neuere Programmiersprache oder in andere übersetzen
2. Reverse Engineering
   - Analyse des Programms
3. Verbesserung der Programmqualität
   - um es besser lesen und verstehen zu können
4. Modularisierung
   - kann Refactoring der Architektur erfordern
5. Daten-Reengineering
   - kann Veränderung der Datenbankschemata u.ä. erforderlich machen

Präventive Wartung durch Refactoring

- Verbesserungen am Programm vornehmen um der Degradierung durch Änderungen vorzubeugen

- Struktur verbessern
- Komplexität reduzieren
- Verständlichkeit erhöhen

Unterschied Reengineering und Refactoring

- Reengineering
  - nachdem ein System eine Weile gewartet wurde und
  - die Wartungskosten steigen
- Refactoring
  - findet kontinuierlich während des gesamten Entwicklungs- und Evolutionsprozesses statt
  - soll die Struktur- und Codedegradierung vermeiden helfen, die Wartung teuer macht

Refactoring und Agile Methoden

- Refactoring ist fester Bestandteil
- Änderungen stehen im agilen Prozess im Mittelpunkt
- daher häufiges Refactoring

Bad Smells

- Martin Fowler et al. haben Situationen (*bad smells*) identifiziert in denen der Code verbessert werden kann
- dazu gehören
  1. Code-Duplizierung
  2. Lange Methoden
  3. Switch-Case-Anweisungen
     - enthalten häufig doppelten Code
  4. Datenklumpen
     - gleiche Gruppe von Datenelementen (Felder, Parameter) wiederholt an mehreren Stellen
  5. Spekulative Generalisierung
     - Superklassen, die nur dafür da sind, falls es mal erweitert werden würde

Primitive Refactoring-Transformationen

- Vorschläge von M. Fowler
- Beispiele
  - Extraktion von Methoden
  - Konsolidierung bedingter Ausdrücke
  - Hochziehen von Methoden in Superklasse

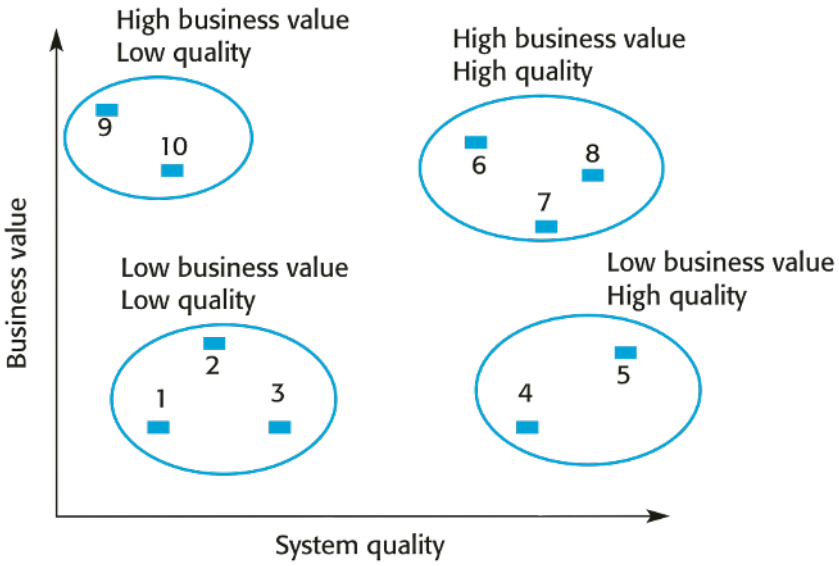
Verwaltung von Altsystemen

- Verständnis, dass Entwicklung und Betrieb/Wartung nicht zu trennen ist, wächst
- aber es sind immer noch viele geschäftskritische Altsysteme im Einsatz
- diese müssen aber auch erweitert werden und z.B. an E-Business angepasst werden
- oft nur begrenztes Budget für die Wartung vorhanden

Strategien

1. Das ganze System ausmustern.
   - wenn das System keinen effektiven Beitrag zu den Geschäftprozessen leistet
2. Das System unverändert lassen und regelmäßig warten.
   - wenn System recht stabil und wenig Änderungswünsche vorhanden sind
3. Das System per Reengineering sanieren, um seine Wartbarkeit zu erhöhen.
   - wenn Systemqualität schlecht und weiterhin Änderungswünsche vorhanden
4. Das System ganz oder teilweise durch ein neues System ersetzen.
   - wenn System, z.B. auf Grund neuer Hardware, nicht in Betrieb bleiben kann

Beispiele für die Beurteilung eines Altsystems



Beurteilung Altsystem (Quelle: I. Sommerville: Software Engineering)

Konsequenzen

1. Niedrige Qualität, niedriger Geschäftswert
   - sollten ausrangiert werden
2. Niedrige Qualität, hoher Geschäftswert
   - Reengineering
3. Hohe Qualität, niedriger Geschäftswert
   - weiter betreiben, solange es wenig kostet
4. Hohe Qualität, hoher Geschäftswert
   - normale Systemwartung weiterführen

Geschäftswert eines Systems beurteilen

mit den Systembeteiligten (z.B. Endbenutzer und Manager) über folgende Themen sprechen

- die Benutzung des Systems:
  - wird es oft oder selten genutzt? ...

- die unterstützen Geschäftsprozesse
  - Festhalten an ineffizienten Geschäftsprozessen? …
- die Stabilität des Systems
  - nicht nur technisches, sondern auch Geschäftsproblem! …
- die Systemausgaben
  - hängt das Geschäft von den Ausgaben des Systems ab? …

Technische Qualität

Folgende Daten können bei der Qualitätsbeurteilung nützlich sein:

- Verständlichkeit des Quellcodes
- Dokumentation vollständig? konsistent? aktuell?
- existiert ein explizites Datenmodell? Daten über mehrere Dateien verteilt? Daten aktuell? konsistent?
- Anwendungsleistung adäquat?
- gibt es für die Programmiersprache moderne Compiler? Wird die PL noch verwendet?
- werden alle Versionen/Bestandteile von einem Konfigurationsmanagmentsystems verwaltet?
- liegen Testdaten für das System vor?
- ist das Personal qualifiziert?