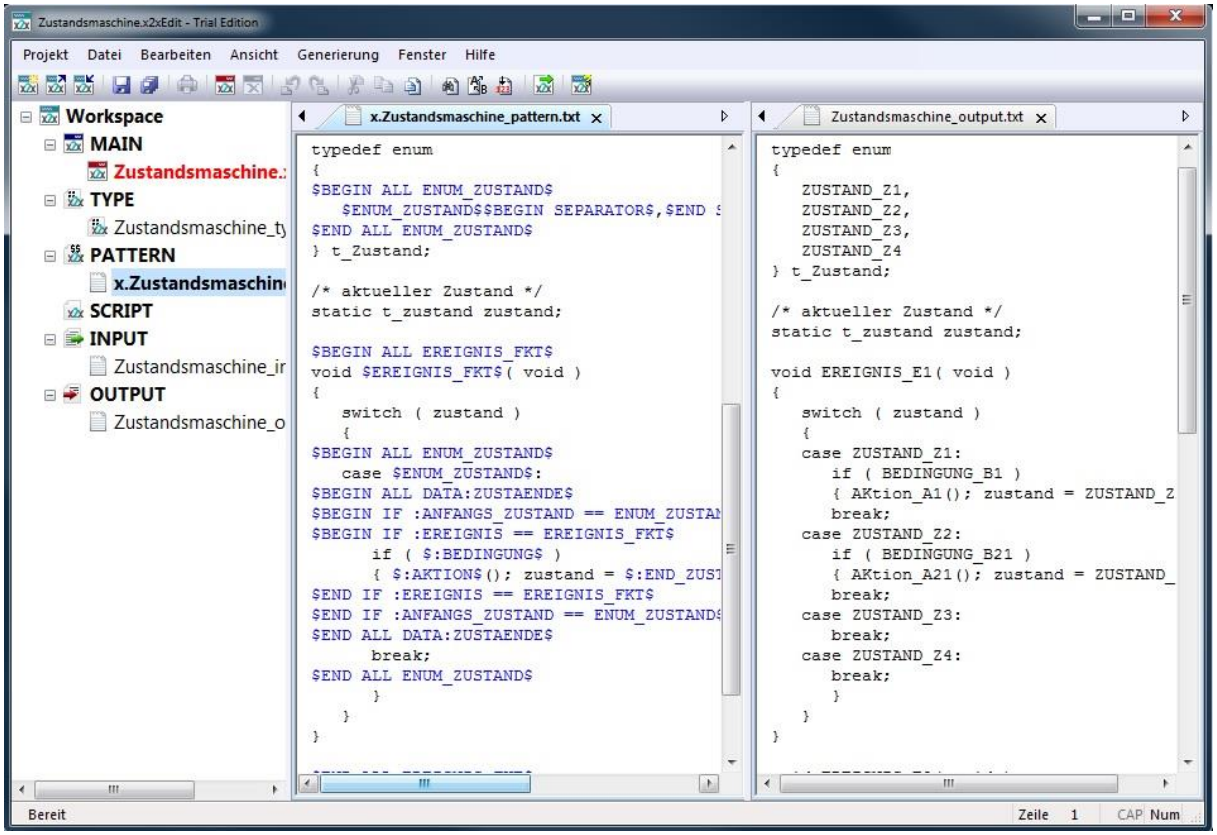
Vom Sourcecode zum Sourcecode-Generator

Auf dem Weg einen Sourcecode-Generator zu schreiben, gilt es einige Hindernisse zu überwinden. Gemeint sind hier nicht Hindernisse die daraus resultieren, dass ein Sourcecode-Generator programmiert werden soll. Diese Zeiten sind Gottseidank vorbei. Ein Sourcecode-Generator wird nicht mehr programmiert, sondern konfiguriert, wobei das westliche Konfigurationsmerkmal die Mustervorlage für den zu generierenden Sourcecode ist. Wer aber glaubt man müsse nur bestehenden Beispielsourcecode als Vorlage nehmen und schon läuft die Sache - der irrt!

Natürlich geht man beim Erstellen der Sourcecode-Mustervorlage am besten immer von bestehendem Beispielcode aus. Der Haken an der Geschichte ist jedoch, dass dieser Beispielcode eben noch nicht durch einen Generator erzeugt wurde und deshalb in der Regel wenig bis keinerlei Systematik aufweisen wird. Das beginnt bei der Zusammensetzung von Variablennamen mit Groß-/Kleinbuchstaben und Unterstrich '_' und endet bei der Frage, ob zum Einrücken von Text Tabulatoren oder Blanks verwendet werden. Wo sollen Leerzeilen sein, welche Art von Kommentaranweisungen (z.B. in C/C++ /*...*/ oder //) soll verwendet werden, wie hat der Header der Datei auszusehen? Glücklicher Weise gibt es in fast allen Softwareentwicklungsabteilungen sogenannte "Coding-Guidelines" in denen all diese Dinge festgelegt sind.



Gegenüberstellung eines Source-Musters zum generierten bzw. Originalcode.

Für das Erstellen eines Codegenerators sind Coding-Guidelines eine wunderbare Sache, denn "Generatoren lieben Regeln". Wie ernsthaft die Coding-Guidelines von den Entwicklern eingehalten wurden, zeigt sich spätestens, wenn wir den Beispielcode mit dem Generator zu erstellen und mit dem händisch implementierten Originalcode vergleichen. (Mit einem geeigneten Merge-Tool kann man die beiden Dateien am Bildschirm direkt gegenüberstellen, wobei Unterschiede farblich hervorgehoben werden.)

Im Idealfall haben die Entwickler die Coding-Guidelines vollständig eingehalten und es werden nur unwesentliche Differenzen sichtbar, die nur die Anordnung des Sourcecodes betreffen, wie zum Beispiel Texteinrückungen. In diesem Fall kann der generierte Code unverändert übernommen werden. Ansonsten muss entweder das Muster für den Generator nachgebessert oder der Originalcode so korrigiert werden, dass er dem generierten Code entspricht. D.h. angepasster Generator und veränderter Originalcode nähern sich in kontrollierten Schritten einander an, bis sie hinreichend übereinstimmen.

Erfahrungen aus der Praxis haben gezeigt, dass in der Regel auf Anhieb ein Großteil (z.B. 80%) des genierten Code direkt mit dem händisch erstellten Code übereinstimmt. Wo keine Übereinstimmung herrscht, liegen häufig Vertauschungen in der Reihenfolge bestimmter Code-Abschnitte vor, die keinen Einfluss auf die Funktion des Codes haben, wenn sie dem generierten Code angepasst werden. (Ein typisches Beispiel hierzu wäre, wenn im Code Enumerationen in eine bestimmten Reihenfolge definiert werden und der Generator genau diese Reihenfolge für ein Switch-Case-Anweisung verwendet.)

Schließlich gibt es noch die berühmte "Ausnahme von der Regel". D.h. der Generator generiert den Code nach einer Regel die fast immer das passende Ergebnis liefert -aber eben nur "fast immer". Genau in dieser Situation muss dann auch das Sprichwort "Die Ausnahme bestätigt die Regel" zitiert werden. Häufig zeigen diese Ausnahmen einen tieferen Einblick in das Verständnis des implementieren Algorithmus. Dies kann im eher seltenen Fall zum Auffinden und Beheben eines tatsächlich vorliegenden Implementierungsfehlers führen, oder was häufiger der Fall ist, aufzeigen dass der Generator den Code nach einer allgemeineren Regel generieren muss, welches als positiven Nebeneffekt, das Gesamtsystem flexibler gestaltet.

Ob sich der generierte Code vom Originalcode nur "unwesentlich" unterscheidet kann durch einen sogenannten Hex-Vergleich ermittelt werden. Dazu wird das Gesamtsystem einmal mit dem Originalcode und parallel dazu einmal mit dem generierten Code compiliert und dann beide Binärcode-Dateien in Hexadezimaldarstellung verglichen. Gibt es dabei nur nicht relevante Unterschiede wie z.B. Compilierungsdatum und -uhrzeit oder Build-Nummer, so kann man absolut sicher sein, dass der generierte Code dem Originalcode entspricht, auch wenn gröbere Anpassungen wie z.B. das Ändern von Variablennamen vorgenommen werden mussten.

Zusammenfassend kann festgehalten werden, dass das Erstellen eines Sourcecode-Generators in iterativen Schritten vor sich geht, wobei sich generierter Code und händisch geschriebener Code bis zur Übereinstimmung annähern. Dabei wird im wahrsten Sinne des Wortes "Ordnung" hergestellt, denn generierter Code folgt naturgemäß einer durch Regeln begründeten Ordnung. In der Praxis impliziert das Herstellen dieser Ordnung den Hauptaufwand, wenn ein Sourcecode-Generator erstellt

wird. Dieser Aufwand ist aber eigentlich unabdingbar, denn wenn die Qualität der Software stimmen soll, dann muss Ordnung vorausgesetzt werden. Andererseits lohnt sich dieser sowiesoo zu leistende Aufwand noch weit mehr, wenn der selbe Generator für mehr als nur ein Modul eingesetzt werden kann, denn dann werden sozusagen "mehrere Fliegen mit einer Klappe geschlagen". (Man denke z.B. an einen Generator für mehrere Zustandsmaschinen im selben Softwareprojekt.)

Und dann bleibt noch das letztendlich absolut schlagende Argument:" Einen Generator schreibt man ja nur deshalb, weil man den Sourcecode für veränderten Input auf Knopfdruck immer neu generieren kann; und zwar "just-in-time". Das heißt, einmal erstellte Ordnung bleibt für immer erhalten. Besser geht's doch nicht! Oder?

Weiterführende Dokumente:

http://www.sss.de/x2x-downloads?file=files/triple-s/downloads/downloads_X2X_2015/Sourcecode_generieren.pdf

http://www.sss.de/x2x-downloads?file=files/triple-s/downloads/downloads_X2X_2015/Modellbasierte%20Softwareentwicklung.pdf

http://www.sss.de/x2x-downloads?file=files/triple-s/downloads/downloads_X2X_2015/Funktionale_Sicherheit.pdf

http://x2x.sss.de