1 Einleitung

»Aus kleinem Anfang entspringen alle Dinge.«

(Marcus Tullius Cicero, 106 – 43 v. Chr., römischer Redner und Staatsmann)

1.1 Warum gerade jetzt dieses Buch?

Die Beherrschung von Komplexität ist wohl die größte Engineering-Herausforderung des 21. Jahrhunderts. Software wird der Rohstoff sein, aus dem zukunftsfähige, smarte Systeme erschaffen werden. Software ist schon jetzt ein zentraler Bestandteil unserer Infrastruktur. Täglich kommen wir Menschen – mehr oder weniger bewusst – mit vielen Systemen in Kontakt, in denen Software zentrale Aufgaben übernimmt: Sei es der Fahrstuhl, die Klimaanlage, der Fahrkartenautomat oder das Auto.

Mit dem Internet der Dinge (*Internet of Things*, IoT) und der vierten industriellen Revolution, sprich: der Zukunftsvision »Industrie 4.0« der deutschen Bundesregierung, wird der Anteil und somit auch der Einfluss von Software auf alle Lebensbereiche des Menschen noch weiter zunehmen.

So wie die Metallverarbeitung eine Schlüsseltechnologie in der industriellen Revolution war, wird Software und Systems Engineering die Schlüsseltechnologie des Informationszeitalters sein.

Im Kontext von Embedded Systems werden sich obige Anforderungen besonders auswirken und modellgetriebene Entwicklung wird hier eine zentrale Rolle im Software und Systems Engineering für eingebettete Systeme einnehmen.

Eingebettete Systeme unterliegen wie jedes IT-System ständigen Innovationen. Moore's Law lässt grüßen. Es gibt jedoch innerhalb der Genesis solcher Systeme immer wieder Umwälzungen, die der scheinbar stetigen Entwicklung sprunghaften Charakter verleihen. Für den Betrachter scheint sich innerhalb kurzer Zeit alles zu ändern. Neue Programmiersprachen, mit denen die Entwickler konfrontiert werden, sind nur die Spitze des Eisbergs. Schaut man aus der heuti-

gen Perspektive auf das Themengebiet Software Engineering zurück (siehe Abb. 1–1), kann die Entwicklung auf drei Umbrüche verdichtet werden.

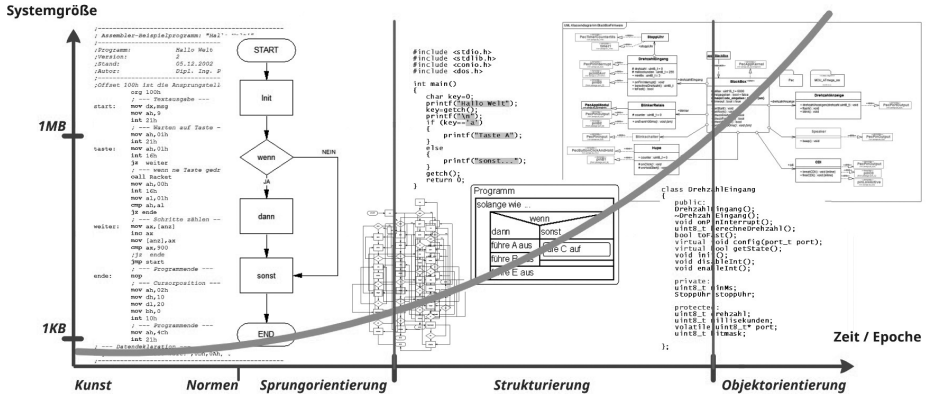

Abb. 1–1 *Entwicklung von Software-Engineering-Paradigmen*

- **1950er- bis 1960er-Jahre:** Initialzustand, maschinennahe Programmierung, dominierende Programmiersprache ist Assembler und es gibt erste Hochsprachen, Programmierparadigma ist die Sprunganweisung, Visualisierung erfolgt mit Flussdiagrammen.
- **1970er-Jahre:** Paradigmenwechsel zur Strukturierung (Funktionsorientierung), dominierende Programmiersprachen verzichten weitestgehend auf die Sprunganweisung, z.B. Pascal und C, Modularisierung, Funktionsbausteine sind zustandslos, Software Engineering erlebt Blütezeit und wird von vielfältigen strukturierten Darstellungstechniken wie strukturierter Analyse (SA), strukturiertem Design (SD), Nassi-Schneiderman-Diagrammen usw. dominiert, definierte Softwareprozesse orientieren sich am Wasserfallmodell.
- **Ab 1990er-Jahre:** Paradigmenwechsel zur Objektorientierung, dominierende Programmiersprachen sind z.B. C++ und Java, Softwarebausteine besitzen Zustände, Software Engineering erlebt einen Vereinheitlichungsprozess der objektorientierten Darstellungstechniken zur Unified Modeling Language (UML), definierte Softwareprozesse orientieren sich am Spiralmodell und werden agil.

Dieser stark verdichtete historische Abriss zeigt, dass der zunehmende Komplexitätsgrad an ausgewählten Punkten zu Paradigmenwechseln geführt hat. Es stellt sich die Frage, ob sich das auf eingebettete Systeme übertragen lässt?

Schaut man sich die dominierende Programmiersprache für eingebettete Systeme an, so stellen wir fest, dass sich C seit geraumer Zeit durchgesetzt hat. Nur noch wenige ausgewählte Aufgabenstellungen werden in Assembler realisiert.

Offensichtlich folgte man hier dem Paradigmenwechsel zur Strukturierung. Es lässt sich sogar eine Kenngröße ausmachen, an der man diesen Wechsel fest-

machen kann. Es ist die Programmgröße für eingebettete Systeme, die Größe des Programmspeichers. Ab 1 bis 16 Kilobyte Programmgröße wird es zunehmend schwieriger, die Aufgabenstellung in Assembler zu lösen. C wurde, obwohl hungriger nach Ressourcen, die adäquatere Programmiersprache für eingebettete Systeme.

Oberhalb der 16 Kilobyte wird nicht mehr ernsthaft darüber diskutiert, das System in Maschinensprache zu programmieren. Für den Paradigmenwechsel zur Objektorientierung lässt sich ebenfalls diese Kennzahl heranziehen. Dieser wurde vollzogen, als die typische Programmgröße deutlich die 1-Megabyte-Grenze überschritten hatte. Spätestens ab 4 Megabyte Hauptspeicher waren alle Diskussionen, ob PC-Programme in C programmiert werden sollen, erledigt. Objektorientierte Programmiersprachen wie C++ und Java haben sich innerhalb kurzer Zeit, zwischen 1990 und 1995, durchgesetzt. Heute verfügen selbst kleine Mikrocontroller wie die ARM-Cortex-M-Familie über mehr als 1 Megabyte Programmspeicher.

1.2 Wie sollte man dieses Buch lesen?

Das Buch spannt einen weiten Bogen über das Fachgebiet der Entwicklung eingebetteter Systeme. Die Modellierung ist die Klammer, die die einzelnen Aspekte zusammenhält. Um das Fachgebiet als Ganzes zu verstehen, sollte das Buch kursorisch gelesen werden. Dabei können Details einzelner Aspekte auch übersprungen werden. Die Abschnitte sind in sich so weit abgeschlossen, dass eine zwingende Reihenfolge für die Bearbeitung nicht vorliegt. Für ausgewählte Leser sind einzelne Abschnitte von besonderem Interesse. Diese sollten intensiver studiert werden. Dafür sind im Anhang wichtige Erklärungen zu finden. Für die praktische Anwendung des Gelernten bietet die Webseite zum Buch unter *www.mdese.de* Werkzeuge, Beispiele und Tutorials an.

Studenten lernen das Fachgebiet Stück für Stück im Überblick kennen und werden für wichtige Aspekte sensibilisiert, deren Inhalte im Studium zu vertiefen sind. Zusammenhänge einzelner Disziplinen werden verstanden und die Möglichkeiten modellgetriebener Technologien können abgeschätzt werden. Dieses Buch ist auch als Nachschlagewerk geeignet.

Entscheider vertiefen ihr Beurteilungsvermögen und werden in die Lage versetzt, moderne modellgetriebene Technologien zu analysieren sowie qualifiziert zu bewerten.

Projektleiter erhalten wichtige Impulse zur Einführung von Modellierungstechniken. Das Verständnis der vorgestellten Technologien ist die Voraussetzung für deren Anwendung. In Kombination mit gesammelter Projekterfahrung gelingt es, für zukünftige Projekte eine neue Synthese mit modellgetriebenen Technologien zu erarbeiten.

Softwareentwickler verstehen den gesamten Prozess der modellgetriebenen Entwicklung und werden in die Lage versetzt, die für sie relevanten Technologien zu beurteilen und anzuwenden. Besonders modellgetriebene Realisierung und Test werden im Detail verstanden.

Hardwareentwickler lernen die Arbeitstechniken der Softwareentwickler kennen und sind in der Lage, mit diesen eine gemeinsame Sprache zu finden.

1.3 Was ist an eingebetteten Systemen so besonders?

Eingebettete Systeme haben im Vergleich zu konventionellen Computern, wie unseren Desktop-PCs oder unseren Notebooks, geringe Ressourcen an Speicher und Rechengeschwindigkeit. Es gibt vielfältige Hardwarearchitekturen von 4 bis 64 Bit. Die verfügbaren Systeme bieten Single- und Multicore sowie sehr unterschiedliche Softwarearchitekturen von Bare-Metal-Programmierung über Real-Time Operation Systems bis Multitask/Multiuser-Betriebssysteme an.

Abb. 1–2 *Programmieradapter und Zielsystem*

Der sinnliche Zugang wird für den Neueinsteiger dadurch erschwert, dass diese eingebetteten Digitalrechner als solche meist nicht zu erkennen, also quasi »unsichtbar« sind. Sie verfügen oft weder über gebräuchliche Eingabegeräte wie Maus und Tastatur noch über grafische Displays. Ein Taster und wenige LEDs bilden in vielen Fällen die einzige Mensch-Maschine-Schnittstelle.

Daher erfordert es auch ein spezielles Equipment für die Programmierung. Der Einsteiger muss sich ein Programmiergerät und wenn möglich Debugger-Hardware für das Zielsystem zulegen.

Zusätzlich sind eine spezielle Entwicklungsumgebung und Compiler nötig, die die gewünschte Hardware auch unterstützen. Die verfügbare Literatur ist entweder proprietär auf die Hardware- und Softwarearchitektur sowie die Entwicklungsumgebung des Chipherstellers fokussiert oder so allgemein gehalten, dass die konkrete Anwendung des Gelernten nur schwer möglich ist.

Von der breit angewendeten Softwaretechnologie im Mikrorechnerbereich ist der Biotop der eingebetteten Systeme eher abgeschnitten.

```
int main() {
  DDRB &= ~(1 << PB0);
  DDRB |= (1 << PB1);

  PORTB |= (1 << PB1);  //PB1 High
  PORTB &= ~(1 << PB1); //PB1 Low
}
```

```
public class HelloWorld
{
      public static void main (String[] args)
      {
            // Ausgabe Hello World!
            System.out.println("Hello World!");
      }
}
```

Abb. 1–3 *Hallo Welt, zwei Welten*

So viel zu den Schwierigkeiten bei der Annäherung an eingebettete Systeme. Es gibt auch Erfreuliches zu berichten: Die Komplexität der Hardware eines eingebetteten Systems ist im Vergleich zu den großen Verwandten PCs, Notebooks und Co. oft noch bis auf die Registerebene durch- und überschaubar. Die gegebene Hardware ist in vielen Fällen bis ins Detail durch den Entwickler selbst determiniert. Es bestehen zahlreiche Alternativen zu einer Architektur, die damit eine entsprechende Vielfalt von Lösungsmöglichkeiten konkreter Aufgabenstellungen eröffnen. Die erlangte Kompetenz ist nicht so einfach nachzubilden und kann dadurch für längere Zeit einen Marktvorsprung darstellen.

1.4 Wie sieht das typische Zielsystem aus?

Dieses Buch fokussiert auf Systeme, die aus softwaretechnologischer Sicht an einem Kipppunkt zu verorten sind. Diese Systeme sind nicht mehr klein, aber auch noch nicht wirklich groß. Sie sind so leistungsfähig und komplex, dass die klassische Entwicklung mit einem Zeileneditor in einer strukturierten Sprache wie C an ihre Grenzen stößt. Es ist sinnvoll, die anstehenden Herausforderungen mit einer modellgetriebenen Entwicklung und in einer höheren Programmiersprache wie z. B. dem objektorientierten C++ zu bewältigen.

Als kleine eingebettete Systeme bezeichnen wir in diesem Kontext Mikrocontroller, meist mit 8-Bit-Verarbeitungsbreite, mit wenigen Kilobyte Programmspeicher und vielleicht maximal 30-MHz-Taktfrequenz.

Abb. 1–4 *8-Bit-Controller im Vergleich zu einer 1-Cent-Münze*

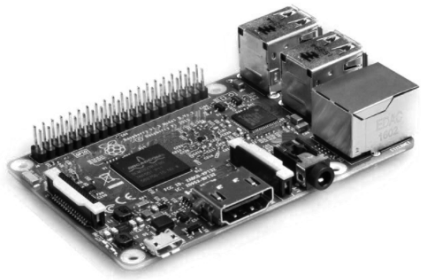

Abb. 1–5 *32 Bit ARM Cortex A8 Singleboard Computer*

Als große eingebettete Systeme bezeichnen wir 32- bis 64-Bit-Multicore-Architekturen, oft mit grafischem Display, einem entsprechenden Betriebssystem, mehreren Gigabyte Speicher und Taktraten im Gigahertz-Bereich. Letztere sind vielleicht noch eingebettete Systeme, aber keine Mikrocontroller mehr, sondern gehören schon zur Klasse der Mikrorechner.

Die kleinen Systeme sind mit den klassischen Mitteln, also hardwarenah in Assembler oder C, beherrschbar. Für die großen Systeme stehen etablierte Betriebssysteme und standardisierte Plattformen zur Abstraktion von der komplexen Hardware zur Verfügung. Zwischen diesen öffnet sich jedoch zurzeit ein großes Feld von Mikrocontrollern, die zum einen zu komplex sind für die althergebrachte Vorgehensweise und zum anderen zu diversifiziert für standardisierte Betriebssysteme.

Da die Übergänge nicht scharf abgrenzbar sind, soll eine Bandbreite von Systemen angesprochen werden, für die eine modellgetriebene Entwicklung, wie sie in diesem Buch beschrieben wird, praktikabel und kommerziell sinnvoll ist (siehe Abb. 1–6). Die untere Grenze bilden leistungsfähige 8- und 16-Bit-Architekturen mit 16 bis 32 KByte Programmspeicher und 1 bis 30 MHz Taktfrequenz. Als typisches Zielsystem sind 32-Bit-Single-Core-Architekturen mit 32 KByte bis 4 MByte Programmspeicher und bis zu 300 MHz Taktfrequenz anzusehen. Die Obergrenze sollten wir bei 32-Bit-Multicore-Systemen mit vielleicht 1 GByte Speicher ziehen.

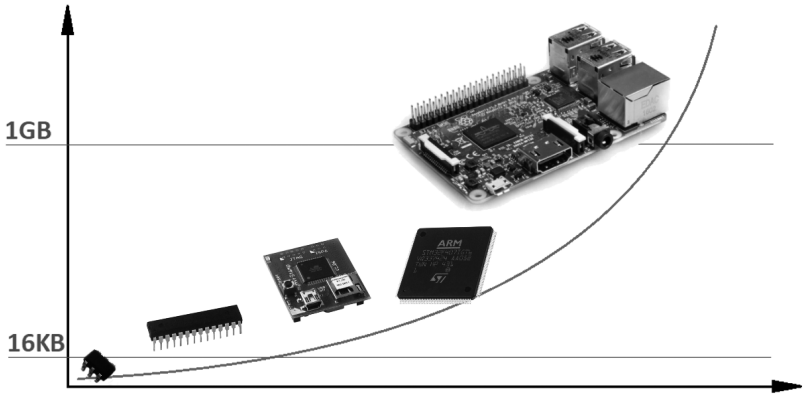

Abb. 1–6 *Bandbreite der Zielsysteme*

In dieser Bandbreite können die hier vorgestellten modellgetriebenen Technologien wertvolle Beiträge zur Qualitätsverbesserung und zum betriebswirtschaftlichen Erfolg leisten. Die zur Vertiefung der Buchinhalte angesprochenen und auf der Webseite verfügbaren Beispiele sind für ein Referenzsystem auf der Basis eines 32 Bit ARM Cortex M4 von Infineon konzipiert. Mit 1 MByte Programmspeicher und 120 MHz Taktfrequenz liegt dieses System genau an der kritischen Schwelle, die den Einsatz modellgetriebener Technologien besonders wirksam werden lässt.

1.5 Fallbeispiele

Die hier kurz charakterisierten Fallbeispiele beziehen sich auf Projekte, die von den Autoren durchgeführt oder begleitet wurden.

1.5.1 Die Anlagensteuerung

Das Projekt »Anlagensteuerung« ist bei einem mittelständischen Unternehmen mit ca. 100 Mitarbeitern zu verorten. Es werden Lüftungsanlagen in Kleinserien und nach Kundenspezifikation gefertigt. Die Entwicklungsabteilung der Firma besteht aus zwei Elektrotechnikingenieuren.

Für die Steuerung der Anlagen haben die Ingenieure im Laufe der Zeit eine kleine, modulare Hardwareplattform entwickelt. Die Ausstattung der Hardware bietet den Grundbedarf an Sensorik, Aktorik und einfachen Benutzerschnittstellen, die für jede Anlage benötigt werden. Zusätzlich kann die Steuerung mit weiteren Sensoren und Aktoren, grafischem Display, Netzwerkanschluss usw. ausgestattet werden.

Vor der Einführung modellgetriebener Technologien bestand das Hauptproblem darin, dass neue oder kundenspezifische Anlagenvarianten von der mechani-

schen und elektrotechnischen Seite her innerhalb weniger Tage realisierbar waren. Die Steuerungssoftware aber, sobald die neuen Anforderungen über eine einfache Parametrisierung hinausgingen, war erst nach Wochen oder gar Monaten verfügbar. Die Auslieferung der Anlagen verzögerte sich und Ressourcen (Kapital, Material, Fertigungs- und Lagerplätze) waren unnötig gebunden.

Mit der Einführung modellgetriebener Technologien wurde die Entwicklungszeit der Software so weit beschleunigt, dass diese jetzt in der Regel mit der Fertigstellung der Hardware zeitgleich zur Verfügung steht. Die Wettbewerbsfähigkeit der Firma wurde mit einer deutlich verkürzten Entwicklungszeit (*Time-to-Market*) entscheidend verbessert.

1.5.2 Die Baumaschine

Ein traditionsreicher Hersteller von robusten Baumaschinen im unteren und mittleren Preissegment sah sich am Markt zunehmendem Wettbewerbsdruck ausgesetzt. Der Verkauf über den Preis konnte nur für begrenzte Zeit die Wettbewerbsfähigkeit erhalten. Als Alternative kam nur eine deutliche Verbesserung der Funktionalität der Baumaschinen infrage.

Die Herausforderung bestand jedoch darin, auf teure Zukaufkomponenten zu verzichten und trotzdem bessere Funktionalität anzubieten. Es wurde die Entscheidung getroffen, in Begleitung durch ein Beratungsunternehmen eine eigene Kompetenz zur Entwicklung von Hydrauliksteuerungen aufzubauen. Dabei wurde von Anfang an auf modellgetriebene Entwicklung gesetzt.

Innerhalb eines Jahres ist es gelungen, eine hauseigene Plattform für Steuergeräte zu entwickeln und vor allem durch modellgetriebene Tests die Konformität mit Normen und Sicherheitsanforderungen begleitend zur Hardwareentwicklung sicherzustellen.

Das Entwicklerteam konnte einen definierten und modellgetriebenen Entwicklungsprozess etablieren. Dabei herrscht ein hoher Grad an Arbeitsteilung. UML-Modelle dienen nicht nur zur Systemgenerierung, sondern auch als normierendes Element in der Kommunikation zwischen den beteiligten Entwicklerteams.

Heute wird ein wesentlicher Anteil der Wertschöpfung des Unternehmens über die Softwareentwicklung erbracht. Verifikation und Validierung neuer Entwicklungen erfolgen bereits auf Modellebene. Nicht nur die Quellcodes und automatisierte Tests, sondern auch weite Teile der Projektdokumentation werden aus den Modellen generiert.

1.5.3 Das Energie-Monitoring-System

Für die wissenschaftliche Untersuchung von besonders energiesparenden Gebäuden (Passivhäusern) wird eine Vielzahl unterschiedlicher Messwerte benötigt. Es müssen Hunderte Sensoren im und am Baukörper verbaut und vernetzt werden. Dabei kommen über verschiedene Kommunikationsmedien sehr unterschiedliche Protokolle und Datenformate zur Anwendung.

Herstellerspezifische Lösungen für moderne Zählersysteme *(Smart Meter)* erschweren die Integration. Das System besteht aus zahlreichen einzelnen Controllern zur Datenerfassung, -übertragung und -aufbereitung.

Die Komplexität der Gesamtlösung lässt sich mit folgenden Parametern umreißen: Individuelle, auf mehrere Hundert unterschiedliche Controller verteilte Softwarelösung, die mehr als zwei Millionen Messwerte am Tag erfasst, verarbeitet und an einen Server überträgt.

Die Lösung wurde innerhalb von weniger als drei Monaten durch ein kleines in der modellgetriebenen Entwicklung erfahrenes Team erstellt und automatisiert auf das Zielsystem übertragen. Späte und nachträgliche Anforderungen des Auftraggebers konnten zuverlässig und zügig implementiert werden. Das System wurde und wird in der Einsatzphase ständig an neue Anforderungen angepasst. Die Anforderungen an das System werden durch Akteure, die an der wissenschaftlichen Auswertung der Daten beteiligt sind, immer wieder erweitert.

Es hat sich herausgestellt, dass modellgetriebene Anpassungen zuverlässiger und um ein Vielfaches schneller sind als vergleichbare Lösungen.

Die Liste der Fallbeispiele lässt sich noch wesentlich länger gestalten. Eines ist allen Erfolgsbeispielen gemeinsam: Der zunächst nicht unerhebliche Aufwand für die Einführung modellgetriebener Technologien hat nicht nur zu moderaten Verbesserungen geführt, sondern zu völlig neuen Dimensionen. Die Auswirkungen sind vor allem in der Softwarequalität, der Prozessqualität, der Entwicklungsgeschwindigkeit und der Wartbarkeit der Systeme festzustellen.

1.5.4 Das Beispielsystem SimLine

Im Buch werden Anwendungsbeispiele anhand eines Referenzsystems erläutert. Dieses Referenzsystem wurde speziell für dieses Buch zusammengestellt und besitzt wichtige Merkmale, die für die genannten Fallbeispiele relevant sind. Zum Beispiel ein grafisches Display wie bei der Anlagensteuerung, ein Fahrzeugbussystem wie in der Baumaschine und Sensorik und Aktorik sowie eine Ethernet-Schnittstelle wie beim Energie-Monitoring-System.

Das Referenzsystem ist das Smart-Home-System SimLine. Es besteht aus einer Plattform, die die Kerninfrastruktur zur Verfügung stellt, um Sensoren und Aktuatoren anzuschließen, die Sensoren auszulesen und über definierbare Regeln die Aktuatoren entsprechend anzusteuern.

Zusätzlich gibt es einzelne SimLine-Sensoren und -Aktuatoren sowie spezielle SimLine-Module mit vordefinierten Regeln, beispielsweise für Einbruch- oder Sturmschutz.

Vertiefende Informationen, Beispielmodelle und Beschaffungsquellen für das SimLine-System finden Sie auf der Webseite zum Buch.

1.6 Das Manifest

Das Manifest *Modeling of Embedded Systems* wurde 2015 von mehreren Personen erarbeitet, die überzeugt sind, dass Projekte den steigenden Anforderungen an die Entwicklung eingebetteter Systeme nur gerecht werden können, wenn sie modellbasierte Methoden verwenden. Gleichzeitig sahen die Autoren, dass trotzdem die Modellierung noch sehr verbreitet eingesetzt wird.

Unter den Autoren und Unterzeichnern des Manifests waren auch Autoren dieses Buches. Das Manifest ist somit auch ein Leitfaden für dieses Buch und Sie werden die einzelnen Thesen des Manifests im gesamten Buch wiederfinden.

Das Manifest postuliert 7 Thesen:

1. ***Beteilige Stakeholder*** durch geeignete domänenspezifische Abstraktionen, Notationen und Sichten.
2. ***Strebe nach langlebigen Modellen*** durch angemessene Abstraktionen, Trennung von Aspekten und Modellierungsrichtlinien.
3. ***Mache Modelle überprüfbar, transformierbar und ausführbar*** durch semantisch klar definierte Sprachen für funktionale und nicht funktionale Aspekte.
4. ***Lerne rechtzeitig*** durch Modelle, die in einer frühen Entwicklungsphase Anforderungen und Spezifikationen überprüfen und Fehler aufzeigen.
5. ***Vermeide Redundanzen und wiederkehrende Arbeiten*** durch Automatisierung und Integration von Modellen für verschiedene Aspekte.
6. ***Mache Modelle zugänglich*** durch eine skalierbare Infrastruktur und benutzerfreundliche und leicht erlernbare Werkzeuge.
7. ***Etabliere eine Modellierungskultur*** durch Ausbildung und Harmonisierung der Modellierungsaktivitäten mit den Entwicklungsprozessen.

Das Manifest wurde erstmals 2015 auf der Konferenz MESCONF *(www.mesconf.de)* in München vorgestellt. Es ist auf einer eigenen Internetseite *www.mdse-manifest.org* publiziert und kann von jedem als Zeichen der Unterstützung unterzeichnet werden.

In Anhang A werden die Thesen des Manifests zur Entwicklung der Skizze eines Reifegradmodells für modellbasierte Softwareentwicklung herangezogen.

Legende

Am Ende der meisten Kapitel gibt es eine Zusammenfassung, die in kurzen Stichpunkten eine Art Essenz bildet. Dabei gibt es folgende Kategorien an Stichpunkten, die entsprechend dem Charakter des jeweiligen Kapitels mehr oder weniger ausgeprägt oder nicht vorhanden sind (z.B. hat Anhang B »Kurzreferenz UML und SysML« keine Zusammenfassung).

- ✓ Checklisten: Die so gekennzeichneten Aussagen sind als Checkliste zu sehen. Sie helfen Ihnen, zu überprüfen, ob Sie die essenziellen Konzepte dieses Kapitels berücksichtigt haben.
- ☞ Hinweise: Die so gekennzeichneten Aussagen weisen noch einmal auf wichtige Aussagen des Kapitels hin.
- 💣 Warnungen: Die so gekennzeichneten Aussagen weisen explizit auf Fehlentscheidungen/Fehlannahmen hin, die in der Praxis immer wieder anzutreffen sind und deren Auswirkungen erfahrungsgemäß besonders aufwands- oder kostenintensiv ausfallen.