

Script generated by TTT

Title: profile1 (01.07.2013)

Date: Mon Jul 01 14:13:27 CEST 2013

Duration: 94:41 min

Pages: 21

Datenstrukturen und Algorithmen

In diesem Kapitel werden einige Klassen von Algorithmen vorgestellt, insbesondere Suchverfahren und Sortierverfahren.

- Fragestellungen des Abschnitts:
 - Welche Möglichkeiten gibt es, Datenmengen im System darzustellen?
 - Welche Möglichkeiten gibt es, in Datenmengen zu suchen?
 - Welche Möglichkeiten gibt es, Datenmengen zu sortieren?
 - Was versteht man unter der Komplexität eines Algorithmus?

[Datenstrukturen](#)
[Suchverfahren](#)
[Sortierverfahren](#)
[Komplexität](#)

Generated by Targeteam

Übliche Komplexitätsklassen

Hauptsächlich verwendete Komplexitätsklassen

- konstant $O(1)$
- logarithmisch $O(\log n)$
- linear $O(n)$
 $O(n) = \text{Komplexität ist z.B. } 3n + 10;$
- überlinear $O(n \log n)$
- quadratisch $O(n^2)$
- polynomial $O(n^k)$, $k > 2$
- exponentiell $O(k^n)$, $k \geq 2$

[Grafische Darstellung](#)

Generated by Targeteam

Interpretierer und Übersetzer

"Interpretierer" (Interpreter): Programmiersystem, das Anweisungen schrittweise zergliedert und Schritte sofort ausführt. Jede Programm-anweisung wird separat betrachtet und Maschinensprache-Unterprogramm für Realisierung der Anweisung aufgerufen.

Alternative: "Übersetzer" (Compiler): wandeln komplettes Programm in Maschinensprache um.

Vorteile von Interpretierern

- Einfacher zu realisieren
- Quellprogramm-bezogenes Testen

Nachteile von Interpretierern

- Ineffizient
- Fehler erst zur Laufzeit entdeckt

Beispiele für Interpretierer(sprachen)

- Basic
- Kommandosprachen von Betriebssystemen, z.B. Shell in DOS Eingabefenster
- Skriptsprachen (z.B. JavaScript)

Generated by Targeteam

Auswahl einer Programmiersprache

Prinzipiell Programmiersprachen gleichwertig: alle Algorithmen formulierbar. Praktische Unterschiede helfen bei Auswahl für Entwicklungsprojekt.

Empfehlungen

wenn dann
einfache Programme, Anwendungserweiterungen	Basic, Visual Basic, Python, Perl
Datenbank-Anwendung und komplexe Programmlogik	C, C++ , Java
Datenbank-Anwendung und einfache Programmlogik	SQL, Reportgenerator
Technisch-wissenschaftliche Anwendung und (Datenbank oder komplexe E/A-Strukturen) und Portabilität	C, C++, Java
(System-Software oder PC-Anwendung) und Portabilität	C, C++, Java
Künstliche Intelligenz-Anwendung	Prolog, LISP
Internet-Anwendung und Portabilität	Java, PHP, Python

Software-Engineering

Softwareerstellung als Ingenieurdisziplin.

Software/Engineering - Definition des Ideals

Die Aufstellung und Befolgung guter Ingenieur-Grundsätze und Management-Praktiken, sowie die Entwicklung und Anwendung zweckdienlicher Methoden und Werkzeuge, mit dem Ziel, mit vorhersagbaren Mitteln, System- und Software-Produkte zu erstellen, die hohe, explizit vorgegebene Qualitätsansprüche erfüllen (nach A. Marco & J. Buxton, 1987)

[Komplexität von Software-Projekten](#)

[Vorgehensmodelle](#)

[Strukturierte Programmierung](#)

[Modellierung](#)

[Modelle für Analyse und Entwurf](#)

Generated by Targetteam

Komplexität von Software-Projekten

Maße für den Umfang von Software

Zahl der Quelltextzeilen der Programme, aus denen das Softwareprodukt besteht (LOC = Lines of Code).

Zeit, die benötigt wird, um eine Programm zu erstellen (Messung in Bearbeiter-Jahre (BJ)).

Klassifikation von Software-Projekten

Projektklasse	Quelltext-Zeilen (LOC)	Bearbeitungsaufwand (BJ)
sehr klein	1 - 1.000	0 - 0,2
klein	1.000 - 10.000	0,2 - 2
mittel	10.000 - 100.000	2 - 20
groß	100.000 - 1 Mio.	20 - 200
sehr groß	1 Mio - ...	200 - ...

Beispiele

Projektklasse	Quelltext-Zeilen (LOC)
Windows XP	ca. 40 Millionen

Spiralmodell

Das **Spiralmodell** ist ein Vorgehensmodell in der Softwareentwicklung, das im Jahr 1986 von Barry W. Boehm beschrieben wurde. Es ist ein generisches Vorgehensmodell und daher offen für bereits existierende Vorgehensmodelle. Das Management kann immer wieder eingreifen, da man sich spiralförmig voran entwickelt.

Inhaltsverzeichnis [Verbergen]

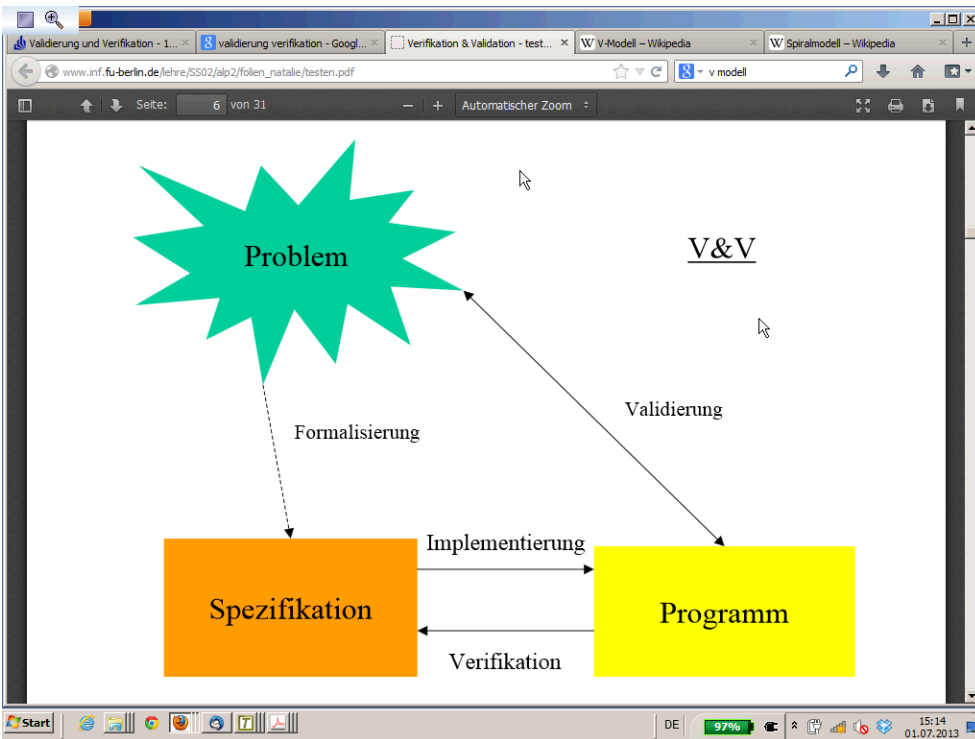
- Beschreibung
- Siehe auch
- Literatur
- Weblinks
- Einzelnachweise

Beschreibung [Bearbeiten]

Das Spiralmodell fasst den Entwicklungsprozess im Software-Engineering als iterativen Prozess auf, wobei jeder Zyklus in den einzelnen Quadranten folgende Aktivitäten enthält:

- Festlegung von **Zielen**, Identifikation von **Alternativen** und Beschreibung von Rahmenbedingungen
- Evaluierung der **Alternativen** und das Erkennen, Abschätzen und Reduzieren von **Risiken**, z. B. durch Analysen, Simulationen oder **Prototyping**
- Realisierung und Überprüfung des Zwischenprodukts
- Planung des nächsten Zyklus der Projektfortsetzung.

Die Risikobetrachtung ist der wesentliche Aspekt, der das Spiralmodell von anderen, zuvor entwickelten Vorgehensmodellen unterscheidet. Dabei werden zunächst alle Risiken, die das Projekt bedrohen, identifiziert und anschließend bewertet. Dann sucht man einen Weg, um das größte Risiko zu beseitigen. Das Projekt gilt als gescheitert, wenn die Beseitigung fehlschlägt. Wenn hingegen keine Risiken mehr existieren, so ist das Projekt erfolgreich abgeschlossen.^[1]



Fehler: Verbindung unterbrochen

Die Verbindung zum Server wurde zurückgesetzt, während die Seite geladen wurde.

- Die Website könnte vorübergehend nicht erreichbar sein, versuchen Sie es bitte später nochmals.
- Wenn Sie auch keine andere Website aufrufen können, überprüfen Sie bitte die Netzwerk-/Internetverbindung.
- Wenn Ihr Computer oder Netzwerk von einer Firewall oder einem Proxy geschützt wird, stellen Sie bitte sicher, dass Firefox auf das Internet zugreifen darf.

[Nochmals versuchen](#)

Wasserfall-Modelle

Entwicklungsprozess einteilen in aufeinander folgende Phasen. Jeweils festgelegt: Ausgangspunkt, Vorgaben, Tätigkeiten, Ergebnisse.

- Problemanalyse und Anforderungsdefinition
- System Definition: fachlicher Entwurf des Datenmodells und des Anwendungsmodells (Funktionen)
- Systementwurf: Festlegung der Struktur der zu entwickelnden Software
- Implementierung: Programmierung und Modultest
- System-Integration und Systemtest
- Installation, Betrieb und Weiterentwicklung

[Graphische Darstellung](#)

Generated by Targeteam

Fehler: Verbindung unterbrochen

Die Verbindung zum Server wurde zurückgesetzt, während die Seite geladen wurde.

- Die Website könnte vorübergehend nicht erreichbar sein, versuchen Sie es bitte später nochmals.
- Wenn Sie auch keine andere Website aufrufen können, überprüfen Sie bitte die Netzwerk-/Internetverbindung.
- Wenn Ihr Computer oder Netzwerk von einer Firewall oder einem Proxy geschützt wird, stellen Sie bitte sicher, dass Firefox auf das Internet zugreifen darf.

[Nochmals versuchen](#)

Validierung und Verifikation - 1... | Validierung verifikation - Goo... | Verifikation & Validation - test... | Seiten-Ladefehler | W' Spiralmodell - Wikipedia

de.wikipedia.org/wiki/Spiralmodell

Benutzerkonto anlegen | Anmelden

Artikel | Diskussion | Lesen | Bearbeiten | Versionsgeschichte | Suchen

WIKIPEDIA

Die freie Enzyklopädie

Navigation
 Hauptseite
 Themenportale
 Von A bis Z
 Zufälliger Artikel

Mitmachen
 Artikel verbessern
 Neuen Artikel anlegen
 Autorenportal
 Hilfe
 Letzte Änderungen
 Kontakt
 Spenden

Drucken/exportieren
 Buch erstellen
 Als PDF herunterladen
 Druckversion

Werkzeuge

Spiralmodell

Das **Spiralmodell** ist ein **Vorgehensmodell** in der **Softwareentwicklung**, das im Jahr 1986 von **Barry W. Boehm** beschrieben wurde. Es ist ein **generisches Vorgehensmodell** und daher offen für bereits existierende Vorgehensmodelle. Das **Management** kann immer wieder eingreifen, da man sich **spiralförmig** voran entwickelt.

Inhaltsverzeichnis [Verbergen]

- Beschreibung
- Siehe auch
- Literatur
- Weblinks
- Einzelnachweise

Beschreibung

Das Spiralmodell fasst den Entwicklungsprozess im **Software-Engineering** als **iterativen** Prozess auf, wobei jeder **Zyklus** in den einzelnen **Quadranten** folgende Aktivitäten enthält:

- Festlegung von **Zielen**, Identifikation von **Alternativen** und Beschreibung von **Rahmenbedingungen**
- Evaluierung der **Alternativen** und das Erkennen, Abschätzen und Reduzieren von **Risiken**, z. B. durch **Analysen**, **Simulationen** oder **Prototyping**
- Realisierung und Überprüfung des **Zwischenprodukts**
- Planung des nächsten **Zyklus** der **Projektfortsetzung**.

Die **Risikobetrachtung** ist der wesentliche Aspekt, der das **Spiralmodell** von anderen, zuvor entwickelten Vorgehensmodellen unterscheidet. Dabei werden zunächst alle **Risiken**, die das **Projekt** bedrohen, **identifiziert** und **anschließend bewertet**. Dann sucht man einen **Weg**, um das **größte Risiko** zu **beseitigen**. Das **Projekt** gilt als **gesehen**, wenn die **Beseitigung** fehlschlägt. Wenn hingegen **keine Risiken** mehr existieren, so ist das **Projekt** **erfolgreich abgeschlossen**.^[1]

Beispiel Getränkeautomat

"Aus", "Wartezustand", "Geld erhalten" und "Getränk gewählt" sind Zustände des Getränkeautomaten.

Generated by Targeteam

Beispiel Getränkeautomat

"Aus", "Wartezustand", "Geld erhalten" und "Getränk gewählt" sind Zustände des Getränkeautomaten.

Generated by Targeteam

Diagrammtypen

Klassendiagramm: spezifiziert die Objektklassen (Entitätstypen) und deren hierarchischen Zusammenhänge.
 z.B. Auto ist eine Unterklasse von Fahrzeug.

statisches Beziehungsdiagramm: modelliert die statischen Beziehungen zwischen Objekten.
 Assoziation: gleichrangige Beziehung zwischen Objekten, z.B. einem Menschen und einer Menge von Büchern, die er liest.
 Aggregation: Zusammensetzung eines Objektes aus einer Menge von Einzelteilen, z.B. eine Stadt hat eine Menge Häuser.

Zustandsdiagramm: modelliert die Zustände von Objekten und wie sie von einem Zustand in den nächsten übergehen.

Anwendungsfalldiagramm (Use Case): zeigt den Zusammenhang zwischen Anwendungsfällen und den daran beteiligten Akteuren.
 z.B. eine Arztpraxis hätte die Akteure Arzt, Arzthelfern und Patient sowie die Anwendungsfälle: Patient anmelden, Diagnose stellen und Abrechnung.

[Sequenzdiagramm](#)

Generated by Targeteam

Diagrammtypen

Klassendiagramm: spezifiziert die Objektklassen (Entitätstypen) und deren hierarchischen Zusammenhänge.
z.B. Auto ist eine Unterklasse von Fahrzeug.

statisches Beziehungsdiagramm: modelliert die statischen Beziehungen zwischen Objekten.

Assoziation: gleichrangige Beziehung zwischen Objekten, z.B. einem Menschen und einer Menge von Büchern, die er liest.

Aggregation: Zusammensetzung eines Objektes aus einer Menge von Einzelteilen, z.B. eine Stadt hat eine Menge Häuser.

Zustandsdiagramm: modelliert die Zustände von Objekten und wie sie von einem Zustand in den nächsten übergehen.

Anwendungsfalldiagramm (Use Case): zeigt den Zusammenhang zwischen Anwendungsfällen und den daran beteiligten Akteuren.
z.B. eine Arztpraxis hätte die Akteure Arzt, Arzthelfern und Patient sowie die Anwendungsfälle: Patient anmelden, Diagnose stellen und Abrechnung.

[Sequenzdiagramm](#)

Generated by Targeteam

Sequenzdiagramm

modelliert den zeitlichen Ablauf und die nachrichtenbasierte Kommunikation zwischen ausgewählten Objekten.

```

sequenceDiagram
    participant Patient
    participant Arzthelferin
    participant Arzt
    Patient->>Arzthelferin: Gesundheitskarte
    Arzthelferin->>Patient: Fordere Praxisgebühr
    Patient->>Arzthelferin: Bezahle Praxisgebühr
    Arzthelferin->>Arzt: Sende Patientenakte
    Arzt-->>Patient: Quittung
  
```

Generated by Targeteam

Sequenzdiagramm

modelliert den zeitlichen Ablauf und die nachrichtenbasierte Kommunikation zwischen ausgewählten Objekten.

```

sequenceDiagram
    participant Patient
    participant Arzthelferin
    participant Arzt
    Patient->>Arzthelferin: Gesundheitskarte
    Arzthelferin->>Patient: Fordere Praxisgebühr
    Patient->>Arzthelferin: Bezahle Praxisgebühr
    Arzthelferin->>Arzt: Sende Patientenakte
    Arzt-->>Patient: Quittung
  
```

Generated by Targeteam

UML2 umfasst in ihren Spracheinheiten verschiedene Möglichkeiten für die Modellierung der Struktur und des Verhaltens eines Systems, muss dabei aber auf einer generischen Ebene bleiben. Sie verwendet zum Beispiel die generischen Begriffe **Aktivität** oder **Artefakt** und kennt den spezifischen Begriff **Geschäftsprozess** aus der Geschäftsmodellierung oder **Enterprise Java Beans** der Java-Plattform nicht. Falls diese Begriffe in der Modellierung benötigt werden, müssen sie über den Erweiterungsmechanismus der Profile zu UML2 hinzugefügt werden. Auch spezielle Notationen, zum Beispiel eine realistischere Zeichnung anstelle des Strichmännchens, das einen **Akteur** darstellt, können UML2 mit Hilfe der Profile hinzugefügt werden. Weiter können Profile Lücken in der Semantik-Definition der UML2 schließen, die dort absichtlich für spezifische Einsatzgebiete offen gelassen wurden (engl. *semantic variation points*). Schließlich können Profile **Einschränkungen** definieren, um die Art und Weise zu beschränken, wie ein Element aus UML2 verwendet wird. Mit Hilfe des Erweiterungsmechanismus' der Profile kann das Metamodell von UML2 jedoch nicht beliebig angepasst werden. So können zum Beispiel keine Elemente aus dem Metamodell von UML2 entfernt, keine Einschränkungen aufgehoben und keine echten neuen Metaklassen, sondern nur Erweiterungen (**Stereotypen**) bestehender Metaklassen, deklariert werden.

Die Spracheinheit definiert zunächst das Konzept eines **Profils**. Ein Profil ist eine Sammlung von **Stereotypen**, und definiert die eigentliche Erweiterung. Ein Profil ist ein Paket und wird auf andere **Pakete angewendet**, womit die Erweiterung, die das Profil definiert, für das entsprechende Paket gilt.

UML2 kennt seit der UML 2.2 einen speziellen Diagrammtyp für Profile. Die graphischen Notationen für Elemente aus dieser Spracheinheit kommen sowohl in **Paketdiagrammen** als auch in **Klassendiagrammen** vor.

Schablonen [\[Bearbeiten\]](#)
Die Spracheinheit Schablonen (engl. *Templates*) umfasst Modellelemente für die Parametrisierung von **Klassifizieren**, **Klassen** und **Paketen**.

Verteilungen [\[Bearbeiten\]](#)

```

classDiagram
    class Applikationsserver
    class BestellungJar["<<artifact>> Bestellung.jar"]
    class RechnungJar["<<artifact>> Rechnung.jar"]
    Applikationsserver --> BestellungJar : <<deploy>>
    Applikationsserver --> RechnungJar : <<deploy>>
  
```

Die Spracheinheit **Verteilungen** (engl. *Deployments*) ist auf ein sehr spezifisches Einsatzgebiet ausgerichtet, nämlich auf die Verteilung von lauffähiger **Software** in einem **Netzwerk**. UML2 bezeichnet eine so installierbare Einheit als **Artefakt** und geht davon aus, dass diese auf **Knoten** installiert werden. Knoten können entweder **Geräte** oder **Ausführungsumgebungen** sein. Eine **Verteilungsbeziehung**, das heißt eine spezielle **Abhängigkeitsbeziehung**, modelliert, dass ein Artefakt auf einem Knoten installiert wird.

Elemente aus dieser Spracheinheit werden normalerweise in **Verteilungsdiagrammen**

W Unified Modeling Langu... W Objektdiagramm - Wiki... W Klassendiagramm - Wik... W Sequenzdiagramm - Wi... W Anwendungsfalldiagra... W Zustandsdiagramm (UM... +

https://de.wikipedia.org/wiki/Klassendiagramm

Benutzerkonto anlegen Anmelden

Artikel Diskussion Lesen Bearbeiten Versionsgeschichte Suchen

WIKIPEDIA
Die freie Enzyklopädie

Navigation
Hauptseite
Themenportale
Von A bis Z
Zufälliger Artikel

Mitmachen
Artikel verbessern
Neuen Artikel anlegen
Autorenportal
Hilfe
Letzte Änderungen
Kontakt
Spenden

Drucken/exportieren
Buch erstellen
Als PDF herunterladen
Druckversion

Werkzeuge

Klassendiagramm

Ein **Klassendiagramm** ist ein Strukturdiagramm der **Unified Modeling Language (UML)** zur grafischen Darstellung (Modellierung) von **Klassen**, Schnittstellen sowie deren Beziehungen. Eine **Klasse** ist in der **Objektorientierung** ein abstrakter Oberbegriff für die Beschreibung der gemeinsamen Struktur und des gemeinsamen Verhaltens von **Objekten (Klassifizierung)**. Sie dient dazu Objekte zu abstrahieren. Im Zusammenspiel mit anderen Klassen ermöglichen sie die Modellierung eines abgegrenzten Systems in der **objektorientierten Analyse und Entwurf**.

Seit den 1990er Jahren werden Klassendiagramme meistens in der Notation der UML dargestellt. Das Klassendiagramm ist eine der 14 Diagrammart der UML, einer Modellierungssprache für **Software** und andere Systeme.

Inhaltsverzeichnis [Verbergen]

- Notation in der Unified Modeling Language
 - Klassen
 - Schnittstellen
- Wichtige Beziehungen
 - Generalisierung
 - Assoziation
 - Komposition und Aggregation
- Formale Semantik
 - Klassen und Attribute
 - Assoziationen
 - Operationen
- Beispieldiagramm
- Literatur
- Weblinks

Strukturdiagramme der UML

- Klassendiagramm**
- Komponentendiagramm
- Kompositionsstrukturdiagramm
- Objektdiagramm
- Paketdiagramm
- Profildiagramm
- Verteilungsdiagramm

Verhaltensdiagramme der UML

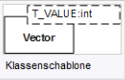
- Aktivitätsdiagramm
- Anwendungsfalldiagramm
- Interaktionsübersichtsdiagramm
- Kommunikationsdiagramm
- Sequenzdiagramm
- Zeitverlaufsdiagramm
- Zustandsdiagramm

Start | DE | 97% | 15:42 | 01.07.2013

W Unified Modeling Langu... W Objektdiagramm - Wiki... W Klassendiagramm - Wik... W Sequenzdiagramm - Wi... W Anwendungsfalldiagra... W Zustandsdiagramm (UM... +

https://de.wikipedia.org/wiki/Klassendiagramm

Beispiel einer aktiven Klasse mit zwei Signalempfängern

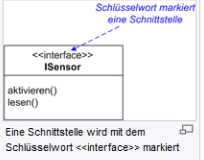


Einige Programmiersprachen ermöglichen eine Parametrisierung von Klassenschablonen (*Class Templates*), um Objekte basierend auf diesen Vorlagenparametern zu erzeugen. Die UML bietet dafür die Notation für *Template Arguments* an. Dabei werden die Vorlagenparameter in einem gestrichelten Rechteck überlappend an die rechte obere Ecke der Klasse eingetragen. Im Beispiel ist eine Klasse „Vector“ mit dem Vorlagenparametertyp „int“ und dem Parameternamen „T_VALUE“ eingetragen.

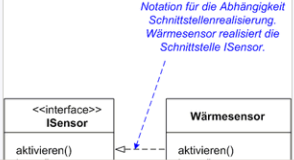
Schnittstellen

[Bearbeiten]

Eine **Schnittstelle** wird ähnlich wie eine Klasse mit einem Rechteck dargestellt, zur Unterscheidung aber mit dem Schlüsselwort **interface** gekennzeichnet. Schnittstellen können seit der UML 2 auch Attribute besitzen^[1].



Eine Schnittstelle wird mit dem Schlüsselwort <<interface>> markiert



Notation für die Abhängigkeit Schnittstellenrealisierung. Wärmesensor realisiert die Schnittstelle ISensor.

Start | DE | 97% | 15:43 | 01.07.2013