JAVA Mag

Java magazin™

Java | Architektur | Software-Innovation

KISS

KEEP IT SIMPLE STUPID

Sonderdruck für www.capgemini.com

Capgemini

Brückenbauer in der Open-Source-Welt – Teil 2

Entwicklungsumgebung automatisieren

Wieviel Zeit braucht ein Entwickler, bis er das Projekt eingerichtet hat und durchstarten kann? Wenn es hierzu eine manuelle Anleitung mit unzähligen Schritten und Screenshots gibt, ist Ärger vorprogrammiert: Das stumpfsinnige Abarbeiten solcher Anleitungen ist erfahrungsgemäß ermüdend für Entwickler. Nach einigen Schritten wird dann doch mal nicht so genau gelesen und schon läuft etwas falsch. Wenn dann am Ende das Ganze nicht funktioniert, ist es meist sehr schwierig und aufwendig, die Fehlerursache zu finden. Und das ist ja erst der Anfang … Was ist, wenn das Projekt über Jahre läuft und die Entwicklungsumgebung kontinuierlich verbessert und weiterentwickelt werden soll?

von Jörg Hohwiller

Das Zauberwort für Probleme dieser Art heißt Automatisierung. Der einzelne Entwickler soll möglichst gar nichts mehr manuell machen. Dann kann nichts mehr schiefgehen und er kann seine Zeit für sinnvollere Dinge nutzen. Um das zu erreichen, gibt es verschiedene Ansätze:

1. Automatisierung von Installation und Konfiguration der Werkzeuge
2. Virtualisierung der Entwicklungsumgebung
3. Verlagerung der Entwicklungsumgebung auf den (Cloud-)Server

Option 3 mag mit Eclipse Che, VS Codespaces etc. langfristig in Unternehmen interessant sein, ist aber (Stand heute) noch nicht vollständig für den produktiven Einsatz ausgereift, auch wenn das Thema Kontrolle von Back-up und Datensicherheit hier für Unternehmen sicher ein attraktiver Nebeneffekt ist. Die Vorstellung, eine vollständig performante, tastaturbedienbare und hoch-effiziente IDE (Integrated Development Environment) im Browser zu haben, ist noch ein Stück Zukunftsmusik.

Dagegen ist Option 2 bereits heute umsetzbar. Eine vollständige Virtualisierung bremst und isoliert die Entwicklungsumgebung jedoch viel zu stark. Deutlich praxistauglicher funktioniert es dagegen mit Tools wie Docker oder Vagrant. Aber auch hier haben wir im Praxistest Schwierigkeiten mit Performance und Isolation gehabt, denn Entwickler möchten aus ihrer Hardware alles rausholen und warten gefühlt ohnehin schon ohne jede Abstraktion zu lange auf Compiler, Linter und Co. Am meisten stört es die Entwickler aber, wenn sie das Gefühl haben, in einer Sandbox eingesperrt zu sein und mit ihren sonstigen Lieblingstools oder OS-Standardfunktionen nicht nahtlos zusammenarbeiten zu können.

Aus diesen Gründen sind wir den Weg von Option 1 gegangen, und so ist devonfw-ide entstanden.

devonfw-ide in a Nutshell

Die Grundidee ist einfach und folgt wie üblich in devonfw dem KISS-Prinzip: Mit ein paar schlauen Skripts automatisieren wir die Installation und Konfiguration der Entwicklungsumgebung vollständig. Wir unterstützen dabei die großen Plattformen Windows, Linux und MacOS. Die Entwicklungsumgebung für ein konkretes Projekt ist nichts weiter als ein Ordner auf dem Ent-

Der Entwickler behält die Kontrolle über seine persönlichen Einstellungen zum Look and Feel und darf auch zusätzliche Plug-ins installieren, die ihm das Leben leichter machen.

wicklerrechner, in dem alles landet, inklusive der Software, Konfigurationen und Code-Repositories. Damit ist alles nach dem Sandbox-Prinzip isoliert, und jeder Entwickler kann unterschiedlichste Projekte parallel als Ordner auf seinem Rechner haben, ohne dass sich verschiedene Versionen von Werkzeugen wie JDK, Maven, Gradle, Node.js usw. gegenseitig ins Gehege kommen. Damit kann die Wartung in einem angestaubten Projekt auch nach Jahren wieder spontan aufgenommen werden, obwohl parallel das Projekt mit Bleeding-Edge-Tools bearbeitet wird. Die Konfiguration eines Projekts erfolgt über ein dediziertes Git Repository, in dem über Konfigurationsdateien alles genau festgelegt werden kann – von den Werkzeugen und ihren Versionen bis zu ihrer Konfiguration, wie z. B. dem Code-Formatter. Die devonfw-ide ist extrem flexibel und unabhängig vom Rest von devonfw. Sie kann daher in jedem Projekt helfen, das gängige Werkzeuge für Java, JavaScript und Co. benötigt.

Sofort einrichten

Wer jetzt denkt, dass sich das vielversprechend anhört, kann direkt loslegen: Einfach auf [1] gehen und unter Download die devonfw-ide herunterladen. Die einzige Vorrausetzung ist eine Git-Installation, die aber jeder Entwickler, der etwas auf sich hält, sicher schon vorliegen hat. Dann kann das Downloadarchiv in einen Projektordner (z. B. *C:\projects\my-project*) ausgepackt werden. Anschließend muss das *setup*-Skript (unter Windows *setup.bat*) gestartet werden. Hier wird man nach einem Settings-URL gefragt, die der URL des zuvor genannten Git Repositories mit der Konfiguration ist. Wer das erstmal schnell ausprobieren möchte, kann einfach Return drücken und mit den Defaulteinstellungen starten. Bei der ersten Einrichtung muss man noch einmalig den Lizenzbedingungen zustimmen, was bei einer Open-Source-Lizenz keine Hürde darstellen sollte. Anschließend wird die Software automatisch aus dem Netz geladen und eingerichtet, was eine stabile Internetverbindung voraussetzt. Dabei poppen einige Fenster auf, die man nicht manuell schließen sollte. Besser lehnt man sich entspannt zurück, trinkt seinen Kaffee und liest einen spannenden Artikel im Java Magazin.

devonfw-ide für Entwickler

Als Entwickler will ich jetzt sofort die Werkzeuge nutzen. Dazu bringt die devonfw-ide das *devon*-Kommando als CLI (Command Line Interface) mit. Hiermit kann ich die Umgebungsvariablen für das aktuelle Projekt (insbesondere *PATH*) direkt in meiner Shell setzen. Mit der Angabe weiterer Parameter kann ich auch direkt meine IDE wie Eclipse oder VS Code starten. Für Windows und MacOS gibt es bereits eine fertige Integration in den Dateimanager des Betriebssystems, sodass man einen Ordner per Rechtsklick direkt in einer Shell öffnen kann und dabei automatisch die Umgebungsvariablen passend zum Projekt gesetzt werden. So ruft *mvn* dann immer Maven in der Version und mit der *settings.xml* passend zu dem Projekt auf, in dem ich mich befinde. Für Mausschubser erstellt das Set-up (siehe oben) auch ein *eclipse-main[.bat]*-Skript, mit dem man Eclipse per Doppelklick öffnen kann.

Der eigentliche Code im Projekt wird in einem Workspace abgelegt. Das ist ein Ordner unterhalb des Ordners *workspaces* innerhalb der Installation der Entwicklungsumgebung. Standardmäßig gibt es immer den Workspace *main*. Hier klone ich meine Code-Repositories und hier liegen auch die Konfigurationen meiner IDE wie z. B. Eclipse (im *.metadata*-Ordner). Wer möchte, kann aber z. B. für weitere Teilprojekte oder Releasezweige eigene Workspaces hinzufügen, indem er einen beliebigen neuen Unterordner im Ordner *workspaces* anlegt. Befinde ich mich innerhalb dieses Workspace und rufe z. B. *devon eclipse* auf, so wird der neue Workspace konfiguriert und Eclipse darin gestartet. So kann ich innerhalb des gleichen Projekts mehrere IDE-Instanzen für unterschiedliche Workspaces als Mandanten parallel öffnen. Bestimmte Einstellungen, wie z. B. der Code-Formatter oder wichtige Compiler-Einstellungen, werden dabei jedoch von devonfw-ide automatisch gemanagt und über alle Workspaces konsistent nach den Projektvorgaben aus dem Settings Git Repository konfiguriert. Wenn sich etwas geändert hat, werden die Entwickler informiert und rufen einfach den Befehl *devon ide update*; so bleiben alle sind auf dem neuesten Stand, egal ob nun Eclipse auf eine neue Version gehoben wurde, ein neues Plug-in dazu kam oder eine spezielle Konfigurationseinstellung geändert wurde.

Der Entwickler behält dennoch die Kontrolle über seine persönlichen Einstellungen zum Look and Feel (z. B. Dark Theme, Perspectives, Schriftgrößen usw.) und darf auch – sofern nicht vom Projekt untersagt – zusätzliche Plug-ins installieren, die ihm persönlich das Leben leichter machen, aber vom Projekt vielleicht als Geschmackssache abgetan werden. Kennt er jedoch ein Werkzeug, ein Plug-in oder einen Kniff in den Einstellungen, der für

alle Entwickler von Vorteil ist, so kann er einen Feature Branch im Settings Git Repository erstellen (und daraus einen Pull-/Merge-Request).

devonfw-ide für Admins

Früher gab es im Projekt den Architekten oder den Chef, der alles vorgegeben hat. Auch eine homogene und konsistente Entwicklungsumgebung braucht etwas Ordnung, funktioniert aber wunderbar in agilen Projekten, wie bereits mit dem Feature Branch angedeutet wurde. Letztlich wird im Settings Git Repository festgelegt, ob z. B. Eclipse als Tool im Projekt installiert sein soll und wenn ja, in welcher Version und mit welcher Konfiguration. Auch können Git Repositories durch einfache Konfiguration beim Set-up automatisch geklont und in die IDE importiert werden. Hierzu nutzt devonfw-ide den Standard-Branch des Settings Git Repositories und konfiguriert alles entsprechend der dortigen Einstellungen. Somit kann jeder im Projekt einen Feature Branch erzeugen, um eine Verbesserung der Entwicklungsumgebung vorzuschlagen. Das kann getestet werden und nach einem zum Projekt passenden Prozess dann in die Entwicklungslinie fließen oder auch abgelehnt werden. Je größer das Team, desto wichtiger ist die Qualitätssicherung. Denn wenn alle Entwickler durch einen Fehler arbeitsunfähig werden, wird es schnell teuer. Mit *git revert* ist bei devonfw-ide immerhin auch dann schnelle Abhilfe möglich, wenn die QS versagt haben sollte.

Die grundlegenden Einstellungen sind in der Datei *devon.properties* im Wurzelverzeichnis des Settings Git Repositories zu finden. Wer in den Tiefen der Eclipse Preferences tunen will, muss sich etwas genauer mit devonfw-ide auseinandersetzen. Für den besagten Fall ist die einfachste Möglichkeit, die gewünschte Einstellung direkt in Eclipse über die PREFERENCES GUI vorzunehmen, Eclipse zu schließen und mit *devon eclipse ws-reverse* automatisiert in das Settings Git Repository zu übernehmen, wo man die Änderungen mit üblichen Git-Diff-Tools nochmal überprüfen kann. Für weitere Details bietet devonfw-ide eine umfangreiche Dokumentation.

Fazit und Ausblick

Jeder Entwickler braucht gute Tools und die müssen richtig konfiguriert und integriert werden. Mit devonfw-ide ist diese Herausforderung bestens zu meistern und neue Entwickler sind in Kürze arbeitsfähig. Aktuell arbeiten wir am devonfw-Dashboard als optionales GUI Frontend für devonfw-ide, das automatisch über Updates informieren und den Einstieg gerade für Mausschubser nochmals vereinfachen wird.

Jörg Hohwiller ist seit 2002 für Capgemini als Architect und Berater tätig. Privat entwickelt er aktiv an vielen Open-Source-Projekten mit. Sein Ziel ist es, gemeinsam mit vielen anderen erfahrenen Entwicklern die IT kontinuierlich besser zu machen.

Links & Literatur

[1] https://devonfw.com