Begriff	Erklärung	Abbildung
CASE	Dies ist eine Abkürzung (engl.) für „Computer Aided Software-Engineering“ und bezeichnet einen Fachbegriff aus dem Software-Engineering. Gemeint sind hier meist die Werkzeuge, welche eingesetzt werden können, um Software-Engineering zu unterstützen. Beispiele sind UML-Tools wie Visual Paradigm (von Visual Paradigm International [Int19]), Rational Rhapsody Developer (von IBM [IBM19b]) oder Enterprise Architekt (von Sparx [Eur19]). Inhaltlich steht die Modellierung von Projektlösungen im Mittelpunkt. Beispiele hierfür sind: ▪ UML ▪ Transformation in verschiedene Programmiersprachen ▪ Datenstrukturen ▪ Reverse Engineering ▪ Generierung von Prototypen. Aber auch Projektmanagement, Prozessmanagement und viele weitere Entwicklungswerkzeuge sind oft zusätzlich mit im Paket.	
UML [OMG19b]	Es handelt sich um eine Abkürzung (engl.) für „**Unified Modeling Language**“. Die UML ist eine Modellierungssprache. „Die UML dient zur Modellierung, Dokumentation, Spezifizierung und Visualisierung komplexer Systeme, unabhängig von deren Fach- und Spezialisierungsgebiet. Sie liefert die Notationselemente gleichermaßen für die statischen und dynamischen Modelle von Analyse, Design und Architektur und unterstützt insbesondere objektorientierte Vorgehensweisen. … Die 1989 gegründete OMG (Objekt Management Group) – ein Gremium mit heute 800 Mitgliedern – ist Hüterin dieses Standards. Das es sich bei Werken der OMG um herstellerneutrale Industriestandards handelt, gewährleistet die Teilnahme aller relevanten Marktvertreter (zum Beispiel Rational Software [IBM], Hewlett-Packard, Daimler AG, I-Logix, Telelogic, Oracle, Micrsosoft, …).“ eine breite Unterstützung in der Industrie. (aus [RQSG12, S. 4])	

Hinweis

Da dieses Buch eine kompakte Übersicht über Software-Engineering bietet und absichtlich viele Themen daher lediglich anschneidet, aber nicht vollständig diskutiert, wird hier und im Verlauf des Buches auf passende ausführlichere Literatur verwiesen. Ein Nachschlagewerk der ausführlichen Natur ist das Buch von Sommerville [Som18] und eine etwas unbürokratische, amerikanische Variante das Buch von Pressman [Pre14]. Zum guten Studium der Modellierungssprache UML sind die „UML Kurzreferenz“ [ÖS14] und „UML glasklar“ [RQSG12] hilfreich.

1.2.1 Der Software-Lebenszyklus

Im Fokus von Software-Engineering steht der sogenannte **Software-Lebenszyklus**. Gemeint ist damit der gesamte Prozess, der zur Erstellung und Erhaltung eines Softwaresystems führt. Bei jeder Art von Softwareerstellung läuft dieser Lebenszyklus ab. Er ist zur einfacheren Unterscheidung der einzelnen Tätigkeiten in sogenannte Phasen unterteilt (siehe Abbildung 1.2).

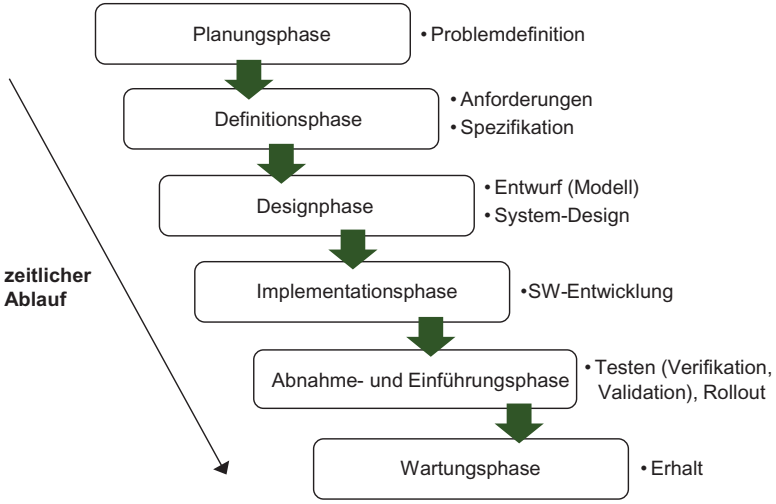

Abbildung 1.2 Software-Lebenszyklus

- **Planungsphase** (s. Kap. 3):
  In dieser Phase wird ein Softwareprojekt neu aufgesetzt, Projektmanagement begonnen, das Lastenheft geschrieben und eine Problemdefinition erarbeitet.
- **Definitionsphase** (s. Kap. 4):
  Nun werden die Anforderungen an eine Software eruiert und in einer Spezifikation dokumentiert.
- **Designphase** (s. Kap. 5):
  Danach erfolgt der Entwurf und die Modellbildung für das Systemdesign der Software. Es entsteht die Softwarearchitektur.
- **Implementationsphase** (vgl. [HV07]):
  In dieser Phase findet die eigentliche Programmierung der Software statt. Zugrunde liegt die in der vorherigen Phase gefundene Softwarearchitektur.
- **Abnahme-/Einführungsphase** (s. Kap.6):
  Um hohe Qualität zu gewährleisten, finden nun das Testen, die Verifikation, die Validation und die Markteinführung (engl. **Rollout**) der Software statt.
- **Wartungsphase** (s. Kap. 7):
  Die Software wird in dieser Phase vor Software-Alterung (engl. Aging) bewahrt; in der Regel werden Fehler der Software gesammelt und wenn möglich behoben. Auch Änderungsanforderungen werden erhoben, um für die aktuelle oder die Folgeversion der Software Verbesserungen erzielen zu können.

Nach der Wartungsphase findet in der Regel entweder die Entwicklung von **Folgeversionen** der Software, manchmal auch deren **Neuentwicklung** oder die **Stilllegung** statt. Bei einer Stillegung wird fallweise ein sogenanntes **End-of-Life-Management** durchgeführt.

1.2.2 Komplexität der Softwareentwicklung

Benutzer von Software erwarten heute einwandfrei funktionierende, effiziente Programme. Software, die nicht den Erwartungen entspricht, wird daher schnell verworfen und Konkurrenzprodukte werden eingesetzt. Softwareproduzenten haben daher großes Interesse daran, qualitativ hochwertige Programme herzustellen.

Warum fällt es schwer, gute Software zu entwickeln?

Grundsätzlich sind oft bereits kleine Softwareprojekte schwer beherrschbar. In der Regel sind es jedoch insbesondere die großen bzw. komplexen, qualitativ hochwertigen Softwareprojekte, die eine Vielzahl von Teilnehmern, angefangen vom Kunden über ein Team von Software-Engineering-Spezialisten, Entwicklern, Testpersonal

und Wartungsspezialisten, benötigen. Jeder Projektteilnehmer bringt unterschiedliche Erfahrungen, Verfahren und Methoden mit, die miteinander abgeglichen und verwendet werden müssen. Unterschiedliche wirtschaftliche Faktoren genauso wie gesetzliche Aspekte müssen Berücksichtigung finden. In internationalen Projekten müssen beispielsweise oft Ländergrenzen, unterschiedliche Kulturen und Sprachen überbrückt werden.
Auch haben verschiedenartige Projektteilnehmer auch unterschiedliche Ziele. Kunden- und Benutzerwünsche liegen oft weit auseinander. Ein Projektleiter hat eine andere Sichtweise auf sein Projekt als ein Entwickler.

Merke

Software-Engineering ist demzufolge eine Wissensquelle, von der alle Projektteilnehmer profitieren können, um ihre Ziele auch erreichen zu können.

Software-Engineering ist somit eine Art **Baukastensystem**, in dem **Standards, Verfahren** und **Methoden** angeboten werden, um einen möglichst erfolgreichen Projektverlauf zu gewährleisten. In dieses Baukastensystem sind das Wissen und die Projekterfahrungen aus zahlreichen, erfolgreichen und nicht-erfolgreichen Vorprojekten eingeflossen. Es bietet erfolgsorientierte Lösungsmöglichkeiten für den gesamten Ablauf eines Softwareprojektes an.

Eigenschaften von Software

Welche Eigenschaften von Software erschweren die Erstellung von qualitativ hochwertiger Software?

Software …

- ist **immateriell**
- wird nicht durch physikalische Gesetze begrenzt
- eine zugrunde liegende Modellbildung ist schwierig
- Anforderungen sind oft unzureichend geklärt, dokumentiert oder spezifiziert
- kann Defekte enthalten, Beispiele sind: Konstruktionsfehler, Spezifikationsfehler oder Portierungsfehler
- Ersatzteile gibt es nicht (nur evtl. sogenannte (engl.) „Patches“ (d. h. ein Austausch von Teilpassagen des Softwarecodes))
- ist schwer zu vermessen (Was ist gute und was ist schlechte Software?)
- ist leicht änderbar
- hat keinen Verschleiß, aber Software-„Aging“ (auf Deutsch: Alterung)
- Veränderungen sind fortlaufend nötig, um Software lauffähig zu halten.

Schwierigkeiten bei der Software-Erstellung
Schwierigkeiten bei der Software-Entwicklung gibt es aufgrund deren Komplexität viele! Deshalb folgt hier keine vollständige Liste der zu bewältigenden Problematiken, sondern ein exemplarischer Ausschnitt möglicher Komplikationen:

- Kommunikationsprobleme mit Projektbeteiligten:
  Wenig Wissen über die Anwendung bei Software-Engineering-Spezialisten und Entwicklern; Anwender hat unklare Vorstellungen des Systems
- Arbeitsabläufe werden durch Software oft verändert:
  Akzeptanz- und Integrationsprobleme
- Software-Varianten:
  Konfiguration und Versionierung gestaltet Software viel komplexer
- Software Einsatz in verschiedenen Umgebungen:
  Portabilitätsprobleme.

Was kann man nun tun, um die genannten Probleme zu minimieren?
Die Antwort, die Software-Engineering-Spezialisten geben, lautet in etwa wie folgt:

Abhilfe schafft die Verwendung von:

- **Standards**
  Beispiel: Planen der Software durch Modellbildung
- **Methoden**
  Beispiel: Einsatz von Projektmanagement
- **Werkzeuge**
  Beispiel: Verwendung von UML-Tools, welche die Zeichenarbeit bei der Erstellung der Softwarearchitektur vereinfachen.

Genau diese Sammlung und Beschreibung der Standards, Methoden und Werkzeuge stehen im Fokus von Software-Engineering und damit auch im Mittelpunkt dieser Arbeit. Aus diesem Grund enthalten die weiteren Kapitel die kompakte Übersicht über die gebräuchlichsten und praxisrelevanten Verfahren dieses Fachgebiets.

1.3 Geschichtlicher Überblick und die Folgen der Software-Krise

In den fünfziger Jahren entstanden die ersten benutzbaren, jedoch teuren Computer, die Makros und bald Prozeduren niederer Programmiersprachen verstanden