Programmieren und Programmiersprachen

Die Menschen haben in ihrer Geschichte immer komplexere Maschinen gebaut, die möglichst viele Funktionen erledigen können. Damit das Produkt "Maschine" und sein Benutzer "Mensch" sich verständigen können, wurden Vorgänge aus der analogen Welt in formalisierte "digitale" Algorithmen und Prozesse übersetzt. Bei Maschinen mit sehr hoher Komplexität, die auch viele Funktionen erledigen können, wird es notwendig, diese in einer für Maschinen verständlichen Sprache anzuweisen. Diese formal-definierten Befehle, die logische und mathematische Anweisungen beinhalten, bilden eine Programmiersprache und die Vorgänge die durch diese Sprachen definiert werden, sind die Algorithmen.
Nach einem kurzen Blick in die Geschichte der Programmiersprachen wird ein Überblick über die verschiedenen Arten der Programmiersprachen gegeben.

Geschichte der Automaten und der Programmierung

Ein Automat ist eine Maschine, die vorbestimmte Abläufe selbsttätig („automatisch“) ausführt. Der Begriff Automatik steht für eine Vorrichtung, die einen Vorgang steuert und regelt. Automat ist ein Lehnwort aus dem Lateinischen automatus „freiwillig, aus eigenem Antrieb handelnd“, zu Altgriechisch αὐτόματος automatos „von selbst geschehend“ (zu autos „selbst“ und der Wurzel men- „denken, wollen“)[8]. Frühe Maschinen und Automaten wie Uhrwerke oder Hebelwerke, die bereits komplexe Berechnungen und Vorgänge darstellen können, sind mechanisch programmierbar.
Das menschliche Interesse an Automaten gab es schon seit Urzeiten der Kultur. Die Menschen versuchen mit Technologien immer komplexere Maschinen zu bauen, um entweder sich das Leben zu erleichtern, sich zu amüsieren, oder die Welt noch genauer zu erklären. Einer der ältesten bekannten Automaten „Der Mechanismus von Antikythera“ (70 bis 60 v. Ch.)[9] ist einer dieser analogen Computer, der wahrscheinlich für die Bestimmung der Zeit und der Position (d.h. Navigation) vorgesehen war und durch mechanische Vorrichtungen programmierbar war.
Die Geschichte der Automaten umfasst die Chronologie der technischen Entwicklung von Maschinen, die vorbestimmte Abläufe selbsttätig ausführen, von der Frühgeschichte bis zur Gegenwart. Neben zahlreichen Mythen und Legenden finden sich hier auch die ersten historisch belegten echten Automaten. Das Hauptinteresse der Automatenbauer galt dabei zunächst überwiegend der Erforschung der Physik und/oder Abbildung von Natur mit technischen Mitteln[10].

---

[8] https://de.wikipedia.org/wiki/Automat

[9] https://de.wikipedia.org/wiki/Mechanismus_von_Antikythera

[10] https://de.wikipedia.org/wiki/Geschichte_der_Automaten

Als erster programmierbarer Automat der neuen Geschichte gilt die „Analytical Engine“, die von Charles Babbage 1834 beschrieben wurde, aber zu seiner Zeit nie gebaut wurde. Ada Lovelace übersetzte 1843 eine ursprünglich französische Beschreibung der „Analytical Engine“ von Federico Luigi Menabrea ins Englische und fügte eigene Notizen hinzu. Die Anmerkungen von Ada enthielten einen tabellarischen Plan zur Berechnung der Bernoulli-Zahlen, was als das erste Programm bezeichnet wird. Daher gilt Ada Lovelace-Byron[11] als der erste Mensch, der je programmiert hat. 1941 realisierte Konrad Zuse mit dem Z3 die erste programmgesteuerte Rechenmaschine, von 1942 an entwickelte er unter dem Namen „Plankalkül“ die erste höhere Programmiersprache. Die Mathematikerin Grace Hopper schuf 1949 den ersten Compiler, der Quellcode in Maschinencode übersetzt[12].

Wie sieht eine Programmiersprache aus

Eine Programmiersprache ist eine formale Sprache zur Formulierung von Datenstrukturen und Algorithmen, d. h. von Rechenvorschriften, die von einem Computer ausgeführt werden können. Sie setzen sich üblicherweise aus schrittweisen Anweisungen aus erlaubten Code-Mustern zusammen, der sogenannten Syntax[13]. Eine sehr bekannte Programmart ist ein Browser. Beispiele dafür sind Mozilla Firefox, Google Chrome, Microsoft Edge oder Apple Safari. Ein Browser ist in erster Linie ein Interpreter für Code. Der von einem Browser interpretierte Basiscode wird Hyper Text Markup Language (HTML) genannt. Ein Browser kann aber auch andere Programmierformate wie Skriptsprachen verarbeiten (Skriptsprachen wie JavaScript enthalten vordefinierte Programmkomponenten). Die Verschränkung unterschiedlicher Programmierformate lässt sich an dem unter Programmieren berühmten Beispiel „Hello World!“ zeigen. Hier ist das Beispiel mit HTML und JavaScript gezeigt.

[11] https://de.wikipedia.org/wiki/Ada_Lovelace

[12] https://de.wikipedia.org/wiki/Programmierung und
https://de.wikipedia.org/wiki/Geschichte_der_Programmiersprachen

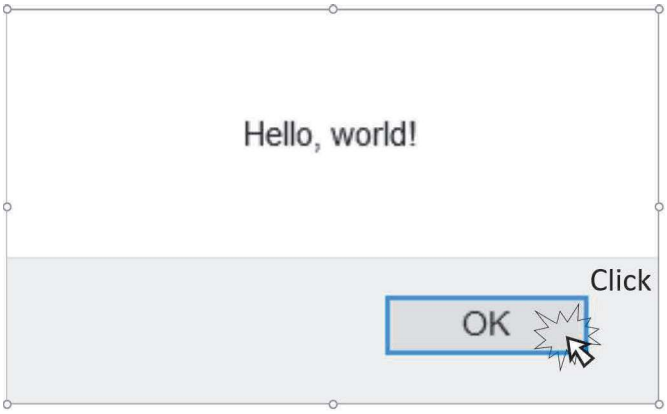
[13] https://de.wikipedia.org/wiki/Programmiersprache

```
<!DOCTYPE HTML>
<!-- Start HTML -->
<html>

<body>

  <p>Before the script...</p>
<!-- Einfügen von JavaScript -->

  <script>
    alert( 'Hello, world!' );
  </script>
<!-- Ende von JavaScript -->

  <p>...After the script.</p>

</body>

</html>
<!-- Ende HTML -->
```

Und das sieht im Browser in 3 Schritten so aus:

1. Start – HTML Anzeige „Before the script“ und Click-Fenster mit „Hello World!“ erscheint

2. JavaScript - Click Fenster mit „Hello World!“ bestätigen



3. Ende - HTML Anzeige „...After the script“ erscheint

Dieser Programm-Code enthält einen automatischen Ablauf in 3 Schritten.

Es ist zu beachten, dass z.B. im Schritt 1 die Hintergrund-Oberfläche des Browsers grau wird und ein Pop-Up Fenster mit „Hello, World!" und „OK" Knopf erscheint. Das ist in dem o.g. Programm-Code nicht enthalten. Das sind Funktionen, die im Code des Browsers selber (d.h. nativ) und durch den Code des JavaScripts (vordefiniertes „alert" Fenster) enthalten sind. Es ist auch der Browser, der interpretativ diesen HTML/JavaScript-Code ausführt, d.h. der Code des Browsers selber „versteht" den HTML und JavaScript Code und kann diesen im Rahmen seiner eigenen Funktion ausführen. Dagegen ist der eigene Code des Browsers pre-compiliert, damit er z.B. in einer Betriebssystem- und Hardware-Umgebung ausführbar ist. Ein handelsüblicher Windows-Rechner macht das mit Hilfe eines Intel-Prozessors, einer NVIDIA Grafikkarte und der Microsoft Windows Betriebssystem-Umgebung.

Heutzutage werden eine große Anzahl von verschiedenen Programmiersprachen und Kombinationen davon verwendet. Die Wahl der Programmiersprache hängt ab von der Notwendigkeit abstrakte Darstellungsfunktionen für die Mensch-Maschinen-Schnittstelle zu realisieren, hardware-nahe Funktionen zu implementieren oder auch fachspezifische Ausdrucksweisen auf den Gebieten der Mathematik, der Ingenieurswissenschaften oder anderer Fachgebiete abzubilden.

Die Programmiersprachen werden nach ihrem logischen Aufbau als objektorientierte, prozedurale oder deklarative Sprachen bezeichnet. Die Merkmale einer Programmiersprache beruhen überwiegend auf den Eigenschaften der angewendeten Struktur- und Syntax-Formen der Sprache (wie z. B. bei Object-Oriented-Programming (OOP), deklarativ ...), aber auch auf dem individuellen Stil des Programmierers, welcher beim Programmiervorgang praktiziert wird (z. B. strukturiert, modular)[14].

Im folgenden Kapitel werden die Grundlagen dieser Programmier-Paradigmen detaillierter erläutert.

Arten von Programmiersprachen

Die Programmiersprachen unterscheiden sich nach den Arten ihrer Anwendung und Anwendungsgebiete voneinander. Die Unterscheidung unterschiedlicher Anwendungen wird auch Programmierparadigma genannt. Die Programmiersprachen werden auch assoziiert mit den Schichten des OSI Modells (siehe Kapitel „Internet und Networking"). Die hardware-nahen Schichten führen Maschinensprache in Form von z.B. Binär-oder Hexadezimal-Codes aus. Eine weitere Abstraktion der Maschinensprache stellt die Assembler-Programmierung dar, die spezifisch für einen Prozessor ist. Auf diese Assembler-Codes werden die Basisfunktionen eines

[14] https://de.wikipedia.org/wiki/Programmierparadigma

Betriebssystems aufgebaut. Das Betriebssystem sorgt mit seinen Treibern dafür, das unterschiedliche Komponenten (motherboard, Video-Karte, Sound-Karte, USB-Schnittstellen, HDMI-Schnittstellen etc) miteinander agieren können. Auf der Ebene der Betriebssysteme wird üblicherweise C oder C++ als Programmiersprache angewendet. Höhere Abstraktions-Schichten zur Implementierung von Anwendungen berücksichtigen dann Audio-, Video-, Bilddarstellungsaspekte sowie Kommunikations-Schnittstellen und Daten-Bankaspekte sowie Sicherheitsaspekte, rechnerspezifische Funktionsweisen etc. Für diese Schichten werden höhere Programmiersprachen verwendet.

Ein Programmierer der mehrere Sprachen beherrscht, um eine spezifische Anwendung oder einen Dienst zu realisieren, nennt sich „Full-Stack Programmierer“[15], d.h. die Sprachen auf unterschiedlichen OSI Ebenen, die für die komplette Funktionalität des Dienstes notwendig sind, stellen den so genannten „Stack“ (d.h. Stapel an Sprachen und Funktionsweisen) dar, um funktionale und nicht-funktionale Eigenschaften des Dienstes zu realisieren, d.h. was der Dienst tut (funktional) und wie er konfiguriert wird, bzw. wie er für den Endverbraucher aussieht (nicht-funktional).

Ein Programmierparadigma ist auch ein fundamentaler Programmierstil. Der Programmierung liegen je nach Design der einzelnen Programmiersprache verschiedene Prinzipien zugrunde. Diese sollen den Entwickler bei der Erstellung von ‚gutem Code‘ unterstützen, in manchen Fällen sogar zu einer bestimmten Herangehensweise bei der Lösung von Problemen zwingen[16].

Die Programmierparadigmen bestimmen die Arten der Programmiersprachen:

1. Die Imperative Programmierung (lateinisch imperare ‚anordnen‘, ‚befehlen‘) ist ein Programmierparadigma, nach dem „ein Programm aus einer Folge von Anweisungen besteht, die vorgeben, in welcher Reihenfolge was vom Computer getan werden soll“.

Die imperative Programmierung ist das am längsten bekannte Programmierparadigma. Diese Vorgehensweise war, bedingt durch den Sprachumfang früherer Programmiersprachen, ehemals die klassische Art des Programmierens. Sie liegt dem Entwurf von vielen Programmiersprachen, zum Beispiel ALGOL, Fortran, Pascal, Ada, PL/I, Cobol, C und allen Assemblersprachen zugrunde.

Abweichende Bezeichnungen: In der Literatur wird dieses Entwicklungskonzept zum Teil auch „imperativ/prozedural“, „algorithmisch“ oder auch „zustandsorientiert“ genannt. Auch die Bezeichnung „prozedurale Programmierung“ wird zum Teil synonym verwendet, was jedoch abweichend auch mit „Verwendung von Prozeduren“ definiert wird.[17]

---

[15] https://usersnap.com/de/blog/7-tipps-full-stack-entwickler

[16] https://de.wikipedia.org/wiki/Programmierparadigma

[17] https://de.wikipedia.org/wiki/Imperative_Programmierung

Diese Sprachen zeichnen sich durch logische Kontrollstrukturen aus, wie zum Beispiel „if… then…", „case… of…", „while… do…", „repeat… until…" .
Bei einer Assemblersprache, kurz auch Assembler genannt (von englisch to assemble ‚montieren'), handelt sich um den Befehlsvorrat eines bestimmten Computertyps (d. h. Befehls-Konstrukte, die dessen Prozessorarchitektur entsprechen).
Die Assemblersprachen bezeichnet man deshalb als maschinenorientierte Programmiersprachen, denn anstelle eines Binärcodes der Maschinensprache können Befehle und deren Operanden durch leichter verständliche mnemonische Symbole in Textform dargestellt werden, z. B. „MOVE" ist ein Befehl und sein Operand als symbolische Adresse ist z. B. eine Puffer-Adresse.[18] Die ursprünglichen Binärcodes der Maschinensprache sind dagegen nur Zahlen (z.B. in Hexadezimal-System, Zahlensystem auf Basis 16) und daher schwer verständlich und schwer handelbar.
In diesem Programmierparadigma unterscheiden sich auch die folgenden Unter-Paradigmen, wie

- Prozedurale Programmierung – d.h. Programme werden in kleinere Teilaufgaben aufgespalten, z.B. sind Sprachen mit solchen Eigenschaften Fortran, Pascal und C
- Modulare Programmierung – d.h. Programme werden in logische Teilblöcke aufgespalten die Module genannt werden, z.B. sind solche Sprachen Modula-2, Ada, Oberon.

Im Rahmen dieser Sprachen wurde auch der sogenannte Abstrakte Datentyp definiert. Ein Abstrakter Datentyp (ADT) ist ein Verbund von Daten zusammen mit der Definition aller zulässigen Operationen, die auf sie zugreifen. Beispiele für ADTs sind mnemonische Symboliken wie characters und strings. Diese ADT-Definition wird später auch als Grundlage der Objektorientierten Sprachen benutzt.

2. Deklarative Programmierparadigmen – Die Idee einer deklarativen Programmierung ist der historisch jüngere Ansatz. Im Gegensatz zu imperativen Programmierparadigmen, bei denen das *Wie* im Vordergrund steht, fragt man in der deklarativen Programmierung nach dem *Was*, das berechnet werden soll. Es wird also nicht mehr der Lösungsweg programmiert, sondern nur noch angegeben, welches Ergebnis gewünscht ist. Zu diesem Zweck beruhen deklarative Paradigmen auf mathematischen, rechnerunabhängigen Theorien. Hier steht die Logik, wie bei der Modularen Programmierung, im Vordergrund. Zu den deklarativen

[18] Ein Puffer ist eine organisierte Ansammlung von bits zur Durchführung von mathematischen Operationen. Ganz frühe Rechner waren mit 8 bit ausgestattet. Moderne Rechner haben bis zu 64 bit Puffer für die Darstellung von Zahlenformaten. Mit steigender Anzahl von bits für die Zahlendarstellung können genauere Berechnungen durchgeführt werden. Die Auswirkungen genauerer Zahlendarstellung führen u.a. zu verbesserter Bild- und Tonwiedergabe an der Mensch-Maschinen-Schnittstelle.

Programmiersprachen gehören die funktionalen Sprachen (unter anderem Lisp, ML, Miranda, Gofer, Haskell, F#, Scala), logische Sprachen (unter anderem XML/XSLT, Prolog), funktional-logische Sprachen (unter anderem Babel, Escher, Curry, Oz) und mengen-orientierte Abfrage-/Suche-Sprachen (unter anderem SQL, RegEx).

3. Objektorientierte Programmierparadigmen – Unter Objektorientierung (kurz OO) versteht man in der Entwicklung von Software eine Sichtweise auf komplexe Systeme, bei der ein System durch das Zusammenspiel kooperierender Objekte beschrieben wird. Einfache Objekte sind Abstrakte Datentypen und ihre Funktionen. Ein String-Objekt hat z.B. die Funktion Wörter eines Satzes zusammenzusetzen. Wenn zwei Strings vorliegen („Hello" und „World") erlaubt die Funktion eines String-Objektes diese zwei Wörter als einen Satz („Hello World") zusammenzusetzen. Der Begriff Objekt ist unscharf gefasst; wichtig an einem Objekt ist nur, dass ihm bestimmte Attribute (Eigenschaften) und Methoden zugeordnet sind und dass das Objekt in der Lage ist, von anderen Objekten Nachrichten zu empfangen beziehungsweise an diese zu senden. Dabei muss ein Objekt nicht gegenständlich sein. Entscheidend ist, dass bei dem jeweiligen Objektbegriff eine sinnvolle und allgemein übliche Zuordnung möglich ist. Ergänzt wird dies durch das Konzept der Klasse, in der Objekte aufgrund ähnlicher Eigenschaften zusammengefasst werden. Ein Objekt wird im Programmcode als Instanz beziehungsweise Inkarnation einer Klasse definiert.
   Objektorientierung wird hauptsächlich im Rahmen der objektorientierten Programmierung verwendet, um die Komplexität der entstehenden Programme zu verringern. Der Begriff existiert jedoch auch für andere, der Programmierung vorgelagerte Phasen der Softwareentwicklung, wie die objektorientierte Analyse und objektorientiertes Design (Synonym objektorientierter Entwurf) von Software. Die Konzepte der Objektorientierung lassen sich zudem auf persistente Daten anwenden. Dabei spricht man von Objektdatenbanken.
   In Programmiersprachen, die nicht auf Objektorientierung eingerichtet sind, werden Daten und Programmteile bewusst getrennt; sie müssen separat deklariert werden. Im Vergleich hierzu erhebt das objektorientierte Programmierparadigma den Anspruch, Daten und zugehörige Programmteile zu einer Einheit zusammenzufassen und somit Organisationsstrukturen aus der realen Welt besser nachzubilden.
   Fast alle höheren Programmiersprachen unterstützen objektorientierte Programmierung. Typische Beispiele für solche Programmiersprachen sind Java und C++, Python.[19]

[19] https://de.wikipedia.org/wiki/Objektorientierung

Es gibt auch weitere Programmierparadigmen, die entweder Sprachvarianten oder Programmierstil-Varianten betreffen, d.h. wie einzelne Programmcodestücke (Textfiles, Quellcodes) intern geschrieben und geordnet werden, bzw. wie die Gesamtanwendung als Code organisiert bzw. kompiliert oder interpretiert wird.
Je nach Kompilierung oder Interpretierbarkeit einer Sprache, gibt es auch die Unterscheidung zwischen compilierbaren, interpretierbaren bzw. Skript-Sprachen[20], z.B.:

- Java – in Bytecode compiliert; durch Java Virtual Machine interpretiert oder weiter in Maschinencode compiliert
- C++ – compiliert
- JavaScript – interpretiert
- C# – in Common Intermediate Language compiliert; durch Virtual Execution System interpretiert
- Python – interpretiert
- PHP – interpretiert
- Ruby – interpretiert

Bei der Compilierung und Interpretierbarkeit wird zwischen folgenden Konzepten unterschieden:

1. Compiler[21]

Ein *Compiler* (auch Kompilierer; von englisch compile ‚zusammentragen' bzw. lateinisch compilare ‚aufhäufen') ist ein Computerprogramm, das Quellcodes einer bestimmten Programmiersprache in eine Form übersetzt, die von einem Computer-Prozessor direkt ausgeführt werden kann. Für jede Hardware-Prozessor-Architektur entsteht durch die Compilierung ein spezifisches Programm in binärem Code. Davon zu unterscheiden sind Interpreter, etwa für frühe Versionen von BASIC, die keinen Maschinencode erzeugen obwohl sie BASIC Compiler genannt wurden.
Es muss zwischen den Begriffen Übersetzer und Compiler unterschieden werden. Ein Übersetzer übersetzt ein Programm aus einer formalen Quellsprache in ein semantisches Äquivalent in einer formalen Zielsprache. (Ein Beispiel wäre ein Übersetzer von Java in C++ oder zurück). Compiler sind spezielle Übersetzer, die Programmcodes aus problemorientierten Programmiersprachen, sogenannten Hochsprachen, in ausführbaren Maschinencode einer bestimmten Architektur oder einen Zwischencode (Bytecode, p-Code oder .NET-Code) überführen. Diese Trennung zwischen den Begriffen Übersetzer und Compiler wird in der Literatur nicht in allen Fällen sauber vorgenommen.

---

[20] https://de.wikipedia.org/wiki/H%C3%B6here_Programmiersprache
[21] https://de.wikipedia.org/wiki/Compiler

Der Vorgang der Übersetzung von einer hohen Sprache in Maschinensprache ist die Compilierung. Das Gegenteil, also die Rückübersetzung von Maschinensprache in Quelltext einer bestimmten Programmiersprache, wird Decompilierung und entsprechende Programme werden Decompiler genannt.
Klassische Compiler werden auch Ahead-of-time-Compiler genannt[22], da der Quellcode im Voraus, also vor der Programmausführung, übersetzt wird. Der Programmcode liegt als Maschinencode vor, Beispiel dafür sind die „direct executables", die unter Windows-Betriebssystem als *.exe-Files bekannt sind.

2. Interpreter[23]

Als Interpreter wird ein Computerprogramm bezeichnet, das eine Abfolge von Anweisungen scheinbar direkt ausführt, wobei das Format der Anweisungen vorgegeben ist. Der Interpreter liest dazu eine oder mehrere Quelldateien ein, analysiert diese und führt sie anschließend Anweisung für Anweisung aus, indem er sie in Maschinencode übersetzt, die ein Computersystem direkt ausführen kann. Interpreter sind deutlich langsamer als Compiler, da sie den Programmcode während der Laufzeit direkt aus einer höheren Sprache interpretieren, ermöglichen aber durch die zeilenweise Ausführung des Codes auch eine Analyse und Lokalisierung der Fehler in Codes[24].

3. Just-in-time-Compilierung[25]

Just-in-time-Compilierung (JIT-Compilierung) ist ein Verfahren aus der angewandten Informatik, um Teil-Programme zur Laufzeit in Maschinencode zu übersetzen. Just in time bedeutet in diesem Kontext „termingerecht", „bei Bedarf" (analog zur Just-in-time-Produktion). Zu diesem Zweck werden Programmbestandteile voranalysiert um zu entscheiden, ob sie zur Installationszeit oder zum Zeitpunkt der Ausführung des Programms compliert werden. Ziel ist es dabei, die Ausführungsgeschwindigkeit gegenüber einem reinen Interpreter zu steigern. JIT-Compiler kommen meist im Rahmen einer virtuellen Maschine (VM)[26] zum Einsatz, wo Plattform-unabhängiger Maschinencode ausgeführt werden soll.
Software wird heutzutage in einer Vielzahl unterschiedlicher Programmiersprachen geschrieben. Viele dieser Programmiersprachen werden typischerweise nicht vor ihrer Ausführung zu Maschinencode kompiliert, sondern stattdessen durch eine virtuelle Maschine ausgeführt. Gründe

---

[22] https://de.wikipedia.org/wiki/Ahead-of-time-Compiler
[23] https://de.wikipedia.org/wiki/Interpreter
[24] https://de.wikipedia.org/wiki/Debugger
[25] https://de.wikipedia.org/wiki/Just-in-time-Kompilierung
[26] Die Virtuelle Maschine ist eine Software, die den interpretierten Code in ein Format für einen betriebssystem- und hardwarespezifischen Prozessor überführt.

dafür sind beispielsweise eine gewünschte Plattformunabhängigkeit oder die dynamische Natur der Programmiersprache. Einige Beispiele für solche Programmiersprachen sind JavaScript und Python. Die Ausführungsgeschwindigkeit der VM entspricht der eines Interpreters und ist häufig geringer im Vergleich zu nativ compilierten Programmen, die aus direkt durch den Prozessor ausführbaren Instruktionen bestehen. Durch Just-in-time-Compilierung versucht man, diesen Nachteil auszugleichen.

Eine weitere Form von Just-in-time Compilierung ist es, den Maschinencode compilierter Programme zur Laufzeit zu modifizieren oder auch zu optimieren. Bei manchen Software Updates ist es nicht notwendig den Ablauf des Programmcodes zu stoppen, sondern es wird nur ein bestimmtes Modul ersetzt und das Programm kann weiterlaufen. Auch bei Analyse von Programmen mit spezifischen Debuggern (Fehleranalyse-Programmen) kann Just-in-time-Compilierung in Frage kommen.