Perlen züchten

Für die Programmiersprache Perl existieren ausgefeilte Plugins sowohl für die gängigen Editoren als auch für ausgewachsene Entwicklungsumgebungen. Ob der Einsatz solcher Plugins oder spezieller Anwendungen für die Programmierung mit Perl besonders hilfreich ist, wird hier nachgeprüft. Zum Einsatz gelangen der Editor Vim, die Perl-Entwicklungsumgebung Padre sowie das Eclipse-Plugin EPIC. Harald Jele

Die Programmiersprache Perl zählt wohl nicht zu jenen, die in den letzten Jahren besonders viel von sich Aufhebens gemacht hat. Trotzdem darf nicht vergessen werden, dass Perl seit Jahrzehnten sehr beliebt, weit verbreitet und in so manchen Anwendungsgebieten einfach nicht wegzudenken ist.
Perl findet sich sowohl in so manchem (komplexen) Einzeiler als auch in umfangreichen Programmen wieder. Und kaum eine Frage, die sich der geübte Perl-Programmierer stellt, ist im CPAN [1] unbeantwortet geblieben.
Der typische Perl-Programmierer scheint einer zu sein, der mit den Jahren seinen Lieblingseditor gefunden hat und damit im Wesentlichen sein Auslangen findet. Wer sich in einem Editor wie dem Vim oder Emacs zurecht findet wie der geübte Hausmann in den Schubladen seiner Küche, vermisst beim Programmieren in Perl wohl selten eine umfangreichere Unterstützung zu dem was er gerade tut.
Daher stellt sich die Frage, ob es denn sinnvoll sein kann, bei der Bewältigung umfangreicherer Projekte, jene Hilfe anzunehmen, die ausgewachsene Entwicklungsumgebungen (IDEs) üblicherweise mitbringen – oder ob es vielleicht doch besser ist, bei jener Arbeitsweise zu bleiben, die einem mit den Jahren vertraut und lieb geworden ist.
Die Beantwortung dieser Frage erfolgt hier in drei Schritten:

- in einem ersten wird eines der beliebten Perl-Plugins „Perl-Support“ für den Editor Vim vorgestellt, das diesen in seinen Funktionen entsprechend erweitert,
- anschließend wird die Anwendung Padre besprochen, die maßgeschneidert Perl-Programmierer bei ihrer Arbeit unterstützen will
- und zu guter Letzt wird das Perl-Plugin EPIC für Eclipse vorgeführt, dessen Erweiterungen den Programmierer Perl-typisch unterstützen soll, wenn er mit dieser sehr ausgereiften IDE zu arbeiten versteht und dies auch im Fall von Perl so belassen möchte.

Alle drei Tools sind Open-Source. Ihr Einsatz ist zudem sowohl unter Linux, als auch unter Mac OS und Windows möglich.
Die Basis für das hier vorgestellte Vorgehen bietet ein frisch installiertes Ubuntu 14.04 LTS.

Das Vim-Plugin für Perl

Ubuntu setzt Vim nicht als einen seiner Standard-Editoren ein und installiert diesen auch nicht automatisch mit. Etwas komfortabler als die konsolenbasierte Ausgabe dieses Editors ist, gerade für wenig Vim-Affine, jene mit graphischer Oberfläche, die im Weiteren hier verwendet wird. Eine Suche danach in den Paketquellen mit

```
apt-cache search gvim
```

bringt mehrere Installationskandidaten ans Tageslicht, wobei sich diese ausschließlich durch die spezifische Ausprägung der Symbole an der Oberfläche unterscheiden.
Ein

```
sudo apt-get install vim-gnome libperl-critic-perl
```

installiert schließlich die Ausgabe mit den für Ubuntu-Anwender wohlbekannten GNOME-Symbolen sowie das Perl-Modul „Critic“, das immer dann zur Anwendung kommt, wenn der Programmierer seinen Perl-Code bezogen auf die Einhaltung der zu verwendenden Syntax überprüfen will.
Das Perl-Plugin für den Vim, das hier zum Einsatz kommt, heißt „Perl-Support“ [2] und wird vom Entwickler Fritz Mehner seit 2003 regelmäßig gepflegt und erweitert. Der Download der entsprechenden Datei `perl-support.zip` gelingt über die leicht auffindbare Web-Seite

```
http://vim.sourceforge.net/scripts/script.php?
script_id=556
```

Das Entpacken der Datei geschieht am besten gleich im Verzeichnis `$HOME/.vim`. Mit

```
unzip perl-support.zip
```

wird die passende Verzeichnisstruktur angelegt. Beim Start des Editors über das Kommando `gvim` wird diesem im Menü ein Eintrag „Perl“ hinzugefügt, über den ab sofort die Funktionen des Plugins angezeigt werden. Der Menüeintrag ist aber nur dann zugänglich, wenn GVim erkennt, dass der Programmierer eine Perl-Datei bearbeiten oder erstellen möchte. Im ersten Fall erkennt GVim dies am Filetype der geöffneten Datei im zweiten muss dieser auf der Kommandozeile von GVim durch

```
:set filetype=perl
```

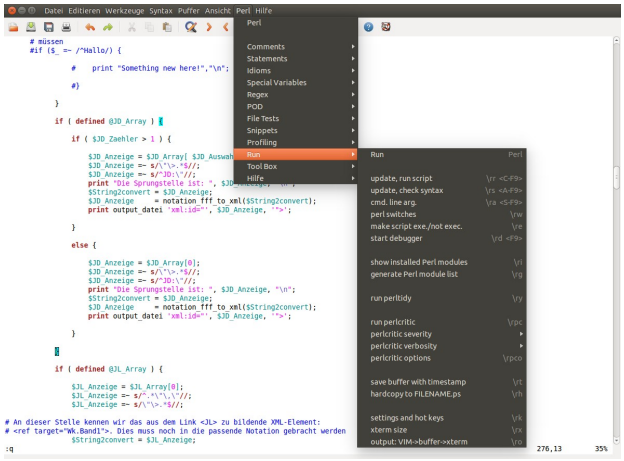

Abbildung 1: Der geöffnete Menüeintrag Perl im Editor.

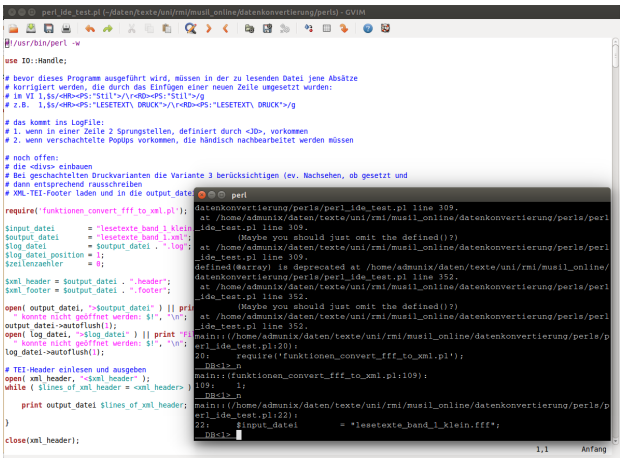

Abbildung 2: Die Textausgabe des gestarteten Debuggers.

gesetzt werden.
Abbildung 1 zeigt einen Ausschnitt der Erweiterungen, die das Plugin dem Editor hinzugefügt hat.
Die ersten Menüeinträge liefern jeweils leere Gerüste dafür, wie die unterschiedlichen Befehle syntaktisch korrekt in Perl umgesetzt werden. Zusammengefasst sind diese in die Gruppen Kommentare (Comments), Deklarationen (Statements) und Ausdrücke (Idioms). Da diese Einteilung zum Teil doch deutlich von der in der Literatur üblichen Gruppierung abweicht, mag sie zu Beginn ein wenig überraschend wirken.
Insgesamt betrachtet ergibt sich mit der weiteren Einteilung jedoch durchaus ein sinnvolle Struktur über die der Programmierer effizient zum Ziel, also letztlich rasch zum passenden Befehl, kommt.
Wählt man einen dieser Einträge aus, so wird das vorbereitete, aber noch leere Code-Gerüst in die geöffnete Datei übernommen. Der geübte Programmierer erspart sich dadurch einiges an Schreibarbeit. Weniger geübte erhalten dadurch einen raschen Einblick in die Syntax von Perl.
Die anschließende Einteilung orientiert sich wesentlich stärker an dem was regelmäßig gebraucht wird: Reservierte Variablennamen, Reguläre Ausdrücke, die POD (Auszeichnungssprache zur Dokumentation) und diverse Möglichkeiten, die Eigenschaften von Dateien zu überprüfen.
In den einzelnen Funktionen zum Menüpunkt „Snippets“ können Abschnitte aus vorhandenem Perl-Code als „Schnipsel“ verwaltet und weiterverwendet werden.
Der nachfolgende Eintrag zum „Profiling“ gestattet einem, die Effizienz seines Codes zu überprüfen. Dafür sind allerdings zusätzliche Perl-Module notwendig, die nicht als fertige Ubuntu-Pakete vorliegen. Am einfachsten werden diese mit dem Tool CPANMinus aus den Paketquellen und folgenden Befehlen direkt aus dem CPAN installiert:

```
sudo apt-get install cpanminus
sudo cpanm Devel::SmallProf
sudo cpanm Devel::FastProf
sudo cpanm Devel::NYTProf
```

Dabei werden auch alle Abhängigkeiten berücksichtigt und die weiteren, notwendigen Module mit installiert.
Der Abschnitt „Run“ schließlich dient dem Ausführen eines Perl-Programms. Zudem können einem Programm zum Starten auch gleich diverse Parameter mit übergeben werden.
In diesem Abschnitt sind des Weiteren der Zugriff auf den Perl-Debugger sowie auf die Tools „Perl-Tidy“ und „Perl-Critic“ verankert. Mit dem Debugger muss man an dieser Stelle bereits vertraut sein, da das GVim-Plugin keine weiteren Hilfestellungen zu dessen Bedienung anbietet. „Perl-Tidy“ soll zu einer ordentlichen Formatierung des geschriebenen Codes führen (was nicht immer gelingen will), „Perl-Critic“ warnt vor schlampiger Syntax, die besser vermieden wird, um nicht in die eine oder andere Stolperfalle zu tappen, die zu einem späteren Zeitpunkt dann mühsam gefunden werden muss.
Sämtliche Ausgaben eines gestarteten Programms sowie der hier verankerten Tools (der Debugger ausgenommen) werden standardmäßig in einen neuen Frame des GVim geschrieben. Wer die Ausgaben lieber in einem Terminal mitlesen mag, der kann dies über den letzten Eintrag „output: Vim → buffer → xterm“ entsprechend konfigurieren.

Fazit zum Vim-Plugin

1. Der allgemeine Mehrwert des Plugins gegenüber dem bloßen Editor ist, dass dieses in einer eigenen Struktur Perl-Spezifisches dem Editor hinzufügt, ohne diesen selbst zu verändern. Für den Einsteiger in die Perl-Programmierung mag dies hilfreich erscheinen, um sich in den Möglichkeiten rascher zurecht zu finden, die diese Programmiersprache bietet. Dem Perl-Experten spart die eine oder andere Funktion regelmäßig Tipparbeit. Ob sich allerdings ein geübter Perl-Programmierer, der bislang mit dem Vim allein zurecht gekommen ist, nicht ohnehin bereits die eine oder andere Krücke zurechtgelegt hat und deshalb das Plugin

nicht oder nur wenig nutzt, sei dahingestellt bzw. kann hier nicht beantwortet werden.
2. Da sich Vim als Editor gut in typische Arbeitsumgebungen integrieren lässt, innerhalb derer mehrere Personen zusammenarbeiten müssen, kann dieser Ansatz, das Plugin zu verwenden, ein durchaus sinnvoller sein. Einen diesbezüglichen Mehrwert (zur Steigerung kollaborativen Arbeitens) liefert dieses aber nicht.
3. Das Plugin ist schlicht und zurückhaltend gestaltet und damit sehr übersichtlich, sodass man sich rasch zurecht findet.
4. Wer in umfangreichen Projekten mitarbeiten muss schätzt den Umstand, dass Programmiercode möglichst übersichtlich dargestellt wird und dieser einem rasch zugänglich wird. Üblicherweise hilft dabei eine ausgefeilte Navigation, über die die eingebundenen Routinen, die gesetzten Variablen, definierte Funktionen, die Schachtelung von Objekte und sämtliche Kontrollstrukturen rasch angesprungen werden können. Diesbezüglich bringt das Plugin keine funktionalen Erweiterungen mit. Auch auf eine automatische Vervollständigung des Perl-Codes, die in modernen IDEs als selbstverständlich vorausgesetzt wird, muss man verzichten.
5. Der Perl-Debugger kann zwar aus dem Editor heraus gestartet werden, die Arbeit mit diesem erfolgt jedoch auf dem herkömmlichen Weg innerhalb eines Terminals. Eine bessere Integration des Debuggers in den Editor liefert das Plugin nicht.
6. Der Code zum Plugin wird laufend gepflegt. Es gibt Kontaktmöglichkeiten zum Entwickler und zu anderen Benutzern über eine Mailingliste [3]. Fragen, die man dort stellt, werden üblicherweise rasch und kompetent beantwortet.
7. Die Dokumentation des Plugins ist umfangreich und gut verständlich.
8. Die Performance entspricht jener des Editors und lässt somit kaum Wünsche offen.

Nachsatz

Bevor dem Autor des Plugins an dieser Stelle zu unrecht vorgeworfen wird, dass dieses viele Funktionen einer modernen IDE nicht abdeckt, muss darauf hingewiesen werden, dass für den Vim unzählig viele Plugins existieren, die noch einiges nachzurüsten in der Lage sind. Allein eine Suche nach „vim plugins“ oder „vim ide“ in der Lieblingssuchmaschine bringt Hinweise auf solche mit Code-Vervollständigung, Code-Browsing, Versionierung und noch einiges mehr. Dass diese sehr spezifischen Funktionen im Perl-Support-Plugin nicht mitgeliefert werden ist daher nur allzu verständlich. Warum sollten Funktionen neu erfunden werden, die es weiland schon in sehr ausgereifter Form gibt?

Padre: eine Perl-IDE

Mit Padre [4] betritt eine Anwendung die Bühne, die selbst in Perl geschrieben ist und ganz gezielt Perl-Programmierer unterstützen und Einsteiger in die Programmiersprache einführen will. Dieser Anspruch ist ein ganz spezifischer und wohl auch einmaliger, da damit ganz gezielt Perl-Programmierer angesprochen werden.
Besonders interessant – und das sei vorweg gleich genannt – ist, dass mit Padre im Kern das Ziel verfolgt wird, nicht nur unterstützend bei der Kodierung zu wirken, sondern zudem dabei, komplexen Code zu restrukturieren (Stichwort „Refactoring“), sodass dieser auf Dauer besser lesbar, wartbar und weniger fehleranfällig ist. Ein weites Anwendungsgebiet, in dem Refactoring mit den Jahren zunehmend interessant werden kann, eröffnet sich dann, wenn Perl-Programmierer ihren Code von Perl5 auf Perl6 migrieren. Durch die in den beiden Versionen mitunter sehr verschiedenen Syntax können automatisierte Tools dem Programmierer hier einiges an mühsamer und fehleranfälliger Arbeit abnehmen.
In den Ubuntu-Paketquellen liegt die Version 1.0, die in einem Terminal mit

```
sudo apt-get install padre
```

installiert wird.
Nachdem das Programm installiert und gestartet ist, kann der Benutzer die Oberfläche im Menüpunkt View → Language von Englisch auf Deutsch umschalten.
Ein erster Blick in das Menü verheißt nur Gutes. Alles was sich das Programmiererherz wünscht, scheint vorhanden zu sein. Auf Grund der Fülle können im Weiteren daher gar nicht alle Menüpunkte genannt oder deren Funktion beschrieben werden. Dafür sollen vielmehr die gar nicht so wenigen Highlights hervorgehoben werden.
Unter Datei → Projektstatistik finden sich einige Zahlen (Anzahl der Zeilen, Zeichen, Kommentare etc.), die aus den Projektdateien gewonnen werden, sowie eine Schätzung des geöffneten Projekts nach dem „Constructive Cost Model“ [5]. Mit diesem wird aufgrund des zugrunde liegenden Codes versucht, einerseits die Zeitdauer zu dessen Erstellung sowie andererseits die damit verbundenen Kosten in US-Dollar anzugeben. Die Zahlen, die mit einer solchen Schätzung verbunden sind, können durchaus interessant oder (wie im vorliegenden Beispielfall) auch ziemlicher Nonsens sein und einen dann zum Schmunzeln anregen, wenn die Schätzung ergibt, dass der Code in 1½ Monaten

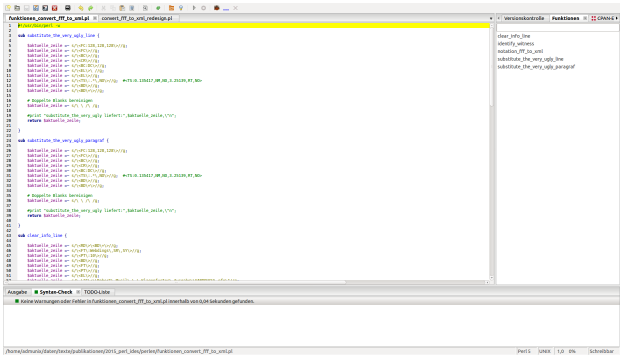

Abbildung 3: Die Rahmendarstellung in Padre.

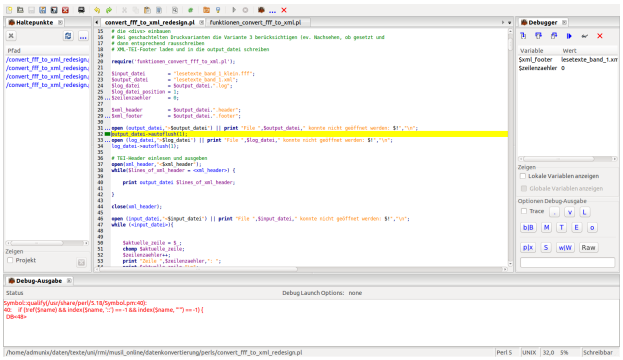

Abbildung 4: Die Integration des Perl-Debuggers in Padre.

geschrieben werden kann, man selbst aber Jahre daran gefeilt hat, um diesen zu warten und zu perfektionieren. Aber das kann Padre nun einmal nicht wissen. Es sieht immer bloß das Endergebnis.

Im Menüpunkt „Bearbeiten“ finden sich all jene Funktionen, die man dort üblicherweise erwartet. Daneben verstecken sich hier jedoch auch die automatische Vervollständigung, die mit STRG+BLANK ausgelöst wird, sowie die Verwaltung von Code-Teilen (Schnipseln), mit der Wiederkehrendes elegant vorrätig gehalten werden kann. Padre vervollständigt dabei sämtlichen Perl-Code als auch die Namen von Variablen, Funktionen, Objekten und dergleichen, die innerhalb einer geöffneten Datei bereits verwendet wurden. Jene, die in anderen Projektdateien vorkommen, werden beim ersten Aufruf leider nicht vervollständigt. Bei umfangreichen Projekten würde eine solche Funktion allerdings sehr viele Ressourcen binden und das ansonsten flüssige Arbeiten merklich bremsen. Zudem sind in diesem Abschnitt das Patchen von Dateien und das Erzeugen einer Differenzdatei (Diff) untergebracht. Beim Testen haben sich hier jeodch Probleme mit der Auswahl und Anzeige der zweiten, abzugleichenden Datei ergeben, sodass diese Funktionen nicht genutzt werden konnten.

Als sehr nützlich stellt sich im Menüpunkt „Suchen“ der Eintrag „Lesezeichen“ heraus. Sind Lesezeichen gesetzt, so können diese hinfort sehr einfach angesprungen und verwaltet werden. Als einziger Wermutstropfen bleibt der Umstand, dass diese Lesezeichen nicht innerhalb der Projektdateien, sondern im Homeverzeichnis des Benutzers verwaltet werden und daher nicht oder nur sehr mühsam über die Versionsverwaltung mit anderen abgeglichen werden können.

Ist man beim Menüpunkt „Ansicht“ angekommen, so zeigt sich an dieser Stelle deutlich, wie viele Tools Padre unter der Haube vereint und in eigenen Rahmen anzuzeigen in der Lage ist:

- über den CPAN-Explorer kann ebendort nach Perl-Modulen gesucht und diese auf einfachem Wege auch gleich installiert werden. Eigenartiger Weise werden immer wieder mal Module nicht angezeigt, die im CPAN sehr wohl zu finden sind und über das Tool CPAN-Minus (das auch bei Padre im Hintergrund werkt) durchaus installiert werden können,
- die „Funktionsliste“ zeigt übersichtlich alle definierten Routinen an und stellt diese zum Navigieren innerhalb des Codes bereit,
- aus der „Projektansicht“ wird deutlich, welche Dateien zu einem Perl-Projekt gehören. Über die angezeigte Liste können die zugehörigen Dateien geöffnet werden,
- mit der „Übersicht“ erhält ein Programmierer einen schnellen Überblick zu den geladenen Perl-Modulen und den eingebundenen Methoden,
- mit „Ausgabe“ werden in einem Rahmen die Ausgaben eines gestarteten Programms sowie jene, die von Perl selbst stammen, angezeigt. Dies geschieht in gleicher Weise, wie man es auf der Kommandozeile eines Terminals gewohnt ist,
- über den „Syntax-Check“ werden die Meldungen des Moduls Perl::Critic angezeigt, wobei sich hier der Level der kritischen Kommentare (üblicherweise von sehr streng bis eher nachlässig) nicht einstellen lässt,
- die „TODO-Liste“ ist ein praktischer Platz, um sich Notizen aufzubewahren. Leider wird auch diese Liste im Homeverzeichnis des Benutzers verwaltet und kann so nur mühsam mit anderen abgeglichen werden,
- mit „Versionskontrolle anzeigen“ erhält man einen Überblick zum Versionsstand lokaler Dateien gegenüber einem Repository, das mit Subversion oder Git vorgehalten werden kann. Leider ist die Konfiguration eines beliebigen Git-Repositories im Test gescheitert. Auch die Recherche nach den Ursachen blieb im Dunkeln. Nach dem Installieren von

  `cpanm Padre::Plugin::Git`

  und

  `cpanm Padre::Plugin::SVN`

  können die beiden Plugins im Plugin-Manager aktivert werden. Mit dem Git-Plugin

hat dies zu keiner funktionierenden Konfiguration geführt. Die Versionskontrolle via SVN reiht sich anschließend als eigener Eintrag im Menüpunkt „Werkzeuge“ ein und kann fortan von dort aus gesteuert werden,
- die Funktion „Code-ausblenden“ sollte in diesem Menüpunkt noch hervorgehoben werden. Damit werden, wie auch in anderen Editoren und IDEs, zusammengehörige Codeteile bei Bedarf ein- bzw. wieder ausgeklappt, um in der Anzeige etwas mehr Übersicht zu bewahren

„Refactor“ [6] verheißt dem Perl-Programmierer laut Beschreibung auf der Homepage, dass hier Funktionen vereint werden, die das Umarbeiten komplexer Programme in besser lesbare unterstützen sollen. Dazu gehört das automatisierbare Umbenennen von Variablennamen sowie das gleichzeitige Einführen einer besser lesbaren Form in „CamelCase“. Zudem stellt das Programm eine Unterstützung bereit, mit der Programmcode aus Routinen herausgelöst und in andere eingefügt werden kann. Leider traten bei allen Versuchen Meldungen zu „unbekannten Fehlern“ auf, sodass auf das Refactoring momentan verzichtet werden muss. Diesbezüglich sind im Padre-Wiki [7] Einträge zu finden, die darauf hindeuten, dass diese Fehler schon einige Jahre mitgeschleppt werden, ohne dass diese je behoben wurden.
Die Funktionen, die zum Menüpunkt „Debugging“ angezeigt werden, sind vorbildlich ausgeführt (siehe Abbildung 4). Die graphische Oberfläche führt zu den wichtigsten Befehlen des Debuggers. In einem Rahmen am unteren Bildschirmrand sind dessen Ausgaben zugänglich. Am linken Rand werden in einem Rahmen die definierten Haltepunkte angezeigt, am rechten Rand sind die Navigationsmöglichkeiten durch den Code zugänglich, die der Debugger bietet. Auch eine Inspektion von Variablenwerten ist dort integriert. Insgesamt kann die Integration des Debuggers als gelungen angesehen werden, da sich hier sowohl Anfänger als auch Fortgeschrittene sofort zurecht finden. Der Debugger mag so als durchaus hilfreich in der oft mühsamen Suche nach Fehlern empfunden werden.
Unter dem Menüpunkt „Werkzeuge“ findet man sämtliche Einstellungen, über die das Verhalten von Padre konfiguriert werden kann, ein Editor für reguläre Ausdrücke, der gut umgesetzt ist und einem durchaus weiterhelfen kann, sowie der Plugin-Manager, über die aus dem CPAN installierte Plugins aktiviert werden müssen, damit sie anschließend auch nutzbar sind.
Auch der Menüpunkt „Hilfe“ verdient seine Beachtung. Typischerweise lesen die wenigsten Programmierer nach und fragen dafür lieber schnell andere. Hier versammelt sich jedoch eine Fülle an Quellen, die einem zu den Themen rund um Padre und Perl weiterhelfen. Als nette Idee kann gewertet werden, dass hier auch die Möglichkeit besteht, via IRC „Live-Kontakt“ zu anderen zu bekommen, was in turbulenten Zeiten (wenn also viele Mitglieder online sind) für den einen oder die andere dann durchaus sinnvoll sein mag.

Fazit zu Padre

1. Der allgemeine Mehrwert von Padre gegenüber einem bloßen Editor ist sicher darin zu sehen, dass eine große Fülle an Perl-Eigenheiten abgedeckt wird. Der Anspruch, dass mit Padre Programmierer hin zu Perl geführt werden, kann allerdings nicht eingelöst werden, da das tatsächliche Kodieren nur über die Angabe externer Quellen innerhalb der Hilfefunktionen unterstützt wird.
2. Padre lässt sich via SVN in typische Arbeitsumgebungen integrieren, die Teamarbeit mit sich bringt. Externe Tools können jedoch nur mit einigem (Programmier-) Aufwand über die definierten Schnittstellen des Programms eingebunden werden.
3. Die Anwendung ist sehr übersichtlich gestaltet, sodass man sich grundsätzlich rasch zurecht findet.
4. Padre bietet viele Möglichkeiten, umfangreichen Code übersichtlich darzustellen und darin zu navigieren. Wer häufig mit fremden Perl-Programmen umgehen oder diese umarbeiten muss, wird diesbezüglich gut unterstützt.
5. Der Perl-Debugger ist sehr gut integriert. Die allermeisten Funktionen, die dieser bietet, sind über graphische Elemente zugänglich. Diese sind zudem übersichtlich umgesetzt.
6. Der Code wird laufend gepflegt. Leider werden lästige Fehler, die nützliche Funktionen verhindern, über Jahre mitgeschleppt ohne dass diese je behoben wurden.
7. Die Dokumentation ist sehr umfangreich. Sowohl die Funktionen als auch die Schnittstellen zum Programm selbst sind umfassend und gut verständlich beschrieben.
8. Die Performance entspricht jener eines typischen Editors und stellt an gängige Hardware keine besonderen Herausforderungen.

EPIC: ein Plugin für Eclipse

Ursprünglich war Eclipse als integrierte Entwicklungsumgebung [8] ganz auf die Unterstützung von Java-Programmierern ausgerichtet. Das Konzept, das über die Jahre entwickelt und im Weiteren verfolgt wurde, sah jedoch vor, eine Anwendung bereitszustellen, die möglichst modular aufgebaut ist und damit für eine Vielzahl an Zwecken eingesetzt werden kann. Eclipse ist in Java programmiert [9]. Seit Version 3 (2004) wird im Kern das Framework Equinox eingesetzt [10]. Die Anwendung selbst besteht aus einer Vielzahl an Plug-

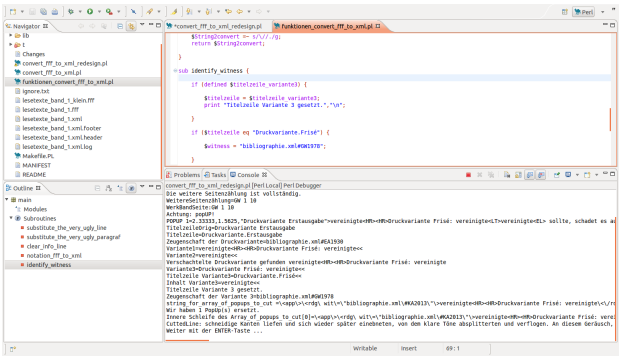

Abbildung 5: Die typische Frame-Ansicht von Eclipse.

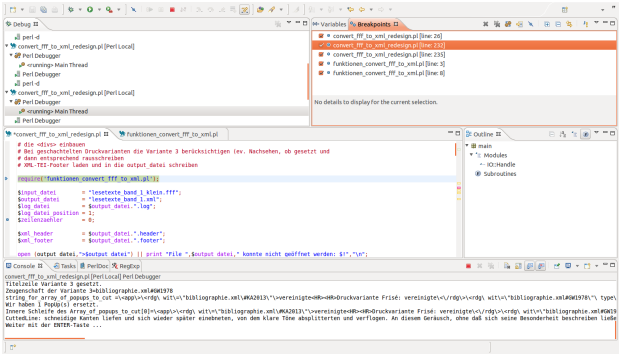

Abbildung 6: Die Integration des Perl-Debuggers in Eclipse.

ins, die sich am Kern einklinken und das Ergebnis wie eine monolithische Anwendung erscheinen lässt. Eclipse selbst wird aufgrund dieser äußerst flexiblen Architektur gern als Basis für andere Anwendungen herangezogen, die entweder ausschließlich zusätzliche Plugins liefern und somit dem Aussehen und der Bedienung von Eclipse sehr nahe kommen – oder aber sich der Eclipse RCP (Rich Client Platform) bedienen und damit zwar die typischen Eclipse-Funktionalitäten nutzen, ohne der Anwendung in Aussehen und Bedienung zwingend sehr ähnlich zu sein [11].
Dieses Grundkonzept ist natürlich prädestiniert, als IDE für eine Vielzahl an Programmiersprachen eingesetzt zu werden.
Mit

```
sudo apt-get install eclipse
```

installiert man zur Zeit aus den Standard-Repositorien die Anwendung in Version 3.8. Die Vielzahl an Paketen, die daraufhin installiert wird, deutet schon darauf hin, dass durchaus Einiges (auch an Ressourcen) gebraucht wird, um mit Eclipse zu arbeiten. Zusätzlich sollte man folgende Perl-Module installieren:

```
sudo apt-get install libpadwalker-perl
libmodule-starter-perl libperl-critic-perl
```

Das passende Perl-Plugin ist EPIC [12] (Perl Editor and IDE for Eclipse) und wird aus Ecplise heraus installiert. Über die Menüeinträge Help → Install New Software gelangt man zum Installationsdialog und gibt dort als Quelle folgenden Link ein:

```
http://e-p-i-c.sf.net/updates
```

Anschließend folgt man den Anweisungen und erhält das Perl-Plugin (momentan) in der Version 0.6.57. Ecplise sucht in den Standard-Einstellungen automatisch nach Updates und weist auf diese hin, sodass neuere Versionen auf Dauer einfach zugänglich bleiben. In äquivalenter Weise kann man in diesem Menüpunkt auch die Sprachunterstützung für Deutsch nachrüsten. Der passende Link dazu ist:

```
http://download.eclipse.org/technology/babel/
update-site/R0.13.0/mars
```

Ist eine Sprache wie Deutsch, oder jede weitere installiert und entspricht diese zudem der Standardsprache des Betriebssystems, so wird diese fortan auch in Eclipse automatisch gewählt. Leider vermisst man anschließend den Menüpunkt, um zwischen den einzelnen Sprachen umzuschalten und die Standardsprache einstellen zu können. Überraschender Weise gelingt dies (hier für Englisch) nur auf der Kommandozeile eines Terminals mit

```
eclipse -nl en
```

Die Bedienung von Eclipse ist durchaus gewöhnungsbedürftig. Ist man jedoch einmal damit vertraut, gelangt man rasch an sein Ziel. Egal, ob man die IDE nun gerade zum Perl-Programmieren, als XML-Editor oder wofür auch immer einsetzt.
Im Unterschied zu den anderen beiden Tools, die hier vorgestellt werden, ist zu beachten, dass in Eclipse viele Funktionen erst zugänglich sind, sobald man die zu erstellenden oder zu bearbeitenden Dateien in ein „Projekt“ integriert (importiert) hat. Das Anlegen einer neuen oder Öffnen einer bestehenden Perl-Datei reicht dazu vielfach nicht.
Ist diese Hürde jedoch genommen, dann bietet ein Projekt-Explorer am linken Bildschirmrand einen einfachen Überblick zu den vorhandenen Dateien, ein Gliederungsrahmen, der unterhalb angezeigt wird, bietet raschen Zugriff auf die Strukturelemente (Module und Funktionen) eines Programmes. Über den Menüpunkt „Source“ sind die Funktionen der beiden Perl-Module Pod::Checker und Perl::Critic zugänglich. Startet man diese, so werden am linken Rand des Editors Warn-Markierungen („EPIC-Markers“) angezeigt, innerhalb derer sich über das entsprechende Navigationssymbol navigieren lässt. Zudem werden am rechten Bildschirmrand der Anwendung geraffte Sprungmarken eingeblendet, über die bequem direkt zu einzelnen Positionen gesprungen werden kann. Der Menüeintrag „Format“ versucht sich via Perltidy an einer einheitlichen Formatierung des Perl-Codes, die nicht immer zu besseren (wohl aber zu sehr einheitlichen) Ergebnissen führt.

Eclipse verändert sein Aussehen (die „Views“) je nach Aufgabe. Das mag für den Anfänger einigermaßen verwirrend erscheinen, da sich einzelne Frames dabei schließen und andere öffnen. Zum Erledigen der jeweiligen Aufgabe (Editieren, Versionieren, Debuggen etc.) ist dieser Ansatz jedoch durchaus sinnvoll. Schließlich konzentrieren sich dabei alle Funktionen darauf, was man gerade erledigen möchte.
Über den Menüpunkt „Run“ können Perl-Programme gestartet und deren Ausgaben in einem Frame angesehen werden. Daneben versteckt sich hier auch die Option, Programme im Debug-Modus zu starten, wobei die Möglichkeiten des Perl-Debuggers hier sehr gut umgesetzt sind, sodass Anfänger zurecht kommen und Fortgeschrittenen ein komfortables Arbeiten mit dem Debugger möglich ist. Erwähnenswert ist die Funktion, dass gesetzte Breakpoints einen Neustart der IDE überleben und so innerhalb des Projekts mitgeführt werden können. Leider werden diese nicht innerhalb der Projektdateien abgelegt, sodass diese innerhalb der Versionskontrolle nicht sinnvoll gehalten oder gar mit anderen abgeglichen werden können. Ist das Perl-Modul Padwalker installiert, zeigt der Debug-Modus zudem sehr komfortabel die aktuellen Inhalte von Variablen an. Beide Funktionen (Run und Debug) sind zudem über entsprechend gestaltete Icons unterhalb des Menüs zugänglich.
Die für eine etablierte IDE wie Eclipse erwartbaren Funktionalitäten sind vollständig vorhanden. Neben den bereits erwähnten sind das: Syntax highlighting, automatische Vervollständigung von Funktionen und Variablen, das Führen von To-do-Listen, die Verwaltung von Code-Schnipseln (hier „Templates“) und eine integrierte Hilfe für Perldoc. Gut versteckt ist der Editor für reguläre Ausdrücke. Dieser findet sich im Menü unter Window → Show View → Other → EPIC → RegExp. Ein Highlight dieses sehr einfach gehaltenen Editors ist, dass ein regulärer Ausdruck auch schrittweise ausgeführt werden kann und dabei der jeweilige Text unterlegt wird, der aufgrund des aktiven Ausdrucks gerade zutrifft (matched). Da das Tool Padre sich umfassend an den Methoden des Refactorings versucht, fällt im Test auf, dass dies auch in Eclipse prominent im Menü vertreten ist. Momentan wird jedoch nur die Funktion unterstützt, dass Code aus bestehenden Subroutinen in neue übernommen werden kann. Dabei wird auf das Perl-Modul Devel::Refactor [13] zurückgegriffen, das sich noch in einem frühen Entwicklungsstadium befindet (momentan v0.05). Die EPIC-Entwickler geben an, Weiterentwicklungen dieses Moduls auch in EPIC zu berücksichtigen.

Wer sich abseits der hier besprochenen Tools kundig machen will, der findet durchaus weitere, sehr brauchbare Werkzeuge, die den Perl-Programmierer in seiner Arbeit unterstützen.

So existiert für den Editor Emacs ein eigener „CPerl-Mode“. Dieser ist in den Ubuntu Paketen enthalten und wird immer dann aktiv, wenn Emacs eine Datei als Perl-Datei erkennt. Dieses Modul ist in weiten Teilen frei konfigurierbar und Emacs-typisch in den Editor integriert. In gleicher Weise wie der Editor Vim kann auch Emacs aufgrund seiner offenen Architektur gezielt zu einer umfangreichen Entwicklungsumgebung ausgebaut werden.

Der Editor Kephra richtet sich, ebenfalls wie Padre, mit seinen Funktionen gezielt an Perl-Programmierer. Obwohl er seit einigen Jahren mit durchaus großem Enthusiasmus entwickelt wird, hat er jedoch den Funktionsumfang und die Stabilität von Padre bei Weitem nicht erreicht. Die Programmierer streben als Ziel einen Editor an, der beim Coding hilft. Die Vielfalt, die eine Entwicklungsumgebung bietet, wird hingegen nicht angestrebt.

Wer IntelliJ IDEA anstelle von Eclipse als IDE einsetzt kann auch als Perl-Programmierer auf ein passendes Perl-Modul zurückgreifen. Der Funktionsumfang ist mit dem Eclipse-Plugin vergleichbar. Die aktuelle Version 1.112 stammt vom 2.11.2015 und steht in der Apache Lizenz 2.0 zum Download bereit. Ein Blick in den Bugtracker zeigt, dass die Programmierer das Plugin regelmäßig warten und bemüht sind, die Weiterentwicklungen der IDE auch in das Perl-Plugin einfließen zu lassen.

Fazit zu EPIC

1. Der Mehrwert von EPIC ist sicher darin zu sehen, dass Programmierer, die mit den Eigenheiten von Eclipse vertraut sind, damit auch eine umfassende Unterstützung für Perl vorfinden. Interessant kann für den einen oder die andere der Umstand sein, dass nicht nur lokal installierte Programme bearbeitet und ausgeführt werden können, sondern auch jene, die innerhalb des CGI eines Web-Servers laufen oder gar auf einem entfernten Rechner betrieben werden (Remote Mode).
2. Eclipse bietet für alle Arbeitsschritte Plugins und lässt sich darüber hervorragend in unterschiedliche Szenarien der Einzel- oder Teamarbeit integrieren.
3. Die Anwendung ist sehr gewöhnungsbedürftig, da das häufige Umschalten in Views rasch zur Verwirrung sorgen kann. Anfänger müssen diesbezüglich mit einem gewissen Aufwand rechnen, bis sie sich rasch zurecht finden.
4. EPIC bietet viele Möglichkeiten, umfangreichen Code übersichtlich darzustellen und darin zu navigieren. Wer häufig mit fremden Perl-Programmen umgehen oder diese umarbeiten muss, wird sehr gut unterstützt.

5. Der Perl-Debugger ist hervorragend integriert und innerhalb einer eigenen View einfach zu bedienen.
6. Der Code wird laufend gepflegt. Die Kommunikation mit den Entwicklern funktioniert via E-Mail, Mailingliste und innerhalb eines Forums ausgezeichnet.
7. Die Dokumentation ist sehr übersichtlich gestaltet und nicht durch kleine Details überfrachtet. Neben dem eigentlich Programmcode wird auch die Dokumentation sorgfältig gepflegt.
8. Wer auf die Performance eines Editors oder gar auf dem Arbeiten innerhalb eines Textterminals besteht, wird sich im Umgang mit Eclipse umstellen müssen. Auf einem durchschnittlichen Arbeitsplatzrechner zeigten sich jedoch keine Engpässe oder irgendwelche Misslichkeiten, die darauf hindeuteten, dass nicht ausreichend Ressourcen vorhanden wären. Die Qualität der Ubuntu-Pakete scheint jedoch nicht die Beste zu sein, da viele Installationen der Originalpakete zu unterschiedlichsten Fehlern geführt haben. Der Einsatz des Ecplise-SDK, das unterschiedlichste Builds anbietet ist über den Link

```
http://download.eclipse.org/eclipse/downloads/
drops4/R-4.5-201506032000/#EclipseSDK
```

erreichbar. Die Installation dieser Ausgabe zeigte letztlich keine Probleme. Zur Anwendung gelangte so die Version 4.5.0, die gegenüber dem Paket aus dem Ubuntu-Standard-Repository zudem flüssiger zu bedienen war.

	Vim-Plugin Perl-Support	Padre	Eclipse-Plugin EPIC
Multi-platform	+	+	+
Syntax highligthing	+	+	+
Syntax checking	-	+	+
Refactoring	-	-	+
Code completion	N	+	+
Code folding	+	+	+
Coding support	+	-	-
Auto-indent	+	+	+
Bookmarks	-	+	+
Code snippets	+	+	+
Context sensitive help	-	+	+
Patch / Diff integration	+	+	+
Versioning	N	+	+
Multilanguage user interface	-	+	+
Functions and outline	N	+	+
Debugging	N	+	+
Perl::Critic	+	+	+
Perl::Tidy	+	+	+
Perldoc support	+	+	+
Project browser	N	+	+
Regex editor	+	+	+
Remote editing	+	+	+
Source navigation	-	+	+
Todo lists	-	+	+
Adjustable line breaks	+	+	+
Documentation	+	+	+

Tabelle 1: Funktionsvergleich der beschriebenen Werkzeuge. Legende: (+) vorhanden, (-) nicht vorhanden, (N) nachrüstbar über weiterer Plugins.

Resümee

Wer sich häufig mit komplexen Perl-Programmen beschäftigen und sich dabei rasch einen Überblick zum vorliegenden Code verschaffen muss, dem kann eine integrierte Entwicklungsumgebung in der Tat sehr helfen.
Ob man dabei seinen geliebten Editor behalten möchte und diesen durch die Verwendung von Plugins schrittweise zu einer IDE ausbaut oder ob

man eine ausgereifte IDE wie Eclipse installiert und sich dazu mittels EPIC-Plugin komfortable Perl-Unterstützung hinzufügt, mag letztlich reine Geschmackssache sein. Dass hier unterschiedlichste Zugangsweisen nebeneinander vorhanden sind, zeigt sich auch im Umstand, dass für die Editorfunktionen von Eclipse auch ein Vim-Plugin existiert, sodass jene, die mit dem Ausbau des Editors hin zu einer IDE irgendwo gestrandet sind, sich hier zumindest ein wenig wohler und aufgehobener fühlen können.
Anwendungen wie Padre stehen zwischen diesen beiden Ansätzen, zeigen im Arbeitsalltag jedoch weder die hohe Flexibilität noch den notwendigen Reifegrad, um letztlich die unterschiedlichsten Bedürfnisse eines Perl-Programmierers möglichst umfassend abzudecken.

Quellen

[1] Comprehensive Perl Archive Network. Over 20 years of Perl libraries at your fingertips.
[http://www.cpan.org]

[2] perl-support.vim : Perl IDE -- Write and run Perl-scripts using menus and hotkeys. Created by Fritz Mehner.
[http://www.vim.org/scripts/script.php?script_id=556]

[3] Mailingliste zum Plugin perl-support.vim
[vim-plugins-list@lug.fh-swf.de]

[4] Padre, the Perl IDE. Perl Application Development and Refactoring Environment.
[http://padre.perlide.org/]

[5] COCOMO (Constructive Cost Model).
[https://de.wikipedia.org/wiki/COCOMO]

[6] Refactoring
[https://de.wikipedia.org/wiki/Refactoring]

[7] WIKI for Padre, The Perl IDE.
[http://padre.perlide.org/trac/wiki/Plugins]

[8] Homepage zu Eclipse.
[https://eclipse.org/]

[9] Wikipedia-Eintrag zu Eclipse (IDE).
[https://de.wikipedia.org/wiki/Eclipse_(IDE)]

[10] Das Framework Equinox von Eclipse.
[http://www.eclipse.org/equinox/]

[11] Eclipse for RCP and RAP Developers.
[http://www.eclipse.org/downloads/packages/eclipse-rcp-and-rap-developers/keplersr2]

[12] EPIC - Perl Editor and IDE for Eclipse
[http://www.epic-ide.org/]

[13] Devel::Refactor - Perl extension for refactoring Perl code
[http://search.cpan.org/~ssotka/Devel-Refactor-0.05/Refactor.pm]

Der Autor

Dr. Harald Jele ist Mitarbeiter an der Universität Klagenfurt.