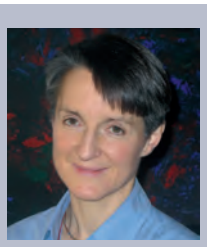
Herausforderungen und Chancen des Software Lifecycle Managements

Maßnahmen gegen Software Obsolescence

Aktuell spricht alles dafür, dass das Thema „Software Obsolescence" Einzug in die Neuauflage der EN62402 „Anleitung zum Obsoleszenzmanagement" hält, die voraussichtlich zum Jahreswechsel 2017/2018 veröffentlicht werden wird. Gleiches gilt für die VDI-Richtlinie 2882, die im Gründruck vorliegt und Anfang 2017 in finaler Version erscheint.

VON HEIKE JORDAN,
GESCHÄFTSFÜHRERIN DER EMLIX GMBH
UND VERANTWORTLICH FÜR
DAS OBSOLESCENCE MANAGEMENT
BEI MAINTENANCE-KUNDEN

Was ist nun konkret mit Software Obsolescence gemeint? Im Gegensatz zu elektronischen Komponenten und Bauteilen, altert Software per se gar nicht. Sie altert jedoch im Kontext, relativ, mittelbar und organisatorisch. Relativ, weil Hardware altert und sich weiterentwickelt, weil Baurechner wegfallen oder weil mit einem modernen Compiler die Baubarkeit nicht mehr gegeben ist. Ein aktuell sehr wesentlicher Aspekt ist die relative Alterung aufgrund von Security Bugs. Verstärkend wirken neue rechtliche Rahmenbedingungen, insbesondere die Neuauflage des IT-Sicherheitsgesetzes. Software altert mittelbar, weil Tools und Programmiersprachen „sterben", Kryptoalgorithmen altern und Serverzertifikate verfallen. Ein plötzlicher Alterungsprozess tritt ein, wenn „der eine" Software-Entwickler das Unternehmen verlässt und niemand mehr mit vertretbarem Risiko am Code etwas ändern kann. Ebenso bewirkt das Jahr-2038-Problem mittelbare Software-Alterung. Bei Systemen, die Unixzeit nutzen, wird es nach 2 hoch 31 sec seit dem 01.01.1970 zu einem Zählerüberlauf und damit zu signifikanten Ausfällen kommen.



Heike Jordan, emlix

„Um der Software Obsolescence im Feld begegnen zu können, sollte der Update-Prozess unbedingt bereits Bestandteil der frühen Design-Phase sein."

Organisatorische Alterung meint einen Verfall, der im Entwicklungs- und Maintenance-Prozess angelegt ist: Die Software wurde nicht personenunabhängig entwickelt und dokumentiert, es gibt kein nachvollziehbares Konfigurationsmanagement und kein verlässliches Versions- oder Schlüsselmanagement. Proprietäre Software altert außerdem automatisch mit ihrem Hersteller.

Hieraus ergeben sich konkrete Herausforderungen, um den Alterungsprozess aufzuhalten. Die organisatorischen Maßnahmen liegen auf der Hand, lassen sich jedoch bei gegebenen Stellenplänen nicht immer problemlos umsetzen. Es sollte jedoch mindestens auf die Zugänglichkeit des entstandenen Source Codes, der für die Reproduktion der Software erforderlichen Werkzeuge sowie eine aussagekräftige Dokumentation geachtet werden. Das Management von Schlüsseln und Zertifikaten reicht typischerweise bis in die Business-Prozesse hinein.

Die technischen und in den Prozessen liegenden Herausforderungen lassen sich an Open Source Software gut verdeutlichen. Die gleichen Überlegungen und technischen Voraussetzungen treffen auch auf proprietäre Software zu, müssen dann allerdings mit dem jeweiligen Lieferanten auf Basis eines entsprechenden Wartungsvertrages abgesichert werden. Open Source Software gestattet es unmittelbar, während der Entwicklung die Software konsequent auf diejenigen Komponenten zu reduzieren, die für die Applikation tatsächlich benötigt werden. Ein solches gehärtetes System enthält schlicht weniger „Bauteile", die einem Alterungsprozess unterliegen und gewartet werden müssen. Ebenso gibt es weniger Abhängigkeiten zwischen Komponenten, die es bei Aktualisierungen zu berücksichtigen gilt. Es geht also um Komplexitätsreduktion.

Um vor dem Hintergrund industrieller Anforderungen im Laufe des meist deutlich mehrjährigen Software-Lebenszyklus' einzelne Komponenten aktualisieren zu können oder Security-Patches aufzubringen, sind die validierbare Reproduzierbarkeit der Software und ein verlässliches Versionsmanagement unbedingte Voraussetzungen. Dies wiederum setzt nicht nur gut dokumentierte Entwicklungsprozesse, sondern auch einen standardisierten Bauprozess und daraus abgeleitet ein Release Mangement voraus (siehe Kasten).

Für Open Source-Systeme kann nur dringend empfohlen werden, die benötigten Software-Pakete direkt aus dem Upstream, also aus der Mainline Community zu beziehen und nicht aus einer Distribution oder aus anderen Quellen, die vom Upstream abweichen. In ein Mainline-System können Paket-Updates und beispielsweise Security-Patches, die im Upstream verfügbar werden, direkt und damit sehr schnell übernommen werden.

Doch die aktualisierte Software-Version selbst ist nur der erste Schritt. Ebenso wichtig ist ein entsprechendes Release Management und das eigentliche Update der im Feld befindlichen Geräte. Die Software auf Embedded-Devices ist gegenüber dem Desktop-/Server-Bereich extrem heterogen, oft hochgradig optimiert und teilweise nicht nur produktspezifisch, sondern auch kundenspezifisch angepasst. Ebenso heterogen ist die Einbausituation und Erreichbarkeit. Um der Software Obsolescence im Feld begegnen zu können, sollte der Update-Prozess unbedingt bereits Bestandteil der frühen Design-Phase sein. Die Dringlichkeit hierfür ist in den vergangenen zwei bis vier Jahren noch ein-

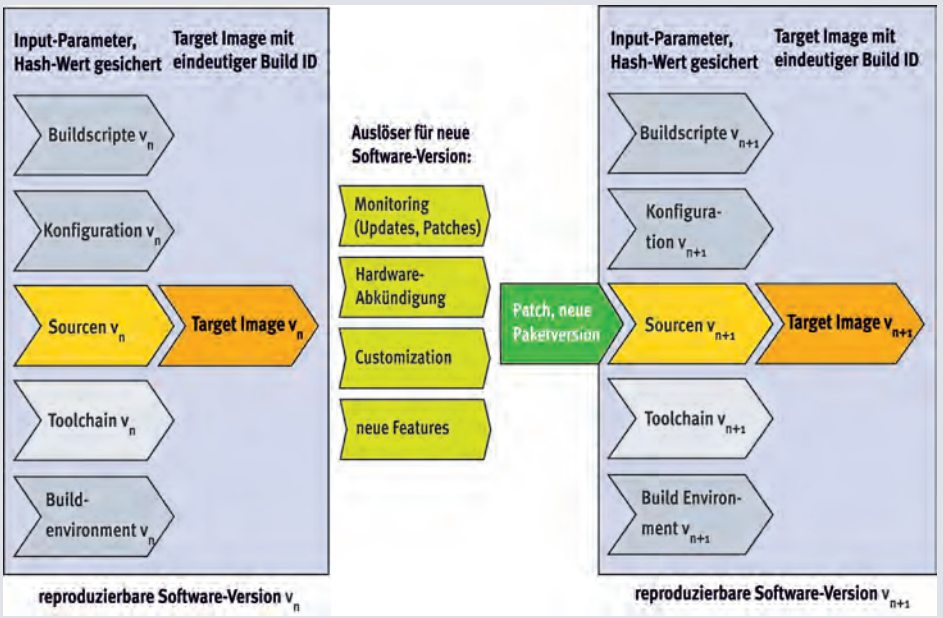
Standardisierte Bauprozesse . . .

. . . gegen Software Obsolescence

Voraussetzung für ein verlässliches Versionsmanagement ist ein reproduzierbarer, Baurechner- und Personen-unabhängiger Bauprozess.

Dass das „Source Code Management"-System, in dem die Software-Komponenten abgelegt werden, auf einem regelmäßig ins Backup gesicherten Server liegen, versteht sich von selbst. Um jedoch eine wartbare Software mit reproduzierbaren Versionsständen zu erhalten, muss auch der Bauprozess standardisiert und reproduzierbar sein.

Neben den Sourcen gehören dazu auch Buildscripte, eine Toolchain, die spezifische Konfiguration sowie eine (virtualisierte) Bauumgebung. Das Target Image, das letztlich auf dem Embedded-Gerät laufen soll, kann nur dann reproduzierbar versioniert werden, wenn alle in den Bauprozess einfließenden Komponenten – Sourcen, Buildscripte, Toolchain, Konfiguration und Buildenvironment – ebenfalls versioniert sind.



Um gegebenenfalls auch Jahre später exakt das gleiche Image auf einem beliebigen Baurechner noch einmal bauen zu können – etwa um ein Paket-Update durchzuführen –, müssen alle genannten Eingangsparameter für die Software-Version versioniert mit abgelegt werden. Das ganze Set wird wieder „ausgepackt", wenn erneut gebaut werden soll. Ändert sich nur einer dieser Parameter, entsteht eine neue Software-Version. (es)

mal deutlich gestiegen – durch „Industrie 4.0" einerseits und die Novelle des IT-Sicherheitsgesetzes andererseits.

Auf der einen Seite wächst der Anteil netzwerkintegrierter Embedded-Geräte sprunghaft. Eine Erreichbarkeit auch für ein Fern-Update ist immer häufiger gegeben. Auf der anderen Seite steigen damit die Security-Anforderungen, also Daten- und Manipulationsschutz, und damit auch die Anforderungen an die Absicherung eines Update-Prozesses. Dies drückt sich bereits sehr konkret in Form von Lieferantenerklärungen aus. In ihnen wird mindestens die Zusicherung verlangt, dass das gelieferte Gerät sicher nach dem Stand der Technik ist und dass es einen Prozess gibt, um diesen Sicherheitsstatus über den Lebenszyklus aufrecht erhalten zu können.

Um später im Feld problemlos Updates durchführen zu können, sollte von Vorneherein ein abgesicherter Update-Prozess implementiert und gründlich getestet werden. Stichworte sind hier die Authentizitäts-Prüfung der Software durch Signatur und/oder Schlüssel, ein Fallback-Mechanismus, aber auch Power Fail Safety sowie ein durchdachtes Rollen- und Berechtigungskonzept. Denn nicht immer soll der Kunde beispielsweise alles dürfen, was ein Service-Techniker darf.

Sobald Infrastruktur und Prozesse für Reproduzierbarkeit, Versionsmanagement und Remote Update verlässlich zur Verfügung stehen, müssen nur noch die „Alterungsfaktoren" frühzeitig erkannt werden. Es muss ein entsprechender Monitoring-Prozess etabliert werden, bei dem regelmäßig Informationen zu neu entdeckten Fehlern einerseits und zur Verfügung stehenden Aktualisierungen (u.a. Security Patches) andererseits eingeholt werden. Dies können entsprechende Meldungen der Kunden zur eigenentwickelten Software sein, die dem Maintenance-Team verfügbar gemacht werden müssen. Bei „Open Source"-Software müssen bestimmte Informationsquellen regelmäßig abgefragt werden, um Aktualisierungen nach einer Relevanzprüfung direkt übernehmen zu können.

Die positive Kehrseite dieser Anforderungen und Aufwände im Kampf gegen Software Obsolescence sind damit einhergehende Chancen. Ein verlässliches Versionsmanagement und abgesicherte Update-Prozesse bieten Investitionssicherheit, weil sie eine schnelle, wirtschaftliche Reaktion auf Veränderungen ermöglichen. Dem Kunden liefern sie höhere Betriebssicherheit und IT-Compliance-Konformität. (es)