Zehn Must-have-Java-Bibliotheken S. 10

Mit Galleon WildFly provisionieren S. 56

Chaos Engineering bei der DB S. 74

JAVA Mag

Java magazin™

Java | Architektur | Software-Innovation

MACHINE LEARNING IN JAVA

www.javamagazin.de

Ausgabe 3.2021

Brückenbauer in der Open-Source-Welt – Teil 4

Architektur validieren

Schulung durch Reviews ist völlig ineffizient, doch das optimale Expertenteam hat man auch nicht immer zur Hand. Fallen grundsätzliche Mängel am Design des Codes erst in der Review auf, ist das Kind schon in den Brunnen gefallen: Der Code muss gegebenenfalls aufwendig refaktoriert werden und der gleiche Fehler wurde in der Zwischenzeit bereits mehrfach wiederholt.

von Jörg Hohwiller

Schlimmstenfalls ist das Antipattern bereits zur Vorlage für viele andere Entwickler im Team geworden. Wenn dann die Experten im Team zum Flaschenhals werden und mit den Reviews nicht nachkommen, erodiert die Architektur und es geht bergab mit der Codequalität. Wie schafft man hier Abhilfe?

Was man braucht, ist eine sofortige Codereview, doch von einer vollständigen Instantreview ist die KI noch meilenweit entfernt. Daher muss man zwischen den einfachen (Anti-)Patterns und dem manuellen Review durch einen menschlichen Experten trennen. Einfache Patterns kann man automatisch erkennen und damit kann das Feedback sehr schnell zum Entwickler gelangen – im besten Fall bereits in der IDE während der Programmierung. Die Architektur und Programmierrichtlinien in devonfw sind sehr genau definiert [1] und daher perfekt für eine automatische Validierung. Hierzu gibt es bereits ausgereifte Werkzeuge, die lediglich konfiguriert werden müssen. Jedoch sind diese fast alle kommerziell, und wenn in devonfw fast alles standardisiert ist, warum soll man dann überhaupt noch etwas konfigurieren müssen?

sonar-devon4j-plugin

Bei statischer Codeanalyse und Open Source sind wir schnell bei SonarQube, das sich in diesem Bereich etabliert hat. Daher haben wir uns auch bei devonfw entschieden, das sonar-devon4j-plugin [2] zu entwickeln, mit dem SonarQube zusätzlich die Architektur- und Programmierrichtlinien automatisch überprüfen kann. Das Plug-in ist im SonarQube Marketplace verfügbar, womit Installation und Updates mit einem Klick als Admin im Web-UI erledigt sind.

Die Abbildung der Architektur auf die Java-Package-Struktur erlaubt es, einfach festzustellen, zu welcher Komponente, welchem Layer und welchem Scope sie gehört. Somit müssen wir nur noch für jede referenzierte Klasse die Eigenschaften vergleichen, um ungewollte Abhängigkeiten aufzudecken. Jedem sollte klar sein, dass man von der Persistenz (z. B. aus einem DAO oder Repository) nicht auf einen Anwendungsfall zugreift, sondern nur umgekehrt. Ein REST Service sollte einen Anwendungsfall aufrufen und nicht daran vorbei direkt auf die Persistenz zugreifen. Genau diese Dinge können automatisch überprüft werden. **Abbildung 1** zeigt die Standardarchitektur von devonfw. Die Pfeile zeigen mögliche Abhängigkeiten zwischen Komponenten und Layern. Erwünschte Abhängigkeiten sind grün, unerwünschte gelb und verbotene rot dargestellt.

Die technische Architektur ist standardisiert, die fachliche Architektur hingegen unterscheidet sich in jedem Projekt. Da die Komponentennamen zur Begriffswelt des Fachbereichs passen sollen, ist hier eine Standardisierung nicht möglich. Damit das Plug-in weiß, welche Komponenten es gibt, erstellt das Projekt eine einfache *architecture.json*-Datei, in der die Komponenten und ihre erlaubten Abhängigkeiten definiert werden, und legt sie mit dem Code in Git ab. Darüber hinaus haben wir aus manuellen Codereviews weitere Patterns gesammelt und als Regeln für SonarQube implementiert.

Da devonfw kein „Closed Shop“, sondern möglichst flexibel sein will, haben wir für jede unerwünschte Ab-

Artikelserie

Teil 1: Besser entwickeln mit devonfw
Teil 2: Bessere Entwicklungsumgebung mit devonfw-ide
Teil 3: Inkrementelle Codegenerierung mit CobiGen
Teil 4: Architektur validieren

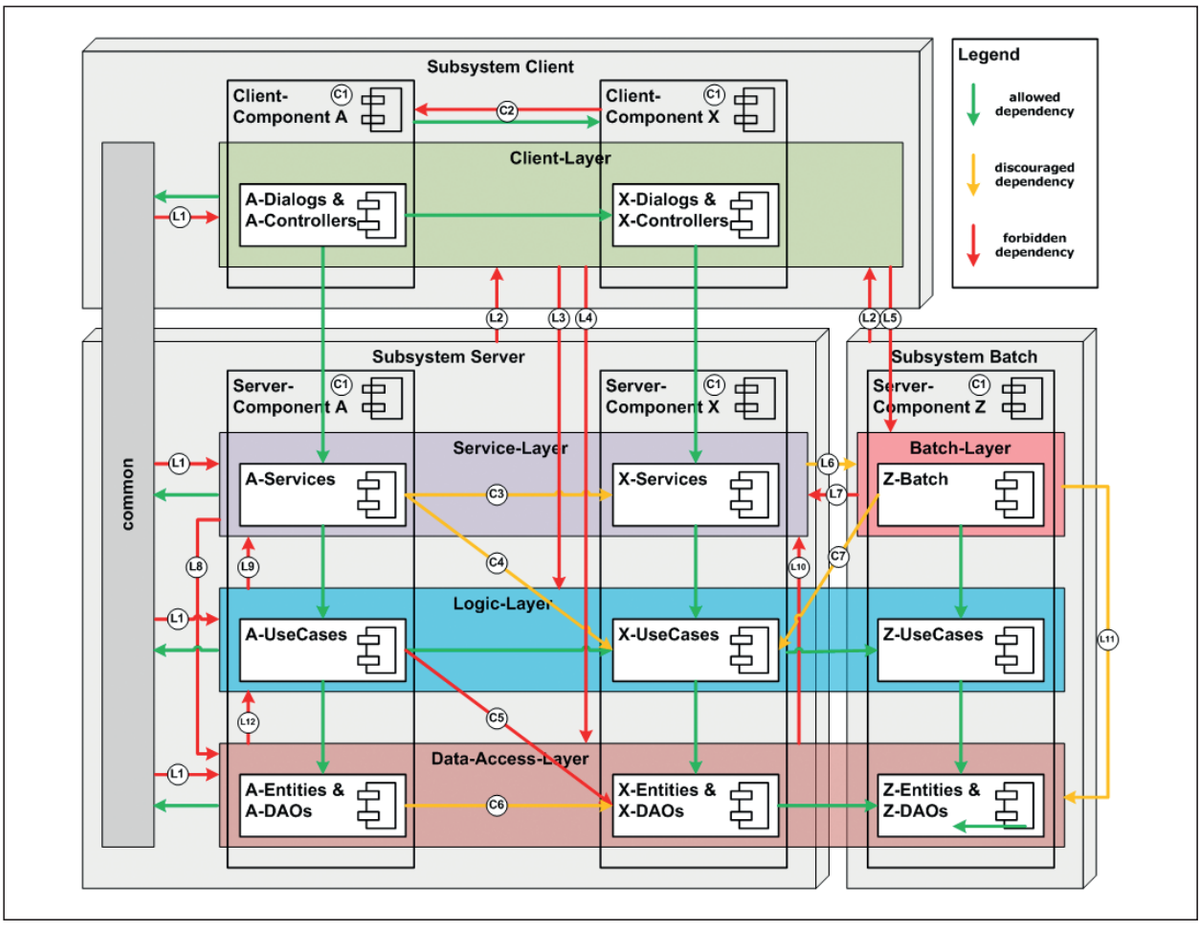


Abb. 1: Die Standardarchitektur von devonfw

hängigkeit eine eigene Regel implementiert – in **Abbildung 1** sind die Abhängigkeitspfeile in den kleinen Kreisen mit den IDs der Regeln beschriftet. Dieser Ansatz erlaubt es Projekten, jedes Detail anzupassen, in dem die Severity geändert oder auch die ganze Regel abgeschaltet werden kann. Das ist nicht als Freifahrtschein gemeint, aber wir wissen, dass es immer unterschiedliche Meinungen gibt, ob etwas ein Blocker ist, und vielleicht auch, ob ein Batch direkt auf die Persistenz (Data Access Layer) zugreifen darf oder immer über die Geschäftslogik (Logic Layer) gehen muss. Wir geben alle Defaults vor und das Plug-in richtet automatisch das entsprechende Quality Profile ein. Aber wer möchte, kann das auf seine Bedürfnisse zuschneiden.

Fazit und Ausblick

Statische Codeanalyse hat mit Checkstyle angefangen zu prüfen, ob Leerzeichen und geschweifte Klammern richtig platziert sind. SonarQube hat das auf eine völlig neue Dimension gehoben und findet höherwertigere und komplexere Bugpatterns bis zur SQL-Injection-Lücke. Mit dem sonar-devon4j-plugin kommt die nächste Dimension der Architekturvalidierung hinzu. Durch die etablierte Infrastruktur von SonarQube kann sie einfach in die bestehende Werkzeugkette integriert und mit SonarLint bis in die Entwicklungsumgebung integriert werden. Damit bekommt der Entwickler die Fehler bereits bei der Entwicklung zu sehen. Reviewer werden massiv entlastet und können sich auf das Wesentliche konzentrieren.

Aktuell erweitern wir das sonar-devon4j-plugin um eine flexible Konfigurierbarkeit des Architektur-Mappings. Damit kann man in der *architecture.json* auch eine Regular Expression für das Package-Schema definieren und per Regex Groups Elemente wie Komponente, Layer und Scope nach den eigenen Wünschen mappen. Damit wird das Plug-in auch für Projekte nutzbar, die eine eigene Architekturkonvention etabliert haben. Für devonfw-Projekte kann man sich diese Konfiguration sparen und bekommt alles out of the box.

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.

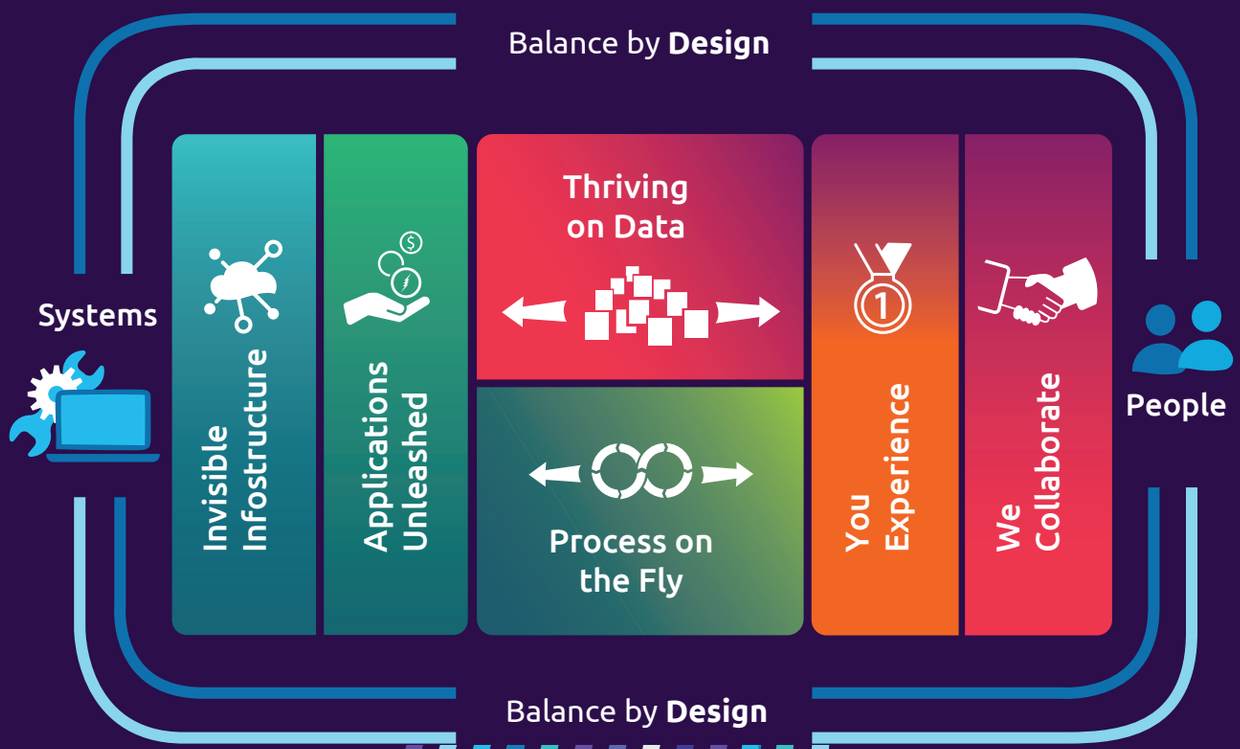
Links & Literatur

[1] https://github.com/devonfw/devon4j/blob/develop/documentation/coding-conventions.asciidoc

[2] https://github.com/devonfw/sonar-devon4j-plugin/

TECHNO VISION | Change Making 2021

BE LIKE WATER



Download the Report

Capgemini