Fraunhofer

Nadia Hammoudeh Garcia | Björn Kahl | Eshan Savla

Softwareintegration in der Robotik

Hrsg.: Thomas Bauernhansl, Marco Huber, Werner Kraus

Im Rahmen des

KI Fortschrittszentrum
LERNENDE SYSTEME UND KOGNITIVE ROBOTIK

Vorwort

Künstliche Intelligenz (KI) ist eine der zentralen Technologien für die Zukunft. Ihre Einführung und der Einsatz fordern Unternehmen im besonderen Maß heraus.

Mit dem KI-Fortschrittszentrum des Fraunhofer IAO und Fraunhofer IPA unterstützen wir Unternehmen dabei, das Potenzial von KI zu erkennen und dieses wirtschaftlich nutzbar zu machen. An der Schnittstelle zwischen anwendungsorientierter Wirtschaft und exzellenter Forschung des Cyber-Valley-Konsortiums entwickeln wir innovative KI-Anwendungen für die Praxis und treiben damit die Kommerzialisierung von KI voran. Erklärtes Ziel ist dabei, menschzentrierte KI-Lösungen zu entwickeln. Denn nur wenn Menschen mit einer neuen Technologie intuitiv interagieren und vertrauensvoll zusammenarbeiten, kann ihr Potenzial optimal ausgeschöpft werden.

Die Studienreihe »Lernende Systeme« des KI-Fortschrittszentrums gibt Einblick in die Potenziale und die praktischen Einsatzmöglichkeiten von KI. Dabei werden übergreifende Themen wie Zuverlässigkeit, Erklärbarkeit (xAI), cloudbasierte Plattformen, Technologien und Einführungsstrategien diskutiert. Zudem werden einzelne Anwendungsbereiche in der Wissensarbeit, Bauwirtschaft, Produktion und dem Kundenservice im Detail beleuchtet.

Die Automatisierungs- und Fertigungstechnik ist heute mehr und mehr Software-definiert. Im Rahmen der vorliegenden Studie wurden Vorgehensweise und Hindernisse für eine effiziente und langfristig tragfähige Softwareentwicklung in der Robotik untersucht. Methodisch wurde der Stand der Praxis der Software- und Systementwicklung in der Robotik bei Integratoren, Anwendern und Komponentenherstellern erhoben und dem Stand der Technik (bzw. Wissenschaft) des Softwareengineering gegenübergestellt. Die Erkenntnisse wurden unter dem Gesichtspunkt der werkzeuggestützten Entwicklung, speziell des modellbasierten Software/System Engineering (MBSE) betrachtet und einerseits konkrete Anforderungen an praktisch einsetzbare MBSE-Tools als auch andererseits Empfehlungen für die Integration von Softwareplattformen wie ROS und modernen modell-basierten Entwicklungsmethoden in die Robotik abgeleitet.

Wir wünschen Ihnen eine spannende Lektüre und freuen uns, wenn wir in Zukunft auch Sie mit unserer Expertise auf Ihrem Weg zur menschzentrierten KI unterstützen dürfen.

Thomas Bauernhansl, Marco Huber, Werner Kraus
Fraunhofer-Institut für Produktionstechnik und Automatisierung IPA

Inhalt

Vorwort **2**

1. State of the Art **5**
1.1. Begrifflichkeiten – Was ist Softwareintegration? 6
1.2. Herausforderungen der Softwareintegration in der Robotik 7
1.3. Stand der Technik der Integration mit ROS 8
1.4. Warum genügt ein Komponenten-basiertes Software-Framework nicht? 10

2. Studie zur Softwaremodellierung und Ergebnisse **12**
2.1. Profil der Teilnehmer 12
2.2. Erfasste Daten 14
2.2.1. Traditioneller Prozess der Software-Entwicklung und seine Anwendung in der Software-Integrationsphase 14
2.2.2. Herausforderungen und Verbesserungen 17
2.2.3. Werkzeuge zur Unterstützung des Integrators 19
2.2.4. Gewünschte Werkzeuge zur Unterstützung des Integrators 19
2.3. Analyse der Ergebnisse 21
2.3.1. Herausforderungen 21
2.3.2. Übliche Vorgehensweisen 21
2.3.3. Anwendbarkeit von Methoden des Softwareentwicklungszyklus 21
2.3.4. Werkzeuge 22
2.4. Kriterien der Validität der Studie 22

3. Lösungsansatz: modellbasierte Softwareentwicklung **24**
3.1. Anwendung der technischen Verfahren und Beschreibung des Lebenszyklus der Systementwicklung . 24
3.1.1. Übersicht 24
3.1.2. Entwurf 26
3.1.3. Implementierung 27
3.1.4. Testen 28
3.1.5. Bereitstellung 28
3.2. Modellgetriebene Softwareentwicklung und -integration 29
3.2.1. Was und wie? 29
3.2.2. Arbeitserleichterungen und Fehlervermeidung 30
3.2.3. Skalierbarkeit, Softwareproduktfamilien und Kundenvarianten effizient umsetzen 31
3.2.4. Vereinfachte Wartung und Analyse alter Software 31

Literaturverzeichnis **32**
KI-Fortschrittszentrum **33**
Fraunhofer **34**
Impressum **36**

1. State of the Art

Die Umsetzung einer Automatisierungsaufgabe von der Problemstellung über die Lösungsidee bis zur fertigen, laufenden Roboteranwendung ist ein hoch komplexer Prozess, an dem viele verschiedene Gewerke beteiligt sind. Frühere Studien haben gezeigt, dass über die Hälfte der Kosten für die Einführung einer Roboteranwendung in der Systemintegration liegen. Ein erheblicher Teil davon entfällt bei moderner, Software-definierter Automatisierung auf die Softwareintegration. Dieser Aufwand fällt bei einer Änderung des Systems, etwa wegen neuer Produktvarianten, in variabler Höhe erneut an – je nachdem, wie wartungs- und änderungsfreundlich die ursprüngliche Lösung gestaltet war. Bei einer monolithischen, kundenspezifischen Lösung können die Änderungskosten praktisch den Neuerstellungskosten entsprechen, bei einer modularen, auf einem »Baukastensystem« beruhenden Lösung können sie deutlich niedriger ausfallen, vorausgesetzt, das zugrunde gelegte Baukastensystem unterstützt effizientes Änderungsmanagement (Re-Engineering).

Die vorliegende Studie untersucht den Stand der Anwendung moderner Software-Engineering- und Systemintegrationsmethoden in der Entwicklung von Roboteranwendungen, wie sie sich im Bereich der reinen Softwareentwicklung – etwa bei Web-, Office- und Geschäftsprozessanwendungen – seit Längerem durchgesetzt haben. Insbesondere wird betrachtet, inwiefern sich Konzepte des Softwareentwicklungs-Lebenszyklus und der modellbasierten Softwareentwicklung und Systemintegration auf die Robotik übertragen lassen. Eine Voraussetzung für die Nutzung dieser Ansätze und die Erzielung von Effizienzgewinnen ist, dass ein technisches Rahmenwerk für modulare, komponentenbasierte Entwicklung sowie entsprechende qualitativ hochwertige Komponenten und Entwicklungswerkzeuge verfügbar sind. Ein solches Rahmenwerk ist mit dem ROS-Framework seit einiger Zeit etabliert; leistungsfähige, modellbasierte Werkzeuge für Entwurf, Integration, Verifikation und Wartung entstehen jedoch erst in den letzten Jahren.

Methodisch baut die Studie auf einer umfassenden Nutzerbefragung in den Jahren 2023/2024 auf, deren Design und Ergebnisse in einem eigenen Kapitel beschrieben werden. In dieser Befragung wurden gegenwärtige Vorgehensweisen, genutzte Entwicklungswerkzeuge und insbesondere Schwächen der aktuellen Entwicklungsmethodik über eine breite Stichprobe von Entwicklern von Roboteranwendungen erfasst. Die dort gewonnenen Ergebnisse werden im weiteren Verlauf mit dem Stand der Technik, speziell mit Fokus auf das ROS-Framework und die modellbasierte Entwicklung, gegenübergestellt und dienen dazu, Stärken sowie Herausforderungen zu identifizieren. Darauf aufbauend werden konkrete Vorschläge für eine effizientere Softwareintegration in der Robotik entwickelt, die auf der konsequenten Anwendung modellbasierter Entwicklungsmethoden und einer angepassten Form des Softwareentwicklungs-Lebenszyklus beruhen.

Abb 1: Modellbasierte Softwareentwicklung für industrielle Roboteranwendungen. Quelle: KI-generiert mit Nano Banana 2, 2026

1.1. Begrifflichkeiten – Was ist Softwareintegration?

Softwareintegration umfasst den Prozess, verschiedene Softwaresysteme und Anwendungen, allgemeine verschiedene Subsysteme oder Komponenten, physisch oder funktional so zu kombinieren, dass sie als einheitliches System zusammenarbeiten. Ziel ist eine nahtlose Kommunikation und Zusammenarbeit zwischen den beteiligten Komponenten. Dazu gehört, Interoperabilität herzustellen und sicherzustellen, dass unterschiedliche Technologien, Plattformen und Implementierungen trotz ihrer Vielfalt zuverlässig zusammenarbeiten. Effektive Softwareintegration ist entscheidend, um vernetzte Ökosysteme und umfassende Systeme mit übergreifender Funktionalität zu schaffen, mithilfe derer Organisationen sich an veränderte Anforderungen anpassen und wettbewerbsfähig bleiben können.

Die Forschung unterscheidet verschiedene Formen der Integration, die jeweils einen eigenen Beitrag zur Verschmelzung disparater Komponenten leisten. Dazu zählen insbesondere Datenintegration, funktionale Integration, Präsentationsintegration, Portalintegration und Prozessintegration. Während Datenintegration primär den konsistenten Austausch und die gemeinsame Nutzung von Informationen über Systemgrenzen hinweg adressiert, zielt funktionale Integration auf die koordinierte Nutzung von Dienste- und Funktionsangeboten verschiedener Komponenten. Präsentations- und Portalintegration betreffen vor allem gemeinsame Benutzerschnittstellen und Zugangspunkte, während Prozessintegration die übergreifende Orchestrierung von Abläufen und Workflows in heterogenen Systemlandschaften in den Blick nimmt.

Im Kontext der Robotik steht vor allem die funktionale Integration im Vordergrund. Individuelle Funktionalitäten – implementiert in Softwaremodulen, Sensoren, Aktuatoren und Steuerungssystemen – werden so zusammengesetzt, dass ein kohärentes und funktionales Robotersystem

entsteht. Die Integration stellt sicher, dass verschiedene Softwaremodule, Sensoren, Aktuatoren und Steuerungssysteme nahtlos zusammenarbeiten, sodass der Roboter seine Aufgaben effizient und zuverlässig ausführen kann. Gleichzeitig bleibt die Robotik nicht auf funktionale Aspekte beschränkt: Datenintegration, etwa für Sensordatenströme oder Qualitätsdaten, sowie Prozessintegration, etwa bezogen auf Produktions- oder Logistikprozesse, spielen eine ebenso wichtige Rolle.

1.2. Herausforderungen der Softwareintegration in der Robotik

Die Integration von Software ist grundsätzlich herausfordernd, weil jedes System seine eigene Architektur, Datenformate, Kommunikationsprotokolle und Funktionalitäten aufweist. Heterogene Systeme zu einem konsistenten Gesamtsystem zu kombinieren, ist daher nicht trivial. Eine zentrale Herausforderung besteht darin, eine nahtlose Interoperabilität zu erreichen, die es den verschiedenen Komponenten erlaubt, effektiv zu kommunizieren und Daten fehlerfrei auszutauschen. Die Vielfalt von Technologien und Plattformen führt häufig zu Kompatibilitätsproblemen und erschwert es, ein vereinheitlichtes System zu etablieren. Hinzu kommt, dass kontinuierliche Aktualisierungen und Änderungen der beteiligten Softwarekomponenten zu Komplikationen führen können, die fortlaufende Anstrengungen erfordern, um die Funktionsfähigkeit eines einmal integrierten Systems langfristig zu erhalten.

Während der Integration kann es leicht zu Dateninkonsistenzen und Fehlern kommen. Nicht übereinstimmende Datenformate, inkompatible Protokolle oder unzureichende Synchronisation können zu Ungenauigkeiten führen und den reibungslosen Informationsfluss zwischen integrierten Systemen behindern. Gleichzeitig sollten integrierte Systeme skalierbar sein, um Wachstum oder Veränderungen in der Organisation zu ermöglichen. Dies erfordert, dass sich die Softwarearchitektur und die Integrationsmechanismen ohne grundlegende Neuentwicklung anpassen lassen.

Die Fehlersuche in integrierten Systemen ist komplex und erfordert anspruchsvolle Werkzeuge und Methoden, um Probleme effektiv zu identifizieren und zu beheben. Häufig sind Integrationsprobleme auf inkonsistente, missverstandene oder unvollständige Spezifikationen zurückzuführen. Konkrete Herausforderungen umfassen unter anderem die Identifizierung des geeigneten Integrationsniveaus, die Abstimmung divergierender Spezifikationen, den Umgang mit unterschiedlichen Implementierungstechnologien, die Bereitstellung und Konfiguration von Teilsystemen sowie die Bewältigung von Interoperabilitätsproblemen. Dies unterstreicht die Notwendigkeit gründlicher Tests sowie von Verifikation und Validierung, um die nahtlose Konvergenz von Teilsystemen zu gewährleisten. Es bedarf eines akribischen und systematischen Validierungsprozesses, um die Zahl und Schwere möglicher Integrationsprobleme zu minimieren.

In der Robotik verschärfen sich diese Herausforderungen noch einmal. Robotersysteme müssen in der realen Welt operieren, mit dynamischen Umgebungen interagieren und auf unvorhergesehene Situationen reagieren. Die Anwendungen werden zunehmend komplexer, da die Einsatzgebiete der Robotik vielfältiger und anspruchsvoller werden. Damit steigen die Anforderungen an Integrationsprozesse und erfordern adaptive Strategien und Methoden, um mit dem komplexen Zusammenspiel von Technologien und Funktionalitäten umgehen zu können. Hinzu kommt, dass die Entwicklung der einzelnen Komponenten und Module in der Robotik bislang nur begrenzt standardisiert ist. Es existieren kaum etablierte Verfahren oder Methoden, die die Qualität und Zuverlässigkeit der Integration systematisch garantieren.

Oft reichen die Integrationsherausforderungen über rein technische Aspekte hinaus und berühren ein breiteres Spektrum von Spezifikationsprozessen, Teststrategien und sich ständig weiterentwickelnden Anwendungen. Dies betrifft insbesondere sicherheitsrelevante Robotersysteme, für die zusätzliche regulatorische Anforderungen gelten. Während regulatorische Vorschriften den gesamten Lebenszyklus eines Hard- und Softwaresystems betreffen und über reine Integrationsfragen hinausgehen, sind die Auswahl geeigneter, zueinander passender Softwarekomponenten, die Auswahl angemessener Rechenhardware und die Verteilung von Funktionen auf Hardwareelemente sowie deren Kommunikation ein zentrales Integrationsproblem.

1.3. Stand der Technik der Integration mit ROS

Ein weit verbreitetes Rahmenwerk für die Entwicklung von Robotersoftware ist das Robot Operating System (ROS). Anhand eines einfachen Beispiels lässt sich veranschaulichen, wie ein Roboter mithilfe vorhandener Softwarekomponenten unter ROS integriert werden kann. Angenommen, ein Unternehmen benötigt einen Roboter, der in einer Produktionslinie Medikamentenschachteln je nach Typ sortiert. Ein Förderband bringt die Schachteln in eine Zelle mit einem Meter Kantenlänge, in der ein Roboterarm mit Greifer und Kamera die Sortieraufgabe übernimmt.

Sobald die Anwendung beschrieben ist, muss die Softwarearchitektur entworfen werden. Ein erster Schritt ist die Auswahl der Middleware. In diesem Beispiel fällt die Wahl auf ROS, weil es zahlreiche vorbereitete Softwarekomponenten und Hardwaretreiber bereitstellt. Anschließend werden die konkreten Komponenten ausgewählt, die benötigt werden. Für die Anwendung sind mindestens ein Roboterarm, ein Greifer und eine Kamera erforderlich. Um die Integration zu vereinfachen, werden Komponenten gewählt, für die es in ROS bereits Treiber gibt.

Die Integration erfolgt in ROS im Kern in drei Schritten. Zunächst werden die einzelnen Treiber konfiguriert, was typischerweise über Konfigurationsdateien im YAML-Format geschieht. Darin werden etwa Parameter für die Hardwarekommunikation oder Betriebsmodi hinterlegt. Im zweiten Schritt wird die Kinematik des Robotersystems beschrieben, indem eine Datei im URDF-Format erstellt wird. Diese beschreibt die Verbindungen zwischen den Hardwarekomponenten sowie die physische Umgebung, in die der sich bewegende Roboter eingebettet ist. Im dritten Schritt werden Ausführungsdateien erstellt, die die Komponenten einschließlich ihrer jeweiligen Konfigurationen starten. Dies geschieht in ROS über sogenannte Launch-Dateien im XML-Format, in denen festgelegt wird, welche Knoten mit welchen Parametern in welcher Reihenfolge gestartet werden.

Werden diese Schritte sauber ausgeführt, werden alle entstehenden Dateien Teil eines neuen Repositoriums, das von den Bibliotheken der genutzten Treiber und weiterer ROS-Pakete abhängt. Um sicherzustellen, dass alle Abhängigkeiten und ihre Versionen konsistent definiert sind, ist es sinnvoll, Verfahren der kontinuierlichen Integration (Continuous Integration, CI) einzusetzen. Typischerweise wird eine virtuelle Umgebung geschaffen, etwa in einem Docker-Container, die auf einem generischen Abbild des vorgesehenen Laufzeitbetriebssystems mit einer bestimmten ROS-Version basiert. In dieser Umgebung wird das Repositorium geklont, und die Abhängigkeiten werden mithilfe gängiger ROS-Werkzeuge, etwa dem Befehl »rosdep«, aus dem jeweiligen Release der gewählten ROS-Version installiert.

Dieser Schritt erfordert oft zusätzlichen Aufwand, weil die Release-Versionen der Abhängigkeiten nicht immer passend sind und der Code im Einzelfall aus dem Quellcode installiert werden muss. Dazu muss der Integrator das zu integrierende Paket verstehen und zusätzliche Schritte durchführen, um die jeweils passende Version zu erhalten. Es empfiehlt sich außerdem, Konsistenztests durchzuführen, um zumindest die von Hand erstellten XML- und URDF-Dateien auf Tippfehler und einfache Inkonsistenzen zu prüfen. In ROS selbst existieren bislang kaum Testtechniken, die bereits in der Entwurfsphase anwendbar wären. Für Tests und Validierung muss der Code in der Regel direkt auf dem realen Roboter ausgeführt werden, und die Interaktionen zwischen den Modulen werden nacheinander empirisch getestet.

Je nach Verfügbarkeit der Hardware ist es teilweise möglich, eine Simulationsumgebung aufzubauen und Teile der Tests in der Simulation durchzuführen. Um Tests an der realen Anwendung durchzuführen, muss zunächst die Bereitstellungsphase abgeschlossen werden: Das Betriebssystem, ROS und die neu integrierte Software mit allen abgeleiteten Abhängigkeiten müssen installiert und die Kommunikationsports der Komponenten so konfiguriert werden, dass die Softwarekomponenten Zugriff auf die Hardwaregeräte des Robotersystems und seiner Umgebung erhalten.

Nach Abschluss der Entwicklung werden alle Komponenten einzeln getestet, und zwar sowohl auf der Ebene der Architektur, also ihrer Kommunikationsschnittstellen, als auch auf Ebene der Funktionalität. So wird geprüft, ob sich der Roboterarm mit der spezifizierten Geschwindigkeit und Beschleunigung bewegt, ob die Kamera korrekt kalibriert ist und scharfe Bilder liefert und ob Sensoren und Aktuatoren innerhalb ihrer Toleranzen arbeiten. Diese Testaufgaben sind sehr aufwendig, dauern üblicherweise deutlich länger als die eigentliche Implementierung und erfordern häufig iterative Konfigurationsänderungen. Zusätzlich müssen die Encoder der Manipulatoren und die Position der Kamera relativ zum Koordinatensystem des Arms kalibriert werden.

Bis zu diesem Punkt ist vor allem der Low-Level-Teil des Systems integriert. Danach folgt die Integration der High-Level-Module. Für das dargestellte Beispiel sind mindestens eine Komponente zur Bewegungsplanung und eine Komponente zur Objekterkennung erforderlich. Nach der Auswahl geeigneter Komponenten werden diese analog zur vorherigen Stufe einzeln konfiguriert. Je leistungsfähiger und komplexer eine Komponente ist, desto spezieller ist sie zu konfigurieren und desto mehr Expertenwissen über die jeweilige Komponente ist erforderlich. Die Qualität der Dokumentation entscheidet maßgeblich darüber, ob sich dieses Wissen selbstständig aneignen lässt oder ob zusätzliche Experten hinzugezogen werden müssen.

Nach der Konfiguration werden erneut alle Komponenten zusammengeführt und das Gesamtsystem getestet. Ergänzend müssen Softwarekomponenten für Systemüberwachung und -diagnose integriert werden, um den Betrieb zu überwachen und Fehler frühzeitig zu erkennen. Zusammen mit dem CI-System erleichtert dies die anschließende Wartungsphase, in der Fehler behoben und Upgrades durchgeführt werden.

Dank ROS und der Auswahl relativ standardisierter Komponenten sind die Schnittstellen der Komponenten untereinander grundsätzlich kompatibel, sodass keine zusätzlichen Adapterkomponenten zur Übersetzung zwischen inkompatiblen Schnittstellen erforderlich sind. In hybriden Systemen, die mehrere Middlewares und Kommunikationsprotokolle kombinieren, entsteht ansonsten in allen Phasen die zusätzliche Schwierigkeit, diese Brücken zu implementieren und zu warten. Der oben ausgeführte Integrationsprozess ist trotz der Unterstützung durch ROS sehr ineffizient und schwer zu standardisieren. Er verhindert eine wirklich systematische und wiederverwendbare Entwicklung neuer Systeme. Gleichzeitig erfordert er sowohl tiefe Expertise in Robotik als auch in der softwaretechnischen Entwicklung und Konfiguration, um überhaupt durchgeführt werden zu können.

1.4. Warum genügt ein Komponenten-basiertes Software-Framework nicht?

Ein komponentenbasiertes Framework wie ROS ist eine wesentliche Voraussetzung für modulare und prinzipiell wiederverwendbare Softwareentwicklung. Es ermöglicht, komplexe Systeme in klar abgegrenzte Teile – Komponenten – zu zerlegen und diese in unterschiedlichen Anwendungen wiederzuverwenden. Die bloße Verfügbarkeit solcher Komponenten, sei es aus externen Quellen oder im Rahmen eines eigenen Entwicklungsprojekts, reicht jedoch nicht aus, um die Ziele effizienter Software- und Systemintegration zu erreichen. Der Grund liegt in der minimalistischen Anforderung an ein generisches Komponentensystem: Damit ein Software-Framework als komponentenbasiert gelten kann, genügt es im Prinzip, dass die Softwareartefakte in separierbare Komponenten strukturiert sind und sich einzelne dieser Komponenten austauschen lassen, ohne den übrigen Code massiv anzupassen.

Für eine wirklich effiziente, qualitätssicherte und werkzeuggestützte Entwicklung müssen über diese Minimalanforderung hinaus verbindliche Regeln definiert sein, wie Komponenten intern strukturiert sind und wie sich aus ihnen Systeme zusammensetzen lassen. Dazu gehören insbesondere formale Schnittstellenbeschreibungen und vereinheitlichte Datentypen, sodass jede Komponente, die beispielsweise Bildinformationen benötigt, diese im gleichen, eindeutig spezifizierten Format erhält. Umgekehrt müssen alle Komponenten, die Bildinformationen bereitstellen, dieses Format zuverlässig einhalten – unabhängig von der verwendeten Programmiersprache, Hardwareplattform oder Middleware-Variante. Die Definition solcher standardisierten Schnittstellen ist jedoch grundsätzlich nicht allgemeingültig lösbar. Tragfähige Lösungen lassen sich immer nur in Bezug auf klar umrissene Anwendungsdomänen finden. Unterschiedliche Frameworks wie ROS, SmartSoft und vergleichbare Ansätze bieten daher domänenspezifische Systematiken für Komponentenschnittstellen mit jeweils unterschiedlichem Detaillierungsgrad und Reifegrad der Standardisierung.

Einheitlich definierte Schnittstellen reichen zudem nur für einen Teil der im professionellen Einsatz benötigten Überprüfungen aus. Sie stellen sicher, dass Komponenten syntaktisch kompatibel sind, also »miteinander reden« können, und ein Softwaresystementwurf auf Datentypkompatibilität geprüft werden kann. Für die industrielle Praxis ist aber ebenso wichtig, dass Komponenten semantisch zueinander passen – dass sie sich also »verstehen«. So sagt der Datentyp »Bildinformation« allein wenig darüber aus, ob eine Sensorik- oder Bildverarbeitungskomponente im konkreten Anwendungskontext überhaupt geeignet ist. Für sicherheitsrelevante oder performanzkritische Anwendungen sind Eigenschaften wie Bildrate, Auflösung, Latenz, Kontrastumfang, Rauschverhalten oder garantierte Verfügbarkeit entscheidend. Erst wenn solche qualitativen und nicht-funktionalen Eigenschaften formal beschrieben und zwischen Komponenten abgestimmt werden, kann bewertet werden, ob eine bestimmte Kombination von Sensoren, Algorithmen und Kommunikationskanälen regulatorische Vorgaben einhält und die geforderte Systemleistung erreicht. Dies betrifft nicht nur die Auswahl der Softwarekomponenten selbst, sondern auch die Auswahl der Rechenhardware, die Dimensionierung von Busverbindungen sowie die Verteilung von Funktionen auf verschiedene Rechenknoten des Robotersystems – allesamt zentrale Integrationsentscheidungen.

Regulatorische Vorschriften zu erfüllen, ist eine Querschnittsaufgabe über den gesamten Lebenszyklus eines Hardware-/Softwaresystems hinweg und geht damit über das unmittelbare Integrationsproblem hinaus. Gleichwohl ist die Integration derjenige Prozessschritt, in dem viele regulatorisch relevante Festlegungen praktisch wirksam werden: Hier wird entschieden, welche Komponenten kombiniert werden, wie sie auf Hardware verteilt sind, welche Redundanzen und Überwachungsmechanismen vorgesehen sind und wie Schnittstellen und Kommunikationspfade konkret ausgestaltet werden. Ein komponentenbasiertes Framework, das diese Entscheidungen

nicht explizit und überprüfbar abbildet, kann die Einhaltung von Normen und Sicherheitsanforderungen daher nur begrenzt unterstützen.

Der Unterschied zwischen bloßer Kombinierbarkeit und semantisch sinnvoller Kombinierbarkeit von Komponenten lässt sich an einem einfachen Beispiel außerhalb der Robotik verdeutlichen. In der Standardbibliothek der Programmiersprache Python existieren zahlreiche Funktionen zur Arbeit mit Dateisystemobjekten. Ursprünglich akzeptierten diese Funktionen schlicht Zeichenketten als Bezeichner für Dateien und Ordner. Zeichenketten sind ein universelles und sehr flexibles Schnittstellenformat, bieten aber keinerlei semantische Information darüber, was konkret kommuniziert wird. Später wurde das »Path«-Objekt eingeführt, das eine vom Betriebssystem abstrahierte Repräsentation von Pfaden zu Dateisystemobjekten darstellt. Funktionen wurden schrittweise so angepasst, dass sie statt beliebiger Zeichenketten explizit »Path«-Objekte akzeptieren und liefern. Auf diese Weise kann ein Pfad zu einem Dateisystemobjekt bereits auf Typ-Ebene von anderen Textdaten unterschieden werden; zudem lassen sich etwa Pfade zu Ordnern von Pfaden zu Dateien differenzieren. Validierungswerkzeuge können dadurch schon vor der ersten Ausführung prüfen, ob übergebene Parameter zulässig sind. Die Einführung semantisch reichhaltiger Typen ermöglicht also frühzeitige, automatisierte Plausibilitätsprüfungen und verhindert ganze Klassen von Laufzeitfehlern.

Übertragen auf die Robotik bedeutet dies: Ein komponentenbasiertes Framework wie ROS stellt zwar die grundlegende Infrastruktur für die Kommunikation zwischen Komponenten bereit und erleichtert die Wiederverwendung von Software. Ohne ergänzende, formale Beschreibung der Semantik von Schnittstellen, Qualitätsmerkmalen, Laufzeit- und Sicherheitsanforderungen bleibt die Systemintegration jedoch im Wesentlichen ein manueller, erfahrungsgetriebener und fehleranfälliger Prozess. Erst durch modellbasierte Ansätze, die solche Aspekte in expliziten, maschinenlesbaren Modellen erfassen, können Integrationsentscheidungen systematisch unterstützt, automatisierte Konsistenz- und Kompatibilitätsprüfungen durchgeführt und Teile des Integrations- und Testaufwands reduziert werden.

Damit spannt sich der Bogen von der heute dominierenden, komponentenbasierten Entwicklung mit Frameworks wie ROS hin zu modellbasierten, durchgängigen Methoden für Entwurf, Integration, Verifikation und Wartung. Die folgenden Kapitel greifen diese Motivation auf, analysieren auf Basis der durchgeführten Umfrage den Status quo in Industrie und Forschung und entwickeln darauf aufbauend Vorschläge, wie modellbasierte Verfahren und ein angepasster Softwareentwicklungs-Lebenszyklus die Effizienz und Qualität der Softwareintegration in der Robotik substanziell verbessern können.

2. Studie zur Softwaremodellierung und Ergebnisse

Um eine Übersicht über den Stand der Softwareintegration in der Robotik zu schaffen, wurde eine Umfrage unter Forschern und Anwendern auf diesem Gebiet durchgeführt. Untersuchungsgegenstand war, wie die Integration von Robotersoftware derzeit durchgeführt wird, um Ähnlichkeiten mit traditionellen Software-Entwicklungsmethoden zu ermitteln und aus den Erfahrungen in anderen Bereichen des Software und System Engineerings Empfehlungen für eine effizientere Systemintegration in der Robotik ableiten zu können.

Die Umfrage behandelte die folgenden Punkte:

- Wie können etablierte Software-Engineering-Methoden (teilweise) auf die Integration von Robotern angewendet werden?
- Unterliegt der Softwareintegrationsprozess in der Robotik bestimmten gemeinsamen Vorgehensweisen?
- Welches sind die größten Herausforderungen bei der Softwareintegration in der Robotik?
- Welche neuen Werkzeuge wären am hilfreichsten, um Integratoren bei der Anwendung etablierter Software-Integrationsprozesse zu unterstützen?

2.1. Profil der Teilnehmer

Im ersten Teil der Umfrage wurden allgemeine statistische Daten erhoben, um die Befragten nach der Art ihres Profils zu klassifizieren und sind in Abbildung 1 abgebildet. Von den 118 Befragten arbeiten 45,3 Prozent in der Privatwirtschaft, 28,2 Prozent an Universitäten und 22,2 Prozent in Forschungszentren. Die restlichen 4,3 Prozent sind Angestellte der Regierung, im Ruhestand oder freiberuflich tätig.

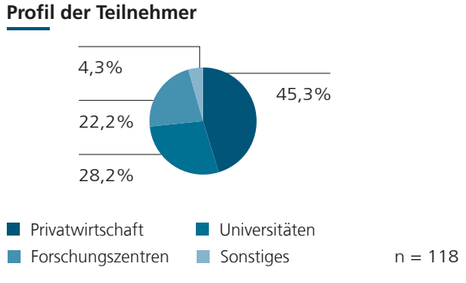

Abb 2: Profil der Teilnehmer

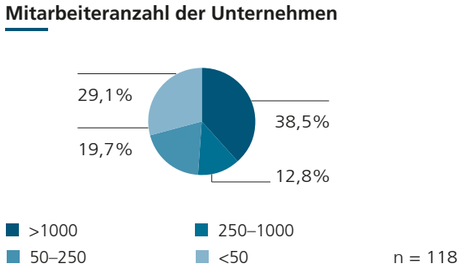

Abb. 2: Mitarbeiteranzahl der Unternehmen

Abbildung 2 stellt die Verteilung der Unternehmensgröße dar. 38,5 Prozent der Befragten arbeiten in einem Unternehmen mit mehr als 1000 Mitarbeitenden, während 29,1 Prozent zu kleinen Unternehmen (unter 50 Mitarbeitenden) gehören. Der Rest verteilt sich auf mittelgroße Unternehmen oder Institutionen, 19,7 Prozent auf Unternehmen mit 50 bis 250 Mitarbeitenden und 12,8 Prozent auf Unternehmen mit 250 bis 1000 Mitarbeitenden. Bei der Frage nach der Haupttätigkeit des Unternehmens zeigte sich, dass 56,4 Prozent der Befragten für ein Forschungsunternehmen arbeiten (und 49,6 Prozent ausschließlich in der Robotik), 31,6 Prozent für Softwareanbieter, 23,9 Prozent für Unternehmen, deren Haupttätigkeit die Integration von Roboteranwendungen ist, und 16,2 Prozent für Roboterhersteller. Mehrfachnennungen waren möglich.

Die Umfrage wurde breit gestreut, damit der Personenkreis, der an der Integration von Robotersystemen beteiligt ist, umfassend abgedeckt ist. Die genutzten Verteiler richteten sich hauptsächlich an Fachleute mit einem technischen Profil und nicht an ein Projektmanagementprofil. Es ist daher nicht überraschend, dass 71,8 Prozent der Befragten Aufgaben der Softwareentwicklung wahrnehmen, gefolgt von Architekturdesign (44,4 Prozent), Systemintegration (36,8 Prozent), Projektmanagement (31,6 Prozent) und Produktverantwortlichen (20,5 Prozent), wobei Mehrfachnennungen möglich waren. Neben den vorgenannten Kategorien konnten Umfrageteilnehmer eigene Kategorien hinzufügen. Unter anderem haben die Befragten Tätigkeiten wie Lehre an der Universität, wissenschaftliche Forschung oder Unternehmensleitung hinzugefügt. Weil die Umfrage primär Software-Entwickler adressiert und weil ein vordefiniertes Integratorprofil in der Robotik fehlte, hat nur eine Person angegeben, dass ihre einzige und ausschließliche Rolle in Projekten die eines Systemintegrators sei. Die am häufigsten genannte Profilkombination ist eine Person, die für die Softwareentwicklung und das Design der Systemarchitektur sowie deren Integration verantwortlich ist.

2.2. Erfasste Daten

2.2.1. Traditioneller Prozess der Software-Entwicklung und seine Anwendung in der Software-Integrationsphase

Dieser Fragenblock beginnt mit einem Erläuterungstext zu den Schritten des Prozesses, die bewertet werden. Dazu gehören die folgenden Schritte:

1. **Analyse der Anforderungen:** Welche Bedingungen muss das resultierende Robotersystem erfüllen?
2. **Entwurf:** Phase, in der der Lösungsansatz durch Prüfung der Benutzeranforderungen, Definition der Architektur und Auswahl der zu verwendenden Komponenten erfasst wird.
3. **Code-Implementierung:** Schreiben des Codes, der den Lösungsansatz realisiert. Dies ist sowohl Code, mit dem das System gestartet werden kann (d. h. Konfiguration der Module und Implementierung des Codes für die Interaktion (oft »Boiler Plate Code« genannt)) als auch Code zur Realisierung der Funktionslogik, wenn noch keine passende Komponente existiert.
4. **Verifizierung, Validierung und Tests:** Bei der Verifizierung und Validierung wird geprüft, ob eine im Schritt »Entwurf« konzipierte und im Schritt »Code-Implementierung« umgesetzte Lösung die erwarteten Produktanforderungen erfüllt. Die Verifizierung prüft, ob die realisierte Lösung (also das fertige Robotersystem) dem Lösungsansatz aus der Entwurfsphase entspricht. Die Validierung prüft, ob der Entwurf und das realisierte System die Anforderungen erfüllen, das Robotersystem also unter Realbedingungen wie vorgesehen funktioniert. Durch systematisches Testen werden sowohl konzeptionelle Fehler als auch Umsetzungsfehler aufgedeckt und behoben. Verifikation und Validierung werden zu unterschiedlichen Zeiten im Entwicklungsprozess zunächst auf Entwurfsbasis, später auf Komponenten- und noch später auf Systemebene durchgeführt.
5. **Bereitstellung:** Dies umfasst alle Aktivitäten, die notwendig sind, um die Software für die praktische Anwendung verfügbar zu machen. Hierzu zählen unter anderem das Zusammenstellen der Software zu installierbaren Softwarepaketen, aber auch die Erstellung von Dokumentationen für die Installation, Inbetriebnahme, Konfiguration und Wartung.
6. **Überwachung und Wartung:** Maßnahmen, die sicherstellen sollen, dass das integrierte Robotersystem die Produktanforderungen langfristig erfüllt. Dies umfasst unter anderem das Aufdecken versteckter Fehler (die in der Validierungsphase nicht gefunden wurden) und Anpassungen an sich ändernde Systemumgebungen, beispielsweise wenn externe Softwaresysteme, mit denen das Robotersystem verbunden wird, aktualisiert werden.

Die erste Frage ist, ob die Entwicklerinnen und Entwickler einen dieser Schritte während des traditionellen Integrationsprozesses befolgen.

In diesem Fall sind einige Unterschiede in den Antworten zwischen Praktikern aus der Industrie und aus dem akademischen Bereich festzustellen. Die Reihenfolge der am häufigsten und am wenigsten durchgeführten Phasen ist für beide Profile gleich: (1) Code-Implementierung, (2) Entwurf, (3) Validierung, Verifizierung und Prüfung, (4) Anforderungsanalyse, (5) Bereitstellung und (6) Überwachung und Wartung, wobei »1« am häufigsten und »6« am wenigsten bedeutet. Es gibt jedoch einen großen Unterschied in der Anzahl der Befragten aus jedem Profil, die die Schritte befolgen. Während in der Industrie 48 Prozent sagen, dass sie alle diese Schritte durchführen, ist dieser Prozentsatz in der Forschung deutlich auf nur 14 Prozent reduziert. Die Schritte, die dieses Gleichgewicht eindeutig stören, sind die Anforderungsanalyse sowie die Überwachung und Wartung des Systems, die von einem großen Teil des akademischen Sektors ausgelassen werden.

Auf die Frage nach den Schritten, die während des Integrationsprozesses befolgt werden sollten (aber nicht immer werden), gibt es jedoch unterschiedliche Antworten, wobei wiederum zwischen den Präferenzen der Industrie und des akademischen Sektors unterschieden werden kann. Hier ändern sich die Trends, und es zeigt sich, dass der akademische Bereich eine viel größere Notwendigkeit sieht, alle Schritte zu befolgen (45 Prozent unterstützen dies), und dass er die Notwendigkeit einer Überwachungs- und Wartungsphase deutlich wichtiger einschätzt. Diese Befragten sehen die Notwendigkeit, dass sie die Art und Weise der Softwareintegration verbessern und weitere Schritte durchführen müssten.

Welche Phasen des Integrationsprozesses werden befolgt?

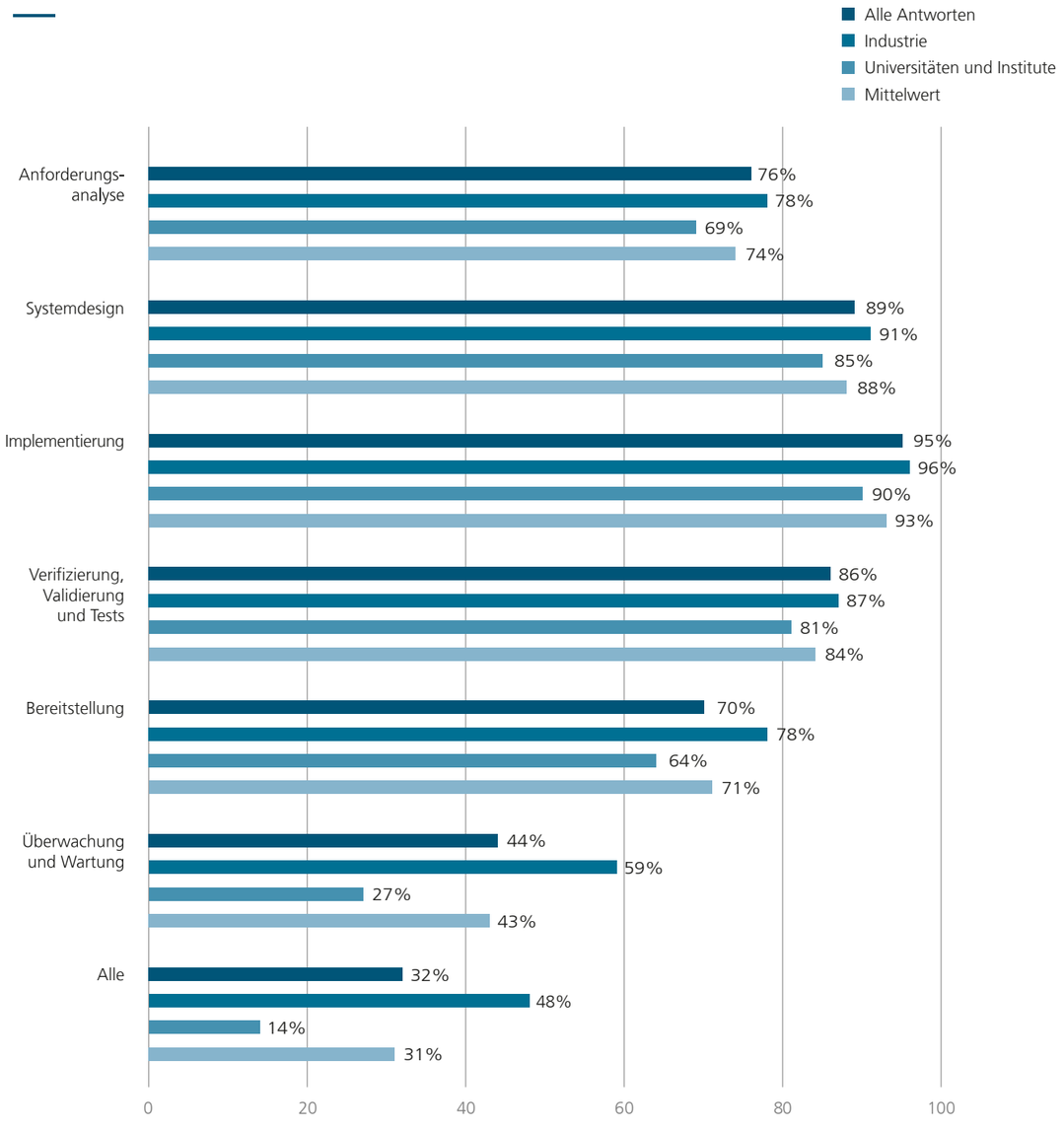

Abb. 4: Verteilung der ANtworten auf die Frage nach den Schritten, die während des Integrationsprozesses durchgeführt werden.

Ebenfalls auffällig ist, dass die Anforderungsanalyse zwar in der Forschung für wichtiger gehalten wird, in der Privatwirtschaft jedoch das Gegenteil der Fall ist. Hier sind die Befragten der Meinung, dass diese Phase nicht notwendig ist. In den Freitextfragen zu den fehlenden Phasen des vorher vorgestellten sechsstufigen Vorgehens finden sich viele Ideen, von denen einige von allen Befragten geteilt werden. Dazu gehören unter anderem die Analyse bestehender Lösungen, System-Prototyping-Phase, Simulation, Dokumentation, Demonstration und Kundenschulung in Verbindung mit Support. Einige Befragte schlagen vor, bei der Integration von Robotersystemen dem V-Modell zu folgen.

Welche Phasen des Integrationsprozesses sollen befolgt werden?

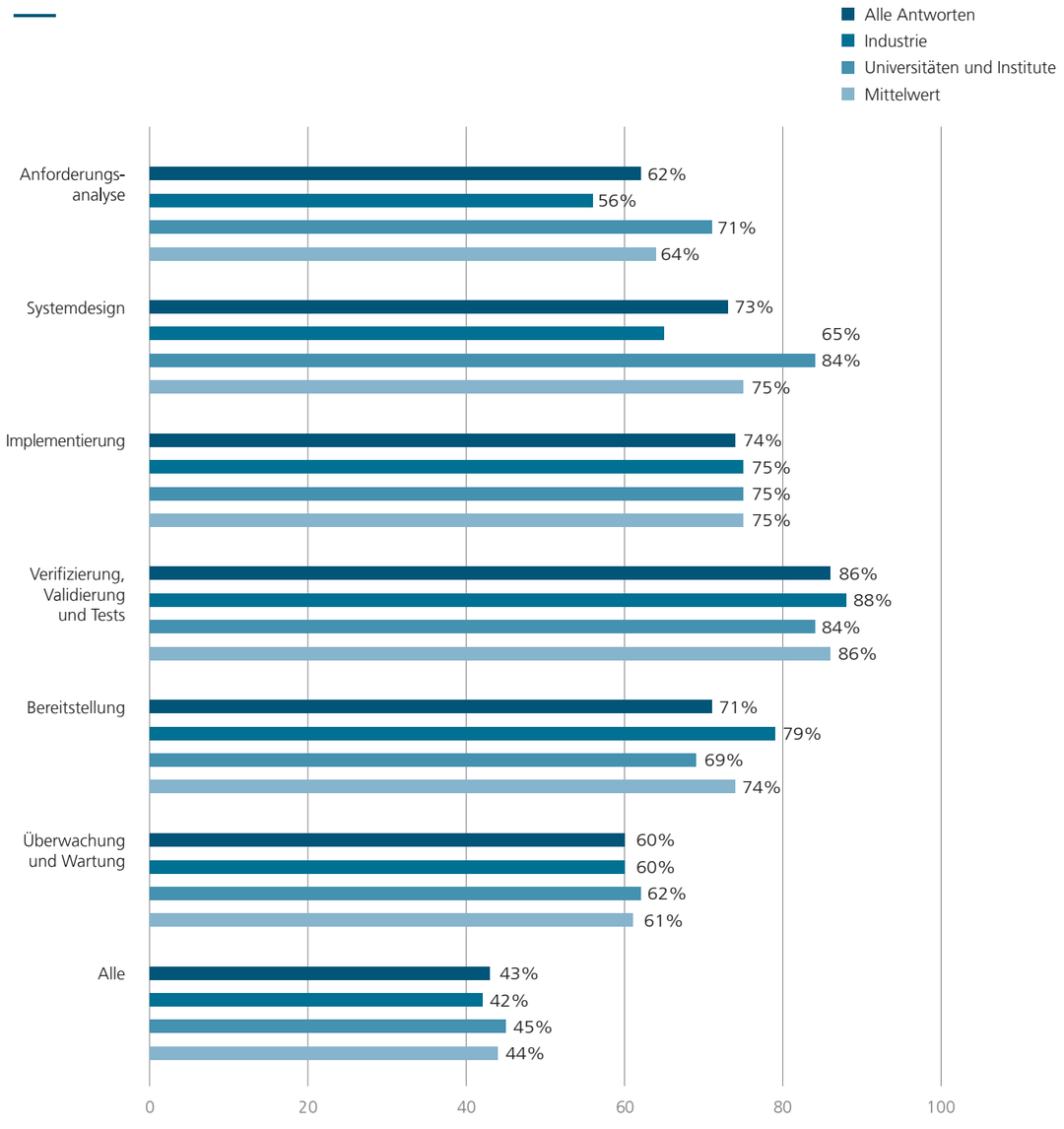

Abb. 5: Verteilung der Antworten auf die Frage nach den Schritten, die während des Integrationsprozesses durchgeführt werden sollten.

Am Schluss dieses Abschnitts steht die Frage, wie die Entwicklungsschritte ausgeführt werden sollten. Hierfür gibt es drei Standardoptionen: (1) linear, d. h. ein Schritt nach dem anderen; (2) zyklisch und kontinuierlich, d. h. nach dem letzten Schritt beginnt der Integrationsprozess von vorn (DevOps-Stil) oder (3) teilweise zyklisch, d. h. nach bestimmten Schritten geht man zurück, um Aufgaben aus vorherigen Phasen anzupassen.
Auch hier sind die Antworten je Profil der Befragten unterschiedlich. So unterstützen 61 Prozent der Wissenschaftlerinnen und Wissenschaftler den teilzyklischen Modus, während 52 Prozent der Befragten aus der Privatwirtschaft den kontinuierlichen zyklischen Modus bevorzugen. Neben anderen Ideen schlagen mehrere Befragte einen Hybridmodus zwischen einem teilweise zyklischen und einem kontinuierlichen zyklischen Prozess vor, d.h. nur einige Phasen sollten kontinuierlich durchgeführt werden. Andere Befragte weisen darauf hin, dass es stark von dem Produkt abhängt, wie der Entwicklungsprozess verlaufen sollte. Entsprechend sollte er individuell an jedes Projekt angepasst werden.

2.2.2. Herausforderungen und Verbesserungen

Um die in der Studie erfassten Herausforderungen (vgl. Abbildung 6 und Liste in 2.3.1) in der Softwareintegration besser einordnen zu können, sind die folgenden zusätzlichen Informationen notwendig:

- Welches Framework wird für die Entwicklung verwendet? Eventuelle Einschränkungen dieses Frameworks können die Ursache für die Herausforderungen sein. Hier waren Mehrfachnennungen möglich. Dabei gaben 80,2 Prozent der Befragten ROS an, die zweithäufigste Wahl war ROS 2 mit 57,7 Prozent, gefolgt von 16,6 Prozent für OPC-UA und 6,3 Prozent für Yarp.
- Von den Befragten gaben 68,7 Prozent an, dass sie Instrumente einsetzten, die dabei helfen, diese Herausforderungen zu vermeiden.

Interessanterweise fanden sich bei der Rangfolge der Herausforderungen während des Integrationsprozesses keine nennenswerten Unterschiede zwischen den Antworten aus der Industrie und der akademischen Gemeinschaft. Beide sehen in der dynamischen Anpassung an die Umgebung die größte Herausforderung für die Entwicklung von Robotersystemen (44 Prozent der Befragten nannten dies an erster Stelle). An zweiter Stelle stehen die Sicherheitszertifizierung (20 Prozent) und die Validierung des fertigen Systems (16 Prozent). Auf der anderen Seite des Spektrums, also bei der kleinsten Herausforderung, finden sich die Wiederverwendbarkeit von Software und die Interoperabilität zwischen Middlewares und Systemen. Dies ist nicht überraschend, da die meisten Befragten ROS (oder ROS 2) verwenden und dieses Framework insbesondere die Wiederverwendbarkeit und prinzipielle Interoperabilität von Softwarekomponenten fördert und sie keine Probleme mit der Interoperabilität haben, da sie keine hybriden Systeme anstreben.

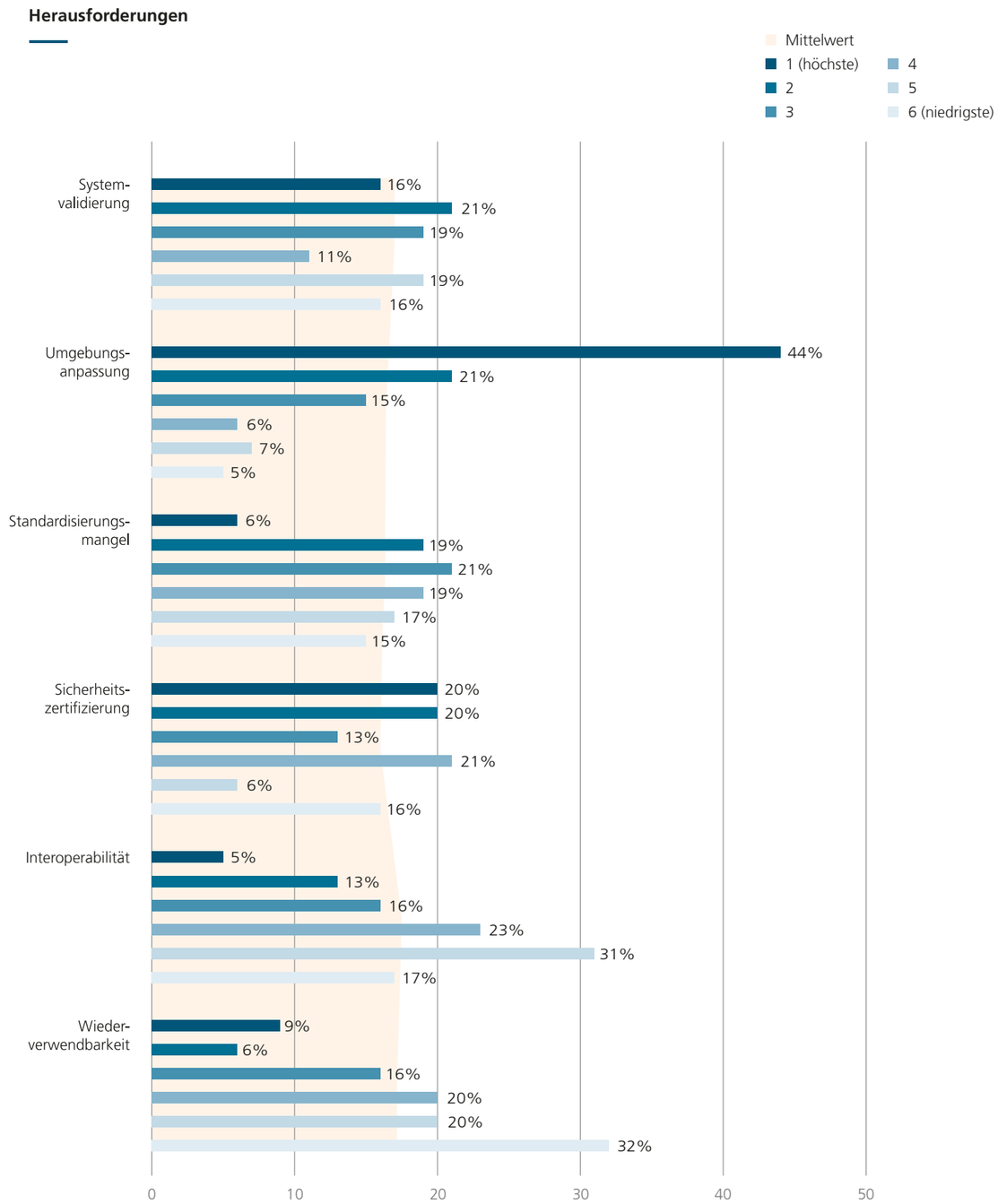

Abb. 6: Ergebnisse der Frage nach den größten Herausforderungen während des Software-Integrationsprozesses. Die Befragten sollten alle vorgeschlagenen Herausforderungen von der höchsten (1) bis zur niedrigsten (6) einstufen.

2.2.3. Werkzeuge zur Unterstützung des Integrators

Dieser Teil der Umfrage zielte darauf ab, die Tools zu ermitteln, die Integratoren aktuell verwenden. Er ist als Freitextabfrage gestaltet, um die Antworten nicht auf einen bestimmten Tooltyp zu beschränken. Aber selbst bei dieser freien Texteingabe nannten 50 Prozent der Befragten Versionskontrollsysteme (wie GitHub oder GitLab) zur Unterstützung bei der Implementierung sowie Systeme zur kontinuierlichen Integration. Der am zweithäufigsten genannte Tooltyp (7 von 53 Antworten) waren Deployment-Containerisierungstools wie Docker (zur einfachen und reproduzierbaren Bereitstellung von Softwarekomponenten und vorkonfigurierten Subsystemen) zusammen mit integrierten Entwicklungsumgebungen wie z.B. Eclipse ode VS-Code (sogenannte IDEs) (7 von 53 Antworten), gefolgt von Simulatoren (wobei viele Gazebo nannten) und einer Auswahl an ROS-spezifischen Tools (wie rosgraph oder rosbag). 6 der Befragten verwiesen auf modellbasierte Werkzeuge, darunter SysML, SmartMDSD oder Simulink. Auf die Frage, in welchem Entwicklungsstadium diese Werkzeuge eingesetzt werden, antworteten 76,3 Prozent, dass sie in der Implementierungsphase verwendet werden, 73,7 Prozent für Validierungs- und Testzwecke und 55,3 Prozent in der Einführungsphase. Dahinter folgten 34,2 Prozent für den Systementwurf und 32,9 Prozent für die Überwachung und Wartung des Systems. Schließlich gaben nur 7,9 Prozent an, dass sie Werkzeuge zur Unterstützung für die Anforderungsanalyse verwenden.

2.2.4. Gewünschte Werkzeuge zur Unterstützung des Integrators

In diesem letzten Abschnitt der Umfrage ging es darum, herauszufinden, welche Bedürfnisse die Robotik-Community in Bezug auf neue Werkzeuge hat. Tatsächlich glauben fast alle Befragten (95,5 Prozent), dass Werkzeuge den Integrationsprozess unterstützen, vereinfachen und standardisieren können.

Die meisten Freitextantworten (20 von 78 Antworten) befürworten den Einsatz von Werkzeugen, um gemeinsame Schnittstellen zu verwalten, und bis zu einem gewissen Grad auch den Einsatz von Standards. Dies gilt auch dafür, die Dokumentationsprozesse zu vereinheitlichen. Weiterhin sehen die Befragten Nutzen bei einem neuen Werkzeug, das sie in den Phasen der Validierung, Verifizierung und Prüfung des Systems unterstützt (17 von 78 Befragten).

Diese Aufgaben der Validierung, Verifizierung und des Tests sind sehr mühsam, wenn sie manuell ausgeführt werden; jede Form der Automatisierung in diesem Bereich wäre daher sehr vorteilhaft. Andere Studienteilnehmer erwähnen Tools zur Codegenerierung, die nicht nur Zeit und Mühe sparen, sondern auch die Fehleranfälligkeit des Codes verringern. Eine Handvoll Teilnehmer geht noch weiter und spricht von modellbasierten Tools, die eine formale Beschreibung sowohl der Hardware- als auch der Softwareeigenschaften von Komponenten ermöglichen. Eine Antwort besagt sogar, dass »Aspekte explizit gemacht und eine Rückverfolgbarkeit von den Anforderungen bis hin zum Quellcode erreicht werden kann«. Im Einklang mit dieser letzten Aussage verweisen 74,1 Prozent der Befragten auf die Nützlichkeit von modellbasierter Softwareentwicklung in der Robotik, 65,7 Prozent auf die Vorteile grafischer Schnittstellen zur intuitiven Modellierung, und 61,1 Prozent sehen auch den Bedarf an Werkzeugen, die Codegeneratoren enthalten. Schließlich wurde auch die formale Verifizierung von Systemen, Simulatoren und datengesteuerten Werkzeugen genannt.

Bei der Frage, in welcher Phase der Softwareintegration Werkzeuge am vorteilhaftesten wären, stehen zwei Optionen mit sehr ähnlichen Ergebnissen an erster Stelle. So wünschen die Roboterentwickler vor allem ein Werkzeug, das den Schritt der Validierung, Verifizierung und Prüfung erleichtert. Am zweithäufigsten gewünscht ist, mit geringem Unterschied, ein Werkzeug für die Entwurfsphase. Demgegenüber sehen die Befragten in der Phase der Anforderungsanalyse den geringsten Bedarf an Unterstützung. Bei diesem Thema gibt es keine bemerkenswerten Unterschiede zwischen den Antworten aus der Wissenschaft und der Privatwirtschaft. Beide Sektoren kommen also zu ähnlichen Schlussfolgerungen.

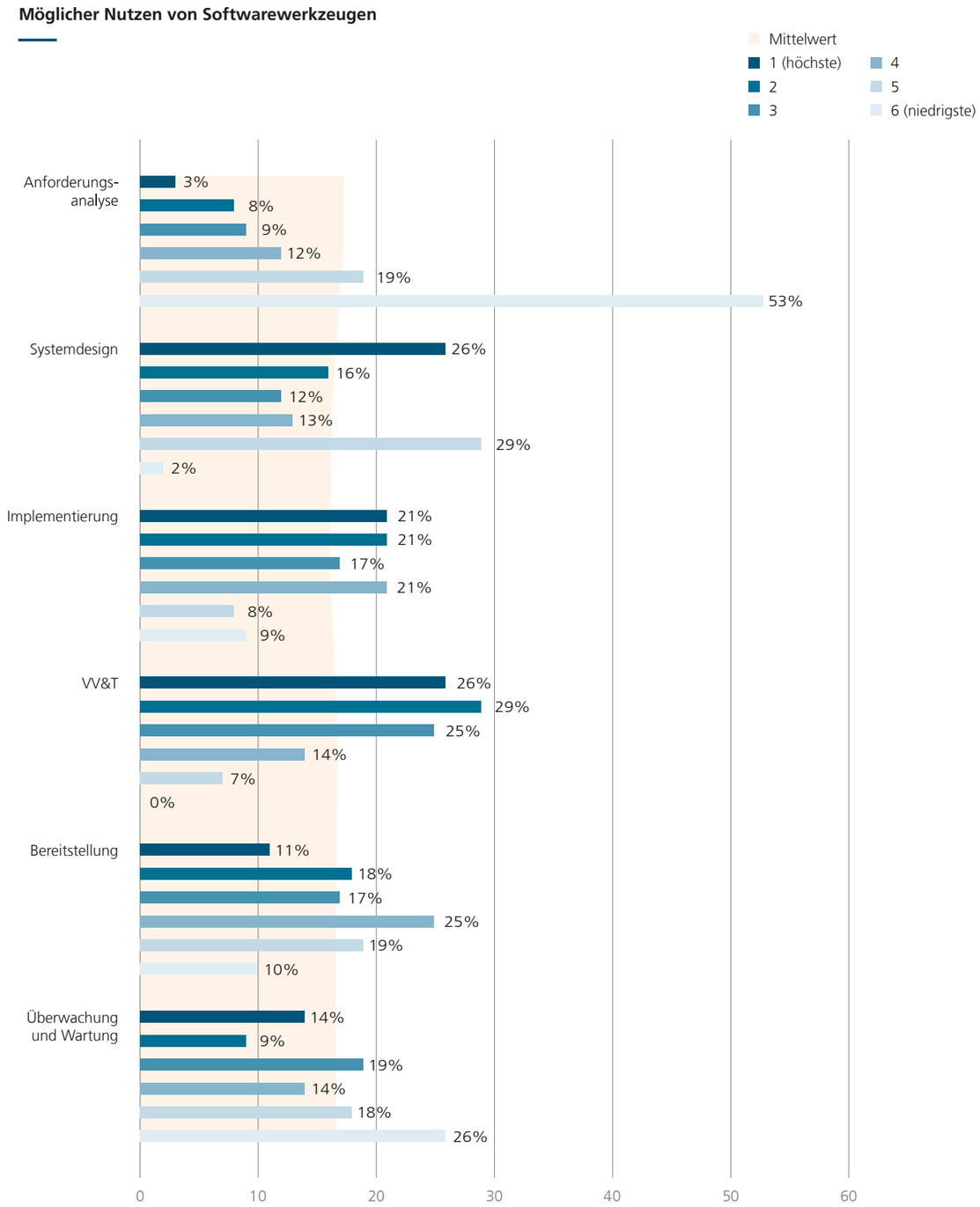

Abb. 7: Zusammenfassung der Antworten auf die Frage, für welche Phase ein Werkzeug vorteilhafter wäre.

2.3. Analyse der Ergebnisse

2.3.1. Herausforderungen

Aus den Ergebnissen der Befragung ergeben sich als wichtigste Herausforderungen bei der Integration von Roboter-Software die folgenden:

1. **Dynamische Anpassung an die Umwelt:** Es ist keine Überraschung, dass dies die Herausforderung ist, die den Befragten am meisten Sorgen bereitet. Robotersysteme müssen in einer realen Umgebung arbeiten und mit ihr interagieren. Auf technischer Ebene gibt es immer bessere Sensoren und Wahrnehmungssoftware, die hierbei helfen. In der Praxis ist es jedoch eine sehr komplexe Angelegenheit, die ein Integrator lösen muss, denn jedes einzelne Szenario hat viele Besonderheiten, die berücksichtigt werden müssen.
2. **Sicherheitszertifizierung:** In den meisten Fällen ist die Zertifizierung einer Roboteranwendung eine sehr mühsame Aufgabe, da sie für jede einzelne Einrichtung auf spezifische und individuelle Weise erfolgen muss. In dieser Hinsicht ist die Normung wahrscheinlich der beste Weg, um diese Herausforderung zu bewältigen, da Normen eine Reihe von Anforderungen enthalten, deren Konformität sich für die verschiedenen Robotersysteme einheitlich bewerten lässt.
3. **Validierung des entstandenen Systems:** Es gibt nur sehr wenige Entwicklungen in dem Bereich der Systemvalidierung, denn um die Validierung (vollständig oder teilweise) zu automatisieren, sind zunächst eine formale Definition der Kundenspezifikationen sowie des gebauten Systems nötig.

2.3.2. Übliche Vorgehensweisen

Zum einen zeigen sich Ähnlichkeiten in den Phasen, die Entwickler bei der Integration eines Robotersystems durchlaufen. So gaben mehr als 80 Prozent der Befragten an, dass sie immer drei Aufgaben im Integrationsprozess durchführen: Systemdesign, Implementierung des Codes, der die Integration ermöglicht, und schließlich eine Validierungs-, Verifizierungs- und/oder Testphase des resultierenden Systems. Zum anderen ist ein Konsens über die Phasen erkennbar, in denen die meisten Werkzeuge verwendet werden: Systemimplementierung sowie die Validierungs- und Testphasen. Dies passt perfekt zu den in Abschnitt 2.2.3 und 2.2.4 erwähnten Werkzeugen. Für die erste Phase wurden in der Mehrzahl der Antworten Versionskontrollsysteme zusammen mit IDEs genannt. Für die zweite Phase wurden Tools genannt, die die Kompilierbarkeit des Codes prüfen können (eine Form der Validierung) und automatisierte Tests ermöglichen, wie CI-Systeme oder Docker-Lösungen. In diesem Zusammenhang ist hervorzuheben, dass die angegebenen Werkzeuge weder auf die Robotik noch auf deren spezifischen Integrationsprozess spezialisiert sind. Dies deutet darauf hin, dass es für diese konkrete Entwicklungsphase keine geeigneten, spezialisierten Werkzeuge gibt.

2.3.3. Anwendbarkeit von Methoden des Softwareentwicklungszyklus

Abschließend geht es darum, zu bewerten, inwieweit sich die vorgestellten sechs Schritte der Softwareintegration, die alle an bewährten Methoden des Softwareentwicklungszyklen (engl. SDLC Methods – Software Development Life Cycle Methods) angelehnt sind, für die Robotik anwenden lassen. Hierfür muss betrachtet werden, welche dieser Methoden während der Systemintegration durchgeführt werden sollten. Alle Phasen haben mehr als 50 Prozent Zustimmung erhalten. Daraus lässt sich schließen, dass die Community im Allgemeinen die Verwendung von SLDC-Methoden unterstützt, allerdings mit Nuancen:

- Die Phasen sollten neu definiert werden, um spezifischen Integrationsaufgaben Rechnung zu tragen. So ist es den Antworten zufolge beispielsweise wichtig, in der Validierungs-, Verifizierungs- und Testphase Simulationen einzusetzen, um die Systeme vor ihrem Einsatz zu testen. In ähnlicher Weise sollte zu Beginn der Systementwurfsphase ein konkretes Einsatzszenario zur Bewertung der bestehenden Lösungen in die Entwurfsspezifikation mit aufgenommen werden.
- Die Phasen haben nicht alle die gleiche Bedeutung. Je nach »Technology Readiness Level« (TRL) können einige beim schnellen Prototypen-Aufbau sogar entfallen, und selbst wenn alle Phasen durchgeführt werden, müssen nicht alle kontinuierlich oder zyklisch ausgeführt werden. Bei einigen genügt es auch, sie nur einmal durchzuführen.

Als erste Instanz einer Methodik für die Softwareintegration in der Robotik scheint die Bewertung der SDLC-Methoden relativ passend zu sein. Diese Basis ist gut, aber es ist auch klar, dass solche generischen Methoden für die Produktentwicklung nicht in Reinkultur angewendet werden können, sondern jeweils an die konkreten Bedürfnisse eines Softwareprojekts angepasst werden müssen. Die in dieser Studie zusammengekommenen Informationen können als gute Basis dienen, um die Methoden an die Besonderheiten der Integration anzupassen und die Abschnitte im Prozess zu identifizieren, die vom Reifegrad der zu entwickelnden Lösung abhängen.

2.3.4. Werkzeuge

Aus den Ergebnissen können mehrere Schlussfolgerungen gezogen werden:

1. Die Community unterstützt die Entwicklung von Werkzeugen für die Integration.
2. Es besteht ein großer Bedarf an Werkzeugen, um den Prozess, Spezifikationen und Schnittstellen zu vereinheitlichen. Viele Befragte fordern eine Standardisierung in der Robotik und weisen darauf hin, dass der Einsatz von Werkzeugen Teil der Lösung sein kann.
3. Die bevorzugte Wahl sind modellbasierte Werkzeuge.
4. Validierungs-, Verifizierungs- und Testfunktionen sowie der Systementwurf sollten die wichtigsten Prioritäten sein, die durch Werkzeuge unterstützt werden sollten.

Es ist eine große Herausforderung, eine Reihe von Werkzeugen oder eine vollständige Toolchain zu entwickeln, die alle diese Punkte zusammen abdecken können. In anderen, stärker konsolidierten Branchen mit komplexen Systemen, z. B. in der Automobil- oder Luftfahrtindustrie, lässt sich jedoch eine gewisse Abhängigkeit zwischen dem Einsatz von Werkzeugen, der Standardisierung, der Verwendung von Modellen und der Gültigkeit und Überprüfung von Anforderungen feststellen. Diese Branchen verfügen über eine starke Basis an Standards. Die Werkzeuge, die den Entwicklern von Softwarelösungen zur Verfügung stehen, verwenden Modelle, die mit diesen Standards übereinstimmen, was die Arbeit der Integratoren (einschließlich Verifikation, Validierung und Tests) erheblich erleichtert. Diese Bereiche sollten bei der Entwicklung neuer Werkzeuge als Inspiration dienen. Eine komplett analoge Entwicklung könnte jedoch zu mühsam sein, da die Robotik nicht über die wichtigste Grundlage, nämlich eine Basis einheitlicher und konsolidierter Standards, verfügt.

2.4. Kriterien der Validität der Studie

In dieser Studie wurden vier Kriterien berücksichtigt, um die erzielten Ergebnisse der Umfrage zu validieren: Konstruktvalidität, Interne Validität, Externe Validität und Validität von Schlussfolgerungen. Die Konstruktvalidität bezieht sich auf das Ausmaß, in dem die Messung von

Variablen in einer Studie die Konstrukte in der realen Welt genau repräsentiert. Im Kontext dieser Studie wurde bewertet, ob abstrakte Sachbegriffe wie System, Tool oder Prozess die gleiche Bedeutung für alle Teilnehmer haben. Die Interne Validität bezieht sich auf das Ausmaß, in dem eine Behandlung oder eine unabhängige Variable tatsächlich für die beobachteten Auswirkungen auf die abhängige Variable verantwortlich ist. Im Kontext der Studie wurde bewertet, inwieweit die Ergebnisse nicht durch den internen Bias der Teilnehmenden beeinflusst wurden. Die Externe Validität bezieht sich auf die Verallgemeinerbarkeit der Studienergebnisse auf andere Teilnehmergruppen oder Umgebungen. Im Kontext dieser Studie wurde die Relevanz der Ergebnisse für diverse Anwendungsfälle sichergestellt. Die Validität von Schlussfolgerungen bezieht sich darauf, ob die gezogenen Schlussfolgerungen aus einer Studie eine statistische Robustheit nachweisen können.

3. Lösungsansatz: modellbasierte Softwareentwicklung

3.1. Anwendung der technischen Verfahren und Beschreibung des Lebenszyklus der Systementwicklung

3.1.1. Übersicht

Der Lebenszyklus eines Systems bezieht sich auf die verschiedenen Phasen oder Stadien, die es durchläuft – von der ersten Konzeption bis zu seiner Außerbetriebnahme. Nach dem heutigen Stand der Technik besteht der Lebenszyklus eines jeden softwarebezogenen Systems aus vier Gruppen: Konzept, Entwicklung, Betrieb und Wartung sowie Ruhestand, vgl. ISO 24748 (ISO/IEC/IEEE, 2018).

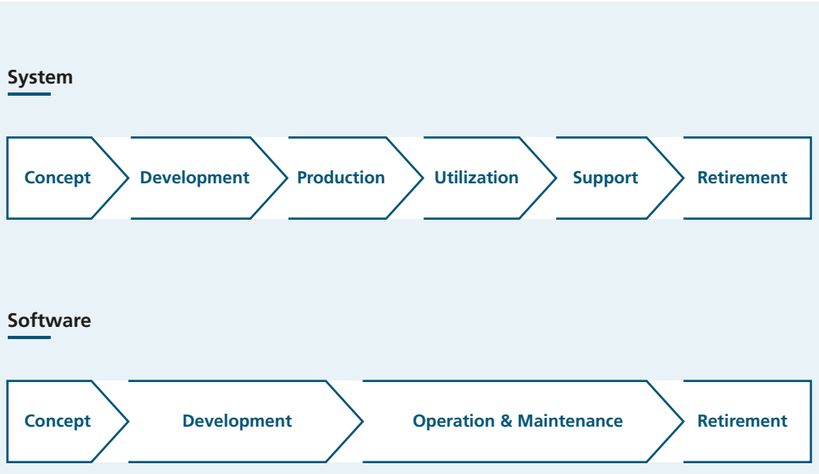

Abbildung 8: Lebenszyklus eines Systems auf der Basis von Normen ISO/IEC/IEEE 24748: Systems and Software

Darauf basierend lassen sich die Aktivitäten im Zusammenhang mit dem Lebenszyklus der Software eines Robotersystems wie folgt einteilen:

Konzept
- Konzeptualisierung: In dieser Phase wird der Bedarf für das System ermittelt und ein Konzept auf abstrakter Ebene entwickelt, um diesen Bedarf zu decken.
- Anforderungsanalyse: In dieser Phase werden die Anforderungen an das System genauer definiert, einschließlich funktionaler Anforderungen (was das System leisten soll) und nicht-funktionaler Anforderungen (wie Zuverlässigkeit, Skalierbarkeit und Leistung).

Entwicklung
- Entwurf: In dieser Phase wird die Systemarchitektur entwickelt und der detaillierte Entwurf der einzelnen Komponenten erstellt. Dazu gehören die Auswahl der Hardware- und Softwarekomponenten sowie die Gestaltung der Schnittstellen zwischen ihnen.
- Implementierung: Diese Phase umfasst den Aufbau und die Integration der Systemkomponenten sowie die Entwicklung der erforderlichen Software.
- Testen: Das Testen stellt sicher, dass das System die Anforderungen erfüllt und wie erwartet funktioniert. Dies umfasst sowohl Einheitstests einzelner Komponenten als auch Integrationstests für das gesamte System.
- Bereitstellung: In dieser Phase wird das System installiert und den Nutzern bereitgestellt.

Betrieb und Wartung
Diese beiden Aktivitäten laufen gleichzeitig und zielen darauf ab, dass das System über einen längeren Zeitraum ordnungsgemäß funktioniert. Der Betrieb ist ein proaktiver und kontinuierlicher Prozess, während die Wartung ein reaktiver Prozess ist, der ausgelöst wird, wenn ein Fehler oder ein Problem auftaucht.

Ruhestand
Dies ist die letzte Phase der Software. Sie umfasst die Beendigung ihrer Nutzung und die Verwaltung der damit verbundenen Aktivitäten.

Diese Studie behandelt nur den Projektabschnitt »Entwicklung«. Im Folgenden werden die einzelnen Aufgaben detailliert erklärt. Diese Beschreibung zielt darauf ab, den Lebenszyklus eines Robotersoftwaresystems in einer anwendungsunabhängigen und generischen Weise zu konzipieren.

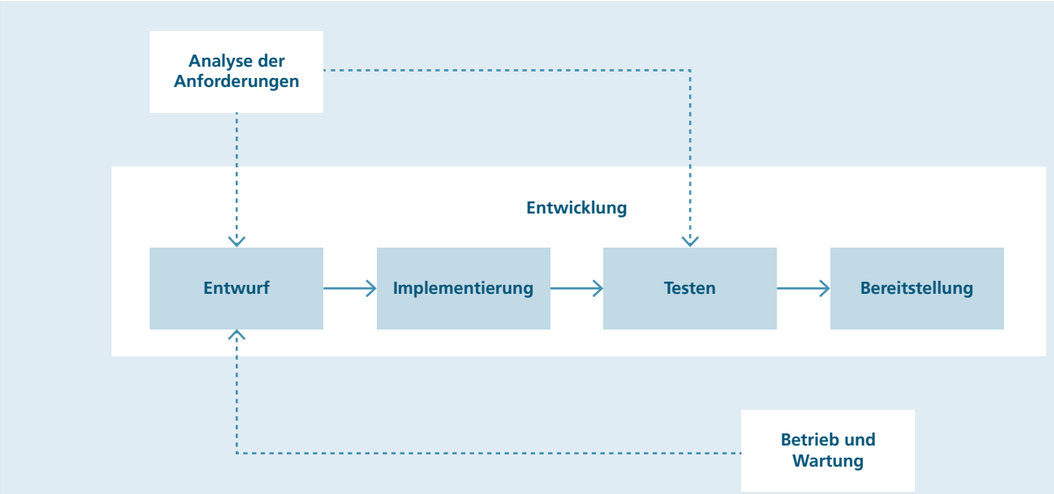

Abbildung 9: Teil des Lebenszyklus, der sich auf die Entwicklungsphase des Systems und seine Schnittstellen mit anderen Phasen bezieht.

3.1.2. Entwurf

Zweck
Der Zweck dieses ersten Schritts in der Softwareentwicklung ist, einen Entwurf oder Plan für das System zu erstellen, der die vorab ermittelten Anforderungen erfüllt. Dazu gehört, die Systemarchitektur, die Schnittstellen, Module, Algorithmen und Datenstrukturen detailliert zu beschreiben. Der Entwurf sollte mit bewährten Verfahren und Standards übereinstimmen und Aspekte wie Wartbarkeit, Skalierbarkeit, Leistung und Sicherheit berücksichtigen.

Eingaben
Für diesen Schritt werden die in der Konzeptphase erzeugten Ergebnisse benötigt, insbesondere der technische Teil der Konzeptualisierung und die endgültige Beschreibung der Anforderungen. Daneben gibt es weitere Überlegungen, die bei der Definition des Systems berücksichtigt werden können:

- Anwendungsfälle, d. h. die Szenarien, die beschreiben, wie externe Systeme und Benutzer mit der Software interagieren werden.
- Funktionsspezifikationen, d. h. spezifische Merkmale und Funktionen, die die Software erfüllen muss.
- Nicht-funktionale Spezifikationen, wie Leistung, Sicherheit und andere Eigenschaften, die die Software haben muss.
- Bestehende Software-Architektur, in die das neue Softwaresystem eingebettet werden soll.
- Entwicklungstechnologien. Dazu gehören Programmiersprachen, Bibliotheken und Frameworks sowie alle anderen Tools, die zur Erstellung, Prüfung und Bereitstellung der Software verwendet werden.

Ausgaben
Das Ergebnis des Entwurfsschritts ist ein Entwurfsdokument oder eine Reihe von Diagrammen, die das Entwicklungsteam für die Implementierung des Systems verwenden kann. Das Entwurfsdokument sollte das System klar und vollständig beschreiben und von den Beteiligten vor der Implementierung überprüft und validiert werden. Die Entwurfsdokumente stellen

eine Blaupause für die Software dar, in der die verschiedenen Komponenten, Funktionen und Merkmale skizziert werden. Um die folgenden Schritte erfolgreich durchführen zu können, ist es empfehlenswert, dass das Entwurfsdokument die folgenden Informationen enthält:

- Systemarchitektur: Der High-Level-Entwurf des Systems, einschließlich der Hauptkomponenten, Module und ihrer Beziehungen. Dazu gehören Diagramme und Beschreibungen der Gesamtstruktur des Systems und seiner Funktionsweise.
- Detaillierter Entwurf der Systemkomponenten: Eine detailliertere Beschreibung jeder Komponente oder jedes Moduls, einschließlich ihres Zwecks, ihrer Eingabe, ihrer Ausgabe und ihrer Schnittstelle zu anderen Komponenten.
- Fehlerbehandlung und Wiederherstellungsstrategie.
- Datenentwurf: Eine Beschreibung der vom System verarbeiteten Daten einschließlich ihrer Typen, Beziehungen und Speicherorte.
- Entwurfsentscheidungen: Eine Beschreibung der Entwurfsentscheidungen, die während des Entwurfsprozesses getroffen wurden, einschließlich der Gründe für die Entscheidungen und aller in Betracht gezogenen Alternativen.

Hilfsmittel
Um ein System zu entwerfen, gibt es Techniken und Instrumente, die diesen Schritt erleichtern können. Unter anderem sind das:

- Entwurfsmuster
- Entwurfssoftware
- Modellierungssprachen
- Gestaltungsprinzipien

3.1.3. Implementierung

Zweck
Beim Implementierungsschritt werden der Entwurf und die Systemspezifikationen in Code umgewandelt, der auf einem Computer ausgeführt werden kann. Dazu gehört, den Entwurf in eine Programmiersprache zu übersetzen und die verschiedenen Komponenten und Module zu implementieren, die im Entwurfsdokument angegeben wurden.

Eingaben
Die wichtigste Quelle für die Implementierung ist das in der Entwurfsphase erstellte Entwurfsdokument.

Ausgaben
Das Ergebnis der Implementierung ist ein funktionierendes System, das die Endnutzer einsetzen und verwenden können. Dazu gehört auch, eine gute und vollständige Dokumentation zu erstellen.

Hilfsmittel
Für die Code-Implementierung im Software-Engineering gibt es eine Reihe von Werkzeugen und verfügbarem Material, die die Entwicklerinnen und Entwickler unterstützen. Hervorgehoben werden können die folgenden:

- Entwicklungsumgebungen, z. B. Werkzeuge zum Entwickeln, Kompilieren und Testen des Codes
- Frameworks, die eine wiederverwendbare, anpassbare und erweiterbare Codebasis bieten
- Etablierte Programmiersprachen und Kodierungsrichtlinien
- Versionskontrollsysteme

3.1.4. Testen

Zweck
Tests dienen dazu, die Funktionalität, Leistung und Qualität des Systems zu bewerten und etwaige Mängel oder Fehler zu erkennen und zu beheben, bevor es den Endnutzern zur Verfügung gestellt wird. Dazu gehört, Testfälle und -szenarien zu erstellen und auszuführen, die vielfältige Eingaben und Anwendungsfälle abdecken, um so Mängel, Fehler und unerwartetes Verhalten zu erkennen.

Eingaben
Für diese Phase werden die Ergebnisse der Anforderungsanalyse, des Entwurfs und der Implementierungsschritte benötigt, d. h. die Anforderungen, das Entwurfsdokument und der implementierte Code. Um den Testcode zu entwickeln und Testschritte durchzuführen, werden möglicherweise einige spezifische Informationen benötigt, wie ein Testplan, Testfälle, Testdaten und die Testumgebung.

Ausgaben
Testberichte dokumentieren die Testergebnisse und zeigen alle gefundenen Fehler oder Probleme auf. Dies beinhaltet die Berichterstattung über den Fortschritt der Testphase und die Bereitstellung von Metriken, um die Softwarequalität zu verbessern.

Hilfsmittel
Für Testzwecke in der Softwareentwicklung gibt es eine ganze Reihe von Hilfsmechanismen. Einige gängige Hilfsmechanismen sind:

- Werkzeuge zur Testautomatisierung
- Testmanagement-Tools
- Kontinuierliche Integration (CI) und kontinuierliche Tests (CT)
- Sondierungstests
- Simulationsumgebungen
- Hardware-in-the-Loop-Testumgebungen

3.1.5. Bereitstellung

Zweck
Jetzt soll das System für die Endnutzer verfügbar gemacht (engl. »deployed«) werden. Dazu gehört die Installation des Systems in der Produktionsumgebung und die Konfiguration des Systems für die ordnungsgemäße Zusammenarbeit mit anderen Systemen und Datenquellen. Der Bereitstellungsschritt umfasst auch Aktivitäten wie das Erstellen und Konfigurieren von Umgebungen, das Installieren des Systems und aller erforderlichen Abhängigkeiten, das Konfigurieren von Datenbanken und anderen Ressourcen sowie das Durchführen aller erforderlichen Datenmigrationen oder -konvertierungen.

Eingaben
Die wichtigsten Inputs für die Bereitstellung (engl. »Deployment«) kommen aus dem Implementierungsschritt: die fertiggestellte Software und die Benutzerdokumentation. Zudem braucht es den Verteilungsplan sowie die Konfigurations- und Installationsdaten, um den Verteilungsschritt durchzuführen.

Ausgaben
Zu den Ergebnissen der Bereitstellung sollte mindestens gehören:

- Eingesetzte Software: Dies sollte eine voll funktionsfähige und stabile Version der Software sein, die in der Zielumgebung installiert und konfiguriert wurde.
- Bereitstellungsdokumentation: Die Bereitstellungsdokumentation enthält alle Informationen und Anweisungen, die für die Installation, Konfiguration und Bereitstellung der Software erforderlich sind.
- Gliederung der Änderungsmanagement-Dokumentation.

Hilfsmittel
Für die Bereitstellung gibt es eine Reihe von bekannten Infrastrukturen, Tools und Frameworks, die diesen Schritt erleichtern. Einige von ihnen sind:

- Automatisierungswerkzeuge, zum Beispiel Tools für kontinuierliche Integration und kontinuierliche Bereitstellung (CI/CD) wie Jenkins, GitLab CI/CD und CircleCI.
- Tools für die Konfigurationsverwaltung, wie Ansible, Chef und Puppet.
- Containerisierungstools: Einige bekannte Lösungen sind Docker und Kubernetes.
- Cloud-Infrastrukturdienste, wie AWS, Microsoft Azure und Google Cloud Platform.

3.2. Modellgetriebene Softwareentwicklung und -integration

3.2.1. Was und wie?

Die modellgetriebene Softwareentwicklung und -integration unterstützt und formalisiert den vorstehend beschriebenen Entwicklungsprozess durch die Einführung von Modellen. Als Modell wird hierbei jede stark formalisierte Beschreibung des Systems, seiner Komponenten und seiner Umgebung bezeichnet, ebenso wie stark formalisierte Darstellungen der Dokumente, die zwischen den fünf Schritten des vorstehenden Entwicklungsprozesses vermitteln. Der zentrale Unterschied zwischen einem Modell und den zuvor genannten Dokumenten wie dem Entwurfsdokument oder der Bereitstellungsdokumentation ist, dass ein Modell einem strikt definierten Format mit klar definiertem Vokabular für die einzelnen beschriebenen Aspekte folgt. Dies ermöglicht, das Modell automatisiert zu verarbeiten und z. B. ein Modell einer Softwarekomponente auf Vollständigkeit und Widerspruchsfreiheit zu überprüfen. Die Festlegung der Regel für diese Überprüfung und die Definition des Formats erfolgt im sogenannten Meta-Modell.

Diese Modelle spielen bei der modellgetriebenen Entwicklung eine Schlüsselrolle im technischen Prozess und treiben den Systemerstellungsprozess voran. Der Hauptzweck dieses Paradigmas besteht darin, von einem Entwicklungsansatz, bei dem der Code im Zentrum steht, zu einem Ansatz zu gelangen, bei dem die Architektur und die Systemanforderungen im Kern stehen (Stahl, 2006).

Das Konzept basiert auf der Verwendung von Modellen zur Beschreibung sowohl des Problems als auch der Lösung, wobei diese Modelle die verschiedenen Abstraktionsebenen darstellen können. Dies führt zu einer erhöhten Kompatibilität zwischen Systemen, vereinfacht den Entwurf und erleichtert die Kommunikation zwischen Anbietern, Systementwicklern und Endnutzern erheblich.

Die Verwendung von Modellen und modellgesteuerten Technologien kann mehrere Vorteile für die Integration von Komponenten in verschiedenen Bereichen bieten. Im Allgemeinen ermöglichen Modelle eine Abstraktion auf verschiedenen Ebenen, die ein besseres Verständnis der Systemkomponenten ermöglicht, ohne sich in unnötige Details zu verlieren. Auch in diesem Zusammenhang bieten Modelle eine visuelle Darstellung der Systemarchitektur und der Komponenten. Dieses visuelle Hilfsmittel verbessert die Kommunikation zwischen den verschiedenen, am Integrationsprozess beteiligten Akteuren, darunter Entwickler, Architekten und Projektmanager.

Darüber hinaus fördert die Verwendung standardisierter Modellierungssprachen (z. B. UML, BPMN) die Konsistenz und stellt sicher, dass alle Beteiligten ein gemeinsames Verständnis des Systems haben. Diese Standardisierung erleichtert eine bessere Planung und Ausführung der Integration.

Im Hinblick auf Hilfsmittel führt die modellgetriebene Entwicklung (Model-Driven Development, MDD) Techniken zur Automatisierung und Codegenerierung ein. Modelle können zur automatischen Generierung von Code oder Konfigurationsdateien verwendet werden, sodass Fehler bei der manuellen Implementierung unwahrscheinlicher werden. Dies beschleunigt den Integrationsprozess und stellt sicher, dass die Komponenten die vorgegebene Architektur einhalten. Darüber hinaus dienen die Modelle als lebende Dokumentation, die automatisch aktualisiert werden kann. Dies stellt sicher, dass die Dokumentation stets mit dem tatsächlichen System übereinstimmt (unter anderem, da Teile des Systems aus den Modellen automatisiert erzeugt werden können) und Entwicklern und anderen Beteiligten eine genaue Referenz bietet. Weiterhin können Modelle genutzt werden, um automatisch Testfälle zu generieren, die eine umfassende Abdeckung der Integrationsszenarien gewährleisten. Dies trägt zur Gesamtqualität und Zuverlässigkeit des integrierten Systems bei.

Im speziellen Fall der Integration unterstützen Modelle die Identifizierung und Definition von wiederverwendbaren Komponenten. Dies fördert einen modularen und skalierbaren Ansatz für die Integration, da wiederverwendbare Komponenten leicht in verschiedene Systeme integriert werden können. Gleichzeitig wird die kollaborative Entwicklung gefördert, da die verschiedenen Interessengruppen gleichzeitig an verschiedenen Aspekten des Systems arbeiten können. Dieser kollaborative Ansatz beschleunigt den Zeitplan für die Integration.

Zusammenfassend lässt sich sagen, dass Modelle und modellgesteuerte Technologien einen strukturierten und effizienten Ansatz für die Komponentenintegration bieten, da sie Abstraktion, Standardisierung, Automatisierung und eine verbesserte Kommunikation zwischen den Beteiligten ermöglichen.

3.2.2. Arbeitserleichterungen und Fehlervermeidung

Ein wesentlicher Vorteil eines werkzeuggestützten, modellgetriebenen Softwareentwicklungsprozesses ist, dass repetitive und fehleranfällige Arbeit wie das manuelle Erstellen von sogenanntem »Boiler-Plate Code« (dem vom konkreten Anwendungsfall weitgehend unabhängigen Gerüst der Software), vermieden wird. Insbesondere bei der Softwareentwicklung für Robotersysteme kommen hier zusätzliche Herausforderungen hinzu, wie die Anpassung an die Rechenhardware, komplexe Hardwareschnittstellen etc., deren Konfiguration zeitaufwendig und bekannt-fehleranfällig ist.

Indem wesentliche Aspekte des Gesamtsystems sehr abstrakt modelliert und auf Modellebene mit Daten angereichert werden, die für die Spezialisierung auf konkrete Systeme nötig sind, bekommen Systementwickler einen einfacheren Überblick über das zu entwickelnde System. Zudem werden sie nicht durch die oft kryptische Syntax konkreter Implementierungen oder

mehrfache Ablage derselben Information belastet. Durch Validierungsregeln, die in den Meta-Modellen zugrunde liegen, können auch Fehler oder ungünstige Vorgehensweisen frühzeitig erkannt und aufgezeigt oder (in Grenzen) automatisch behoben werden.

Doppelte und damit Inkonsistenzen befördernde Eingaben der gleichen Information, z. B. in einer Konfigurationsdatei im technischen System und in einer Dokumentationsdatei, können vermieden werden, indem die Information an genau einer Stelle im Modell abgefragt und dann Dokumentation und technische Konfiguration automatisiert generiert werden. Dies spart Zeit und Arbeit und verhindert Fehler.

3.2.3. Skalierbarkeit, Softwareproduktfamilien und Kundenvarianten effizient umsetzen

Durch Trennung der eigentlichen Anwendungslogik innerhalb von Softwarekomponenten (wie z. B. eines konkreten Objekterkennungsalgorithmus) von der Systemkomposition eines vollständigen Softwaresystems für eine gegebene Roboteranwendung kann die Adaption eines einmal entwickelten Softwaresystemkonzepts auf unterschiedliche Kundenanwendungen deutlich einfacher werden. Diese Vereinfachung rührt daher, dass die Anwendungslogik innerhalb der Komponenten gegen eine stabile Schnittstellendefinition der Komponenten entwickelt und daher geändert werden kann, ohne die Systemkomposition zu tangieren. Umgekehrt kann die Systemkomposition auf Modellebene verändert werden, ohne Wissen über die konkrete Implementierung einer Komponente zu haben. Und konkurrierende Implementierungen einer Komponente von unterschiedlichen Anbietern können auf Modellebene ausgetauscht werden, ohne hierfür manuell eine Zeile Code schreiben zu müssen. Die konkrete Realisierung der Software für eine Roboteranwendung wird immer automatisch aus den Komponenten und Informationen über die Systemkomposition auf Anforderung neu generiert. Dieser Vorteil bleibt in ähnlicher Form erhalten, selbst wenn der wesentliche Aspekt der Anwendungslogik (und damit des Knowhows) für eine Roboteranwendung weniger in den Komponenten selbst als mehr in der Auswahl und Verschaltung der Komponenten, also der Systemkomposition, liegt.

In allen Fällen ermöglicht die Anpassung der Softwarelösung für eine Roboteranwendung auf Modellebene, diese deutlich effizienter über verschiedene, aber ähnliche Anwendungsszenarien zu skalieren, als dies bei direkter Anpassung auf Code-Ebene der Fall wäre. Die Vorteile eines Komponentensystems und der modellorientierten Entwicklung kombinieren und multiplizieren sich hierbei. Beispielsweise ließe sich in der Systemkomposition zunächst ein Gesamtsystem entwickeln, das alle in einem Anwendungsgebiet denkbaren Features enthält. Dann könnten kundenspezifisch verschiedene Features (Komponenten oder Subsysteme) deaktiviert und so eine Softwareproduktfamilie realisiert werden, die in jeweils passender Kundenvariante generiert und ausgerollt wird.

3.2.4. Vereinfachte Wartung und Analyse alter Software

Soweit die modellgetriebenen Werkzeuge neben der Modell-zu-Text-Transformation (d. h. Code-Generierung) auch den umgekehrten Weg der Modellextrahierung unterstützen, wie das am Fraunhofer IPA entwickelte RosTooling (Hammoudeh Garcia, Deshpande, Santos, Kahl, & Bordignon, 2021) , eröffnen sich neue und effizientere Wege in der Wartung, Analyse und Aktualisierung existierender Altsysteme. Hierbei wird das Altsystem automatisiert analysiert und in seine konstituierenden Komponenten und die Komposition des Systems zerlegt. Das so erzeugte Modell kann dann analog zu einem durch Systementwickler erstellten Modell mit den gleichen Werkzeugen manipuliert und eine veränderte Fassung des Altsystems wieder generiert werden. Dies erlaubt zum Beispiel die semiautomatische Übersetzung von Robotersoftware im ursprünglichen ROS Framework nach ROS2. Selbst in Fällen, in denen keine automatische vollständige Analyse gelingt, erleichtern die erzeugten Modelle immer noch die anschließende manuelle Analyse eines Altsystems. Sie bieten auch Ansatzpunkte für Aktualisierungen des Systems.

Literaturverzeichnis

Balasubramanian, K. a. (2009). System Integration using Model-Driven Engineering. In Designing software-intensive systems: methods and principles (S. 474-504). IGI Global.

Hammoudeh Garcia, N., Deshpande, H., Santos, A., Kahl, B., & Bordignon, M. (2021). Bootstrapping MDE development from ROS manual code: Part 2---Model generation and leveraging models at runtime. Software and Systems Modeling.

ISO/IEC/IEEE. (Nov 2018). ISO/IEC/IEEE 24748. Systems and software engineering — Life cycle management — Part 1: Guidelines for life cycle management.

Ng, M. Q. (2009). ROS: an open-source Robot Operating System. In Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics.

Stahl, T. a. (2006). Model-Driven Software Development: Technology, Engineering, Management.

Tola, D. a. (2022). Towards Easy Robot System Integration: Challenges and Future Directions. In 2022 IEEE/SICE International Symposium on System Integration (SII) (S. {77-82).

Trowbridge, D. a. (2004). Integration Patterns. Microsoft Corporation.

KI-Fortschrittszentrum

Das KI-Fortschrittszentrum »Lernende Systeme und Kognitive Robotik« unterstützt Firmen dabei, die wirtschaftlichen Chancen der Künstlichen Intelligenz und insbesondere des Maschinellen Lernens für sich zu nutzen. In anwendungsnahen Forschungsprojekten und in direkter Kooperation mit Industrieunternehmen arbeiten die Stuttgarter Fraunhofer-Institute für Produktionstechnik und Automatisierung IPA sowie für Arbeitswirtschaft und Organisation IAO daran, Technologien aus der KI-Spitzenforschung in die breite Anwendung der produzierenden Industrie und der Dienstleistungswirtschaft zu bringen. Unterstützt werden sie dabei vom Institut für Arbeitswissenschaft und Technologiemanagement IAT der Universität Stuttgart. Finanzielle Förderung erhält das Zentrum vom Ministerium für Wirtschaft, Arbeit und Tourismus Baden-Württemberg.

Mission
Das KI-Fortschrittszentrum ist der anwendungsorientierte Zweig von Cyber Valley, Europas größter Forschungskooperation im Bereich der Künstlichen Intelligenz. Darüber hinaus ist das KI-Fortschrittszentrum Bestandteil von S-TEC, dem Stuttgarter Technologie- und Innovationscampus: www.s-tec.de

Das KI-Fortschrittszentrum schlägt die Brücke von der KI-Spitzenforschung in den Mittelstand und macht KI-Technologien für die Wirtschaft in Baden-Württemberg und darüber hinaus nutzbar. Als führender Innovationspartner für den Mittelstand arbeitet das Zentrum an Themen, die für den Einsatz von KI und Robotik branchenübergreifend von zentraler Bedeutung sind, beispielsweise Autonomie, Effizienz und Nachhaltigkeit, Mensch-Maschine-Interaktion sowie Vertrauen. Das KI-Fortschrittszentrum informiert Unternehmen über Technologietrends und deren Einsatzpotenziale und unterstützt sie bedarfsgerecht und niedrigschwellig bei der Entwicklung und Umsetzung von ambitionierten KI-Innovationen, damit sie die wirtschaftlichen Chancen der KI künftig noch besser nutzen können.

Vision
Das KI-Fortschrittszentrum ist ein Leuchtturm für erfolgreichen Technologietransfer in den Mittelstand und ermöglicht Unternehmen einen wirtschaftlichen und verantwortungsvollen Einsatz von Künstlicher Intelligenz und Robotik für unternehmerischen Erfolg sowie individuellen und gesellschaftlichen Nutzen.

Studienreihe »Lernende Systeme und Kognitive Robotik«
Die Studienreihe »Lernende Systeme und Kognitive Robotik« gibt Einblick in die Potenziale und die praktischen Einsatzmöglichkeiten von KI. Nähere Informationen und die aktuellen Versionen der Studien finden Sie unter: https://www.ki-fortschrittszentrum.de/veroeffentlichungen/

Fraunhofer

Die Fraunhofer-Gesellschaft mit Sitz in Deutschland ist eine der führenden Organisationen für anwendungsorientierte Forschung. Im Innovationsprozess spielt sie eine zentrale Rolle – mit Forschungsschwerpunkten in zukunftsrelevanten Schlüsseltechnologien und dem Transfer von Forschungsergebnissen in die Industrie zur Stärkung unseres Wirtschaftsstandorts und zum Wohle unserer Gesellschaft. Seit ihrer Gründung als gemeinnütziger Verein im Jahr 1949 nimmt sie eine einzigartige Position im Wissenschafts- und Innovationssystem ein.

Knapp 32 000 Mitarbeitende an 75 Instituten und selbstständigen Forschungseinrichtungen in Deutschland erarbeiten das jährliche Finanzvolumen von 3,6 Mrd. €. Davon entfallen 3,1 Mrd. € auf das zentrale Geschäftsmodell von Fraunhofer, die Vertragsforschung. Im Vergleich zu anderen öffentlichen Forschungseinrichtungen bildet die Grundfinanzierung durch Bund und Länder lediglich das Fundament des jährlichen Forschungshaushalts. Sie ist die Basis für wegweisende Vorlaufforschung, die in den kommenden Jahren für Wirtschaft und Gesellschaft bedeutend wird. Das entscheidende Alleinstellungsmerkmal ist der hohe Anteil an Wirtschaftserträgen, der Garant ist für die enge Zusammenarbeit mit Wirtschaft und Industrie und die stetige Marktorientierung der Fraunhofer-Forschung: 2024 beliefen sich die Wirtschaftserträge auf 867 Mio. € des laufenden Haushalts. Ergänzt wird das Forschungsportfolio durch im Wettbewerb eingeworbene öffentliche Projektmittel, wobei eine ausgewogene Balance zwischen öffentlichen und wirtschaftlichen Erträgen angestrebt wird.

Wir produzieren Zukunft
Das Fraunhofer-Institut für Produktionstechnik und Automatisierung IPA, kurz Fraunhofer IPA, realisiert hoch innovative und nachhaltige Lösungen in der Produktionstechnik und Automatisierung für verschiedenste Zukunftsbranchen. Dies können Methoden, Komponenten und Geräte bis hin zu kompletten Maschinen und Anlagen sein. Die Lösungen stehen stets in Verbindung mit den strategischen Eckpfeilern des Instituts »Mass Sustainibility« und »Mass Personalization«. Seine Hauptaufgabe sieht das Institut im Wissens-, Innovations- und Technologietransfer von Forschungsergebnissen in Applikationen, um die Wettbewerbsfähigkeit der Unternehmen zu stärken. Dabei versteht es sich als unabhängiger Ansprechpartner, der neutral berät und Unternehmen mit genau auf deren Bedürfnisse zugeschnittenen Projektteams unterstützt.

Impressum

Kontaktadresse
Fraunhofer-Institut für Produktionstechnik und Automatisierung IPA
Nobelstraße 12, 70569 Stuttgart

Björn Kahl
Telefon +49 711 970-1346
bjoern.kahl@ipa.fraunhofer.de

Eshan Savla
Telefon: +49 711 970-6079
eshan.savla@ipa.fraunhofer.de

Titelbild
© Joi_studio - Adobe Stock

Gefördert durch das Ministerium für Wirtschaft, Arbeit
und Tourismus Baden-Württemberg

Kontakt

KI-Fortschrittszentrum
Fraunhofer IAO und Fraunhofer IPA
Nobelstraße 12
70569 Stuttgart
www.ki-fortschrittszentrum.de

Björn Kahl
Telefon +49 711 970-1346
bjoern.kahl@ipa.fraunhofer.de

Eshan Savla
Telefon: +49 711 970-6079
eshan.savla@ipa.fraunhofer.de

Fraunhofer IPA
Nobelstraße 12
70569 Stuttgart
www.ipa.fraunhofer.de

Gefördert durch

Baden-Württemberg
Ministerium für Wirtschaft,
Arbeit und Tourismus

Partner

Cyber Valley

S-TEC

Universität Stuttgart
Institut für Arbeitswissenschaft und
Technologiemanagement IAT