Software – eine Chance für KMU und Start-ups: Herstellungsprozesse ohne große Produktionsanlagen, umfangreiche Belegschaft für einfache Tätigkeiten, Warenlogistikleistungen usw. erleichtern den Markteinstieg gerade für diese Unternehmen in die Raumfahrt. Da Software außerdem weitgehend unabhängig von den Weltraumbedingungen ist, bietet sie großes Potenzial für Raumfahrt-Spin-ins und Spin-outs.

Software – a great opportunity for SMEs and start-ups. Being able to make do without any major production facilities, a big workforce performing simple jobs, or logistics operations makes it easier for these space companies to enter the space tech market. The fact that software is largely unaffected by the operating conditions prevailing in space makes it a high-potential business for space spin-ins and spin-outs.

Gehirne von Raumfahrtmissionen

Ohne Software läuft in der Raumfahrt fast nichts

Von Dr. Christian Prause und Uwe Soltau

Sie wiegt nichts, steuert dennoch fast alles und trägt somit zum erfolgreichen Gelingen von Programmen maßgeblich bei: Software. Heutzutage ist sie ein Teil von praktisch allen Weltraumsystemen und als sogenannte Embedded Software oder Firmware oft schon in einfachsten Geräten enthalten. Gleichzeitig ist sie – im Gegensatz zu Hardware – von Schwerelosigkeit, Vakuum, Erschütterungen, extremen Temperaturen und Temperaturschwankungen und den verschiedenen Arten von Strahlung – also den physikalischen Bedingungen einer Weltraummission – unabhängig. Das bedeutet jedoch nicht, dass Software ausführende Hardwareteile wie Prozessor oder Arbeitsspeicher nicht doch ausfallen könnten. Aus diesem Grund verwenden zum Beispiel Ariane-Raketen mehrere Bordcomputer. Softwarekomponenten sind miteinander und gleichzeitig mit Hardware verbunden. Sie kommunizieren, konkurrieren und steuern Hardware. Fortschritte auf dem Gebiet der Software und ihrer Entwicklung machen Raumfahrtmissionen – über einen langen Zeitraum gesehen – zuverlässiger, flexibler und günstiger.

Brains of missions

Without software space technology could not run successfully

By Dr Christian Prause and Uwe Soltau

It weighs nothing but controls almost everything and contributes significantly to a successful mission: software. Today, it figures in virtually all space systems, often forming part of even the most basic devices as embedded software or firmware. At the same time, it is – unlike hardware – unaffected by microgravity, vacuum, shocks, extreme temperatures and temperature fluctuations, and various kinds of radiation – in other words: the physical conditions of a space mission. However, this is not to say that the hardware components which execute software, such as processors or memory, do not fail. It is for this reason that Ariane launchers, for example, use several on-board computers. Software components are closely interconnected and linked with hardware. Software communicates with other components and competes for resources. Progress in the field of software and its development makes space missions more dependable, more flexible and cost-efficient in the long run.

Autoren: **Dr. Christian Prause** (links) arbeitet in der Abteilung Produktsicherung im DLR Raumfahrtmanagement. Er ist der Fachgebietsleiter Software-Qualitätssicherung in Bonn. **Uwe Soltau** ist Fachgruppenleiter Industriepolitik in der Abteilung Raumfahrt-Strategie und Programmatik und KMU-Beauftragter des DLR Raumfahrtmanagements.

Authors: **Dr Christian Prause** (left) works at the Product Assurance department of the DLR Space Administration. He is in charge of software quality assurance. Working at the department for Space Programme Planning, **Uwe Soltau** is head of Industrial Policy and the SME-Commissioner of the DLR Space Administration.

Von Software wird viel erwartet: Sie soll die unterschiedlichsten Geräte an Bord eines Satelliten korrekt verbinden, ansteuern und bedienen sowie plötzlich auftretende Fehlfunktionen entdecken, isolieren und abfangen. Um dem Satelliten eine gewisse Selbstständigkeit zu geben, muss die Software die Zustandsparameter der Geräte kennen und analysieren. Dazu gehören Parameter, die deren inneren Zustand beschreiben, wie Batterieladezustand oder verfügbarer Speicher, aber auch diejenigen des äußeren Zustands, wie Tag/Nacht oder Position. Software gibt einem Satelliten somit ein beschränktes „Selbstverständnis". Sie ist dessen Gehirn – also das Organ, das man aufgrund seiner Komplexität in der Medizin bislang am wenigsten verstanden hat. In Software selbst steckt dabei eine noch viel höhere technische Komplexität – je vielschichtiger ihre Umgebung, desto anspruchsvoller die Prozesse. Besteht ein kleines Programm mit wenigen Zeilen vielleicht noch aus überschaubar vielen Logiken und Rechenpfaden, sind schon bei doppelter Größe exponentiell mehr Pfade möglich. Daher lässt sich ein umfangreiches Programm nicht mehr vollständig durch Tests überprüfen. Das Testen muss also mit anderen Verfahren kombiniert werden.

Immun gegen Alterung und Verschleiß

Soft- und Hardwaretests unterscheiden sich nicht nur voneinander, weil Programme gegenüber den physikalischen Bedingungen des Weltraums immun sind: Überprüft man ein Stück Hardware, so kann dieses beim zehnten, hundertsten oder

Software is expected to do a lot. It is supposed to correctly control and operate a wide range of devices on board a satellite and to detect, isolate, and recover sudden faults. To give a satellite a certain degree of autonomy, software has to know and analyse the interior (e.g. battery charge, available memory) and exterior condition (e.g. day/night, position) of each device, thereby providing a satellite with a limited degree of self-perception. Software is a satellite's brain – the organ which to this day is most imperfectly understood in medicine because of its complexity. The technical complexity inside of software tends to increase very rapidly – the greater the variety of facets in its environment, the more demanding the processes will be. While a small program comprising a simple logic and a few lines of code may consist of a manageable number of computational paths, that number grows exponentially even when the length of a program is merely doubled. Therefore, a lengthy program cannot be checked in its entirety merely by testing, so that testing needs to be combined with other methods.

Immune to ageing and wear

Software and hardware tests differ not only because programs are immune to the physical conditions of space: a piece of hardware you are testing may suddenly break after ten, a hundred or a thousand test runs. Software, on the other hand, knows neither ageing nor wear: two identical inputs will always produce two identical results. However, this also applies if a program is

Links: Ariane-Raketen verwenden mehrere Bordcomputer. Fällt ein Rechner aus, so kann ein anderer einspringen, um einen Systemausfall zu vermeiden. Rechts: Der Mars Climate Orbiter (MCO) der NASA sollte günstiger gebaut werden. Dadurch wurde er gegen die ursprüngliche Planung mit nur einem anstatt symmetrisch mit zwei Solarpanelen konstruiert. So musste als Designkompromiss eine Software entwickelt werden, die die Orientierungsänderungen der asymmetrischen Sonde ausgleichen sollte. Ein Einheitenfehler führte zu einer falschen Navigation und der Orbiter verglühte im Jahr 1999 in der Mars-Atmosphäre.

Left: Ariane launchers use more than one on-board computer. If one of them fails, another can take over, so a full system failure can be ruled out. Right: NASA intended to build a less costly version of the Mars Climate Orbiter (MCO). As a result, it was designed with only one instead of two symmetrically arranged solar panels. To compensate for this design modification, the software developers had to come up with a compromise solution that would make up for the altered orientation of the asymmetrical space vehicle. A unit error ended up corrupting the orbiter's navigation, causing it to burn up in the Martian atmosphere in 1999.

tausendsten Mal plötzlich defekt sein. Nicht so Software, denn sie kennt weder Alterung noch Verschleiß: Sind zwei Eingaben gleich, dann wird das Ergebnis auch immer das Gleiche sein. Das gilt jedoch auch, wenn ein Programm fehlerhaft ist. Daher funktioniert Redundanz im klassischen Sinn nicht: Der Fehler würde immer wieder beziehungsweise überall auftreten. Das musste die Ariane 5 auf ihrem allerersten Flug am 4. Juni 1996 erfahren. Nach einem perfekten Start trat ein Softwarefehler in einem eigentlich nur vor dem Start benötigten Teil des Navigationssystems auf. Der Hauptcomputer konnte jedoch nicht auf das Ersatzsystem umschalten, da es wegen des gleichen Fehlers bereits zuvor abgestürzt bzw. abgeschaltet war. Die Steuerdüsen verursachten daraufhin eine abrupte Bahnänderung, die die Rakete unkontrolliert fliegen und auseinanderbrechen zu lassen drohten. Um noch Schlimmeres zu verhindern, musste die Selbstzerstörung ausgelöst werden. Entsprechend muss man feststellen: Es half nicht, dass die Ariane mehrere Rechner mit der gleichen Software an Bord hatte. Redundanz ist nur möglich, wenn unterschiedliche Programme verwendet werden – ein kostspieliges Unterfangen. Da allerdings auch ein Fehlstart oder eine Fehlfunktion sehr teuer ist, kann es – ganz im Sinne von Philip Crosby – billiger sein, die Software direkt beim ersten Mal richtig zu entwickeln.

Ohne Produktionskosten in den Weltraum

Der Entwicklungsaufwand für Software ist bei gleichen Komponenten-Gesamtkosten wesentlich umfangreicher als für Hardware – da die Herstellung von Software zu nahezu 100 Prozent Entwicklungsarbeit ist. Diese Prozesse sind organisatorisch aufwändig, da mehr Entwickler zusammenarbeiten. Im Hardwarebereich sind außerdem Aufgabengebiete wie zum Beispiel Thermal, Struktur und Optik leichter zu trennen. Gleichzeitig leben Software-Entwicklungsarbeiten von Kreativität. Dafür braucht es jedoch Freiraum, der auf Kosten der Nachhaltigkeit gehen kann. Das Bild des einfachen Programmierers, der Anweisungen stupide in Programmcodes umsetzt, ist eine Illusion. Vielmehr sind Softwareentwickler die prototypischen „knowledge workers" mit einer hochgradig wissensintensiven Arbeit. Die Herausforderung: Kreativität und Freiheit müssen mit organisatorischer Steuerung für Nachhaltigkeit in einem ausgewogenen Gleichgewicht gehalten werden. Dafür fallen – anders als bei Hardware – so gut wie keine Produktionskosten an. Der abschließende Herstellungsprozess kann somit praktisch vollständig automatisiert erfolgen und nahezu perfekt reproduzierbar ablaufen. Dies entspricht den kühnsten Träumen von Hardwareproduzenten zum Beispiel aus dem Automobil- oder Halbleiterbereich. Es gibt praktisch keine einfachen Tätigkeiten

faulty. This is why redundancy in the classical sense will not work: an error would occur at all times and/or everywhere. This is what happened to Ariane 5 on its very first flight on June 4, 1996. After a perfect lift-off, a software error occurred in a part of the navigation system that really was needed only before the launch. However, the main computer was unable to switch to the standby system since it had crashed before because of the same error. Subsequently, the control jets caused the rocket to change its path abruptly, so that it threatened to break apart. To prevent an even worse outcome, self-destruction had to be initiated. So we have to conclude that it did not help Ariane to have several computers with identical software on board. Therefore, redundancy is an option only if different programs are used – a costly undertaking. However, as an aborted launch is very costly as well, it may be cheaper, following Philip Crosby, to develop the software right the first time.

To space without production costs

Assuming that the total component cost is the same, the development effort for software is materially more extensive than for hardware – because building software consists to nearly 100 per cent of development work. Organising a process of software development is complex because more developers are working together. In the hardware sector, moreover, the tasks of individual developers working on, for example, thermal, structural, or optical issues are easier to keep separate. At the same time, development work thrives on creativity. Creativity, however, calls for latitude, which may have a negative impact on sustainability. The idea of a simple programmer mindlessly translating instructions into program codes is illusory. Rather, software developers are archetypal knowledge workers doing a highly scientific job. The challenge: there has to be a balance between creativity and latitude on the one hand and organisational control for sustainability on the other. Therefore, the cost of producing software – unlike that of hardware – is practically nil. Thus, software may be made in a process that is virtually fully automated and almost perfectly reproducible, fulfilling the wildest dreams of hardware producers from, for example, the motor-vehicle or semiconductor industry. Software production involves practically no simple activities, such as soldering, gluing, screwing, painting, commissioning, carrying or packaging. Routine work is continuously being eliminated by automation.

Großes Potenzial für KMU

Gerade einmal gut 50 Jahre alt – seit der berühmten NATO-Konferenz in Garmisch-Partenkirchen 1968 – ist die Softwaretechnik eine junge und empirische Wissenschaft, die sich noch stark in der Entwicklung befindet. Dieser Markt bietet gerade KMU und Start-ups Raum für einen einfachen Einstieg, da keine großen Produktionsanlagen, keine umfangreiche Belegschaft für einfache Tätigkeiten, Warenlogistikleistungen usw. benötigt werden. „Open Source"-Software gestattet auch kleinen Unternehmen mit kleinem Kapital Zugang zu modernen Produktionsmitteln. Da Software weitgehend unabhängig von den physikalischen Eigenschaften ihrer Umgebung ist, bietet sie großes Potenzial für Spin-ins in und Spin-outs aus der Raumfahrt.

Great potential for SMEs

Having just about reached the age of 50 after the famous NATO conference of 1968 in Garmisch-Partenkirchen, software engineering is a young and empirical discipline of science that is still very much in its development stage. It offers the chance of an easy market entry to SMEs and start-ups because there is no need for large production facilities and numerous employees to handle basic jobs, logistics services, etc. 'Open Source' software affords access to modern means of production even to small enterprises with little capital. Being largely independent of its physical production environment, software offers a great potential for spin-ins and spin-outs in the space industry.

Dabei gliedert sich Software in drei große Themenbereiche:

- Flugsoftware, die als Komponente die Steuerung im Flug übernimmt,
- Ground-support Software, bestehend aus Teilen für die Zeit vor, während und nach einem Überflug, und
- Softwarewerkzeuge, um Missionen, Hardware und Software zu gestalten und zu entwickeln.

Software falls into three major categories:

- Flight software, which handles in-flight control as a component,
- ground support software comprising segments addressing the time before, during, and after a flyover, and
- software tools to design and develop missions, hardware and software.

wie Löten, Kleben, Schrauben, Streichen, Kommissionieren, Tragen oder Verpacken in der Softwareproduktion – Routinearbeiten werden kontinuierlich durch Automation eliminiert.

Software rettet Weltraummissionen

Dennoch dient Software oftmals als „Heftpflaster" für Designkompromisse – dies kann eine Gefahr für die Sicherheit von Missionen oder aber zugleich eine Chance auf Kosteneinsparung sein. Der Mars Climate Orbiter (MCO) der NASA ist hierfür ein klassisches Beispiel: Hätte das „Faster-Better-Cheaper"-Regime nicht auf eine starke Senkung der Projektkosten gedrängt, wäre der Satellit des Discovery-Programms wie geplant symmetrisch mit zwei Solarpanelen statt einem einzelnen gebaut worden. So aber wurde als Designkompromiss eine Software entwickelt, die die Orientierungsänderungen der Sonde, die aus dem Strahlungsdruck der Sonne auf die asymmetrische Struktur entstehen, ausgleichen sollte. Die Folge: Ein Einheitenfehler in der Software führte zu einer falschen Navigation und der Orbiter verglühte im Jahr 1999 in der Mars-Atmosphäre. Ungeachtet des Fehlschlags hat der NASA-Ingenieur Eberhardt Rechtin zu Recht auf das Potenzial von Software zur Kostensenkung hingewiesen. Die Flexibilität von Kommunikationssatelliten durch Software zeigt es. Dabei sollte man jedoch klar feststellen, dass Software viel mehr Missionen gerettet hat als durch sie verloren gegangen sind. Denn sie lässt sich – im Gegensatz zur Flughardware – nach dem Start noch ändern, wovon auch oft Gebrauch gemacht wird. So kann Software Hardwareausfälle noch ausgleichen, wo andernfalls mit Einschränkungen oder gar dem Verlust der Mission gerechnet werden müsste. Ein anderes Beispiel sind FDIR (Fault Detection, Isolation and Recovery)-Softwaresysteme, die es dem Raumfahrzeug erlauben, flexibel auf problematische Ereignisse zu reagieren. Da Software kein Gewicht hat, erhöhen zusätzliche Funktionen, anders als Hardware, auch nicht das Startgewicht. Daher werden immer häufiger auch kritische Funktionen – manchmal sogar ausschließlich – durch Software abgesichert. In der Vergangenheit mussten gefährliche Funktionen immer zusätzlich durch Hardwaremaßnahmen abgesichert werden. Um diesem Umstand Rechnung zu tragen, wird eine überarbeitete Fassung bestimmter Raumfahrtstandards Software und Hardware in Bezug auf ihre Funktion als kritische Komponente auf eine Stufe stellen.

Investitionen in Software zahlen sich langfristig aus

Im Automobilbereich entfallen inzwischen über zwei Drittel aller Innovationen auf Software. Bei Militärflugzeugen ist im Zeitraum von 1960 bis 2000 der Anteil an Funktionalität, die von Software zur Verfügung gestellt wird, von acht auf 80 Prozent angewachsen. Der Umfang des Codes ist von 1.000 Zeilen auf sechs Millionen angestiegen und schätzungsweise 24 Millionen im modernsten Kampfflugzeug – der F-35. Eine Zahl, die in der Raumfahrt bislang noch nicht erreicht wird. Dennoch zahlt sich das nun aus: Für wissenschaftliche Missionen hat sich das Verhältnis von Hard- zu Softwarekosten inzwischen von 10:1 zu 1:2 umgekehrt. Im Bereich der Flugsoftware überstiegen vor 20 Jahren die Kosten erstmals die Marke von zehn Prozent an den Gesamtkosten. Neuere Zahlen aus dem amerikanischen Raum sehen die Kosten für die Entwicklung von Flugsoftware heute sogar bei 20 bis 30 Prozent des gesamten Raumfahrzeugs.

Software as a space mission safeguard

Software often serves as a 'band-aid' for compromises in design; while this, on the one hand, could pose a threat to the safety of a mission or a way to save on costs. One classical case in point is NASA's Mars Climate Orbiter (MCO): if the faster-better-cheaper regime had not urged a massive reduction in the cost of the project, this satellite of the discovery programme would have been built symmetrically as planned, featuring two solar panels instead of one. Instead, the engineers decided to go for a design compromise, based on a software package that would compensate changes in the attitude of the probe caused by the pressure exerted by solar radiation on the asymmetrical structure. The consequence: a metric/imperial unit conversion error in the software led to faulty navigation, and the orbiter burned up in the atmosphere of Mars in 1999. Leaving this mishap aside, the famous NASA engineer Eberhardt Rechtin was quite right in insisting on the cost-cutting potential of software. The flexibility of communication satellites by software prove this statement. It should also be clearly stated in this context that software has saved many more missions than were lost because of it. Software, unlike flight hardware, can, for example, still be modified after take-off, an option which is frequently used. Thus, hardware failures may be compensated by software instead of a restriction oder loss of a mission. Another example are Fault Detection, Isolation and Recovery (FDIR) systems implemented in software that allow a spacecraft to flexibly react to problematic events originating from inside or outside the spacecraft. As opposed to hardware, software has practically no weight. Hence, functions implemented in software do not increase launch weight. This is why more and more critical functions are realised in software, sometimes even exclusively. In the past, hazardous functions always had to be backed up by hardware solutions. To account for this situation, revised versions of certain space standards will place software and hardware on the same level as far as their function as critical components is concerned.

Investments in software pay off in the long run

In the motor vehicle sector, more than two thirds of all innovations relate to software by now. In military aircraft, the proportion of functionality provided by software rose from eight to 80 per cent in the period from 1960 to 2000. Code lengths have increased from 1000 to six million lines and by guess up to 24 million in the latest fighter yet F-35. A mark, which has not been reached in the space flight sector. However, in the space sector, too, software investments have been massive, and they are now paying off: in scientific missions, the ratio between hardware and software costs has been reversed, from 10:1 to 1:2. It was 20 years ago that the cost of flight software first rose above the mark of ten per cent of the overall cost. More recent figures from the American region even indicate that the present cost of developing flight software ranges between 20 and 30 per cent of the entire spacecraft.