

# API-Usability der auf Templatemetaprogrammierung basierenden Softwarebibliothek „SeqAn“

Björn Kahlert

Dissertation  
zur Erlangung des Grades  
Doktor der Naturwissenschaften (Dr. rer. nat.)

Fachbereich Mathematik und Informatik  
Freie Universität Berlin

Berlin, 2015



## *Verfassererklärung*

Ich, Björn Kahlert, erkläre: Ich habe die vorgelegte Dissertation selbständig, ohne unerlaubte fremde Hilfe und nur mit den Hilfen angefertigt, die ich in der Dissertation angegeben habe. Alle Textstellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Schriften entnommen sind, und alle Angaben, die auf mündlichen Auskünften beruhen, sind als solche kenntlich gemacht. Bei den von mir durchgeführten und in der Dissertation erwähnten Untersuchungen habe ich die Grundsätze guter wissenschaftlicher Praxis eingehalten.

Unterschrift:

---

Ort & Datum:

---

Datum der Disputation: **27.10.2015**

Dekan des Fachbereichs Mathematik und Informatik:

**Prof. Dr.-Ing. Jochen Schiller**

Gutachter:

**Prof. Dr. Lutz Prechelt, Freie Universität Berlin, Deutschland**

**Prof. Dr.-Ing. Mira Mezini, Technische Universität Darmstadt, Deutschland**



# Zusammenfassung

**Ausgangslage** Die Usability der bioinformatischen, auf C++-basiierenden Softwarebibliothek *SqAn* ist wenig ausgeprägt, denn SeqAn wurde mit dem ultimativen Ziel einer hohen Performance entwickelt. Die Grundlage des Entwurfs bilden *Templatemetaprogrammierung* und *generische Programmierung*. Obwohl diese Techniken die Usability einer API stark beeinflussen, ist die API-Usability-Forschung diesbezüglich vollkommen erkenntnislos.

**Zielsetzung** Das Ziel dieser Arbeit besteht darin, die API-Usability von SeqAn zu erforschen, um mit diesem Wissen SeqAns Usability zu verbessern und verallgemeinerbare Erkenntnisse für andere auf Templatemetaprogrammierung basierenden APIs zu gewinnen.

**Methode** Für die Erforschung der API-Usability von SeqAn wird die empirische, praktisch annahmenlose Grounded Theory Methode nach Strauss und Corbin gemeinsam mit einer neuartigen und den speziellen organisatorischen Rahmenbedingungen gerechten Datenerhebungsmethode eingesetzt. Letztere kombiniert diverse subjektive und eine hochstrukturierte, objektive Datenquelle miteinander. Eigens für diese Arbeit wird das qualitative Datenanalysewerkzeug *API Usability Analyzer* entwickelt, welches den methodischen Gegebenheiten gerecht wird, indem es eine Reihe von Funktionen bietet, die selbst bei den kommerziellen Lösungen wie ATLAS.ti nicht zu finden sind.

**Ergebnisse** Diese Arbeit stellt die Ergebnisse der API-Analyse in Form einer Grounded Theory dar, die nachzeichnet, wie fehlende Erkenntnisse zu den verwendeten Programmierparadigmen und mangelndes Verständnis von den SeqAn-Anwendern, die sich in API-Anwender und API-Endanwender unterscheiden lassen, zu der schlechten API-Usability führte. Es werden die gefundenen Usability-Probleme und mit welchen Strategien diese von den Anwendern bewältigt werden, ausführlich vorgestellt. Weiterhin werden Maßnahmen zur Behebung der Usability-Probleme vorgeschlagen. Unter anderem wird die Dokumentation, unter Berücksichtigung von so genannten Sprachentitätstypen, umfassend überarbeitet und die Nutzung von SeqAn durch API-Endanwender ermöglicht. Weitere Beiträge stellen ein umfassender wissenschaftlicher Literaturüberblick zum Thema API-Usability und das bereits erwähnte qualitative Datenanalysewerkzeug dar.

**Konklusion** Die in dieser Arbeit erarbeiteten und umgesetzten Maßnahmen haben die API-Usability von SeqAn erfolgreich verbessert. Die auf diesem Weg gewonnenen Erkenntnisse lassen allgemeine Rückschlüsse auf den Entwurf von auf Templatemetaprogrammierung basierenden Softwarebibliotheken zu. Darüber hinaus eignet sich das eingesetzte Verfahren für die Evaluation anderer APIs. Abschließend werden Empfehlung für den Bau leistungsfähiger, qualitativer Datenanalysewerkzeuge für Forschungsvorhaben vorgestellt, die ebenfalls die Grounded Theory Methode einsetzen.



## *Danksagung*

Mein Dank gilt den folgenden Personen, ohne deren Beitrag diese Arbeit nicht möglich gewesen wäre:

- Lutz Prechelt für sein Vertrauen in meine Person, die Betreuung und die vielen hilfreichen Anmerkungen.
- Julia Schenk, Franz Zieris, Stephan Salinger und Holger Schmeisky für die wertvollen Diskussionen über meine Forschung.
- Meiner Familie und meinen Freunden, die meine langen Phasen der konzentrierten Isolation ertragen haben. Dabei danke ich ganz besonders meiner Mutter, die mich stets kompromisslos unterstützte. Alexander Zelasny, der keine inhaltliche Auseinandersetzung scheute und für mich eine unermessliche Unterstützung war. Marcus Wolschke, der mir die Luft gab, die ich für den Abschluss meiner Arbeit benötigte. Mario Zelasny und Niels Berkholz, die mich unterstützten, indem sie mir unzählige Störfaktoren vom Hals hielten. Andrea Tank für ihren aufopfernden Einsatz bei der Korrektur meiner Arbeit. Und schließlich auch Martin Merl und Olaf Apel für die intellektuelle Auseinandersetzung mit meinem Thema.
- Einzelnen Mitgliedern des Völklinger Kreis e.V. — ganz besonders Harm-Peter Dietrich und Patrick Dommaschk für deren Bestärkung meines Lebensweges.
- Den Teilnehmern der SeqAn-BioStore-Workshops und der PMSB-Praktika für deren Bereitschaft zur Datenaufzeichnung.
- Und allen dafür, dass sie mich stets ermutigt haben, weiter zu machen.



# INHALTSVERZEICHNIS

<b>Verfassererklärung</b>	<b>3</b>
<b>Zusammenfassung</b>	<b>5</b>
<b>Danksagung</b>	<b>7</b>
<b>Inhaltsverzeichnis</b>	<b>8</b>
<b>Abbildungsverzeichnis</b>	<b>15</b>
<b>Tabellenverzeichnis</b>	<b>19</b>
<b>Abkürzungen</b>	<b>22</b>
<b>1 Einleitung</b>	<b>25</b>
1.1 Motivation . . . . .	27
1.1.1 Bedeutung der Bioinformatik . . . . .	28
1.1.2 Bioinformatik und Programmieren . . . . .	29
1.1.3 Relevanz von API-Usability . . . . .	30
1.1.4 Nutzen von API-Usability . . . . .	31
1.1.5 Synthese und Zusammenfassung . . . . .	32
1.2 Definitionen . . . . .	33
1.3 SeqAn . . . . .	35
1.3.1 Entwurf . . . . .	36
1.3.2 Beispiel . . . . .	38
1.3.3 Bestandteile . . . . .	41
1.3.4 Anwender . . . . .	44
1.3.5 Zusammenfassung . . . . .	45
1.4 Methode der Grounded Theory . . . . .	47

1.4.1	Methode der Grounded Theory in der Informatik . . . . .	47
1.4.2	Generationen und Strömungen der Methode der Grounded Theory . . . . .	48
1.4.3	Methode der Grounded Theory im Detail . . . . .	49
1.4.4	Gütekriterien . . . . .	52
1.4.5	Anwendung der Methode der Grounded Theory in dieser Arbeit . . . . .	53
<b>2</b>	<b>Forschungsstand</b>	<b>57</b>
2.1	Überblick . . . . .	59
2.1.1	Grundlagen . . . . .	59
2.1.2	API-Usability . . . . .	60
2.2	Programmverständnisforschung . . . . .	63
2.2.1	Grundlagen (Shneiderman u. Mayer 1979) . . . . .	63
2.2.2	Top-Down (Brooks 1983) . . . . .	65
2.2.3	Bottom-Up (Pennington 1987) . . . . .	68
2.2.4	Top-Down vs. Bottom-Up (Shaft u. Vessey 1998) . . . . .	71
2.2.5	Fact Finding (LaToza u.a. 2007) . . . . .	74
2.3	Usability-Evaluation . . . . .	77
2.3.1	Klassische Usability-Evaluationsmethoden . . . . .	77
2.3.2	Cognitive Dimensions Framework (Green 1989; Green u. Petre 1996; Blackwell u. Green 2000; Blackwell u. Green 2003) . . . . .	78
2.4	API-Usability-Evaluation . . . . .	89
2.4.1	End-User Programming & End-User Software Engineering (Ko u.a. 2011) . . . . .	89
2.4.2	Klassische Usability-Evaluation . . . . .	91
2.4.3	Personas (Clarke 2007) . . . . .	96
2.4.4	Metriken (Stylos u. Myers 2007) . . . . .	98
2.4.5	Lernbarrieren (Ko u.a. 2004) . . . . .	99
2.4.6	Abstrakte API-Aktivitäten (Stylos 2009) . . . . .	101
2.4.7	API-Direktiven (Dekel 2011) . . . . .	103
2.4.8	Grundlagen / Psychologie / Strategien . . . . .	105
2.4.9	Weitere Methoden & Techniken . . . . .	108
2.5	API-Usability-Verbesserung . . . . .	111
2.5.1	“Participatory Programming: Developing Programmable Bioinformatics Tools for End-Users” (Letondal 2006) . . . . .	111
2.5.2	“Methods towards API Usability: A Structural Analysis of Usability Problem Categories” (Grill u.a. 2012) . . . . .	113
2.5.3	“An Empirical Study of API Usability” (Piccioni u. a. 2013) . . . . .	116
2.6	Gruppierte Erkenntnisse bezüglich der API-Usability-Forschung . . . . .	119
2.6.1	Dokumentation . . . . .	119
2.6.2	Tutorials . . . . .	120
2.6.3	Beispiele . . . . .	120
2.6.4	Benennung . . . . .	121
2.6.5	Konsistenz . . . . .	122
2.6.6	API-Entwurf . . . . .	123
2.7	API-Werkzeuge . . . . .	129
2.7.1	Werkzeuge für API-Entwickler . . . . .	131
2.7.2	Werkzeuge für API-Anwender . . . . .	131
2.7.3	Werkzeuge für API-Anwender und -Endanwender . . . . .	135

2.7.4	Werkzeuge ausschließlich für API-Endanwender . . . . .	138
<b>3</b>	<b>Forschung</b>	<b>141</b>
3.1	Übersicht . . . . .	143
3.1.1	Rahmenbedingungen . . . . .	143
3.1.2	Planung . . . . .	144
3.1.3	Tatsächlicher Verlauf . . . . .	145
3.1.4	Schwierigkeiten . . . . .	148
3.2	Phase 1: Behebung grober API-Usability-Probleme . . . . .	151
3.2.1	Datenerhebung . . . . .	151
3.2.2	Datenanalyse . . . . .	153
3.2.3	Ergebnisse . . . . .	154
3.2.4	Verbesserungen . . . . .	158
3.2.5	Validierung . . . . .	168
3.3	Phase 2: Planung und Durchführung der Datenerhebung . . . . .	173
3.3.1	Vergleich mit anderen Studien . . . . .	173
3.3.2	Planung der Datenerhebung . . . . .	175
3.3.3	Gruppendiskussion . . . . .	177
3.3.4	Cognitive-Dimensions-Fragebogen . . . . .	179
3.3.5	Programmierfortschritte-Erhebung . . . . .	181
3.3.6	Zusammenfassung . . . . .	190
3.4	Phase 3: Entwicklung des API Usability Analyzers . . . . .	193
3.4.1	Motivation . . . . .	195
3.4.2	Beispiel: Diff-Dateien . . . . .	196
3.4.3	Beispiel: Doclog-Datei . . . . .	199
3.4.4	Herausforderungen für ein GTM-Datenanalysewerkzeug . . . . .	201
3.4.5	Entwurf . . . . .	207
3.4.6	Funktionsweise von APIUA und Implementierung der GTM . . . . .	212
3.4.7	Zusammenfassung . . . . .	226
3.5	Phase 4: Durchführung der GTM-Datenanalyse . . . . .	229
3.5.1	Vergleich mit anderen Studien . . . . .	229
3.5.2	Analyse mit Hilfe der Methode der Grounded Theory . . . . .	231
3.5.3	Probleme . . . . .	239
3.5.4	Zusammenfassung . . . . .	243
3.6	Zusammenfassung der Forschung . . . . .	245
<b>4</b>	<b>Ergebnisse</b>	<b>247</b>
4.1	Theorie: Folgen von SeqAn-Entwurfsentscheidungen . . . . .	249
4.1.1	Anwender . . . . .	251
4.1.2	Entwurfsentscheidungen . . . . .	254
4.1.3	Folgen . . . . .	260
4.1.4	Usability-Probleme & Strategien . . . . .	263
4.2	Zusammenfassung . . . . .	277
4.2.1	Theorie über die Entstehung und Auswirkungen von Entwurfsentscheidungen in SeqAn . . . . .	277
4.2.2	Einsichten . . . . .	278
4.2.3	Ur-Ursache . . . . .	279
4.3	API-Usability-Verbesserungsvorschläge . . . . .	283

4.3.1	Bewertung von Fatalität und Aufwand . . . . .	283
4.3.2	Maßnahmenkatalog . . . . .	284
4.3.3	Maßnahme: Frameworkumbau . . . . .	285
4.3.4	Maßnahme: STL-Angleichung . . . . .	285
4.3.5	Maßnahme: Intransparenzbeseitigung . . . . .	293
4.3.6	Maßnahme: Inkonsistenzbeseitigung . . . . .	295
4.3.7	Maßnahme: Shortcuts . . . . .	295
4.3.8	Maßnahme: Fail-Fast . . . . .	295
4.3.9	Maßnahme: Dokumentation . . . . .	296
4.3.10	Maßnahme: Werkzeugunterstützung . . . . .	298
4.3.11	Maßnahme: Kollaborationsplattform . . . . .	298
4.3.12	Probleme ohne Lösung . . . . .	299
4.3.13	Weitere Maßnahme: Usability-Priorisierung . . . . .	300
4.3.14	Zusammenfassung . . . . .	300
4.4	Verbesserung der API-Usability von SeqAn . . . . .	303
4.4.1	Prozessverbesserungen . . . . .	303
4.4.2	Frameworkumbau . . . . .	304
4.4.3	STL-Angleichung . . . . .	304
4.4.4	KNIME als API-Endanwender-Werkzeug . . . . .	305
4.4.5	Inkonsistenzbeseitigung . . . . .	306
4.4.6	Shortcuts . . . . .	307
4.4.7	Fail-Fast . . . . .	307
4.4.8	Dokumentation . . . . .	308
4.4.9	Kollaborationsplattform . . . . .	322
4.4.10	Usability-Priorisierung . . . . .	322
4.4.11	Zusammenfassung . . . . .	322
4.5	Güte, Validierung und Verallgemeinerbarkeit . . . . .	325
4.5.1	Güte . . . . .	325
4.5.2	Validierung . . . . .	328
4.5.3	Verallgemeinerbarkeit . . . . .	335
4.5.4	Zusammenfassung . . . . .	338
<b>5 Fazit</b>		<b>341</b>
5.1	Ausgangslage . . . . .	341
5.2	Zielsetzung . . . . .	341
5.3	Methode . . . . .	342
5.4	Ergebnisse . . . . .	343
5.5	Konklusion . . . . .	349
<b>6 Ausblick</b>		<b>351</b>
6.1	API-Usability . . . . .	351
6.2	SeqAn . . . . .	354
6.3	Qualitative Datenanalysewerkzeuge mit Unterstützung der GTM . . . . .	356
6.4	UI-Entwicklung mittels Websprachen . . . . .	357
<b>A API Usability Analyzer</b>		<b>359</b>
A.1	Services . . . . .	359
A.2	Browser . . . . .	366

<b>B Literaturergänzungen</b>	<b>375</b>
B.1 Heuristiken der Heuristischen Evaluation . . . . .	375
B.2 Cognitive Dimensions Fragebogen . . . . .	376
<b>C Rohdaten</b>	<b>379</b>
C.1 Programmierfortschritte . . . . .	379
C.2 Interviewnotizen “ATLAS.ti” . . . . .	379
C.3 Ausgefüllte Cognitive-Dimensions-Fragebögen, 15.09.2013 . . . . .	381
C.4 Gruppendiskussions-Transkript, 06.09.2014 . . . . .	381
<b>D Forschungsergebnisse: Heuristische Evaluation</b>	<b>383</b>
<b>E Forschungsdokumente</b>	<b>385</b>
E.1 Einverständniserklärung zur Datenerhebung . . . . .	386
E.2 Online-Umfrage . . . . .	386
E.3 Feedback-Zettel . . . . .	389
E.4 Cognitive-Dimensions-Fragebogen . . . . .	393
<b>Glossar</b>	<b>395</b>
<b>Literaturverzeichnis</b>	<b>399</b>



# ABBILDUNGSVERZEICHNIS

1.1 SeqAn-Logo . . . . .	36
1.2 Paradigmatisches Modell . . . . .	51
2.1 Typische Abhängigkeiten zwischen kognitiven Dimensionen. (Blackwell u. Green 2003)	88
2.2 Einordnung von Endanwender-Programmierern entlang der Dimensionen <i>Programmierungsfahrung</i> und <i>Zielgruppe</i> . . . . .	90
2.3 Qualitätseigenschaften von APIs (Robertson 2007) . . . . .	99
2.4 Kategorisierung von API-Designentscheidungen (Robertson 2007) . . . . .	100
2.5 Lernbarriere — Treffen einer Annahme . . . . .	102
2.6 Lernbarriere — Unüberwindbare Barrieren . . . . .	102
2.7 Abstrakte Programmieraktivitäten (Stylos 2009) . . . . .	104
2.8 API-Direktiven-Taxonomy (Monperrus et al. 2011) . . . . .	106
2.9 Visuelle Darstellung der Usability der Siena API . . . . .	109
2.10 API-Evaluation nach Grill et al. (2012) . . . . .	113
2.11 Problemlösung bei Verwendung von Konstruktoren mit und ohne Parametern . . . . .	125
2.12 Einteilung von API-Werkzeugen . . . . .	130
2.13 Codeeditor mit Anwendungsbeispielen (Neal 1989) . . . . .	132
2.14 Beispiel eines Eclipse-Javadoc-Hilf dialogs für polymorphen Code (Dekel 2011) . . . . .	132
2.15 CodeBroker (Ye u. Fischer 2002) . . . . .	133
2.16 Mica-Webapplikation (Stylos u. Myers 2006) . . . . .	134
2.17 Jadeite-Dokumentationssystem (Stylos et al. 2009a) . . . . .	135
2.18 Apatite-Dokumentationssystem (Eisenberg et al. 2010b) . . . . .	135
2.19 Codelets-Integration in Webeditor (Oney u. Brandt 2012) . . . . .	136
2.20 Precise-Eclipse-Plugin (Zhang et al.) . . . . .	137
2.21 Strathcona: Formulierung einer Beispiel-Suchanfrage (Holmes u. Murphy 2005) . . . . .	138
2.22 Strathcona: Einfügen des gewählten Beispielcodes (Holmes u. Murphy 2005) . . . . .	138
2.23 FrUit-Eclipse-Plugin (Bruch et al. 2006) . . . . .	139
2.24 API-Explorer: Konstruktion einer Instanz (Duala-Ekoko u. Robillard 2011) . . . . .	139
2.25 API-Explorer: Auffinden der zur <b>Message</b> -Klasse in Beziehung stehenden <b>Transport</b> -Klasse (Duala-Ekoko u. Robillard 2011) . . . . .	140

---

3.1	BioStore-Logo . . . . .	143
3.2	BioStore-Komponenten . . . . .	144
3.3	Zeitlicher Verlauf dieser Arbeit . . . . .	147
3.4	Erste Analyse in Apple Keynote . . . . .	154
3.5	Mangelhafter Dokumentationseintrag . . . . .	156
3.6	Commit-Nachrichten-Format . . . . .	159
3.7	SeqAn-Installation unter Windows . . . . .	164
3.8	Vergleich Dokumentationseintrag <b>Infix</b> . . . . .	165
3.9	Boxen zur Auszeichnung von Inhaltstypen . . . . .	167
3.10	Erste Verbesserung des neuen Anfänger-Tutorials . . . . .	168
3.11	Revision der ersten Verbesserung des neuen Anfänger-Tutorials . . . . .	169
3.12	Zweite Verbesserung des neuen Anfänger-Tutorials . . . . .	169
3.13	Ausschnitt aus dem deutschsprachigen Cognitive-Dimensions-Fragebogen . . . . .	181
3.14	Bestätigung der Aktivierung der Datenerhebung . . . . .	186
3.15	Eintragung der ID im Feedback-Zettel . . . . .	188
3.16	APIUA: Startbildschirm . . . . .	193
3.17	APIUA: Offenes Kodieren . . . . .	194
3.18	Beispielhafte Diff-Datei . . . . .	199
3.19	Beispielhafte Doclog-Datei . . . . .	202
3.20	APIUA: Verallgemeinerung von Relationen . . . . .	207
3.21	APIUA: Wertzuweisung bei Umstrukturierungen . . . . .	207
3.22	APIUA: Architektur — Plugins . . . . .	208
3.23	APIUA: Kode-Ansicht . . . . .	208
3.24	APIUA: Eigenschaften-Ansicht . . . . .	209
3.25	APIUA: Architektur — Schichten . . . . .	209
3.26	Vergleich APIUA und ATLAS.ti: Kode-Darstellung . . . . .	213
3.27	Vergleich APIUA und ATLAS.ti: Kodeinstanz-Darstellung . . . . .	214
3.28	APIUA: Memo-Editor . . . . .	215
3.29	APIUA: Gruppendiskussion . . . . .	216
3.30	APIUA: Exploration . . . . .	218
3.31	APIUA: Cognitive Dimensions . . . . .	218
3.32	APIUA: Gruppendiskussion . . . . .	219
3.33	APIUA: Axiales Kodieren . . . . .	220
3.34	APIUA: Modellierung — Unterrelationen . . . . .	221
3.35	APIUA: Modellierung — Implizite Relationen . . . . .	223
3.36	APIUA: Modellierung — Hypothetische Relationen I . . . . .	223
3.37	APIUA: Modellierung — Hypothetische Relationen II . . . . .	224
3.38	APIUA: Modellierung — Hypothetische Relationen III . . . . .	224
3.39	Skalierbarkeit von API-Evaluationsverfahren (Farooq u. Zirkler 2009) . . . . .	230
3.40	Ergebnisse des ersten Analyseversuchs . . . . .	232
3.41	Analyse von Programmierfortschritte-Daten mit Hilfe von APIUA . . . . .	234
3.42	Analyse der Cognitive-Dimensions-Fragebögen mit Hilfe von APIUA . . . . .	235
3.43	Kausale axiale Kodierung . . . . .	236
3.44	Analyse der Gruppendiskussion mit Hilfe von APIUA . . . . .	237
3.45	Axiale Kodierung des Konzepts  Inkonsistenzen bzgl. STL . . . . .	238
3.46	Selektives Kodieren in APIUA . . . . .	240
3.47	Selektives Kodieren in APIUA mit Hilfe zusammengefasster hypothetischer Relationen . . . . .	240

3.48 Axiales Kodiermodell zur Gruppendiskussion . . . . .	242
4.1 Theorie: Folgen von SeqAn-Entwurfsentscheidungen . . . . .	250
4.2 Ursachen und Folgen des Problems ● Inkonsistenzen bzgl. STL . . . . .	292
4.3 Workflow-Engine KNIME . . . . .	294
4.4 Beispiel-SeqAn-Workflow in KNIME . . . . .	306
4.5 KNIME-Konfigurationsdialog für einen CTD-basierten Knoten . . . . .	307
4.6 SeqAn-Installation unter Windows . . . . .	309
4.7 Beispiel für ein überarbeitetes Tutorial . . . . .	310
4.8 Alte und neue Dokumentation im Vergleich — in normaler Breite . . . . .	312
4.9 Entwicklermodus — Dox-Quelle für den aktuellen Dokumentationseintrag . . . . .	315
4.10 Entwicklermodus — Anzeige aller Links für das aktuell selektierte Element . . . . .	316
4.11 Kurzbeschreibung des Sprachentitätstyps <i>Interface-Metafunktion</i> . . . . .	318
4.12 Ausführliche Beschreibung der in SeqAn verwendeten Sprachentitätstypen . . . . .	319
4.13 Alte und neue Dokumentation im Vergleich — in smaler Breite . . . . .	321
A.1 Nebula-Browser-basierter Rich-Text-Editor . . . . .	367
A.2 Nebula-Browser: Veranschaulichung einer Utility-Klasse . . . . .	368
A.3 Nebula-Browser mit geladener Wikipedia-Startseite . . . . .	369
A.4 Nebula-Browser mit Bootstrap-Erweiterung . . . . .	370
A.5 Nebula-Browser: Drag'n'Drop . . . . .	372



# TABELLENVERZEICHNIS

2.1	Fact Finding: Unterschiede zwischen Anfängern und Experten . . . . .	75
2.2	Qualitative Unterschiede zwischen professioneller und Endanwender-Softwaretechnik .	90
3.1	In einer Diff-Datei enthaltene Informationen . . . . .	198
3.2	Anforderungen an GTM <sup>G</sup> -Datenanalysewerkzeuge . . . . .	206
3.3	Gegenüberstellung von GTM-Begriffen . . . . .	213



# LISTINGSVERZEICHNIS

1	Beispiel: Typisches SeqAn-Programm . . . . .	39
2	Beispiel: Typisches SeqAn-Programm in Java I . . . . .	40
3	Beispiel: Typisches SeqAn-Programm in Java II . . . . .	41
4	Automatisch generiertes Code-Beispiel Buse u. Weimer (2012) . . . . .	131
5	Argument-Parser: Beispielhafte Schnittstellenbeschreibung in C++ . . . . .	161
6	Argument-Parser: Beispielhafte Hilfeseite . . . . .	162
7	Argument-Parser: Beispielhafte Fehlerausgabe . . . . .	162
8	Beispiel: Daten zur Arbeitsumgebung . . . . .	184
9	Beispiel: Einfache Diff-Datei . . . . .	197
10	Beispiel: Auszug aus einer Doclog-Datei . . . . .	200
11	CRTP-Demonstration: Allgemein . . . . .	287
12	CRTP-Demonstration: SeqAn . . . . .	289
13	DDDoc-Beispiel für die Template-Spezialisierung <b>SimpleScore</b> . . . . .	313
14	Dox-Beispiel für die Template-Spezialisierung <b>SimpleScore</b> . . . . .	314
15	Dox-Beispiel für die zur Klasse <b>Score</b> gehörige Interface-Funktion <b>scoreGapOpen</b> . . . . .	314
16	Beispiel: CodeStore.xml . . . . .	362
17	Beispiel: ACM-Datei . . . . .	364



# ABKÜRZUNGEN

**API** Application Programming Interface

**GTM** Grounded Theory Methode

**GT** Grounded Theory

**HE** Heuristische Evaluation

**CDF** Cognitive Dimensions Framework

**CD** Cognitive Dimension / kognitive Dimension



## KAPITEL

### 1

## EINLEITUNG

Dieses Kapitel befasst sich mit der Motivation, die sich hinter der Erforschung der API-Usability von SeqAn verbirgt. Es beantwortet die Fragen, was API-Usability überhaupt ist und weshalb sie so wichtig ist. Anschließend wird erläutert, worum es sich bei SeqAn exakt handelt, und welchen Zweck SeqAn erfüllt. Abgeschlossen wird dieses Kapitel mit der Vorstellung, der in dieser Arbeit verwendeten Grounded Theory Forschungsmethode.



## MOTIVATION

Diese Arbeit verfolgt das Ziel, die Benutzerfreundlichkeit<sup>1</sup> (engl. *Usability*) der Softwarebibliothek *SeqAn* im Rahmen einer explorativen empirischen Fallstudie zu verbessern. Wie ich<sup>2</sup> im nächsten Kapitel noch erläutern werde, ist die klassische Usability-Verbesserung von Endanwender-Programmen bereits gut erforscht und, für sich genommen, lediglich anspruchsvolles Handwerk. Anders sieht es aus, wenn beispielsweise neuartige Bedienkonzepte entwickelt, evaluiert und verbessert werden sollen.

Im Falle von SeqAn sind gleich zwei Aspekte interessant:

1. SeqAn ist keine Anwendersoftware, mit der durch eine GUI<sup>G</sup> interagiert wird, sondern eine Softwarebibliothek, die mittels einer weiter unten definierten Applikationsprogrammierschnittstelle (engl. *application programming interface*, kurz: API) verwendet wird.  
Oder anders ausgedrückt: Während man in “normalen Programmen” herum klickt, muss man bei SeqAn mit Hilfe einer selbst gewählten Entwicklungsumgebung programmieren.
2. SeqAn löst keine Informatik-eigenen Probleme wie Persistierung oder Logging, sondern bietet Funktionen zur Lösung von Problemen aus der Bioinformatik an.

Schaut man genauer hin, ergeben sich drei anspruchsvolle, zu lösende Problemfelder:

1. Im Bereich der Forschung wurde der Usability von APIs weit weniger Aufmerksamkeit geschenkt, als der Usability von Endanwender-Programmen (Grill et al. 2012). Erst seit Kurzem hat das Interesse an API-Forschung wegen des ansteigenden Gebrauchs von APIs zugenommen (Daughtry et al. 2009a). 2009 fand das erste und bis dato einzige Treffen der *Special Interest Group “API Usability”* auf der CHI 2009<sup>3</sup> statt (Daughtry et al. 2009b). Wie ich im nächsten Kapitel zeigen werde, ist der Korpus an empirischer Grundlagenforschung klein und es fehlen umfassende wissenschaftliche Literaturstudien.

<sup>1</sup> Die Übersetzung des englischen Begriffs *Usability* ins Deutsche ist kompliziert. Nach dem Inhalt der DIN EN ISO 9241 müsste man den Begriff “Gebrauchstauglichkeit” verwenden, wohingegen — nach meinem persönlichen Eindruck — der Begriff “Benutzerfreundlichkeit” von der deutschen Bevölkerung besser verstanden und vorgezogen wird. Ich werde daher alle drei Begriffe synonym verwenden. In Abschnitt 2.3 gehe etwas genauer auf dieses Problem ein.

<sup>2</sup> Diese Arbeit ist in der Ich-Form verfasst. Die Begründung für diesen Entschluss ist in der verwendeten Forschungsmethode begründet, die ich ab Seite 47 vorstelle.

<sup>3</sup> <http://www.chi2009.org>

2. SeqAns Alleinstellungsmerkmal ist seine kompromisslose Effizienz. Um diese zu erreichen, basiert SeqAn auf dem sehr anspruchsvollen Programmierparadigma *Templatemetaprogrammierung*<sup>4</sup>, zu der es keinerlei Erkenntnisse in Bezug auf die Usability gibt.
3. Das Spektrum an Anwendern von SeqAn ist breit und reicht vom professionellen Entwickler bis hin zum Biologen mit minimalen Programmierfertigkeiten. Letzterer gehört der Gruppe der *Endanwender-Programmierer* an (Tisdall 2001). Die Forschung im Bereich der Endanwender-Programmierung hat sich allerdings vornehmlich auf visuelle Programmiersprachen<sup>5</sup> konzentriert (Gieselmann 2014; Ko et al. 2011), zu der die von SeqAn verwendete Programmiersprache C++ nicht gehört.

Im Folgenden möchte ich klären, was Bioinformatik überhaupt ist, weshalb Bioinformatiker APIs benötigen und welchen Nutzen APIs im Speziellen wie auch im Allgemeinen haben.

### 1.1.1 Was ist überhaupt Bioinformatik?

Die Bioinformatik ist eine interdisziplinäre Wissenschaft, die Methoden der Informatik auf biologische Daten und Fragestellungen anwendet (Gibas u. Jambeck 2002). Quellen für biologische Daten sind dabei Sequenzierungen von Genomen, Expressionsprofile von Proteinen, Strukturaufklärung von Proteinen und Interaktionen zwischen Biomolekülen wie Proteinen, RNAs und niedermolekularen Verbindungen (Rarey 2010).

Mit jedem Jahr werden immer größere Datenmengen produziert. Allein das menschliche Genom umfasst etwa 3 Milliarden Basenpaare<sup>6</sup> (Tisdall 2001). Deutlich kleinere Genome können bei Auswertungsschritten bereits mehrere Terabytes große Datenmengen verursachen (Reinert et al. 2014). Der Zuwachs an Daten hat mittlerweile ein exponentielles Wachstum angenommen. Für die Vorbereitung, Auswertung und Analyse dieser Datenmassen werden Werkzeuge benötigt, die durch die Bioinformatik bereitgestellt werden (Gibas u. Jambeck 2002).

Das Hauptaufgabengebiet der Bioinformatik liegt in der Sequenzanalyse. Darunter wird die Analyse von biomolekularen Sequenzen, wie der DNA, RNA oder Aminosäuren verstanden. Diesem Vorgehen liegt die Beobachtung zu Grunde, dass eine hohe Sequenzähnlichkeit für gewöhnlich eine signifikante,

---

<sup>4</sup> Dass es sich bei der Templatemetaprogrammierung um die Hauptursache für SeqAns fragliche Usability handeln würde, schien allen Beteiligten mehr oder minder klar zu sein. Wenn ich im diesen Sinne von der Templatemetaprogrammierung spreche, handelt es sich allerdings um einen Vorgriff auf mein in dieser Arbeit vorgestelltes Forschungsergebnis. Bewusst habe ich auch andere Ursachen in Betracht gezogen, die ich ebenfalls in dieser Arbeit vorstelle.

<sup>5</sup> Definition nach Schiffer (1998): "Eine visuelle Sprache ist eine formale Sprache mit visueller Syntax oder visueller Semantik und dynamischer oder statischer Zeichengebung."

<sup>6</sup> Das Basenpaar beschreibt ein Paar Nukleobasen. Die wohl bekanntesten Basen sind **A**dénin, **C**ytosin, **G**uanin und **T**hymin. Sie treten bei der DNA als die Paare A-T und C-G auf.

funktionale oder strukturelle Ähnlichkeit impliziert (Reinert et al. 2011). Zu den ersten Bioinformatikern gehören Needleman u. Wunsch (1970), auf deren Algorithmus heute noch viele Methoden zur Sequenzanalyse basieren (Hansen 2013).

In den vergangenen Jahren sind die Kosten für Hochdurchsatz-Sequenzierungsverfahren, die auch als *Next Generation Sequencing* (NGS) bezeichnet werden, rapide gefallen. Dies führt zu einer extrem schnellen Entwicklung im Feld der Bioinformatik (Li et al. 2012). So haben beispielsweise Alexander et al. (2012) ein Verfahren entwickelt, das einen potentiell vorliegenden Schilddrüsenkrebs ohne chirurgischen Eingriff mit hoher Wahrscheinlichkeit ausschließen kann. Dieses Verfahren kann Patienten die unnötige Entfernung ihrer Schilddrüse und die damit einhergehende lebenslange Hormontherapie ersparen.

Bereits heute ist die Sequenzierung eines menschlichen Genoms für weniger als 1.000\$ zu haben (Young 2014); in wenigen Jahren wird die 100\$-Grenze unterschritten sein (Reinert et al. 2014). Diese Entwicklung ermöglicht vollkommen neue Anwendungsformen, z.B. in den Bereichen der personalisierten Medizin, Metagenomik und der klinischen Forschung (Reinert et al. 2014). SAP-Mitgründer Dietmar Hopp spricht sogar davon, “dass Life Sciences die nächste Welle nach der Informationsverarbeitung sein wird” und hat bereits mehr als eine Milliarde Euro in den Bereich der Biotechnologie investiert (Ziegler 2015). Auf staatlicher Ebene werden, teil bereits seit Jahren, nationale Großprojekte verfolgt (Young 2014). Ein solches ist das *Genomics England* Projekt<sup>7</sup>, das durch den staatlichen Gesundheitsanbieter *National Health Service*<sup>8</sup> geplant wird. Dieses Projekt sieht vor, bis 2017 die Gendaten von insgesamt 100.000 Patienten vollständig zu erfassen und in einem ersten Schritt für die Erforschung von seltenen Krankheiten, Krebs und Infektionskrankheiten zu nutzen.

Die auch als *Bioethik* bezeichnete ethische Auseinandersetzung setzt sich in einer überraschenden Breite sowohl mit ganz grundsätzlichen (de Bouvet et al. 2006), wie auch methodischen (Vayena et al. 2015), als auch didaktischen (Gasparich u. Wimmers 2014) Fragestellungen auseinander. Auch hoch aktuelle Themen, wie der immer noch andauernde Ebola-Ausbruch in Westafrika im Frühjahr 2014, sind Gegenstand des ethischen Diskurses (Blais u. White 2015). Auf nationaler Ebene beschäftigt sich der Ethikrat der Bundesrepublik Deutschland auch mit der Bioethik<sup>9</sup>. Auf internationaler Ebene ist dafür das bereits 1993 eingerichtete *International Bioethics Committee*<sup>10</sup> zuständig.

## 1.1.2 Warum müssen Bioinformatiker und Biologen programmieren?

Wie bereits erwähnt, entwickelt sich die Bioinformatik rasant. Während die Datenmengen stetig steigen, entstehen immer wieder neue Algorithmen und Hypothesen (Letondal 2006). Diese Entwicklung

7 <http://www.genomicsengland.co.uk>

8 <http://www.nhs.uk>

9 <http://www.ethikrat.org/veranstaltungen/anhoerungen/praediktive-genetische-diagnostik-multifaktorieller-erkrankungen>

10 <http://www.unesco.org/new/en/social-and-human-sciences/themes/bioethics/international-bioethics-committee/>

führt dazu, dass Probleme formuliert werden, für deren Lösung es noch keine (Tisdall 2001) oder nur unzureichende (Letondal 2006) grafische Werkzeuge existieren.

Während es zu den fachbedingten Tätigkeiten der Bioinformatik gehört, Code schreiben zu müssen, trifft dies nun auch auf Biologen zu. Anders wäre es ihnen nicht möglich, neuartige Hypothesen zu überprüfen (Letondal 2006; Tisdall 2001). Damit bilden Biologen klassische Vertreter der Endanwender-Programmierer.

Unabhängig von verfügbaren grafischen Werkzeugen gibt es einen immensen Automatisierungsbedarf in Form von Stapelverarbeitungsskripten (engl. *batch scripts*) und *Workflows*. Abgesehen von der geringeren Fehlerquote wird Automatisierung durch den enormen Zeitgewinn motiviert: Während die händische Auswertung von Gen-Abschnitten einen Monat oder mehr beanspruchen kann, kann ein in zwei Wochen entwickeltes Skript dieselbe Arbeit in 30 Minuten leisten (Tisdall 2001). Bedenkt man das Alter der Quelle und die fortschreitende Entwicklung von Informatik und Bioinformatik, wird dieser Zeitgewinn sicherlich gewachsen sein.

Die Bioinformatik verfügt bereits über ein großes Repertoire an Algorithmen, die Lösungsbestandteile neuartiger Probleme darstellen. Existierende Implementierungen werden über Softwarebibliotheken, Frameworks, etc. bereitgestellt; sie wiederzuverwenden verringert die Anzahl an produzierten Defekten in den entwickelten Werkzeugen und erhöht zugleich die Effizienz der Arbeit.

### 1.1.3 Warum ist die Benutzerfreundlichkeit von APIs überhaupt so wichtig?

Moderne Softwareentwicklung macht ausgiebig Gebrauch von dem Prinzip der Wiederverwendung (Piccioni et al.). Wiederverwendbarer Code wird am häufigsten in Form von *Frameworks* und *Softwarebibliotheken* bereitgestellt (Henning 2007; Jeong et al. 2009), wobei letztere am häufigsten von Anwendern genutzt werden (Feilkas u. Ratiu 2008). Die Nutzung von existierendem qualitativem Code hat gegenüber der Neuimplementierung potentiell die Vorteile geringerer Kosten sowie eine höhere Sicherheit und Stabilität (Stylos 2009).

Der Zugriff auf eine Softwarebibliothek wird über eine API realisiert. Weil es sich dabei um die Schnittstelle zwischen Anwender und Softwarebibliothek handelt, ist die Usability ein fundamentales Ziel bei der Entwicklung der API (Piccioni et al.).

APIs sind schwer zu verwenden (Stylos 2006) und schwer zu erlernen, weil nicht nur die einzelne Klasse, sondern auch das Zusammenspiel der Klassen wichtig ist (Bruch et al. 2006). Das gilt sogar für etablierte APIs wie dem Java Development Kit<sup>11</sup> (Stylos et al. 2009a). In besonders schweren Fällen kann der Verwender einer API sogar so stark ausgebremst werden, dass er schneller ohne die Verwendung der API gewesen wäre (Robertson 2007).

---

<sup>11</sup> <http://www.oracle.com/technetwork/java/javase>

Eine schlechte API-Usability verringert die Produktivität und Codequalität (Zibran et al. 2011) bei der Entwicklung von Programmen mit Hilfe dieser API. Inzwischen ist eine *API-Ökonomie* entstanden (Hülsenbusch 2014), bei der sich die Benutzerfreundlichkeit natürlich auch auf die Wettbewerbsfähigkeit auswirkt (Stylos 2009). Ist eine API zu schwer zu erlernen, wechseln die Anwender eben auf eine andere API (Sunshine et al. 2014). Aus humanistischer und arbeitspsychologischer Sicht wirkt sich die Arbeit mit einem benutzerunfreundlichen System negativ auf Wohlbefinden und Motivation aus (Sarodnick u. Brau 2006).

### 1.1.4 Lohnt sich wirklich der Aufwand zur Verbesserung der Benutzerfreundlichkeit von APIs?

Diese Frage ist nicht nur für die Leser und Geldgeber dieser Arbeit sowie für die Autoren von APIs von Wichtigkeit, sondern auch für mich selbst. Schließlich stand ich vor der Wahl, eine Promotion über einen Zeitraum von drei oder mehr Jahren zu verfolgen und fachlich wie auch motivatorisch zu meistern.

Etablierte APIs werden tausendfach häufiger verwendet als entwickelt (Ellis et al. 2007). Die Mehrkosten für die Autoren von APIs sind vielfach kleiner als der Schaden, den benutzerunfreundliche APIs ihren Anwendern anrichten (Henning 2007). Die Verantwortung für eine benutzerfreundliche API liegt also klar bei seinen Autoren. Außerdem ist eine einmal veröffentlichte API quasi ewig — es existiert also nur eine Chance, sie korrekt zu entwerfen (Bloch 2006a).

Wird die API-Usability nicht explizit beim Entwurf beachtet, ist sie weiter nichts als ein extrem unwahrscheinlicher Zufall (Stylos 2009). Bereits kleine und unschuldig anmutende Mängel tendieren dazu, sich negativ in überproportionalen Größenordnungen auszuwirken (Henning 2009). Robertson (2007) bezeichnen dies trefflich als “death by 1000 paper cuts”.

Für die besonders anspruchsvolle Anwendergruppe der Endanwender-Programmierer, zu denen auch Biologen (Tisdall 2001) und teilweise auch Bioinformatiker (Letondal 2006) gehören, ist das Schadpotential besonders groß. Schließlich verfügt diese Gruppe über weit weniger Erfahrung im Bereich der Softwaretechnik und kann Usability-Probleme weniger gut kompensieren, als ihre erfahrene Kollegen (Shaft u. Vessey 1998).

Die Verbesserung der API-Usability ist möglich und wurde in einigen Studien gezeigt (Gerken et al. 2011; Grill et al. 2012; Letondal 2006; Piccioni et al.). Wie meine, im Abschnitt 2 vorgestellte Literaturrecherche zeigt, ist der Wissensstand in diesem Bereich jedoch noch stark fragmentiert. Ein Grund dafür ist — neben der Vernachlässigung von API-Usability zu Gunsten der klassischen Usability — die Vielzahl an Einflussfaktoren, zu denen, neben der API selbst, auch die dazugehörige Dokumentation, die gewählte Programmiersprache und Entwicklungswerkzeuge gehören (Stylos u. Myers 2008).

### **1.1.5 Synthese und Zusammenfassung**

APIs bilden die Schnittstelle zu wiederverwendbarem Code. Ohne diese Schnittstelle ist der Nutzer gezwungen, Code zu duplizieren, was zu einer ganzen Fülle von Nachteilen führt (Spinellis 2006). Glücklicherweise wird wiederverwendbarer Code häufig in Form von Softwarebibliotheken gekapselt und durch APIs zugänglich gemacht.

Eine benutzerunfreundliche API führt zu einer Reihe von Problemen, die sich negativ auf die ganze Spannbreite von persönlichen bis hin zu wirtschaftlichen Erfolgsfaktoren auswirken. Um diesen potentiellen Schaden abzuwenden, müssen sich die Entwickler einer API aktiv für dessen Usability einsetzen. Bei ihnen liegt aus fachlichen und ökonomischen Gründen die Verantwortung dafür. Leider ist diese Anforderung leichter formuliert als erfüllt, denn die Forschung im Bereich der API-Usability steht der Forschung im Bereich der klassischen Usability um einiges nach. Hinzu kommt, dass es an umfassenden wissenschaftlichen Literaturstudien fehlt.

Die Bioinformatik entwickelt Werkzeuge, mit denen Problemstellungen gelöst werden können, die unter anderem zu einem besseren Verständnis von Krankheiten und deren Heilungsmöglichkeiten führen. Dazu greifen Bioinformatiker auf Softwarebibliotheken zurück. Eine solche Softwarebibliothek ist SeqAn, die das Leben ihrer Anwender durch den intensiven Gebrauch des Programmierparadigmas Template-metaprogrammierung nicht unbedingt leicht macht. Zu allem Überfluss ist dieses Paradigma in Bezug auf seine Benutzerfreundlichkeit vollkommen unerforscht.

Die allgemein formulierten Nachteile einer schlechten API-Usability bedeuten für den Bereich der Bioinformatik, dass die Forschung im besten Fall ausgebremst wird. Eine gute API-Usability hingegen erlaubt es nicht nur Bioinformatikern effizienter zu arbeiten, sondern kann — unter Berücksichtigung von Erkenntnissen aus der Endanwender-Softwaretechnik — Biologen in den Zustand versetzen, sich ebenfalls bioinformatisch zu betätigen. Dass dies tatsächlich möglich ist, konnte in einer Arbeit gezeigt werden (Letondal 2006), die ich im Abschnitt 2.5.1 vorstelle.

Als Forscher und Autor dieser Dissertation beschäftige ich mich also mit den Aufgaben, (1) eine umfassende wissenschaftliche Literaturstudie durchzuführen, (2) die Usability von Templatemetaprogrammierung im Allgemeinen und von SeqAn im Speziellen empirisch zu erforschen, (3) die besonderen Anforderungen von Endanwender-Programmierern zu berücksichtigen, (4) Vorschläge für die Verbesserung der Usability von SeqAn zu formulieren und (5) diese umzusetzen.

## DEFINITIONEN

An dieser Stelle möchte ich knapp auf Übersetzungsschwierigkeiten vom Englischen ins Deutsche eingehen: Bekanntermaßen ist die Informatik durch englischsprachige Begriffe geprägt. Außerdem wird der mit Abstand größte Teil wissenschaftlicher Veröffentlichungen im Bereich der Informatik in englischer Sprache verfasst. Für viele englische Begriffe gibt es etablierte Übersetzungen (z.B. *interface* & *Schnittstelle*). Andere Begriffe lassen sich hingegen nur mühselig (z.B. *usability* & *Gebrauchstauglichkeit*/*Nutzbarkeit*/*Benutzerfreundlichkeit*) oder gar missverständlich (z.B. *framework* & *Rahmenwerk*) übersetzen.

Aus diesen Gründen werde ich für den Rest dieser Arbeit nach Möglichkeit den deutschen Begriff verwenden. Würde eine Übersetzung jedoch unnötig kompliziert sein oder gar missverstanden werden können, verwende ich den englischen Begriff. Wenn es sich um kein direktes Zitat handelt, schreibe ich englische Substantive mit großem Anfangsbuchstaben und verwende den sächlichen Artikel. Bei Abkürzungen, die sowohl den englischen als auch deutschen Begriff beschreiben (z.B. *API*), nutze ich den Artikel des deutschen Begriffs (z.B. *die API*). Von diesen Vereinbarungen mache ich gleich bei der folgenden ersten Definition Gebrauch.

Basierend auf existierenden Definitionen (Dekel 2011; Gerken et al. 2011; Henning 2007; McLellan et al. 1998; Robillard 2009; Stylos 2009; Zibran et al. 2011; Zibran 2008), definiere ich *API* wie folgt:

Eine *Applikationsprogrammierschnittstelle* (engl. *application programming interface*, kurz: *API*) ist eine Schnittstelle zwischen Anwender und dem der Schnittstelle zu Grunde liegenden Code, der eine bestimmte Funktionalität implementiert. Eine API liegt in Gestalt einer Softwarebibliothek, eines Frameworks, SDKs, Toolkits, etc. vor, umfasst alle dazu in Verbindung stehenden Ressourcen (Installationsdateien, Dokumentation, etc.) und definiert dabei eine Dienstleistung, welche die Komplexität der Implementierung mittels Abstraktion versteckt. Bei APIs handelt es sich um eine fortgeschrittene Form der Code-Wiederverwendung, die den Bau komplexer Software ermöglicht.

Obwohl es eine Reihe Veröffentlichungen im Bereich der API-Usability gibt, hat sich erstaunlicherweise noch keine grundlegende einheitliche Terminologie für die folgenden drei relevanten Parteien entwickelt. Einige Arbeiten verwenden zumindest den Begriff *API user* — ohne ihn jemals explizit eingeführt zu haben. Ich schlage daher folgende Begriffe vor:

Ein *API-Entwickler* (engl. *API developer* bzw. *API designer*) ist eine Person, die an der Entwicklung einer API beteiligt ist. Neben der eigentlichen Programmierung gehören auch Konzeption, Entwurf oder Wartung in ihre Arbeitsbereiche.

Ein *API-Anwender* (engl. *API user*) ist eine Person, die eine oder mehrere APIs nutzt, um ein Problem programmatisch zu lösen.

Da sich diese Arbeit mit einer Software befasst, die nicht ausschließlich von professionellen Softwareentwicklern genutzt wird, muss sie sich auch mit der Endanwender-Softwaretechnik auseinandersetzen. Abschnitt 2.4.1 befasst sich näher mit diesem Thema. An dieser Stelle möchte ich lediglich den Begriff des *Endanwender-Programmierers* herausgreifen.

Ein *Endanwender-Programmierer* (engl. *end-user programmer*) ist eine Personen, deren Ziel — im Gegensatz zum professionellen Programmierer — nicht das Programmieren selbst ist. Stattdessen ist Programmieren für sie ein “notwendiges Übel” zur Erreichung ihres Ziels innerhalb ihrer Expertendomäne.

Auf den Bereich der API-Usability übertragen, können API-Entwickler und API-Anwender um eine weitere Rolle ergänzt werden:

Ein *API-Endanwender* (engl. *API end-user*) ist ein spezieller Endanwender-Programmierer, der eine API zum Erreichen seines Ziels nutzt.

Abhängig vom Kontext verwende ich den Begriff *API-Anwender* als Überbegriff für *API-Anwender* und *API-Endanwender* — also als undifferenzierte Bezeichnung für alle Anwender einer API.

Wie ich weiter unten noch erläutern werde, ist die dieser Arbeit zu Grunde liegende Forschungsmethode die *GTM<sup>G</sup>*. Diese Methode verwendet den englischen Begriff *Code*, der leider mit dem Code-Begriff aus der Informatik kollidiert. Für den Rest dieser Arbeit werde ich für den GTM-Code-Begriff *Kode* bzw. *Konzept* und für den Informatik-Code-Begriff *Code* bzw. *Quellcode* benutzen.

Leider steht der GTM-Konzept-Begriff wiederum im Konflikt mit dem Konzept-Begriff aus der Templatemetaprogrammierung. Daher werde ich für den GTM-Konzept-Begriff *Konzept* und für den Templatemetaprogrammierungs-Konzept-Begriff *Concept* verwenden. Die Eselsbrücke lautet: Für nicht-technische Gebiete verwende ich die deutsche Übersetzung und für technische Konfliktbegriffe das englische Wort im Original.

## SEQAN

SeqAn<sup>12</sup> ist eine quelloffene C++-Softwarebibliothek, die von Andreas Gogol-Döring im Rahmen seiner Dissertation (Gogol-Döring 2009) entwickelt wurde. Sie dient ihren Anwendern zur “Verarbeitung großer Mengen biomedizinischer Daten, insbesondere von Gen- und Proteinsequenzen” und mir als Forschungsgegenstand für API-Usability.

Typische Anwendungen sind (Reinert et al. 2014):

### **Indices**

Suche von Mustern in Sequenzen

### **Alignment**

Lokaler, globaler, semi-globaler, approximativer, etc. Vergleich von zwei oder mehr Sequenzen

### **Read-Mapping**

Zusammensetzung einer in vielen Fragmenten (engl. *reads*) vorliegenden Sequenz

### **Assemblies**

Auffinden von genomischen Varianten

Bei der Entwicklung von SeqAn hatte Gogol-Döring (2009) zwei Anwendergruppen im Auge:

**Softwareentwickler**, die, neue Softwarewerkzeuge für die biologische und medizinische Forschung entwickeln und

**Algorithmendesigner**, die neue Algorithmen entwickeln, testen und vergleichen wollen.

Für die Verwendung von SeqAn stehen zwei Möglichkeiten zur Verfügung: Einerseits kann SeqAn als Softwarebibliothek verwendet werden. Dieser Anwendungsfall ist der von mir in dieser Arbeit beleuchtete. Andererseits gibt es die Möglichkeit, die im Abschnitt 1.3.3.5 beschriebenen und auf SeqAn basierenden Werkzeuge zu nutzen — dieser Anwendungsfall wird in dieser Arbeit nicht beleuchtet, da es dabei nicht um API-, sondern um klassische Usability geht.

Die immer größeren Datenmengen erfordern automatisierte Analyseverfahren, die eine hohe Sorgfalt und Verfahrenssicherheit aufweisen. Die wiederum stetig fallenden Sequenzierungskosten machen solche Verfahren auch für kleinere Pharmafirmen und Labore interessant (Reinert et al. 2014). Für diese Anwendergruppe ist der Einsatz von SeqAn besonders interessant.



ABBILDUNG 1.1: Logo der Softwarebibliothek *SeqAn*

Bei der quelloffenen, freien Bereitstellung der Bibliothek gestaltet sich die Monetarisierungsform, die auf den Verkauf von Nutzungslizenzen setzt, schwierig. Daher setzen die SeqAn-Entwickler auf ein Beratungsmodell, bei dem die Nutzung von SeqAn kostenlos ist und die Möglichkeit besteht, beratende und entwickelnde Tätigkeiten kostenpflichtig in Anspruch zu nehmen.

### 1.3.1 Entwurf

SeqAn wurde mit dem Ziel entwickelt, eine hohe Performanz, Allgemeingültigkeit, Erweiterbarkeit und Integration mit anderen Bibliotheken zu garantieren und dabei benutzerfreundlich zu sein (Gogol-Döring 2009; Gogol-Döring u. Reinert 2009). In die Entwicklung von SeqAn sind circa 18 Personenjahre investiert worden (Reinert et al. 2014). Wie man unschwer erahnt, wird diese Arbeit zeigen, dass es um die Benutzerfreundlichkeit nicht gut bestellt ist.

Die größte Auffälligkeit beim Entwurf von SeqAn ist, dass auf objektorientierte Programmierung zu Gunsten von Templatemetaprogrammierung verzichtet wird. Der Entwicklung von SeqAn liegen vier von Gogol-Döring (2009) als “Techniken” beschriebene Entwurfsansätze zu Grunde:

#### Generische Programmierung

SeqAn setzt generische Programmierung ein — ein Paradigma, das auch der *C++ Standard Template Library (STL)* zu Grunde liegt (Plauger 2001). Bei diesem Ansatz werden Algorithmen und Datenstrukturen derart entworfen, dass sie auf allen Datentypen operieren, welche die an sie gestellten Minimalanforderungen erfüllen.

#### Template-Subclassing

Diese Technik wird verwendet, um in SeqAn Polymorphismus zu ermöglichen. Dazu erhalten Klassen (z.B. `Class`) mindestens einen Templateparameter `TSpec`, mit dessen Hilfe dann Verfeinerungen beliebiger Tiefe vorgenommen werden können (z.B. `Class<SubClass>`, aber auch `Class<SubClass<SubSubClass>>`). Dies ermöglicht es, spezialisierte generische Funktionen für bestimmte Eingaben bereitzustellen, die ein Problem effizienter lösen, als dies die Standardimplementierung zulässt.

Gepaart mit der generischen Programmierung führt diese Technik zu höchst performanten Programmen, denn die korrekte Implementierung für einen Funktionsaufruf wird zur Kompilier- und nicht zur Laufzeit ermittelt.

### Globales Funktionen-Interface

Dieser Ansatz ist mehr eine Notwendigkeit als ein Feature, denn der Einsatz von Template-Subclassing erfordert die Bereitstellung aller verfeinerbaren Funktionen in einem gemeinsamen Namensraum, was in globalen Funktionen resultiert. Damit verlieren die Funktionen technisch gesehen ihre Zugehörigkeit zu einer Klasse. Semantisch existiert diese Zugehörigkeit dennoch. Die semantisch zu einem Typ gehörenden Funktionen werden als globales Funktionen-Interface bezeichnet. Anstatt also `obj.fn()` aufzurufen, müssen API-Anwender `fn(obj)` verwenden.

Für die API-Anwender ergeben sich Probleme, die entweder zu Stande kommen, weil eine Funktion aufrufbar ist, die für den in Frage stehenden Typ eigentlich gar nicht definiert sein sollte (z.B. die `length`-Funktion<sup>13</sup>) oder weil gleichnamige Funktionen — abhängig von ihrer Eingabe — nun plötzlich vollkommen verschiedene Dinge erledigen (z.B. die `label`-Funktion<sup>14</sup>). Die Folgen werden ausführlich in den Ergebnissen in Kapitel 4 erläutert.

Um die Kompatibilität zu anderen Bibliotheken, wie der *C++ Standard Library (STD)*, herzustellen, werden *Shims* verwendet. Dabei handelt es sich um globale Funktionen, welche die existierenden Memberfunktionen einfach durch einen entsprechenden Funktionsaufruf kapseln. Beispiel: Die Funktionen des globalen Funktionen-Interfaces für SeqAn-Strings können auch auf C++-Strings operieren, indem sie auf die jeweilige Memberfunktionen des C++-Strings zugreifen. Das globale `length`-Shim für C++-Strings wurde wie folgt implementiert, wobei der relevante Aufruf in Zeile 6 stattfindet:

```

1 template <typename TChar, typename TCHARTraits, typename TAlloc>
2 inline typename Size< std::basic_string<TChar, TCHARTraits, TAlloc> >::Type
3 length(std::basic_string<TChar, TCHARTraits, TAlloc> const &me)
4 {
5     SEQAN_CHECKPOINT;
6     return me.length();
7 }
```

### Metafunktionen

Metafunktionen sind Funktionen, die nicht zur Lauf-, sondern zur Kompilierzeit ausgeführt werden. In SeqAn werden sie hauptsächlich dazu verwendet, zu einem gegebenen Eingabetyp den dazugehörigen Ausgabetyp zu berechnen. Dieser Ansatz findet beispielsweise bei der Erzeugung von Iteratoren Anwendung, denn der Typ des Iterators hängt von der Eingabe der erzeugenden

---

<sup>13</sup> <http://docs.seqan.de/seqan/develop/?p=ContainerConcept#length>

<sup>14</sup> <http://docs.seqan.de/seqan/develop/?p=Fragment#label> bzw.  
<http://docs.seqan.de/seqan/develop/?p=EdgeIterator#label>

Laufzeit-Funktion `begin` ab<sup>15</sup>. Implementiert werden Metafunktion in SeqAn mittels Klassen-templates. Das folgende Beispiel zeigt die Anwendung der Metafunktion `Iterator` in den Zeilen 2 und 3:

```

1 Dna5String genome = "TATANNNNGCGCG";
2 Iterator<Dna5String>::Type it = begin(genome);
3 Iterator<Dna5String>::Type itEnd = end(genome);
4
5 while (it != itEnd)
6 {
7     std::cout << *it;
8     ++it;
9 }
10 std::cout << std::endl;
```

## Tag Dispatching

Mit Hilfe der generischen Programmierung und dem Template-Subclassing ist ein C++-Compiler in der Lage, die passendste Implementierung für einen Funktionsaufruf zur Kompilierzeit zu ermitteln. Allerdings möchte man auf diesen Prozess, beispielsweise bei der Wahl zwischen alternativen Alignment-Algorithmen, Einfluss nehmen können. Dazu kann der entsprechenden Funktion ein sogenannter, als Member-lose Klasse implementierter, *Tag* übergeben werden. Für die Auswahl einer Alignment-Implementierung stehen beispielsweise `NeedlemanWunsch` und `MyersHirschberg` zur Verfügung.

### 1.3.2 Beispiel

Um den obigen technischen Erläuterungen ein wenig Leben einzuhauen, stelle ich an dieser Stelle ein typisches, knappes SeqAn-Programm vor und wie dessen Entsprechung in Java aussehen könnte. Das Beispiel soll einerseits dabei helfen, einen besseren Eindruck von der Gestalt von auf SeqAn basierenden Programmen zu bekommen. Andererseits soll es das Verstehen dieser Arbeit erleichtern, weil ich immer wieder auf die Entwurfsentscheidungen von SeqAn zu sprechen kommen werde und diese in der Vorstellung meiner Analyseergebnisse eine große Rolle spielen.

Das folgende Programm berechnet das Alignment von zwei DNA-Sequenzen und gibt die Abweichung (*Score*), eine textgrafische Darstellung und die reverse DNA-Sequenz der ersten Sequenz aus:

```

1 int main()
2 {
```

---

<sup>15</sup> <http://docs.seqan.de/seqan/develop/?p=ContainerConcept#begin>

```

3   typedef String<Dna> TSequence;
4   typedef Align<TSequence, ArrayGaps> TAlign;
5
6   TSequence seq1 = "ATAAGCGTCTCG";
7   TSequence seq2 = "TCATAGAGTTGC";
8
9   TAlign align;
10  resize(rows(align), 2);
11  assignSource(row(align, 0), seq1);
12  assignSource(row(align, 1), seq2);
13
14  int score = globalAlignment(align, Score<int, Simple>(0, -1, -1));
15  std::cout << "Score: " << score << std::endl;
16  std::cout << align << std::endl;
17
18  std::cout << "Complement of sequence #1: ";
19  Iterator<TSequence, Rooted>::Type it = end(seq1);
20  for (goEnd(it); !atBegin(it); goPrevious(it))
21  {
22      std::cout << getValue(it);
23  }
24  std::cout << std::endl;
25  return 0;
26 }
```

LISTUNG 1: Beispiel — Typisches SeqAn-Programm.

**Template-Subclassing:** In Zeile 3 wird eine Spezialisierung der String-Klasse speziell für Nukleotide definiert. In Zeile 4 wird die Datenstruktur definiert, die sowohl als Ein- als auch Ausgabe für die Berechnung des Alignments in Zeile 14 dient. Der zweite Template-Parameter spezifiziert, in welcher Weise das Alignment intern dargestellt wird. Diese Wahl beeinflusst das Laufzeitverhalten für bestimmte Operationen. **Globales Funktionen-Interface:** In den Zeilen 11 und 12 wird die globale `assignSource`-Funktion verwendet, die semantisch eine Memberfunktion der `Align`-Klasse ist.<sup>16</sup> Im Gegensatz dazu ist die `globalAlignment`-Funktion technisch, wie auch semantisch, globale. **Generische Programmierung:** Die `resize`-Funktion in Zeile 10 ist ein hervorragendes Beispiel, denn sie operiert auf jeder Eingabe, deren Größe verändert werden kann. Tatsächlich ist das gesamte Programm generisch. Schließlich kann das Alignment für beliebige Elemente berechnet werden, wenn `String` in Zeile 3 entsprechend spezialisiert wird und der dort angegebene Datentyp den Gleichheitsoperator unterstützt. In Zeile 19 wird die **Metafunktion Iterator** aufgerufen. Diese Funktion gibt ein `struct` mit dem Feld `Type` zurück, das den Datentyp enthält, der einen Iterator für `TSequence` speichern kann. Das Wissen um die konkrete Implementierung der `Iterator`-Klasse ist nicht nötig.

---

<sup>16</sup> Tatsächlich ist die `assignSource`-Funktion eine Memberfunktion der `Gaps`-Klasse, was an dieser Stelle aber nicht zum Verständnis beiträgt.

Würde man das obige Programm 1:1 nach Java übersetzen, würde das Programm in etwa wie folgt aussehen:

```

1  private static class TSequence extends String<Dna> {};
2  private static class TAlign extends Align<TSequence, ArrayGaps> {};
3
4  public static void main(String[] args) {
5      TSequence seq1 = new TSequence("ATAAGCGTCTCG");
6      TSequence seq2 = new TSequence("TCATAGAGTTGC");
7
8      TAlign align = new TAlign();
9      align.setSize(2);
10     align.getRows().assignSource(seq1, 0);
11     align.getRows().assignSource(seq2, 1);
12
13     int score = Alignment.globalAlignment(align, new Score<Integer, Simple>(0, -1,
14         -1));
15     System.out.println("Score: " + score);
16     System.out.println(align);
17
18     System.out.print("Complement of sequence #1: ");
19     for(Iterator<TSequence, Rooted> iterator = seq1.end(); iterator.hasPrevious();) {
20         System.out.print(iterator.prev());
21     }
22 }
```

LISTUNG 2: Beispiel — Unmittelbare Übersetzung eines typischen SeqAn-Programms in Java

Die unmittelbare Java-Übersetzung wirkt etwas künstlich. Ein Java-Entwickler würde wahrscheinlich die folgende mittelbare Variante als die für ihn am vertrautesten empfinden:

```

1  public static void main(String[] args) {
2      String<Dna> seq1 = new String<Dna>("ATAAGCGTCTCG");
3      String<Dna> seq2 = new String<Dna>("TCATAGAGTTGC");
4
5      Align<String<Dna>, ArrayGaps> align = new Align<String<Dna>, ArrayGaps>(seq1,
6          seq2);
```

```

6
7     int score = Alignment.computeGlobalAlignment(align, new Score<Integer, Simple>(0,
8         → -1, -1));
9     System.out.println("Score: " + score);
10    System.out.println(align);
11
12    System.out.print("Complement of sequence #1: ");
13    Iterator<String<Dna>> iterator = seq1.iterator(seq1.size());
14    while(iterator.hasPrevious()) {
15        System.out.print(iterator.previous());
16    }
17 }
```

LISTUNG 3: Beispiel — Mittelbare Übersetzung eines typischen SeqAn-Programms in Java

Wie man an der Java-Übersetzung sehen kann, unterscheidet sich die Programmierung mit SeqAn deutlich von der objektorientierten Programmierung. In Kapitel 4 diskutiere ich die sich daraus ergebenden Probleme. Ursachen sind insbesondere die technische Auflösung von Memberfunktionen durch die Verwendung von generischer Programmierung im Zusammenspiel mit dem globalen Funktionen-Interface (siehe Abschnitt 4.1.2) und der intensive Gebrauch von Metafunktionen (siehe Abschnitt 4.1.2), welche von den SeqAn-Anwendern nur unzureichend verstanden werden.

Die im Zuge dieser Arbeit umgesetzten und in Abschnitt 4.4 vorgestellten API-Usability-Verbesserungen umfassen ein Tutorial<sup>17</sup>, das sich explizit an SeqAn-Anfänger richtet. Der gewillte Leser erfährt dort noch detaillierter, weshalb und wie die eben beschriebenen vier Grundtechnologien in SeqAn eingesetzt werden.

### 1.3.3 Bestandteile

Die Softwarebibliothek SeqAn besteht aus den im Folgenden vorgestellten Komponenten:

#### 1.3.3.1 Softwarebibliothek

Die herunterladbare Softwarebibliothek, ist das, was man unter SeqAn im engeren Sinne versteht. Dabei handelt es sich um eine Archiv-Datei, die zur Verwendung entpackt werden muss. Das so entstehende

---

<sup>17</sup> <http://seqan.readthedocs.org/en/master/Tutorial/FirstStepsInSeqAn.html>

SeqAn-Verzeichnis enthält die SeqAn-Quellen und insbesondere die `seqan.h`-Header-Datei, die in das eigene Softwareprojekt eingebunden werden kann.

Ursprünglich war SeqAn — entgegen seines Namens — nur als Framework nutzbar, was Usability-Probleme zur Folge hatte. Auf diese Beobachtung gehe ich in den Abschnitten 4.1.2 und 4.1.2 ein.

### 1.3.3.2 Website

Die SeqAn-Website <http://www.seqan.de> ist der erste Anlaufpunkt für SeqAn-Anwender. Die Website informiert über Neuigkeiten rund um SeqAn, bietet Download-Möglichkeiten für die Bibliothek selbst und Anwendungen an und verweist auf alle weiter unten aufgeführten Ressourcen.

### 1.3.3.3 Dokumentation

Die Dokumentation umfasst im engeren Sinne die Beschreibungen der Klassen, Metafunktionen, Funktionen, etc., die von SeqAn bereitgestellt werden. Ursprünglich basierte sie auf der eigens für SeqAn entwickelten Dokumentationssprache *DDDoc* (Gogol-Döring 2009). Die Dokumentation wurde im Zuge dieser Arbeit jedoch auf eine standardkonformere Eigenentwicklung umgestellt, was im Abschnitt 4.3.9 ausführlich erklärt wird.

Die Dokumentation ist Bestandteil der herunterladbaren Archiv-Datei, kann jedoch auch Online unter <http://docs.seqan.de> konsultiert werden. Unter dieser Adresse stehen auch Revisionen zu den Major-Releases von SeqAn bereit.

Im weiteren Sinne umfasst die Dokumentation auch die sogenannten *Tutorials*.

### 1.3.3.4 Tutorials

Tutorials<sup>18</sup> sind unter <http://seqan.readthedocs.org> konsultierbare Anleitungen für die verschiedenen SeqAn-Funktionen. Die Spannbreite reicht von Installationanleitungen über den Gebrauch von Iteratoren bis hin zum Alignment mehrerer Sequenzen.

Die Tutorien wurden ursprünglich über eine selbst verwaltete *trac*<sup>19</sup>-Installation unter <http://trac.seqan.de> gehostet.

---

<sup>18</sup> Eigentlich gibt es für den englischen Begriff *Tutorial* die deutsche Übersetzung *Tutorium*. Der englischsprachige Begriff *Tutorial* ist jedoch derart im Vokabular der SeqAn-Entwickler und -Anwender verwurzelt, dass ich diesen Begriff dem deutschen ebenfalls vorziehe.

<sup>19</sup> <http://trac.edgewall.org>

### 1.3.3.5 Anwendungen

Wie bereits weiter oben beschrieben, gibt es — neben der Nutzung von SeqAn als Softwarebibliothek — die Möglichkeit, lediglich die auf SeqAn basierenden und im Folgenden genannten Werkzeuge zu verwenden. Die Werkzeuge wurden hauptsächlich im Rahmen wissenschaftlicher Veröffentlichungen innerhalb der Bioinformatik-Arbeitsgruppe entwickelt und werden, wenn sie nicht durch eine neue Anwendung abgelöst wurden, von den SeqAn-Entwicklern gepflegt.

#### **ALF — Alignment Free Sequence Comparison**

Berechnung der paarweisen Ähnlichkeit von Sequenzen unter Verwendung von Alignment-freien Methoden

#### **ANISE and BASIL** Assemblierung und Erkennung von eingefügten Sequenzen

**Breakpoint Calculator** Berechnung paarweiser und über drei Wege versteckter Breakpoints

#### **DFI — Deferred Frequency Index** Sequenzielle Mustersuche

**Fiona** Automatische Korrektur von Sequenzfehlern in Sequenzfragmenten (*reads*)

#### **Gustaf — Generic mUlti-spliT Alignment Finder**

Erkennung von Mappings, die einen speziellen Breakpoint-Typ (strukturelle Variante) überschneiden

#### **INSEGT — INtersecting SEcond Generation sequencing daTa with annotation**

Analyse von Alignments von RNA-Sequenzierungs-Reads unter Verwendung von Gen-Annotationen

#### **JST — Journaled String Tree** Datenstruktur zur Nutzung von Daten-Parallelität in einer Menge von ähnlichen Sequenzen

**Lambda** Für Protein-Reads optimiertes Local-Alignment-Werkzeug

**Masai** Schnelle und sensitive Read-Mapper

**Mason** Read-Simulator für Illumina-, 454- und Sanger-Reads

**MicroRazerS** Schnelles Alignment kleiner RNA-Reads

#### **NGS ROI — Region of Interest Analysis for NGS Data**

Framework für die Post-Alignment-Datenanalyse

#### **PISAtoolbox**

Schnelle und adaptive Erzeugung von Markov-Ketten variabler Reihenfolge (*variable order Markov chains*, kurz: *VOMC*)

#### **Rabema — Read-Alignment-BEnchMArk** Durchführung von Read-Alignment-Benchmarks

**RazerS 3** Schnelles, voll-sensitives Read-Mapping (abgelöst durch *Yava*, siehe unten)

**SeqAn::T-Coffee** Segment-basiertes Alignment mehrerer Sequenzen

**SeqCons — Sequence Consensus** Konsistenz-basierter Algorithmus zur Konsens-Berechnung

#### **SnpStore — SNP and Indel Calling in Mapped Read Data**

Berechnung von Vorkommnissen von SNPs und Indels

**SplazerS — Split Read Mapping as an Extension of RazerS** Read-Mapping von Split-Reads

**Stellar — The Swift Exact Local Aligner** Voll-sensitiver, paarweiser Local-Aligner

**String Similarity Search/Join** Berechnung von Sequenz-Ähnlichkeit

**Yara — Yet another read aligner**

Werkzeug für das Alignment von DNA-Sequenzen unter Verwendung von Referenz-Genomen

Die genannten Anwendungen können unter <http://www.seqan.de/projects/> bezogen werden.

### 1.3.3.6 Hosting-Plattform

Die SeqAn-Entwickler verwenden *GitHub*<sup>20</sup> als Hosting-Plattform. Sie dient nicht nur als Code-Repositorium, sondern auch als Issue-Tracker und allgemein als Kommunikationsmöglichkeit zwischen API-Entwicklern und -Anwendern. Das SeqAn-Projekt ist unter <https://github.com/seqan/seqan> erreichbar.

Der Wechsel zu GitHub wurde erst im Laufe dieser Arbeit vollzogen. Zuvor wurde SeqAn mit einer selbstverwalteten Subversion-Installation gehostet.

### 1.3.4 Anwender

Zu den Anwendern von SeqAn gehören<sup>21</sup>:

#### Akademisch

- Charité Universitätsmedizin Berlin
- Computational Biology Research Center
- Freie Universität Berlin
- Institute Pasteur
- John Hopkins University, Salzberg Lab
- Max Delbrück Center for Molecular Medicine
- Max Planck Institute for Molecular Genetics
- MRC Biostatistics Unit
- Robert Koch Institute
- Saarland University
- University of Queensland
- University of Tübingen

---

<sup>20</sup> <https://github.com>

<sup>21</sup> <http://www.seqan.de/seqan-users/> und [http://www.knime.org/files/ugm2013\\_talks/knime\\_ugm\\_2013\\_knutreinert\\_final.pdf](http://www.knime.org/files/ugm2013_talks/knime_ugm_2013_knutreinert_final.pdf)

## Kommerziell

- Febit Biomed GmbH
- Illumina, Inc.
- Molecular Health
- Progenity, Inc.
- Stillwater Supercomputing, Inc.

## Partner

- Nvidia Corporation
- New York Genome Center

### 1.3.5 Zusammenfassung

SqAn ist eine quelloffene, umfangreich dokumentierte Softwarebibliothek, die typische Aufgaben der Sequenzanalyse löst. Sie wendet sich sowohl an Softwareentwickler wie auch an Forscher aus den Lebenswissenschaften — insbesondere der Bioinformatik und der Biologie.

Zu den Anwendern von SqAn gehören sowohl bekannte nationale und internationale, als auch akademische und kommerzielle Einrichtungen. Das SqAn-Projekt verfügt darüber hinaus über eine Partnerschaft mit der Nvidia Corporation<sup>22</sup>, die bei der Nutzung von GPU<sup>23</sup> im Fokus steht.

Die Softwarebibliothek unterscheidet sich von anderen Lösungen durch ihre kompromisslose Effizienz. Dazu basiert sie auf der Programmiersprache C++ und verwendet einen in der Bioinformatik einmaligen Ansatz — das Programmierparadigma Templatemetaprogrammierung. Dieses Paradigma erlaubt die Einsparung der beim Einsatz von Polymorphie und überschriebenen Funktionen auftretenden, virtuellen Lookups, wie man sie aus der objektorientierten Programmierung kennt.

Usability war ein erklärtes Ziel beim Entwurf von SqAn. Die Erfüllung dieser Anforderung wird von Gogol-Döring (2009) jedoch vornehmlich und nur knapp argumentativ, aber nicht empirisch überprüft, was sich durch meine Forschung als unzureichend herausstellen wird.

Bevor ich aber meine Forschung vorstelle, erläutere ich im nächsten Abschnitt die verwendete Forschungsmethode.

---

22 <http://www.nvidia.com/content/cuda/spotlights/knut-reinert-fuberlin.html>

23 Grafische Prozessoreinheit (engl. *graphics processor unit*), also ein für grafische Berechnungen optimierter Prozessor



## METHODE DER GROUNDED THEORY

Für die in dieser Arbeit vorgestellte Erforschung der API-Usability von SeqAn habe ich die Methode der Grounded Theory (GTM)<sup>24</sup> eingesetzt. Wie die im nächsten Kapitel vorgestellte Literaturrecherche zeigen wird, existiert wenig Grundlagenforschung im Bereich der API-Usability. Die GTM wiederum ist eine Forschungsmethode, die sich besonders gut für explorative Studien eignet, bei denen der “Gegenstandsbereich noch neu und unerforscht ist” (Mayring 2002, S. 107). Die API-Usability von SeqAn mittels GTM zu erforschen, ist also nur sinnvoll<sup>25</sup>.

Weil die GTM die Art der Forschung stark beeinflusst — ja sogar bestimmt und wesentlich leitet — stelle ich die GTM im Folgenden vor.

### 1.4.1 Methode der Grounded Theory in der Informatik

Zur GTM gibt es viel Literatur in den Sozialwissenschaften, die hohe Anforderungen an seine Leser aus der Informatik stellt. Einerseits fehlt Informatikern häufig die sozialwissenschaftliche Ausbildung. Andererseits treffen Informatiker mit ihrer formalwissenschaftlichen und ingenieurtechnischen Prägung auf das Problem, dass sich empirische sozialwissenschaftliche Forschung für sie weniger eindeutig darstellt.

Das stellt Informatiker, welche die GTM nutzen wollen, vor gewisse Probleme. Die GTM-Literatur zeichnet sich darüber hinaus nicht durch ihre Präzision aus. Wichtige Begrifflichkeiten wie *Kode*, *Konzept* und *Kategorie* werden “je nach Kontext teilweise (fast) synonym, teilweise (eher) differenzierend benutzt” (Mühlmeyer-Mentzel 2011). Auch der instruktionelle, anwendungserläuternde Gehalt unterscheidet sich stark.

Im Rahmen meiner durchgeführten, mehr als 200 Veröffentlichungen berücksichtigenden wissenschaftlichen Literaturrecherche, konnte ich im Bereich der API-Usability relevanten Forschung lediglich acht Arbeiten finden, bei denen die GTM zumindest thematisiert wurde. Davon zogen zwei Studien (Cao et al. 2010; Ko u. Riche 2011) die GTM lediglich in Betracht. Unter den übrigen sechs setzten drei Arbeiten (Gerken et al. 2011; de Souza et al. 2004; Sunshine et al. 2014) die GTM nur geringfügig und intransparent ein. Bei den anderen drei Arbeiten (Yamashita 2012; Yamashita u. Moonen, 2013) wurde die GTM nachvollziehbar, jedoch nicht vollständig verwendet. Ich konnte keine Arbeit finden, in der die GTM in vollem Umsatz eingesetzt wurde.

<sup>24</sup> Es gab mehrfach den Versuch, die Begriffe GT<sup>G</sup> bzw. GTM ins Deutsche zu übersetzen. Mittlerweile ist es aber üblich, die beiden englischen Begriffe zu verwenden. (Mey u. Mruck 2007)

<sup>25</sup> Eine differenzierte Betrachtung der Eignung der GTM für meine Forschungszwecke stelle ich im Abschnitt 3.5 auf Seite 229 vor.

Der Aspekt der geringen GTM-Verwendung wird auch von Glass et al. (2002) bestätigt (< 1%). Salinger (2013) beschreibt in seiner Arbeit, dass die “Nachvollziehbarkeit des Forschungsprozesses” schlecht ist und zitiert indirekt, dass bei “einigen GTM-Studien [...] es sich sogar um unzulässige Etikettierungen [handelt], durch die ein eher unstrukturiertes qualitatives Vorgehen (nachträglich) legitimiert werden soll [(Oates 2006; Suddaby 2006, S. 147 bzw. S. 633)]”.

Offenbar gibt es also wenig Erfahrung im korrekten Gebrauch der GTM in der Informatik und im Speziellen in der API-Usability-Forschung. Eine Ausnahme bildet die Dissertation “Ein Rahmenwerk für die qualitative Analyse der Paarprogrammierung” (Salinger 2013), bei der der Autor zunächst vor demselben Problem stand. In seiner Dissertation setzt sich Salinger intensiv mit der GTM auseinander. Dieser wissenschaftliche Beitrag erlaubt Informatikern, aber auch Forschern anderer Wissenschaften, einen einfacheren Einstieg in die GTM-basierte Forschung.

## 1.4.2 Generationen und Strömungen der Methode der Grounded Theory

Innerhalb der GTM gibt es verschiedene Strömungen, die grob in zwei Generationen unterschieden werden können (Mey u. Mruck 2011):

### Erste Generation

Die GTM hat ihren Ausgangspunkt in der Arbeit “The Discovery of Grounded Theory: Strategies for Qualitative Research” von Glaser u. Strauss (1967). Zwischen den beiden Autoren kam es in den folgenden Jahren zu wissenschaftlichen Differenzen, die in zwei GTM-Strömungen mündete. Glaser (1978) verfolgte eine “puristisch orientierte” (Haselhoff 2010) GTM, während Strauss (1987); Strauss u. Corbin (1990) eine “geradlinigere” (Salinger 2013) GTM entwickelten.

### Zweite Generation

Zu dieser Generation, die die letztere Strömung der ersten Generation weiterentwickelte, gehört einerseits die konstruktivistische GTM (Bryant u. Charmaz 2010; Charmaz 2006), bei der nicht nur die “Handlungen von Forschungsteilnehmenden”, sondern auch “Handlungen der Forschenden” betrachtet werden (Mey u. Mruck 2011, S. 188). Außerdem ist hier die “situative” GTM bzw. *Situationsanalyse* (Clarke 2005a) zu nennen, die u.a. auch der Erforschung von “nicht-menschlichen Objekten (Technologien, Tiere, Diskurse, historische Dokumente, bildliche Darstellungen usw.) mühelos gerecht [wird].” (Mey u. Mruck 2011, S. 209)

### 1.4.3 Methode der Grounded Theory im Detail

Der Inhalt dieses Abschnitts 1.4.3 basiert im Wesentlichen auf der Arbeit von Salinger (2013), was auch sämtliche Zitate einschließt. Dabei beziehen sich die folgenden GTM-Erläuterung auf die GTM nach Glaser u. Strauss (1967) und ihrer Weiterentwicklung durch Strauss u. Corbin (1990).

Die GTM dient dazu, Erkenntnisse aus Daten zu extrahieren, die in “relevante konzeptionelle Kategorien, Eigenschaften und Hypothesen” verwandelt werden und aus denen schließlich eine GT gebildet wird — also eine in den Daten verankerte Theorie, die “nichts anderes als ein Ausdruck der in den Daten verborgenen Ordnung” (Glaser u. Strauss 2005, S. 50) darstellt.

Dabei fängt der Forscher zunächst mit einer groben Fragestellung an, die sich durch die Auseinandersetzung mit der Materie immer mehr verfeinert. Der GTM liegen die in den nächsten beiden Abschnitten beschriebenen Grundprinzipien und Phasen zu Grunde.

#### 1.4.3.1 Grundprinzipien

##### 1. Iteratives Vorgehen

Im Gegensatz zum klassischen sequentiellen Vorgehen, d.h. Planung, Datenerhebung, Datenanalyse und Theoriebildung, sieht die GTM einen iterativen Prozess, der ständig zwischen Phasen wechselt.

##### 2. Theoretisches Sampling

Die Idee hinter diesem Prinzip ist, dass Daten nur in Teilschritten erhoben werden. Dabei orientieren sich Zeitpunkt, Form und Zweck einer jeden weiteren Datenerhebung an den bisherigen Erkenntnissen.

##### 3. Ständiges Vergleichen

Diese Methode dient dazu, Unterschiede und Ähnlichkeiten zwischen den Daten zu finden. Wird beispielsweise ein interessanter Datenpunkt (*Phänomen*) mit einer konzeptionellen Überschrift (*Kode* bzw. *Konzept*<sup>26</sup>) versehen, müssen auch die anderen Phänomene dieses Kodes erneut verglichen werden. Dadurch entsteht ein wachsendes Verständnis von dem Kode. Möglicherweise stellt sich durch den Vergleich heraus, dass der Kode in zwei Kodes aufgetrennt werden muss, weil durch den neuen Datenpunkt der alte Kode zu breit in seiner Bedeutung geworden wäre, oder sich ein anderes Verständnis entwickelt hat. Ständiges Vergleichen ist auch relevant für das Verstehen der Eigenschaften eines Kodes.

Außerdem begrenzt “das ständige Vergleichen sowohl die entstehende Theorie sowie auch [den] Analyseprozess selbst”. Mit zunehmender “Ausarbeitung der Kategorien” — an dieser Stelle reicht die Definition *Kode- oder Konzeptfamilie* — “stellt sich im Laufe der Untersuchungen die

---

<sup>26</sup> Ich verwende *Kode* und *Konzept* in dieser Arbeit synonym.

so genannte *theoretische Sättigung* ein.“ Das heißt, es werden immer weniger neue, noch nicht von der Theorie beschriebene Phänomene gefunden, was dem Forscher dabei hilft, seine GT, die “eigentlich nie fertig ist”, abzuschließen.

Während des ständigen Vergleichens werden Memos verschiedener Art verfasst, welche die erworbenen Erkenntnisse der Forschung, neben der eigentlichen Theoriemodellierung, festhalten.

#### 4. Ignorieren von vorhandener Theorie

Die Idee der GTM ist das “Entdecken und Entwickeln von Kategorien”. Dabei soll die “Literatur über Theorie bez. des Untersuchungsfeldes zunächst so weit wie möglich [ignoriert werden]”, um eine Kontamination der eigenen Konzepte zu vermeiden. Strauss u. Corbin (1996) gestatten jedoch explizit die bewusste, sensible Verwendung von Literatur sowie beruflicher und privater Erfahrung.

##### 1.4.3.2 Phasen

Der Kodierprozess von Strauss u. Corbin (1990) sieht die folgenden drei Phasen vor, die jedoch nicht sequentiell durchlaufen werden. Diese Phasen werden leicht verständlich von Salinger (2013, S. 62 ff.) beschrieben. Daher beschränke ich mich auf eine sehr knappe Wiedergabe und verweise auf die Quelle für mehr Details.

##### 1. Offenes Kodieren

Beim offenen Kodieren werden “kleine [Daten]-Einheiten, z.B. Sätze, Abschnitte oder Episoden, ‘aufgebrochen’ und konzeptionalisiert”, wie es weiter oben unter *ständiges Vergleichen* beschrieben wurde. Dabei werden die der Konzeptionalisierung zu Grunde liegenden Überlegungen in *Kodememos* festgehalten.

Bei der Konzeptionalisierung geht es zunächst nicht um “die ‘wahre’ Bedeutung”, sondern darum, iterativ ein immer besseres Verständnis zu gewinnen, das sich der Wahrheit innerhalb des Forschungsprozesses immer mehr annähert.

Um eine für die GT hinreichende Abstraktion zu erreichen, werden gleichartige Kodes zu *Kategorien* zusammengefasst.

Für die *konzeptionelle Dichte* werden außerdem *Eigenschaften* formuliert und *dimensionalisiert*. So könnte beispielsweise der Kode / das Konzept / die Kategorie *Schmerz* die Eigenschaft *Intensität* mit der Dimension (*hoch — niedrig*) besitzen. Jedes Phänomen dieses Konzepts würde dann eine *dimensionale Ausprägung*, z.B. *mittel*, haben.

##### 2. Axiales Kodieren

Beim axialen Kodieren werden die in Form von Kodes “aufgebrochenen Strukturen in neuer Art zusammengefasst” und in Beziehung gesetzt. Während Glaser (1978) die Art der Beziehungen vollkommen offen lässt, schlagen Strauss u. Corbin (1990) dazu das *paradigmatische Modell* vor.

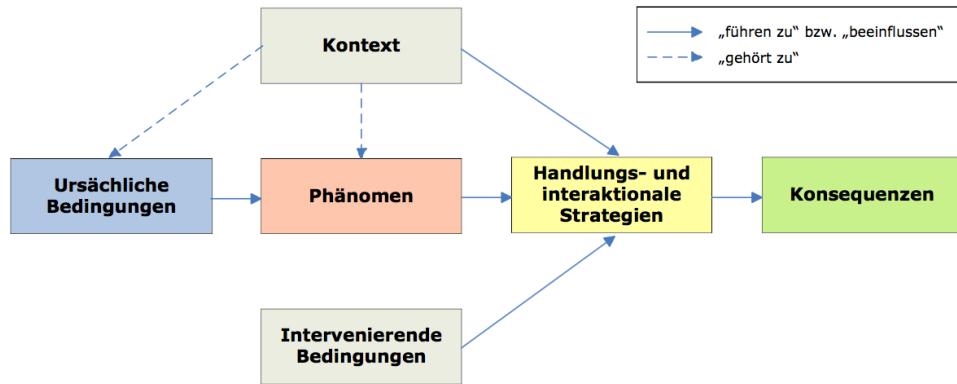


ABBILDUNG 1.2: Paradigmatisches Modell (Salingер 2013, S. 67)

Das in Abbildung 1.2 abgebildete paradigmatische Modell verfolgt im Wesentlichen den Zweck, “dem Forscher ein [erkenntnistheoretisches] Schema an die Hand zu geben, mit dem er bewusst das menschliche Handeln und Interagieren, das Verwenden von Strategien beim Umgang mit Situationen bzw. Situationsinterpretationen und die daraus resultierenden Konsequenzen untersuchen kann”.

Vereinfachend technisch ausgedrückt, könnte man das paradigmatische Modell als auseinandersetzungsfördernde Schablone für die Zusammenhänge von Konzepten begreifen. Im Zentrum steht dabei ein Phänomen, auf welches das Handeln gerichtet ist. Bezogen auf dieses Phänomen interessieren dann beispielweise (kausale) ursächliche Bedingungen, eingesetzte Strategien und die Konsequenzen<sup>27</sup>.

Das paradigmatische Modell hilft dabei, Hypothesen in Beziehung zu setzen, sie zu verifizieren sowie Konzepteigenschaften und Variationen von Phänomenen zu suchen. Man wechselt also stetig zwischen induktivem und deduktivem Denken, was noch einmal im Abschnitt 3.4.6.4 eine Rolle spielen wird.

In der Literatur scheint es keine Bezeichnung für eine konkrete Ausprägung / Instanz eines paradigmatischen Modells zu geben. Daher führe ich an dieser Stelle den Begriff *axiale Kodierung*<sup>G</sup> für Instanzen des paradigmatischen Modells ein, die auf Phänomen-Ebene Zusammenhänge beschreiben. Für Instanzen des paradigmatischen Modells, die auf Konzept-Ebene arbeiten, verwende ich den Begriff *axiales Kodiermodell*<sup>G</sup>.

Geht es dem Forscher lediglich um eine Konzeptentwicklung, genügen offenes und axiales Kodieren.

27 Die Bestandteile *Kontext* und *intervenierende Bedingungen* des paradigmatischen Modells sind ein Paradebeispiel für die schlechte Begriffsschärfe, auf die man in der gängigen Literatur trifft. Erst durch Salingер (2013) haben ein Kollege und ich verstanden, worin der genaue Unterschied zwischen *Kontext* und *intervenierende Bedingungen* besteht. Passendere Bezeichner wären für Kontext *spezieller Kontext* und für intervenierende Bedingungen *allgemeiner* oder *breiter Kontext*. Nicht die *intervenierenden Bedingungen*, sondern der *Kontext* ist es, der die “Ausprägungen der Eigenschaften des im Mittelpunkt der Betrachtung stehenden Phänomens” beschreibt. Die *intervenierenden Bedingungen* beschreiben lediglich den “breiteren strukturellen Kontext”.

### 3. Selektives Kodieren

Selektives Kodieren geht über die reine Konzeptentwicklung hinaus und ermöglicht die Entwicklung einer GT, was der GTM nicht zuletzt ihren Namen gibt. Dabei werden die vorhandenen Erkenntnisse “systematisch zu einem Bild der Wirklichkeit” entwickelt.

Im Zentrum steht das zentrale Phänomen, das als *Kernkategorie* bezeichnet wird und gegebenenfalls erst noch entwickelt werden muss. Ähnlich zum axialen Kodieren werden um die *Kernkategorie* “herum andere Kategorien integriert”. Dieser Prozess sollte mit der Motivation erfolgen, eine *Geschichte* zu erzählen, welche die folgenden Fragen beantwortet:

- Was ist im Untersuchungsbereich am auffallendsten?
- Welche Phänomene werden wiederholt in den Daten gespiegelt?
- Was halte ich für das Hauptproblem?

Zwar werden diese Phasen nicht sequentiell durchlaufen, dennoch ist das offene Kodieren zu Beginn, und das selektive Kodieren gegen Ende der Forschung häufiger zu erwarten.

#### 1.4.4 Gütekriterien

Die GTM ist ein “handlungs- und interaktionsorientiertes Verfahren zur systematischen, qualitativen Analyse von Daten” (Salinger 2013) und unterscheidet sich damit wesentlich von quantitativer Forschung.

Die Gütekriterien quantitativer Forschung können nicht ohne weiteres auf qualitative Forschung übertragen werden. Stattdessen muss die Güte qualitativer Forschungsergebnisse argumentativ nachgewiesen werden. (Mayring 2002)

Im Folgenden fasse ich die sechs allgemeinen Gütekriterien qualitativer Forschung (Mayring 2002) zusammen:

**Verfahrensdokumentation** Für qualitative Forschung gibt es keine standardisierten Techniken und Messinstrumente, auf die verwiesen werden könnte. Weil qualitative Forschung auf einen Gegenstand bezogen ist, muss das Zustandekommen der Ergebnisse im Detail dokumentiert werden, um den Forschungsprozess für andere nachvollziehbar zu machen.

**Argumentative Interpretationsabsicherung** Bei der qualitativen Forschung entstehen die Ergebnisse durch die Interpretationen des Forschers. Um diese adäquat prüfen zu können, müssen sie argumentativ begründet werden. Relevant ist dabei, ob Vorverständnis und Interpretationen des Forschers übereinstimmen. Die Interpretation muss schlüssig, Brüche erklärt und Alternativdeutungen erläutert sein.

**Regelgeleitetheit** Trotz der Offenheit qualitativer Forschung muss das Ergebnis systematisch zu Stande kommen. Darauf Bezug nehmend, müssen die einzelnen Analyseeinheiten festgelegt, systematisch und schrittweise bearbeitet sein. Abweichungen von dieser Systematik müssen begründet werden.

**Nähe zum Gegenstand** Forschung soll sich nahe am Forschungsgegenstand bewegen und ihr angemessen sein. Dieses Gütekriterium besteht im Nachweis dieser Nähe.

**Kommunikative Validierung** Die Gültigkeit der Ergebnisse lässt sich nachweisen, indem diese beispielsweise mit den Beforschten besprochen werden.

**Triangulation** Dieses Kriterium macht die Güte des Forschungsergebnisses von den verschiedenen eingesetzten Lösungswegen abhängig. Dazu gehören verschiedene Datenquellen, Interpreten, Theorieansätze und Methoden.

Steinke (1999) formuliert in ihrem Buch ebenfalls “Kriterien qualitativer Forschung”, welche die zuvor genannten Gütekriterien anders zusammenfassen. Drei dieser acht Kriterien möchte ich an dieser Stelle nennen, weil sie Aspekte der obigen Kriterien für meine Begriffe treffender bzw. prägnanter gruppieren:

**Intersubjektive Nachvollziehbarkeit** Die ~ soll eine “(kritische) Verständigung über eine empirische Studie zwischen Forschern und Lesern” ermöglichen. Dazu gehört u.a. die Dokumentation des Forschungsprozesses und die Anwendung kodifizierter Verfahren.

**Reflektierte Subjektivität** Da in der qualitativen Forschung der Forscher mit seiner Subjektivität selbst Bestandteil des Forschungsprozesses ist, muss er seine eigene Rolle bei der Theoriebildung reflektieren.

**Limitation** Dieses Kriterium befasst sich mit den Grenzen der Verallgemeinerbarkeit der entwickelten Theorie.

Die Gütekriterien zeigen, wie stark die Güte qualitativer Forschung von dem Förschenden abhängt. Das ist der Grund, weshalb ich mich entschieden habe, für die Darstellung meiner Arbeit die Ich-Form zu verwenden.

## 1.4.5 Anwendung der Methode der Grounded Theory in dieser Arbeit

### 1.4.5.1 GTM-Variante

Für die in dieser Arbeit vorgestellte Forschung habe ich die GTM nach Strauss u. Corbin (1990) im Rahmen einer explorativen, empirischen Fallstudie verwendet. Lediglich die explizite Auffassung von

nicht-menschlichen Objekten als Akteure — zu denen explizit auch Technologien gehören — habe ich der “situativen” GTM nach Clarke (2005a) entnommen.

Damit habe ich mich auch gegen die GTM nach Glaser (1978) entschieden. Dessen Purismus hat sicherlich seine Berechtigung, jedoch habe ich Zweifel, dass gerade GTM-Anfänger in einer überschaubaren Zeit ein hinreichend entwickeltes Ergebnis erzielen.

#### 1.4.5.2 Notationen

In dieser Arbeit verfolge ich das methodische Ziel einer hohen Nachvollziehbarkeit. Zu diesem Zweck vereinbare ich folgende Notationen:

- Kodes, Konzepte und Kategorien werden wie folgt notiert: ● Bezeichner
  - Beispiel: ● Versagensverschleppung
  - Diese Darstellung wird durch das \code-Makro erzeugt, das Bestandteil meines L<sup>A</sup>T<sub>E</sub>X-Pakets `apiua` ist. Das Beispiel wurde durch den Befehl `\code{apiua://code/-9223372036854775615}` konstruiert.
  - Der Bezeichner ist verlinkt und verweist eindeutig auf das entsprechende Element innerhalb meiner Forschungsdaten. Im angezeigten Beispiel lautet die URI `apiua://code/-9223372036854775615`.
  - Da zur Auflösung der Links eigentlich das im Abschnitt 3.4 beschriebene Datenanalysewerkzeug installiert sein muss, verweisen die Links stattdessen auf eine ebenfalls öffentlich zugängliche HTML-Fassung meiner Forschungsdaten.
  - Das HTML-Dokument listet alle gefundenen Kodes/Konzepte/Kategorien/Relationen hierarchisch auf. Zu jedem Eintrag wird mein Memo und alle von mir entdeckten Phänomene (*Instances*) dargestellt. Eventuell vorhandene Memos in Bezug auf einzelne Phänomene werden ebenfalls angezeigt. Häufig verfügen abstrakte Kodes nicht über unmittelbare Phänomene. In diesem Fall empfiehlt es sich, die Phänomene von Unterkodes zu betrachten.
- Relationen werden wie folgt notiert: ● Bezeichner 1  $\xrightarrow{\text{Name}}$  ● Bezeichner 2
  - Beispiel: ● Intuitive Entwurfsentscheidungen  $\xrightarrow{\text{führen zu}}$  ● Usability-Problemen
  - Unbenannte Relationen haben das Format: ● Bezeichner 1  $\rightarrow$  ● Bezeichner 2
  - Die Elemente einer Relation sind ebenfalls verlinkt.
- Wichtigkeit wird durch Opazität notiert: ●●●
  - Standardmäßig wird ein semi-transparenter Kreis (●) verwendet.

- Ein vollfarbiger Kreis (●) markiert besonders wichtige Kodes, Konzepte und Kategorien.
  - Ein transparenter Kreis (○) markiert weniger wichtige Kodes, Konzepte und Kategorien.
  - Die verwendeten Farben geben Aufschluss über die Zusammengehörigkeit von Kodes, Konzepten und Kategorien. Deren Zusammengehörigkeit wird durch ähnliche Farben gekennzeichnet.
- Zitate der von mir erfassten Rohdaten sind mit dem Open-Access-Symbol versehen: ☘
    - Beispiel: “to make things more funny there are places in SeqAn where the subscript operator returns values and not references” ☘
    - Das Open-Access-Symbol ist mit dem entsprechenden, öffentlich zugänglichen Rohdatum verlinkt, wie dies auch bei Kodes, Konzepten und Kategorien der Fall ist.
  - Bezeichnungen mit einem Glossar-Eintrag verfügen über ein hochgestelltes “G”: <sup>G</sup>
    - Beispiel: EUSE<sup>G</sup>
    - Annotiert sind je Kapitel nur die ersten Vorkommnisse.
    - Jedes Vorkommnis ist mit dem entsprechenden Glossar-Eintrag verlinkt.

#### 1.4.5.3 Gliederung

Obwohl klassisches, aus der quantitativen Forschung bekanntes Vorgehen nicht die Idee der GTM ist, werden Teile dieser Arbeit so gegliedert, wodurch ich lediglich das Lesen erleichtern möchte. Tatsächlich sind aber viele Tätigkeiten parallel verlaufen.

So stelle ich meine im Kapitel 3 vorgestellte Forschung in sequentiellen Phasen dar, welche sich tatsächlich aber überlappt und teilweise auch gegenseitig stark beeinflusst haben. Die zeitaufwändige erste Beseitigung “grober”, teils offensichtlicher Usability-Probleme reichte in die eigentliche GTM-Forschung hinein, die wiederum im stetigen Wechsel zur Entwicklung eines eigens für meine Forschung entwickelten qualitativen Analysewerkzeugs stand.

Bevor ich jedoch meine Forschung vorstelle, gebe ich im nächsten Kapitel zunächst eine umfassende Literaturübersicht. Diese Literaturübersicht habe ich nicht zu Beginn meiner Forschung, sondern erst ein Jahr später erarbeitet. Auf diese Weise konnte ich meinem Anspruch an eine explorative Studie gerecht werden und einen sensiblen Umgang mit bestehender Literatur erreichen.



## KAPITEL

### 2

## FORSCHUNGSSTAND

In diesem Kapitel gebe ich einen umfassenden Überblick über den Forschungsstand im Bereich der API-Usability. Auf einen Teil der Veröffentlichungen gehe ich dabei genauer ein. Einerseits möchte ich damit dem Leser den Komfort bieten, nicht selber jede Studie nachlesen zu müssen, um meine Arbeit zu verstehen. Andererseits sehe ich in einer genaueren Darstellung auch die teilweise Erfüllung von den im vorangegangen Kapitel vorgestellten Gütekriterien für qualitative Forschung — namentlich *Argumentative Interpretationsabsicherung* (Mayring 2002) und *intersubjektive Nachvollziehbarkeit* (Steinke 1999).

Die GTM<sup>G</sup> nach Strauss u. Corbin (1990) erlaubt explizit die bewusste Verwendung von theoretischem Vorwissen für die eigene Forschung. Da ich eine zu frühe Beeinflussung durch die existierende Literatur vermeiden wollte, habe ich mich erst in einer späteren Phase (etwa nach einem Jahr) mit dem tatsächlichen Forschungsstand intensiv beschäftigt. Meinen Literaturverständnisprozess darzulegen und damit nachvollziehbar zu machen, ist mein Verständnis davon, die vorgenannten Gütekriterien gerecht zu werden.

Der größte Beitrag dieses Kapitels besteht aber sicherlich darin, erstmalig einen umfassenden Überblick über den Forschungsstand der API-Usability zu geben. Zwar gibt es bereits wissenschaftliche Literaturstudien in diesem Bereich, jedoch sind alle vergleichsweise knapp und beschränken sich ausschließlich auf Teilespekte der API-Usability (Zibran et al. 2011; Zibran 2008) und API-Usability-Evaluation (Barth 2011).

Einen umfassenden Überblick, wie es ihn für den Bereich des im Abschnitt 2.4.1 vorgestellten Endbenutzer-Softwareentwicklung bereits gibt (Ko et al. 2011), sollen die folgenden 80 Seiten<sup>1</sup> geben.

Da nicht jede vorgestellte Arbeit von gleicher Relevanz für meine Forschung ist und um die Lektüre dieses Kapitels zu erleichtern, sind die für das Verständnis meiner Forschung *besonders* relevanten Passagen durch einen vertikalen schwarzen Strich am Seitenrand hervorgehoben.

Zur Zitierweise in diesem Kapitel ist zu sagen, dass alle Aussagen, wenn nicht anders angegeben, sich auf die in der jeweiligen Überschrift genannten Quelle beziehen.

---

<sup>1</sup> Der Umfang dieser Literaturübersicht ist dem Umstand geschuldet, dass dieses Forschungsgebiet noch relativ jung ist und es noch vergleichsweise wenig allgemein anerkanntes Wissen gibt. Außerdem habe ich manche Studien detaillierter erläutert, wenn diese besonders relevant für mein Forschungsvorhaben waren.

## ÜBERBLICK

Dieser Abschnitt gibt einen Überblick über die verfügbare, für API-Usability relevante Literatur. In den übrigen Abschnitten dieses Kapitels gehe ich auf eine Reihe der hier genannten Arbeiten genauer ein.

### 2.1.1 Grundlagen

Bevor von API-Usability die Rede war, fand bereits Forschung statt, derer sich die relativ neue Disziplin mehr oder minder explizit bedient.

#### 2.1.1.1 Programmverständnis

Dabei sind die Arbeiten im Bereich der Programmverständnisforschung zu nennen. Shneiderman u. Mayer (1979) kritisierten bereits, dass sich die bisherige Forschung nur auf spezielle Aspekte des Programmierprozesses konzentrierte und unter anderem den Prozess des *Programmverstehens* ignorierte. Die späteren Arbeiten zum *Top-Down-* (Brooks 1983) und zum *Bottom-Up*-Vorgehen (Pennington 1987) griffen beide zu kurz und wurden von Shaft u. Vessey (1998) in Einklang gebracht. LaToza et al. (2007) stellen ein alternatives Verständnismodell vor.

#### 2.1.1.2 Klassische Usability

Der zweite verwandte Bereich ist die bereits gut erforschte klassische Usability. Sie bietet eine Reihe von Evaluationsmethoden, wie *GOMS* (Card et al. 1983), *EVADIS II* (Oppermann u. Reiterer 1992), das *Cognitive Walkthrough* (Wharton et al. 1994), die *HE<sup>G</sup>* (Nielsen u. Molich 1990) samt Erweiterungen (u.a. Sarodnick u. Brau 2006) und den *Usability-Test* (Faulkner 2003). Mit dem Zusammenspiel der beiden letztgenannten Methoden befasst sich Fu et al. (2002). Die Bücher von Nielsen (2005); Sarodnick u. Brau (2006) bieten einen hervorragenden Einstieg in die Materie.

Neben den “üblichen Verdächtigen” gibt es noch den Exoten *Cognitive Dimensions Framework* (Green 1989, 1996). Dabei handelt es sich nicht um eine Evaluationsmethode, sondern um ein Diskussionswerkzeug für Notationen im Allgemeinen. Blackwell u. Green (2000) haben auf ihrer Grundlage einen an Notation-Anwender gerichteten Fragebogen zur Datenerhebung entwickelt.

### 2.1.1.3 Adaptionen

Einige Verfahren aus dem Bereich der klassischen Usability wurden auch für die Evaluation der API-Usability verwendet. Dazu gehört das *Think-Aloud*-Protokoll (Beaton et al. 2008b; Ellis et al. 2007; Stylos u. Clarke 2007).

Die HE wurde gleich auf verschiedene Art und Weise zu diesem Zweck genutzt. Die ursprüngliche Version nach Nielsen u. Molich (1990) wird in ihrer Anwendung zur API-Evaluation von Beaton et al. (2008b) diskutiert. Weiterhin gibt es den Versuch, API-spezifische (Grill et al. 2012) wie auch API-Dokumentations-spezifische (Watson 2012) Heuristiken zu entwickeln. Eine grundlegende Arbeit existiert von Correia et al. (2009). Das Cognitive Walkthrough für die API-Usability-Evaluation wird von Beaton et al. (2008b) diskutiert.

Für das Cognitive Dimensions Framework existiert ein Fragebogen, welcher der Evaluation von APIs dient (Kadoda 2000). Eine weitere Arbeit (Clarke u. Becker 2003) präsentiert die Adaption des Cognitive Dimensions Framework auf APIs. Erläuterungen zu deren Anwendung für die API-Usability-Evaluation existieren (Clarke 2003, 2004, 2005b), wurden aber nur in wenigen Fällen (Blackwell u. Green 2003; Piccioni et al.; Roast et al. 2000) genutzt.

Bis heute wird der klassischen Usability eine weitaus größere Aufmerksamkeit geschenkt, als der API-Usability (Grill et al. 2012).

## 2.1.2 API-Usability

Auf dem Feld der API-Usability-Forschung selbst wurde die Pionierarbeit insbesondere von Rosson u. Carroll (1996) geleistet (Robillard u. DeLine 2010). Diese empirische Arbeit studiert die beobachteten Wiederverwendungsstrategien von API-Anwendern zum Erlernen einer API für die Implementierung von grafischen Benutzeroberflächen in SmallTalk. Eine ähnliche Studie (Basili et al. 2000) für C++ existiert ebenfalls. McLellan et al. (1998) untersuchen eine kommerzielle API, die von professionellen Entwicklern genutzt wird.

Ein wichtiger benachbarter Forschungsbereich ist die *Endanwender-Softwaretechnik* (engl. *end-user software engineering*, kurz: EUSE), bei welcher der Entwickler kein professioneller Entwickler ist, sondern “gezwungener Maßen” programmieren muss und sich softwaretechnischer Disziplinen ausgesetzt sieht. Eine hervorragende wissenschaftliche Literaturstudie existiert von Ko et al. (2011). Ko et al. (2004) untersuchen in diesem Bereich generische Barrieren, auf die Endanwender beim Erlernen einer API treffen.

Die Komplexität des Programmierprozesses selbst wird von Daughtry et al. (2009a); Sarodnick u. Brau (2006) und bezüglich der API-Verwendung von Bruch et al. (2006) thematisiert. Abstrakte Aktivitäten

beim Gebrauch von APIs werden von Stylos (2009) beschrieben. Dabei wird die Strategie der *Verständnisvermeidung* (Lange u. Moher 1989; Stylos u. Myers 2008), das *Vocabulary Problem* (Furnas et al. 1987; Good et al. 1984), wie auch das Lernen mittels *Beacons* beschrieben. Wiederverwendung wird in der Literatur in Bezug auf die *Implementierungswiederverwendung* (Lange u. Moher 1989), die *Anwendungswiederverwendung* (Fairbanks et al. 2006; Rosson u. Carroll 1996) und in Bezug auf Endanwender (Ko u. Myers 2005) erforscht.

Für den konkreten Entwurf von APIs gibt es Erkenntnisse zu Konstruktoren im Allgemeinen (Zibran et al. 2011), aber auch im Vergleich zur Fabrikmethode (Ellis et al. 2007) und zum *create-set-call*-Muster (Stylos u. Clarke 2007). Die Klassenzugehörigkeit von Methoden (Stylos u. Myers 2008) ist genauso wie die Typisierung der gewählten API-Sprache (Mayer et al. 2012) Gegenstand der Forschung. Roberts u. Johnson (1997); Schmidt u. Buschmann beschäftigen sich mit der Kapselung in APIs. Einen breiten Überblick zum Thema API-Entwurf bietet Zibran et al. (2011).

Zahlreiche Arbeiten beschäftigen sich mit der Dokumentation von APIs. Eine umfassende Feldstudie (Robillard u. DeLine 2010) zum Erlernen von APIs, belegt, dass die schwerwiegendsten Usability-Probleme in der API-Dokumentation zu finden sind. Eine Fülle von Dokumentations-Aspekten wie *API-Direktiven* (Dekel 2011; Monperrus et al. 2011), Wartung (Hou et al. 2005; Shi et al. 2011), Benennung (Aguiar 2000; Blinman u. Cockburn 2005; Bloch 2006a; Cwalina u. Abrams 2008; Stylos et al. 2009a; Teasley 1993), Aufbau und Darstellung (Hou et al. 2005; Jeong et al. 2009), Zielgruppenspezifität (Fairbanks et al. 2006; Ko u. Riche 2011; Nykaza et al. 2002; Pugh 2006), Tutorials (Nykaza et al. 2002), Beispiele (Basili et al. 2000; Hou et al. 2005; Jeong et al. 2009; Tenny 1988) und die Bedeutung von Webinhalten als Alternative zu API-Dokumentationen (Parnin u. Treude 2011; Stylos u. Myers 2006) wurden untersucht.

Die Arbeit von Robertson (2007) stellt Qualitätsfaktoren von APIs vor. Wichtige Erkenntnisse bezüglich der Wichtigkeit, die Zielgruppe gut zu verstehen (Nykaza et al. 2002), führten schließlich zur Entwicklung von drei, speziell im API-Umfeld gebräuchlichen *Personas* (Clarke 2007; Stylos u. Clarke 2007) als Möglichkeit, grob Anwendergruppen umschreiben zu können.

### 2.1.2.1 API-Usability-Evaluation

API-spezifische Evaluationsmethoden basieren auf *Guidelines*, wie die für Java (Bloch 2008), C++ (Meyers 1992) oder der STL (Meyers 2001). O'Callaghan (2010) hingegen stellen das *API-Walkthrough* und Farooq et al. (2010); Farooq u. Zirkler (2009, 2010) das *API (Usability) Peer Review* vor. *Metrix* (de Souza u. Bentolila 2009) wiederum ist eine objektiv-summative Evaluationsmethode. Ein weiteres Verfahren besteht im Gebrauch von Textanalysetechniken (Watson 2009).

Besonders interessant sind die folgenden Studien, die allesamt eine Mischung verschiedener Methoden zur API-Usability-Evaluation und — bis auf eine Ausnahme (Piccioni et al.) — auch zur API-Usability-Verbesserung einsetzen. Zu diesen Verfahren gehören das *Concept-Maps*-Verfahren (Gerken

et al. 2011) und die leider in der Forschung bisher vollkommen ignorierte Methode von Grill et al. (2012). Eine andere Arbeit (Letondal 2006) stellt die partizipatorische Entwicklung einer integrierten Entwicklungsumgebung für Bioinformatiker vor. Die Arbeit von Stylos et al. (2008) beschreibt eine Anwendergruppen-spezifische API-Usability-Verbesserung.

### **2.1.2.2 API-Werkzeuge**

Zahlreiche Forschungsergebnisse sind in Form prototypischer Anwendungen veröffentlicht worden.

Zu den Werkzeugen, die sich an API-Entwickler wenden, gehört ein Assistent zur Erstellung von API-Dokumentationen (Dahotre et al. 2011), eine elektronische Alternative zur klassischen API-Dokumentation (Berglund 2003), ein modulares Dokumentationssystem (Horie u. Chiba 2010) und ein Algorithmus, der automatisch Code-Beispiele erzeugen kann (Buse u. Weimer 2012).

Werkzeuge, die sich an API-Anwender und teilweise auch API-Endanwender richten, unterstützen beim Auffinden oder Verwenden von Beispielen (Bruch et al. 2006; Dagenais u. Ossher 2008; Dohnert et al. 2014; Holmes u. Murphy 2005; Neal 1989; Oney u. Brandt 2012; Stylos u. Myers 2006; Wightman et al. 2012; Ye u. Fischer 2002), bei der Beachtung von API-Verwendungsregeln (Bruch et al. 2006; Dekel 2011; Feilkas u. Ratiu 2008), oder bei der Exploration der API-Dokumentation (Bruch et al. 2006; Duala-Ekoko u. Robillard 2011; Eisenberg et al. 2010a, b; Stylos et al. 2009a). Wiederum andere stellen eine bessere Codevervollständigung bereit (Hou u. Pletcher 2010; Omar et al. 2012; Zhang et al.). Eine Arbeit (Hou et al. 2005) beschreibt Kollaborationsplattformen als Ergänzung zur API-Dokumentation.

Ausschließlich an API-Endanwender richtet sich eine integrierte Entwicklungsumgebung (Gross et al. 2011) und ein spezieller API-Endanwender-Debugger (Ko u. Myers 2004).

Auf den folgenden Seiten werde ich eine Reihe von Studien genauer vorstellen.

## PROGRAMMVERSTÄNDNISFORSCHUNG

Im vorangegangenen Abschnitt 2.1 habe ich einen Überblick über den API-Usability-Forschungstand gegeben. Dieser Abschnitt soll einige Forschungsergebnisse näher beleuchten.

Grundmotivation hinter der hier vorgestellten Forschung ist es, herauszustellen, wie Menschen Programmtexte verstehen und wie das gewonnene Wissen mental repräsentiert wird. Hintergrund ist die Beobachtung, dass etwa 50% der Zeit eines Programmierers auf Wartungsarbeiten entfallen, bei denen sich der Programmierer hauptsächlich mit dem Code anderer Entwickler befasst (Pennington 1987). Andere Quellen gehen sogar von 50%-70% (Layzell u. Champion 1993), 50%-75% (Boehm 1984) oder 50%-80% (Shaft u. Vessey 1998) Kostenanteil aus, der im Laufe des Lebens von Software auf die Wartung entfällt. Des Weiteren verbringen Entwickler 50%-90% ihrer Zeit mit dem Verstehen von Programmtext (Shaft u. Vessey 1998).

Während meiner Forschung beobachtete ich, dass Anwender in bestimmten Problemsituationen anfingen, Beispiel-Code aus der SeqAn-Online-Dokumentation zu kopieren (siehe Abschnitt 4.1). Das hier vorgestellte Wissen hat sich als gewinnbringend erwiesen, diese Strategie besser zu verstehen.

### 2.2.1 Grundlagen (Shneiderman u. Mayer 1979)

Shneiderman u. Mayer (1979) kritisierten die punktuelle Forschung bzgl. der Programmiertätigkeiten und versuchten daher, das breite Feld der Programmierer-Tätigkeiten (Komposition / Programmieren, Programmverständnis, Debugging, Programmänderung und Lernen) auf der Grundlage vorangeganger Forschung und eigner kontrollierter Experimente zu integrieren. Mit Verweis auf die Eigenschaften von Kurz- und Langzeitgedächtnis erklärt Shneiderman, wie im Arbeitsgedächtnis Informationen aus den beiden anderen Gedächtnissen zu neuen Strukturen verschmelzen.

Im Langzeitgedächtnis wird Wissen mehrschichtig abgelegt und in zwei Arten unterschieden:

**Semantisch** heißt der Teil, der generisches, sprachenunabhängiges Wissen umfasst. Die Spanne dieses Wissens ist groß und reicht von grundlegenden Dingen wie der Bedeutung einer Zuweisung, bis hin zu hoch abstraktem Wissen wie dem Transaktionshandling von Flugreservierungssystemen.

**Syntaktisch** wird der Teil bezeichnet, bei dem die gespeicherten Informationen zwar präziser und detaillierter, aber auch beliebiger und damit leicht vergesslicher sind.

Das Lernen von Programmiersprachen illustriert den Unterschied sehr gut. Sobald man eine Programmiersprache und damit semantisches und syntaktisches Wissen erworben hat, lernt sich eine zweite

Sprache mit ähnlichen Konzepten viel leichter, weil man dann zu großen Teilen nur noch syntaktisches Wissen erlernen muss. Dass syntaktisches Wissen weniger integriert wird, sieht man bei Programmierern, die mehrere Programmiersprachen beherrschen, beispielsweise daran, dass sie irrtümlich falsche Schleifenkonstrukte oder Semikolone verwenden, obwohl die Sprache sie nicht als Anweisungsende verwendet.

In einem Experiment erklärten die beiden Autoren, dass Programmieranfänger — im Gegensatz zu ihren erfahrenen Kollegen — das arithmetische **IF** schlechter beherrschen als das logische **IF**, weil letzteres mehr der semantischen Struktur entspricht.

In einem zweiten Experiment zeigten die Autoren, dass abstrakte Kommentare, die also von der Syntax abheben, besser geeignet sind, um ein mentales Modell der semantischen Struktur aufzubauen und mit der vorhandenen Struktur zu vergleichen.

Etwas Ähnliches zeigte ein weiteres Experiment: Programmierer und Nicht-Programmierer sollten sich jeweils ein kleines Programm und dann ein zeilenweise durchmisches Programm einprägen. Lediglich Programmierer konnten eines der beiden Programme wiedergeben - und zwar das nicht durchmischte Programm, wobei Variationen auftraten, welche die Semantik nicht betrafen. Die Autoren führen dies darauf zurück, dass lediglich die geordnete Form in eine semantische Struktur überführt werden konnte. Da die Nicht-Programmierer über keine semantischen Kenntnisse verfügten, scheiterten sie in beiden Fällen.

Shneiderman und Mayer nennen viele ähnliche Unterscheidungen von anderen Publikationen, wie z.B. "know how" und "know what" (Polya 1957).

In Bezug auf die fünf genannten Programmiertätigkeiten fassen die Autoren wie folgt zusammen:

**Komposition / Programmieren** Zwei Strategien sind anzutreffen, wobei sich manchmal beide Strategien und bei anderen Problemen nur eine Strategie eignet:

1. Top-Down als Prozess, bei dem "rückwärts" gearbeitet wird und das Ziel immer wieder aufs Neue vom generellen auf immer speziellere Ziele "reformuliert" wird.
2. Bottom-Up als Prozess, bei dem "vorwärts" gearbeitet wird und Low-Level-Code generiert dieser dann immer wieder aufs Neue "reformuliert" bis das Ziel erreicht wird.

**Programmverstehen** bedeutet für den Programmierer die Aufgabe, mithilfe seines syntaktischen Wissens von der Sprache eine semantische, mehrschichtige Struktur zu konstruieren. Dieser Prozess gipfelt im Verständnis, was das Programm tut und erfordert keinesfalls ein Zeile-für-Zeile-Verständnis. Mit Verweis auf Miller (1956), der zeigte, dass das durchschnittliche Arbeitsgedächtnis nur sieben so genannte "Chunks" gleichzeitig umfassen kann, fügen Shneiderman und Miller hinzu, dass zusammenhängende Code-Passagen besser zu einem Chunk zusammengefasst werden könnten, als verteilte Passagen.

**Debugging** wird von Autoren in drei Fehlerquellen unterschieden:

1. Fehler, die der Compiler erkennt und damit nicht weiter von Interesse sind (z.B. nicht erlaubtes Zeichen im Bezeichner einer Variablen).
2. Fehler, die ihre Ursache in der falschen Transformation von der internen Semantik zum Quellcode haben (z.B. Variable zu oft hochgezählt).
3. Fehler, die ihre Ursache in der falschen Transformation von Problemlösung zur internen Semantik haben (z.B. nicht Berücksichtigung eines Anwendungsfalls).

**Modifikation** von Quellcode wird als Zusammenspiel von Implementierung, Verstehen und Debugging beschrieben.

**Lernen** erstreckt sich über die folgenden beiden Extreme:

1. Beim klassischen Ansatz wird der Fokus auf die Syntax gelegt. Erlerner der Sprache erfahren also viel mehr zur syntaktischen Struktur als zu den motivationalen Hintergründen oder dem Problemlösen.
2. Dem gegenüber steht der Problemlösungsansatz, beim dem abstraktes und sprachenunabhängiges Wissen vermittelt wird und den Erlerner in die Lage versetzt, Aufgaben in seine Teilprobleme zu zerlegen.

Weder der eine noch der andere Ansatz sind pauschal richtig. Es kommt auf das Ziel an. In der Lehre sind allerdings häufig Mischformen zu finden. Interessanterweise kann das Erlernen einer weiteren Programmiersprache, die sich von ihren semantischen Strukturen stark von den bereits erlernten Programmiersprachen unterscheidet, schwer sein, da sie mit den bereits bekannten semantischen Strukturen interferieren.

Maschinen-bezogenes Detailwissen wie Compileroptimierungen, Zahlenbereiche, etc. gehören dem syntaktischen Wissen an, da diese Art von Wissen in der Regel nicht integriert, sondern lediglich häufig wiederholt wird.

### 2.2.2 Top-Down (Brooks 1983)

Die Top-Down-Theorie stammt von Brooks (1983) und erklärt den Prozess des Programmverständnisses. Sie wurde zu großen Teilen analytisch entwickelt und stützt sich lediglich stellenweise auf empirische Erhebungen anderer Veröffentlichungen.

Brooks nennt vier Hauptquellen, die das unterschiedliche Verhalten von Programmierern bedingen:

1. vom Programm gelöstes Problem
2. intrinsische Eigenschaften wie Programmiersprache, Programmänge, etc.

3. Aufgabentyp wie Modifizieren oder Debuggen
4. individuelle Unterschiede auf Seiten des Programmierers

Die Hauptpunkte seiner Theorie fasst Brooks wie folgt zusammen:

1. Der Programmierprozess ist die Konstruktion von Abbildungen von der Problemdomäne (ggf. über mehrere Zwischendomänen) auf die Programmierdomäne.
2. Beim Verstehen eines Programms werden diese Abbildungen oder Teile davon rekonstruiert.
3. Der Rekonstruktionsprozess ist erwartungsgetrieben durch Erzeugung, Bestätigung und Verfeinerung.

Der Anwender macht auf Grundlage des Programmnamens und seiner Dokumentation eine Hypothese über Hauptoperationen und Datenstrukturen

Hypothesen nennen nicht explizit die Komponente, sondern eher die Aufgabe, z.B. "Dieses Ding sortiert die Kundenliste" statt "Merge-Sort".

### 2.2.2.1 Verständnisprozess

1. Zunächst entwickelt der Anwender seine primäre Hypothese, welche die wahrscheinlichste globale Struktur (Input, Output, Haupt-Datenstrukturen, Prozessequenzen) adressiert und bereits beim Hören des Programmnamens und einer deskriptiven Phrase entsteht.  
Ausnahme: Ist die Domäne sehr fremd, benötigt der Programmierer eine ausführliche Beschreibung, um seine primäre Hypothese entwickeln zu können.  
Dieser Schritt wird beeinflusst durch Kenntnisse der Problemdomäne und den Domänen, die sich zwischen Problem und Lösung (Programmcode) befinden.<sup>2</sup>
2. Anschließend konstruiert der Anwender Teilhypotesen im Sinne einer Tiefensuche, bis direkt verifizierbare Einheiten erzeugt wurden.
3. Die jeweilige Teilhypothese wird mit Hilfe von *Beacons*<sup>3</sup> verifiziert. Die Verifikation findet nicht blind statt. Der Anwender sucht auch in der Breite und nimmt irrelevante starke Beacons war, die später die Grundlage neuer Hypothesen bilden<sup>4</sup>.
4. In diesem Schritt werden die relevanten Beacons mit der Hypothese synthetisiert. Beispiel: Die Hypothese, es könnte sich um eine Suche handeln, könnte mit dem Beacon "Sieht irgendwie

<sup>2</sup> Ohne vorweg greifen zu wollen, möchte ich meine Verwunderung darüber ausdrücken, dass Brooks trotz dieser Erkenntnis von einem Top-Down-Vorgehen spricht.

<sup>3</sup> Für das Verständnis reicht an dieser Stelle die Übersetzung "Indiz" oder "Hinweis". Beispiel: Ein Codestück, das eine Datenstruktur sortiert, könnte man an geschachtelten Schleifen mit Austauschoperationen und sinnvollen Variablennamen erkennen (Gellenbeck u. Cook 1991). Das Beacon würde dann aus den geschachtelten Schleifen und/oder den sinnvollen Namen bestehen.

<sup>4</sup> Streng genommen ist dieses Vorgehen kein Top-Down-Vorgehen in Reinkultur.

wie Bubblesort aus.” zu “Es handelt sich um eine Suche mittels Bubblesort.” konkretisiert und verifiziert werden.

5. Werden die vorangegangen Schritte hinreichend oft durchlaufen, entsteht schließlich eine Baumstruktur, in der Hypothesen die Knoten und die Beacons die Kanten bilden. Dabei entspricht die Wurzel der primären Hypothese.

### 2.2.2.2 Mögliche Problemtypen

1. Kann kein Beacon gefunden werden, der zur Hypothese passt, liegt entweder eine falsche Hypothese vor oder der Anwender verfügt nicht über die notwendigen Kenntnisse, um relevante Beacons zu finden.
2. Ein Beacon passt zu verschiedenen und sich widersprechenden Hypothesen.
3. Ein Beacon passt zu keiner Hypothese.

### 2.2.2.3 Lösungsschritte bei Problemen

1. Tritt ein Problem auf, wird eine alternative Hypothese basierend auf der aktuellen und seiner Elternhypothese gebildet.
2. Sind alle Alternativen ausgeschöpft, werden alternative Hypothesen basierend auf Schwesternhypothesen gebildet.
3. Kann das Problem auch dann nicht gelöst werden, wird der Teilbaum für ungültig erklärt und die Elternhypothese durch eine Alternativhypothese ersetzt. Je mehr “Rückschläge” es gibt, desto gründlicher wird der Quellcode gelesen.

### 2.2.2.4 Implikationen

1. Da das Verstehen eines Programms darin besteht, die Abbildung von Problem zur Lösung zu rekonstruieren, ist ein Verständnis für das Problem ein sehr einflussreicher Faktor.
2. Der Abstraktionsgrad der benötigten Beacons muss in etwa dem der zu validierenden Hypothesen entsprechen. Damit ergibt sich gerade bei den abstrakten Hypothesen die Notwendigkeit einer Dokumentation des Programmdesigns oder ähnliches.
3. Dokumentationen großen Umfangs sind zweischneidig. Auf der einen Seite steigt mit redundanten Informationen die Wahrscheinlichkeit, dass zumindest ein Beacon erkannt wird. Andererseits könnte die Masse an redundanten Beacons den Blick auf einmalige Beacons versperren.

4. Modifikationen des Quellcodes erfordern eine intensivere Auseinandersetzung als das Verstehen der Funktionsweise.
5. Brooks nennt drei Einflussfaktoren auf das Programmverständnis eines Programmierers:
  - (a) Programmierkenntnisse
  - (b) Domänenkenntnisse (insb. bei Formulierung der primären Hypothese)
  - (c) Verständnis-Strategien (z.B. Programmfluss nachvollziehen vs. I/O-Funktionen identifizieren)

Entgegen seiner Bezeichnung enthält das Top-Down-Vorgehen auch Bottom-Up-Elemente. Eines besteht beispielsweise im “Aufschnappen” von irrelevanten Beacons während der Hypothesen-Verifikation. Einen reinen und viel zitierten konträren Ansatz stelle ich im nächsten Abschnitt vor.

### 2.2.3 Bottom-Up (Pennington 1987)

Die Bottom-Up-Theorie stammt von Pennington (1987) und befasst sich ebenfalls mit dem Prozess des Programmverstehens. Im Gegensatz zur Top-Down-Theorie von Brooks (1983) ist Penningtons Theorie empirisch entstanden und legt den Fokus auf erfahrene Entwickler.

#### 2.2.3.1 Forschung

Pennington führte zwei Laborexperimente durch, bei denen 80 Entwickler kleine Programme verstehen und diesbezüglich Fragen beantworten mussten. In einem zweiten Experiment mussten 40 der 80 Entwickler etwas größere Programme (rund 200 Zeilen) verstehen, verändern und dann Verständnisfragen beantworten. Die Programme waren sowohl in *FORTRAN* als auch in *COBOL* geschrieben.

Pennington wollte herausfinden, welche der folgenden vier Abstraktionen die dominante beim Verstehen von Programmtexten ist.

Die vorgeschlagenen Abstraktionen werden durch folgende Fragen geleitet:

**Funktional** Was soll erreicht werden? In welche Teilziele wird das Hauptziel zerlegt?

**Datenfluss** Welches Datum wird in welcher Reihenfolge von wem bearbeitet? Was muss wie verändert werden?

**Kontrollfluss** Wann wird welcher Befehl ausgeführt?

**Bedingung-Aktionen** Unter welchen Bedingungen wird welche Aktion ausgeführt? Was soll unter welchen Bedingungen gemacht werden?

Beispiel: Mahlzeit Spaghetti zubereiten
<b>Funktional</b> Spaghetti kochen, Sauce zubereiten, Spaghetti und Sauce vermischen, ...
<b>Datenfluss</b> Spaghetti kommen aus dem Geschäft und dann in den Topf, Topf kommt aus dem Schrank und landet auf der Herdplatte, ...
<b>Kontrollfluss</b> Einkaufen, Wasser kochen, Nudeln kochen, ...
<b>Bedingung-Aktionen</b> Wenn das Wasser kocht, kommen die Nudeln in den Topf. Wenn 8min vergangen sind, werden die Nudeln aus dem Wasser genommen.

Da diese Abstraktionen parallel existieren und der Mensch gezwungen ist, die kognitive Belastung zu minimieren (Miller 1956), ergibt sich die Frage:

*Gibt es eine Abstraktion, die dem mentalen Modell eines erfahrenen Programmierers am besten entspricht wenn er ein Programm studiert? Wenn ja, welches?*

### 2.2.3.2 Ergebnisse

Die Auswertung der beiden Experimente hat ergeben, dass die Probanden Fragen bzgl. des Kontrollflusses am häufigsten korrekt beantworten konnten und diese Abstraktion die dominante ist. Fragen zur Funktion wurden am häufigsten falsch beantwortet.

### 2.2.3.3 Theoretischer Hintergrund

Kintsch (1988) haben das psychologische Prozessmodell “Construction-Integration-Model” vorgestellt, mit dem man den Prozess des Textverständens erklären kann. Pennington bedient sich zweier Repräsentationsstrukturen aus diesem Modell, um gemeinsam mit den Ergebnissen ihrer Experimente einen ungefähren Prozess für das Programmtextverstehen zu beschreiben.

**Mikro- und Makrostruktur** bilden eine der Repräsentationstrukturen. Die Mikrostruktur besteht aus so genannten Proposition-Bedeutungsträgern, die aus einem Prädikat und mindestens einem Argument bestehen (z.B. *ESSEN(FUSILLI)*). Mikropositionen entspringen entweder dem Text oder dem Wissen des Lesers. Textbasierte Mikropositionen entsprechen den Anweisungen im Quellcode. Mikropositionen werden in der Makrostruktur nach definierten Regeln zu Makropositionen zusammengesetzt (z.B.  $ESSEN(FUSILLI) \wedge ESSEN(MACARONI) \Rightarrow ESSEN(SPAGHETTI)$ ). Dabei entspricht eine Proposition einem *Chunk* nach Miller (1956).

**Textbasis und Situationsmodell** bilden die andere Repräsentationsstruktur. Die Textbasis beschreibt die sich in einem Text befindlichen expliziten Propositionen — egal ob Mikro- oder Makroproposition. Die expliziten Propositionen sind praktisch nie vollständig und müssen vom Leser mittels eigenen Wissens zu einem kohärenten Situationsmodell vervollständigt werden. Makropropositionsbezeichnungen entsprechen den Konzepten des Situationsmodells (Kintsch u. van Dijk, 1978).

Beispiel: Ein Text erzählt von einem Mann, der gemeinsam mit seiner Frau nach Frankreich fährt. Möglicherweise interpretiert der Leser nun, dass es sich um einen Urlaub handeln müsste. Hier wäre genau diese Vorstellung - weil in der Textbasis nie explizit erwähnt - das Situationsmodell.

#### 2.2.3.4 Verständnisprozess

Der von Pennington angedeutete Prozess des Programmtextverständens lässt sich wie folgt beschreiben:

1. Anweisungen / Mikropropositionen werden mithilfe von syntaktischen Markierungen (z.B. Leerzeilen, Schleifen) segmentiert.
2. Den entstandenen Segmenten / Makropropositionen wird eine Rolle zugeschrieben. Beispiel: Eine Schleife, bei der Nullen einer Variablen zugewiesen werden, wird wahrscheinlich die Rolle der Initialisierung einer Variablen zugeschrieben.
3. Während der Kontrollfluss gut verstanden wurde, entwickelt sich zunehmend das Datenfluss- und Funktionsverständnis. Makropropositionen höherer Ordnung und ein mächtigeres Situationsmodell entstehen, indem Zusammenhänge über Segmentsgrenzen hinweg verstanden werden. Beispiel: Die Berechnung eines Durchschnitts erfordert ein Segment zur Initialisierung von Variablen und eines, das durch eine Schleife repräsentiert wird.
4. Ein ausgeprägtes Funktionsverständnis bildet sich erst nach einer längeren und intensiveren Auseinandersetzung mit dem Programmtext. Lange u. Moher (1989) fand für diesen Punkt im Rahmen einer einwöchigen Beobachtung einer Objective-C-Entwicklerin ebenfalls Evidenz.

Dieser Beschreibung kann man gut entnehmen, dass das Verständnis von Prozeduren und des Kontrollflusses dem Funktionsverständnis vorausgeht. Entgegen Penningtons Behauptung aus einer Veröffentlichung des gleichen Jahres (Pennington 1987), die dort vorgeschlagenen Abstraktionsebenen (Prozeduren, Kontrollfluss, Datenfluss, Zustand und Funktion) würden sequentiell durchlaufen werden, konnte Teasley (1993) darauf Bezug nehmend lediglich zeigen, dass es eine deutliche Verzögerung zwischen den ersten vier Abstraktionsebenen und der Funktionsebene gibt. Teasley konnte damit also lediglich einen Verstehensprozess, bestehend aus zwei Phasen, bestätigen.

Pennington konnte beobachten, dass die Probanden bei der Beschreibung von Prozeduren und des Kontrollflusses mehrheitlich eine technische Terminologie, also einen Begriff aus der Lösungsdomäne

verwendet haben. Hingegen kamen bei der Beschreibungen von funktionalen Aspekten vorzugsweise Begriffe aus der Problemdomäne zum Einsatz. Diese Beobachtung harmonisiert mit dem Situationsmodellbegriff von van Dijk and Kintch (1983), da dieser ja erst die Brücke von der Lösungswelt in die Problemwelt schlägt.

Ein wichtige Frage, nämlich ob die Programmiersprache einen Einfluss auf die präferierte Repräsentation hat, bleibt unbeantwortet. Für beide Theorien gibt es Evidenz (Hayes u. Simon 1975; Spoehr et al. 1984). Green et al. (1980) zeigte bereits, dass sich Eigenschaften der Programmiersprache auf das mentale Modell auswirken. Ein offensichtliches Beispiel sind aspektorientierte Sprachen, die Bedingungen und Aktionen in besonderer Weise betonen.

### 2.2.3.5 Zusammenfassung

1. Programmverstehen findet auf verschiedenen Abstraktionsebenen statt.
2. Die Kontrollfluss-Abstraktion wurde in ihrer Anwendung am häufigsten beobachtet.
3. Mikrostruktur-Elemente werden zu Makrostrukturen zusammengesetzt.
4. Explizite im Programmcode manifeste Propositionen werden zu einem Situationsmodell vervollständigt.
5. Es gibt Indizien, die darauf hinweisen, dass die Programmiersprache selbst Einfluss auf das mentale Modell des Anwenders hat.

Brooks (1983) vornehmlich analytische Herleitung der Top-Down-Theorie und Penningtons empirisch entwickelte Bottom-Up-Theorie sind beide nachvollziehbar und wertvoll. Dennoch scheinen sie nicht miteinander vereinbar. Im nächsten Abschnitt wird eine Theorie vorgestellt, die beide Modelle miteinander vereinbart.

### 2.2.4 Top-Down vs. Bottom-Up (Shaft u. Vessey 1998)

*Diese Arbeit ist relevant für die von mir in Kapitel 4 vorgestellten ● Strategien, die ich bei den SeqAn-Anwendern beobachten konnte.*

Offensichtlich sind sowohl Brooks Top-Down-Theorie als auch Penningtons Bottom-Up-Theorie nachvollziehbar und gleichzeitig widersprüchlich, auch wenn Brooks in der Literatur mehr Beachtung findet (Shaft u. Vessey 1998).

Gemeinsam ist beiden Modellen, dass sie davon ausgehen, dass Anfänger und Fortgeschrittene die gleichen mentalen Modelle benutzen, die sich lediglich im Detailgrad unterscheiden (Corritore u. Wiedenbeck 1999).

Auch wenn Blinman u. Cockburn (2005) Shneiderman nur teilweise korrekt zitieren<sup>a</sup>, haben sie korrekt erkannt, dass in beiden Theorien das Konzept von Beacons verwendet wird. Dieses Vorkommen ist für Blinman u. Cockburn (2005) Ursache für die spätere Erforschung von Beacons (Aschwanden u. Crosby 2006; Crosby et al. 2002; Gellenbeck u. Cook 1991; Wiedenbeck 1986, 1991).

Beide Theorien sind insoweit zu kritisieren, als dass Brooks (1983) in seiner Top-Down-Theorie auch Bottom-Up-Elemente verwendet<sup>b</sup> und einige Schlussfolgerungen von Pennington (1987) auf teilweise statistisch nicht signifikanten Ergebnissen<sup>c</sup> basieren.

Gemein ist beiden Theorien, dass sie beide auf der Idee des *Chunkings* (Miller 1956) basieren. Auch für die Fact-Finding-Theorie von LaToza et al. (2007) (Abschnitt 2.2.5) ist das Chunking ein grundlegender Mechanismus.

Die Integration beider Theorien gelang Shaft u. Vessey (1998) mit einer Untersuchung, die sich auf eine vorangegangene Veröffentlichung von Glass u. Vessey (1992) stützt, welche die Bedeutung von Anwendungswissen für Programmierer historisch nachzeichnet und diskutiert. Die Untersuchung soll zeigen, inwiefern das Domänenwissen eines Entwicklers Einfluss auf sein Programmverständnis hat. Dazu wurden 24 Entwickler mit jeweils einem Programm bekannter und unbekannter Anwendungsdomäne konfrontiert. Jeder Teilnehmer wurde instruiert, während der Konfrontation laut zu denken (*think aloud*). Außerdem mussten sie im Anschluss Verständnisfragen beantworten.

Shaft u. Vessey (1998) fanden folgendes heraus:

1. Ein Teil der Programmiererschaft hat ein von seiner Domänenkenntnis abhängiges Vorgehen beim Verstehen von Programmen (flexibel), während der andere Teil sein Vorgehen nicht von seiner Domänenkenntnis abhängig macht (Top-Down, Bottom-Up).
2. Die Kombination aus Domänenkenntnis und Vorgehen hat Einfluss auf das Programmverständnis.
3. Hat der Programmierer viele Domänenkenntnisse, ist das Top-Down-Vorgehen am effizientesten. Je weniger Domänenkenntnisse vorhanden sind, desto überlegener ist das Bottom-Up-Vorgehen. Programmierer mit einem flexiblen Vorgehen erzielen in beiden Szenarien gute Ergebnisse.

Interessanterweise gab es keinen Programmierer, der beim Programm aus einer bekannten Anwendungsdomäne den Bottom-Up- und beim Programm aus einer unbekannten Anwendungsdomäne den Top-Down-Ansatz gewählt hat.

Shaft u. Vessey (1998) stellen fest, dass weder Top-Down, noch Bottom-Up die besten Ergebnisse erzielen, sondern Programmierer, die ihr Vorgehen ihrem existierenden Domänenwissen anpassen (flexibles Vorgehen).

Die vereinfachte Zusammenfassung lautet also: Sowohl die Top-Down- als auch die Bottom-Up-Strategie findet zum Programmverstehen Anwendung. Entwickler mit flexilem Vorgehen wenden das Top-Down-Verfahren mit größerer Wahrscheinlichkeit an, wenn sie über Domänenkenntnisse verfügen. Je weniger davon vorhanden sind, desto wahrscheinlicher ist ein Bottom-Up-Vorgehen.

*a* Blinman u. Cockburn (2005) geben an, Shneiderman (1977) sei die erste Veröffentlichung, in der der Bottom-Up-Ansatz vorgestellt worden wäre. Allerdings werden weder Top-Down noch Bottom-Up mit einem Wort erwähnt. Shneidermans Erklärungen deuten höchstens letzteren Ansatz an. Die darauf folgende Veröffentlichung (Shneiderman u. Mayer 1979) wird als das Paper angeführt, in dem der Bottom-Up-Ansatz weiterentwickelt wäre. Allerdings wurde er hier überhaupt erst namentlich genannt. Außerdem verlieren Blinman und Cockburn kein Wort darüber, dass Shneiderman u. Mayer (1979) den Top-Down-Ansatz in einem vergleichbaren Umfang behandelt haben.

*b* Brooks beschreibt, dass Hypothesen mittels Beacons verifiziert werden. Dabei zur Kenntnis genommene Beacons, die nicht mit der zu verifizierenden Hypothese in Verbindung gebracht werden, werden trotzdem vom Programmierer wahrgenommen. Haben sich hinreichend viele zusammengehörige Beacons angesammelt, können daraus neue Hypothesen entstehen. Derart entstandene Hypothesen sind demnach Bottom-Up und nicht Top-Down entstanden.

*c* Im ersten Experiment misst Pennington, wie hoch der Anteil durch die Teilnehmer korrekt beantworteter Fragen bzgl. der verschiedenen Verständnisebenen ist. Fragen zur Kontrollflussebene wurden mit 79% am häufigsten und Fragen zur Funktionsebene mit 64% am seltensten korrekt beantwortet.

### Definition: Beacons

Ein Beacon ist definiert als eine Menge von Eigenschaften, die typischerweise auf die Präsenz einer bestimmten Struktur oder Operation innerhalb des Codes hinweisen. (Brooks 1983; Wiedenbeck 1986)

Beacons spielen eine der Schlüsselrollen beim Verstehen von Programmtexten. (Crosby et al. 2002)

Als Beacons dienen unter anderem wiedererkennbare Muster (Wiedenbeck 1986) — z.B. der bekannte Dreizeiler zum Tauschen von Werten zweier Variablen — und sinnvolle Bezeichner (Gellenbeck u. Cook 1991; Teasley 1993).

Erfahrene Programmierer stehen mehr Strategien zum Verstehen von Programmtexten zur Verfügung. Sie sind dadurch nicht auf die Anwesenheit bestimmter Beacons angewiesen (Teasley 1993). Sind beispielsweise Variablen, Funktionen, etc. nicht sinnvoll benannt, können erfahrene Programmierer mehr Informationen aus dem Quelltext extrahieren als Anfänger. Dennoch beschleunigt eine sinnvolle Namensgebung das Verständnis beider Gruppen (Crosby et al. 2002).

## 2.2.5 Fact Finding (LaToza u.a. 2007)

LaToza et al. (2007) sehen ein wachsendes Interesse an der Verbesserung von Softwaretechnik-Werkzeugen. Durch die Anwendung von Programverständnismodellen haben sich allerdings keine nennenswerten Erfolge ergeben.

Mittels einer Laborstudie über 3h mit 13 Entwicklern (0-10 Jahre Programmiererfahrung), die das Design von jEdit (54 KLOC) ändern sollten, fanden LaToza et al. (2007) heraus, dass Programmverstehen durch den Glauben an Fakten getrieben wird. Die gefundenen Fakten bilden dabei die Grundlage für von den Entwicklern getroffenen Entscheidungen.

Die gefundenen Fakten unterschieden sich auf unterschiedlichen Dimensionen:

**Rolle** Fakten können *änderbar* sein, d.h. eine Änderung eines solchen Fakts kann zielführend für eine Aufgabe sein. *Beschränkende* Fakten hingegen müssen gelten und Änderungen solcher Fakten haben unerwünschte Folgen. *Teure* Fakten sind solche, deren Untersuchung mit einem hohen Aufwand verbunden ist.

**Verlässlichkeit**, mit der Fakten Glauben geschenkt werden kann. Einfluss auf diese Dimension hat, wie explizit man einen Fakt gesehen hat und ob es sich bei dem Fakt lediglich um eine Schlussfolgerung handelt.

**Abstraktion** Fakten können auf unterschiedlichen Abstraktionsebenen angesiedelt sein. So ist ein Fakt, der von einem unnötigen Cache spricht abstrakter, als ein Fakt, der von einer Zählschleife spricht, die mindestens 10 mal durchlaufen wird.

**Domäne** Code-Fakten adressieren die Implementierung, wohingegen Anforderungsfakten das Applikationsverhalten bzgl. der Anwendungsdomäne betreffen.

**Intention** Vom ursprünglichen Programmierer unbeabsichtigterweise eingeführte Fakten haben im Gegensatz zu beabsichtigten mit größerer Wahrscheinlichkeit versteckte Abhängigkeiten.

Als Hauptantreiber für das Finden von Fakten gilt *Unsicherheit*, die Entwickler in einem individuellen Maß versuchen, auszuräumen. Die Sensibilität für versteckte Nebenbedingungen war besonders ausgeprägt.

Die Autoren unterschieden die Teilnehmer in Anfänger und Experten. Dabei stellten sie fest, dass die beiden Gruppen zwar die gleichen Praktiken durchführten, diese sich in ihrer Qualität allerdings stark unterschieden:

<b>Praktik</b>	<b>Anfänger</b>	<b>Experten</b>
Suche	Anschauen irrelevanter Funktionen	Übergehen irrelevanter Funktionen
Defektbehebung	Beschreibung und Behebung von Symptomen	Erklärung und Behebung der Urgrundsache
Abstraktion der Chunks	konkret / Statement-Ebene	abstrakt, z.B. "Caching"
Schnelligkeit bei Implementierungen	langsam	schnell

TABELLE 2.1: Unterschiede zwischen Anfängern und Experten während der Ausübung verschiedener Praktiken

Auch wenn die Erkenntnisse banal erscheinen, können Sie dabei helfen, den Gedankengang eines Anfängers oder Experten besser nachvollziehen zu können. Einen weiteren Anwendungsfall nennen die Autoren selbst — wenn auch nur sehr schwammig: Ein Werkzeug, das automatisch Fakten aus Programmcode extrahieren und mit Kontextinformationen nach Relevanz filtern könnte, würde den Verstehungsprozess beschleunigen.

Welche Arten von Fakten es genau gibt, beantwortet die Forschung zum Thema API-Direktiven.



## USABILITY-EVALUATION

Unter der Bezeichnung Usability-Evaluation wird die Bewertung von Benutzerschnittstellen in Bezug auf ihre Benutzerfreundlichkeit begriffen. Was genau unter Usability verstanden wird und welche Termini genau welchen Aspekt davon beschreiben, ist nicht eindeutig und variiert in der Literatur. Nicht einmal Einigkeit besteht darüber, ob nun *Usability* oder *User Experience* dem jeweils anderen Konzept untergeordnet ist. (Kahlert 2011)

Auch wenn die korrekte Übersetzung nach dem Inhalt der DIN EN ISO 9241 *Gebrauchstauglichkeit* lautet, ist mein persönlicher Eindruck, dass in der deutschsprachigen Bevölkerung *Benutzerfreundlichkeit* gebräuchlicher ist. Daher werde ich die beiden Begriffe in dieser Arbeit synonym verwenden.

Sarodnick u. Brau (2006) definieren Usability, basierend auf der ISO-Norm, als “Ausmaß [...], in dem ein technisches System durch bestimmte Benutzer in einem bestimmten Nutzungskontext verwendet werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen”. Es geht also um den *Benutzer*, sein *Arbeitsmittel*, die von ihm zu erledigende *Arbeitsaufgabe* und die *Umgebung*, in der das alles stattfindet.

Usability-Evaluationsmethoden können auf verschiedene Art unterschieden werden. Sarodnick u. Brau (2006) beschreiben die drei Dimensionen *analytisch-empirisch*, *summativ-formativ*<sup>5</sup> und *subjektiv-objektiv*<sup>6</sup>.

Schränken wir den Begriff des Benutzers nicht weiter ein, kommen zur Evaluation, eine ganze Reihe erprobter und im folgenden Abschnitt beschriebener Evaluationsmethoden.

### 2.3.1 Klassische Usability-Evaluationsmethoden

Die Evaluation der Usability von grafischen Benutzeroberflächen ist bereits gut erforscht (Kahlert 2011).

Zu den klassischen Vertretern dieser Gattung gehören Gestaltungsrichtlinien (DIN EN ISO 9241<sup>7</sup>, Eclipse User Interface Guidelines<sup>8</sup>, etc.), formal-analytische Verfahren (GOMS<sup>9</sup>, EVADIS II<sup>10</sup>, etc.).

<sup>5</sup> Summative Verfahren überprüfen die Qualität eines (fast) fertigen Systems, wohingegen formative Verfahren der Verbesserung des sich in der Entwicklung befindlichen Systems eingesetzt werden.

<sup>6</sup> Subjektive Verfahren erheben subjektive Meinungen/Ansichten/Darlegungen der Benutzer. Bei einem Interview wäre dies beispielsweise der Fall. Objektive Verfahren erfassen Daten, die direkt beobachtet wurden.

<sup>7</sup> [http://www.iso.org/iso/catalogue\\_detail.htm?csnumber=52075](http://www.iso.org/iso/catalogue_detail.htm?csnumber=52075)

<sup>8</sup> [http://wiki.eclipse.org/User\\_Interface\\_Guidelines](http://wiki.eclipse.org/User_Interface_Guidelines)

<sup>9</sup> Card et al. (1983)

<sup>10</sup> Oppermann u. Reiterer (1992)

Inspektionsmethoden (Heuristische Evaluation<sup>11</sup>, Cognitive Walkthrough<sup>12</sup>, etc.) und schließlich *der* Usability-Test (Faulkner 2003).

Sind Fragen der Kosteneffizienz zweitrangig, gilt leider für die Qualität der Ergebnisse: viel hilft viel. Jede Usability-Evaluationsmethode erkennt seinen ganz individuellen Ausschnitt aus der Menge aller vorliegenden Usability-Probleme (Fu et al. 2002; Kahlert 2011; Nielsen 2005; Sarodnick u. Brau 2006). Besonders beliebt ist der Methodenmix aus Heuristischer Evaluation und Usability-Test (Fu et al. 2002).

### **2.3.2 Cognitive Dimensions Framework (Green 1989; Green u. Petre 1996; Blackwell u. Green 2000; Blackwell u. Green 2003)**

*Diese Arbeiten sind relevant für den im Abschnitt 3.3 vorgestellten Cognitive-Dimensions-Fragebogen und für den in Kapitel 4 unternommenen Versuch der Validierung meiner Ergebnisse.*

Das CDF<sup>G</sup> ist eine Alternative zu den etablierten Usability-Evaluations-Methoden, wie dem teuren (Faulkner 2003) Usability-Test oder der zu begrenzt einsetzbaren (Nielsen 2005) Heuristischen Evaluation. Dabei wird das zu untersuchende System<sup>a</sup> bzgl. seiner Ausprägung in einer Vielzahl von so genannten *kognitiven Dimensionen* (engl. cognitive dimensions, kurz: CD) beurteilt und seine Eignung für die Durchführung verschiedener Aktivitäten diskutiert.

Ein Beispiel: Man möchte mehr zu der Usability einer Programmiersprache erfahren. Für die Aktivität des Programmierens mittels der betrachteten Programmiersprache würde man die tatsächliche Ausprägung entlang der verschiedenen kognitiven Dimensionen ermitteln und dieses *Profil* mit dem zu diskutierenden *Idealprofil* vergleichen.

Das CDF wurde ursprünglich nicht als Usability-Evaluations-Methode, sondern als Diskussionswerkzeug entworfen. Es sieht für die Dimensionen kein Maß vor, sondern leistet seinen Beitrag mit der Benennung der kognitiven Dimensionen selbst. Erst diese Benennung abstrahiert umschweifende Formulierungen zu einem Terminus, den jeder Diskussionsteilnehmer ähnlich versteht. Der hohe Diskussionsanteil und die Breite der kognitiven Dimensionen sollen das CDF dazu befähigen, auch neuartige (notationelle) Systeme auszuwerten. (Green u. Blackwell 1998)

2003 gab es bereits 60 Veröffentlichungen zum Cognitive Dimensions Framework (Blackwell u. Green 2003). Es ist aber immer noch Forschungsgegenstand. Die im Idealfall disjunkten Dimensionen überlappen sich noch und haben untereinander Abhängigkeiten. Die Anwendung gestaltet sich schwierig, worauf ich noch in den Abschnitten 3.3.4.2 und 4.5.2.1 eingehen werde.

---

<sup>11</sup> Veröffentlicht von Nielsen u. Molich (1990). Es existieren diverse Erweiterungen / Anpassungen, u.a. von Sarodnick u. Brau (2006)

<sup>12</sup> Wharton et al. (1994)

Bis heute wird das CDF eben doch “nur” als Diskussionswerkzeug, und nicht als Evaluationsmethode verwendet. In der Literatur konnte ich nur wenige nennenswerte Ausnahmen (Blackwell u. Green 2003; Roast et al. 2000) finden.

### 2.3.2.1 Entwicklung

Das CDF hat seinen Ursprung in der Arbeit von Green et al. (1980), in der fünf Aspekte<sup>b</sup> hergeleitet wurden, die später als kognitive Dimensionen (engl.: cognitive dimensions; kurz: CD<sup>G</sup>) bezeichnet wurden und signifikanten Einfluss auf die Benutzerfreundlichkeit von Programmiersprachen haben.

Die fortgesetzte Forschung führte zur der Arbeit “Cognitive Dimensions of Notations” (Green 1989). Ähnlich dem Bestreben naturwissenschaftlicher Fächer wie der Physik, die laut der Autoren auf den Grunddimensionen Masse, Länge und Zeit basiert, wird ein Satz von fünf Grunddimensionen für die Bewertung von Programmiersprachen vorgestellt. Zu den in der späteren Literatur am häufigsten zitierten Dimensionen gehört *Role-expressiveness* (Blackwell u. Green 2000; Clarke 2005b; Clarke u. Becker 2003; Corritore u. Wiedenbeck 1999; Crosby et al. 2002; Farooq et al. 2010; Farooq u. Zirkler 2010; Green u. Blackwell 1998; Green u. Petre 1995; Gross u. Kelleher 2009; Ko u. Myers 2004; Teasley 1993). Diese Dimension beschreibt die Einfachheit, mit welcher der Leser der Notation die Rolle der einzelnen Komponenten innerhalb des Systems erkennen kann.

Mit der Arbeit von Green u. Petre (1995) wurden die kognitiven Dimensionen auf 14 (ausführlich in Blackwell u. Green (2003) erläutert) erweitert und als Framework zur Evaluation von visuellen Programmiersprachen und deren Entwicklungsumgebungen verwendet, was den kognitiven Dimensionen ihre bis dahin größte wissenschaftliche Aufmerksamkeit brachte.

### 2.3.2.2 Aufbau

Antriebsmotor für die Forschung waren die fehlenden wissenschaftlich fundierten Kenntnisse zum Entwurf von neuartigen (Computer-)Schnittstellen. Solche Schnittstellen werden als *notationelle Systeme* bezeichnet und sollen nicht nur auf einfache Konzepte wie der Desktop-Metapher, sondern auf jede Art informationeller Artefakte anwendbar sein. Ein notationelles System besteht aus zwei Komponenten “System = Notation + Environment” (Green 1989, 1996, genauer in Blackwell u. Green (2003)):

**Notation** Der ungewöhnliche Begriff Notation wurde gewählt, um eine sehr große Spanne an Interaktions- und Kommunikationsstrukturen zu beschreiben. Synonym wird auch der Begriff *Interaktionssprache* gebraucht. Beispiele für Notationen sind Programmiersprachen, Protokolle, die Notensprache oder die Auszahlung von Geld an einem Bankautomaten.

**Umgebung** Für sich betrachtet ist eine Notation wertlos. Sie entfaltet erst eine Eignung, wenn sie in ein Umfeld eingebettet wird, d.h. einen Zweck erfüllt, über ein Medium mit ihr kommuniziert werden kann und möglicherweise Untersysteme, wie eine Suchen-Funktion, bereitstellt. Beispiele unterschiedlichster Abstraktionsstufe für Umgebungen sind Eclipse, Textverarbeitungsprogramme, oder der Vorlesungssaal des Instituts für Informatik der Freien Universität in der Takustr. 9, 14195 Berlin.

**Interaktionsmedium** Das Interaktionsmedium ist das Bindeglied zwischen dem Nutzer und der Notation. Typische Interaktionsmedien sind Stift & Papier, Bildschirm und Maus & Tastatur. Interaktionsmedien haben eine Reihe von Eigenschaften, zu denen insbesondere *Persistenz* (Stift & Papier sind persistent, der Klick auf einen Push-Button ist transient) und *Reihenfolge* (engl. order; Arbeit mit Stift & Papier hat einen geringen Ordnungsgrad, Interaktion mittels Schallwellen ist sequentiell und hat damit einen hohen Ordnungsgrad) gehören.

**Teilgeräte** Es gibt zwei Typen von Teilgeräten, die in aller Regel eigene Notationen aufweisen und eine eigene CDF-Analyse erfordern: *Hilfsgeräte* ermöglichen eine andere Perspektive auf die Notation, wie das zum Beispiel die Gliederungsansicht mancher Text-Editoren realisiert. *Redefinitionsgeräte* erlauben die Änderung der Hauptnotation. Die Konfigurationsoberfläche der Kurzwahlfunktion von Telefonen ist ein Beispiel für Redefinitionsgeräte.

Eine Notation ohne eine Umgebung, in der sie verwendet wird, ist also weder gut noch schlecht. Erst durch ihre Passung zur Umgebung (Anwender, Einsatzzweck, etc.) ergibt sich eine Wertung. Blackwell u. Green (2000) spitzen die Aussage sogar zu, indem sie sagen, dass Usability eine auf Notation und Umgebung basierende Funktion ist.

Penningtons Spaghetti-Beispiel (siehe 2.2.3.1) eignet sich gut zur Veranschaulichung: Einem Datenflussdiagramm kann man ohne weiteres entnehmen, welche Stationen die Spaghetti durchläuft, bis sie auf dem Teller landet. Welche Funktionen das geschriebene Programm bietet, lässt sich hingegen leichter einer Funktionsbeschreibung entnehmen.

**Kognitive Dimension** Eine kognitive Dimension einer Notation wird von Green informell als eine Charakteristik definiert, die:

- angibt, wie Informationen strukturiert und repräsentiert werden,
- viele Notationen unterschiedlicher Art gemeinsam haben,

- durch seine Wechselwirkung mit der menschlichen kognitiven Architektur einen starken Einfluss auf die Art hat, wie (1) die Notation gebraucht und (2) opportunistisches Planen ermöglicht wird.

Blackwell u. Green (2003) fassen die 14 kognitiven Dimensionen frei übersetzt wie folgt zusammen:

#### **Viskosität (engl. Viscosity)** Änderungsresistenz; Kosten für Änderungen

Ein viskoses System erfordert viele Aktionen, um ein Ziel zu erreichen. Ein Beispiel für eine hohe *Wiederholungs-Viskosität* wäre die Änderung des Formats von Überschriften in *Microsoft Word* ohne die Verwendung von Formatvorlagen. Unter Verwendung von Formatvorlagen hätte man eine geringe Wiederholung-Viskosität. Ein Beispiel für eine hohe *Domino-Viskosität* wäre die Änderung einer API. In einem solchen Fall müssten alle Zugriffe angepasst werden, was möglicherweise weitere Anpassungen erfordert, bis sich das Programm wieder in einem konsistenten Zustand befindet.

#### **Sichtbarkeit (engl. Visibility)** Fähigkeit, Komponenten einfach zu betrachten

Systeme, die Informationen in Verkapselungen “verbergen” (“bury”), verringern die Sichtbarkeit. Derartige Systeme beeinflussen explorative Aktivitäten negativ, wenn es um das Lösen von Problemen geht.

**Vorzeitige Festlegung (engl. Premature Commitment)** Diese kognitive Dimension befasst sich mit Einschränkungen bzgl. der Reihenfolge, in welcher der Anwender etwas tun kann. Diese Einschränkungen zwingen den Anwender in einem bestimmten Moment Entscheidungen zu treffen, an dem die dafür notwendigen Informationen noch nicht vorliegen.

Dieser Fall liegt beispielsweise vor, wenn der Bezeichner einer Klasse gewählt werden muss, obwohl noch nicht klar ist, welche Aufgabe die besagte Klasse haben wird. Ein weiteres Beispiel sind Konfigurationsassistenten, die Einstellungen erfragen, die der Anwender vor Inbetriebnahme des Systems nicht zuverlässig festlegen kann.

Für vorzeitige Festlegungen sind zwei Ursachen typisch: Entweder hatten die Entwickler des Systems eine andere Vorstellung der natürlichen Abfolge von Dingen, oder technische Erfordernisse führen zu einer “unnatürlich” wahrgenommenen Ordnung.

#### **Versteckte Abhängigkeiten (engl. Hidden Dependencies)**

*Wichtige Verknüpfungen zwischen Entitäten sind nicht sichtbar*

Versteckte Abhängigkeiten liegen vor, wenn Abhängigkeiten zwischen Komponenten nicht vollständig sichtbar sind. Jede versteckte Abhängigkeit besteht aus drei Eigenschaften

**Asymmetrisch vs. symmetrisch** Asymmetrische Abhängigkeiten sind unidirektional, zeigen also nur in eine Richtung. Dies ist der Fall bei Links im Word Wide

Web, denn die referenzierte Seite weiß nichts von dem Link, der auf sie zeigt. Symmetrische Abhängigkeiten sind bidirektional. Mit Einrückung formatierte Schleifen in Java beispielsweise haben symmetrische Abhängigkeiten, weil man optisch erkennen kann, auf welches Ende sich der Schleifenanfang bezieht und umgekehrt.

**Lokal vs. entfernt** Lokale Abhängigkeiten zeigen nur einen sehr kleinen Ausschnitt aus dem Abhängigkeitsgraph. Der POSIX-Befehl `ls` zeigt — ohne Parameter aufgerufen — zum Beispiel nur die Unterverzeichnisse und im Verzeichnis enthaltenen Dateien. Im Gegensatz dazu ist der vollständige Verzeichnisbaum ein Beispiel für entfernte Abhängigkeiten.

**Implizit und explizit** Implizite Abhängigkeiten werden im Gegensatz zu expliziten Abhängigkeiten nur über Umwege, oder möglicherweise gar nicht angezeigt. Transitive Abhängigkeiten sind häufig implizite Abhängigkeiten — zumindest so lange, wie man sie nur mit erhöhtem Aufwand ermitteln kann.

Die Suchkosten für eine Abhängigkeit richten sich nach der Länge und Verzweigung der Abhängigkeit sowie dem Aufwand, einer einzelnen Abhängigkeitsverknüpfung zu folgen.

### **Rollenerkennbarkeit (engl. Role Expressiveness)**

*Einfache Erkennbarkeit des Zwecks einer Komponente*

Bei Systemen mit einer ausgeprägten Rollenerkennbarkeit kann die Rolle einer Komponente innerhalb des Systems leicht hergeleitet werden. Die Bewertung der Rollenerkennbarkeit erfordert ein gutes Gespür für kognitive Repräsentationen.

**Fehleranfälligkeit (engl. Error-Proneness)** Die Notation lädt zu Fehlern ein und das System schützt nicht davor.

**Abstraktion (engl. Abstraction)** *Typen und Verfügbarkeit von Abstraktionsmechanismen*

Jenseits der Informatik finden sich viele Beispiele - von Kurzwahlnummern bis hin zur juristischen Person. Abstraktionen gehören praktisch zu den Grundpfeilern der Informatik. Ohne Sie müsste man Transistoren zusammenwürfeln und hoffen, dass etwas Sinnvolles dabei entsteht. Avancierte Abstraktionsmechanismen führen aber möglicherweise zu einer erhöhten Fehleranfälligkeit oder versteckten Abhängigkeiten.

### **Sekundäre Notation (engl. Secondary Notation)**

*Zusätzliche Informationen jenseits der formalen Syntax*

Anwender benötigen häufig Möglichkeiten, Dinge zu notieren, die von der primären Notation nicht vorgesehen wurden. Vorstellbare Ausprägungen sind die Möglichkeit, Kommentare in Programmiersprachen zu verfassen, oder die Farben von Elementen in einem UML-Diagramm ändern zu können. In beiden Fällen wird die primäre Notation nicht beeinträchtigt.

**Domänennähe (engl. Closeness of Mapping)** *Nähe der Repräsentation zur Domäne*

Diese Eigenschaft beschreibt, wie stark die Notation die fachliche Domäne repräsentiert. Die Verwendung der gleichen fachlichen Terminologie erhöht die dimensionale Ausprägung.

**Konsistenz (engl. Consistency)**

*Ähnliche Bedeutungen werden durch ähnliche Syntax ausgedrückt*

Anwender schlussfolgern die Struktur eines Informationsartefakts mithilfe von Mustern. Wenn ähnliche Informationen verschiedenartig dargestellt werden, wird dieser Mechanismus gestört.

**Diffusität (engl. Diffuness)** *“Geschwäzung” der Notation*

Manche Notationen benötigen mehr Platz als andere Dimensionen. Vor- und Nachteile hat das viele: So führen lange Bezeichner in der Regel zu einer besseren Rollenerkennbarkeit, verbrauchen aber auch mehr Arbeitsfläche.

**Schwierige mentale Operationen (engl. Hard Mental Operations)**

*Hohe Beanspruchung kognitiver Ressourcen*

Eine Notation kann Dinge komplex und schwer verarbeitbar machen. Dies ist der Fall, wenn beispielsweise der Anwender viele Informationen unterschiedlicher Abstraktionsgrade für seine Lösung gebrauchen muss.

**Vorläufigkeit (engl. Provisionality)** *Grad der Verbindlichkeit von Aktionen*

Ein System mit hoher Vorläufigkeit kann sich zum Beispiel dadurch auszeichnen, dass es ein Whiteboard zum skizzieren oder Möglichkeiten zum schadlosen Durchspielen von “Was wäre wenn”-Gedanken bereitstellt.

**Fortschreitende Evaluation (engl. Progressive Evaluation)**

*Arbeitsstand kann jederzeit überprüft werden*

Die Dimension beschreibt, in welchem Grad es möglich ist, frei den Moment zu bestimmen, in dem man evaluieren möchte. Viele interpretierte Sprachen haben häufig eine höhere dimensionale Ausprägung als kompilierte Sprachen, da letzte weniger tolerant mit fehlenden Typdeklarationen umgehen.

**Aktivität** Eine kognitive Dimension kann nur hinsichtlich eines Systems und einer Aktivität bewertet werden. Dabei unterscheidet das CDF zwischen den folgenden fünf generischen Aktivitäten (Blackwell u. Green 2003):

**Inkrementierung** beschreibt die Bildung von Inkrementen, wie das zum Beispiel beim Programmieren durch das Hinzufügen von Code-Zeilen geschieht.

**Transkription** beschreibt die Überführung von einer Notation in eine andere.

**Modifikation** des existierenden Artefakts.

**Suche** nach einem bekannten Ziel, wie dem Ort eines Funktionsaufrufs.

**Exploratives Verstehen** unterschiedlichster Art, wie der nach der Grundlage einer Klassifikation oder der Struktur eines Algorithmus.

**Exploratives Design** umfasst zum Beispiel das Zeichnen von Skizzen und andere Techniken, die nicht das finale Produkt darstellen.

Blackwell u. Green (2003) raten davon ab, stark ausdifferenzierte Aktivitäten in der Analyse zu verwenden, da dies die Analyse zeitraubend macht, spezialisiertes Wissen erfordert und nicht die “labile Natur von alltäglichen Aktivitäten” erfasst.

**Profil** Als Profil werden die Ausprägungen entlang der kognitiven Dimensionen für eine Aktivität unter Zuhilfenahme des zu untersuchenden Systems verstanden (Blackwell u. Green 2003).

**Idealprofil** Ein Idealprofil ist ein Profil, das für eine Aktivität die ideale Ausprägung entlang der kognitiven Dimensionen aufweist — unabhängig vom System. Für vier Aktivitäten schlagen Green u. Blackwell (1998) (S. 42) ein jeweiliges Idealprofil vor. Diese Idealprofile sind jedoch sehr grob, empirisch nicht belegt und umfassen lediglich sechs der 14 CDs. Das Idealprofil für die Aktivität *exploratives Design* lautet beispielsweise:

**Viskosität** schädlich

**Versteckte Abhängigkeiten** akzeptabel für kleine Aufgaben

**Vorzeitige Festlegung** schädlich

**Abstraktion** schädlich

**Sekundäre Notation** sehr schädlich

**Sichtbarkeit** wichtig

Blackwell u. Green (2003) betonen, dass die vorgeschlagenen kognitiven Dimensionen angesichts des schlechten Forschungsstandes noch nicht disjunkt und im hohen Maße untereinander Abhängigkeiten haben. Man kann den Forschungstand mit dem allgemeinen Wissen zur Pflanzenpflege vergleichen. Jeder weiß, dass eine gut gedeihende Pflanze Wasser, Licht und Nährstoffe braucht. Allerdings wirken diese Faktoren nicht vollständig unabhängig sondern stehen in Wechselwirkung (Green 1989). Komplexe — im Unterschied zu paarweisen — Abhängigkeiten liegen aber in der Natur der Sache. So kann man ein Gas nicht erwärmen ohne dass bei konstantem Volumen der Druck oder bei konstantem Druck das Volumen steigt (Green u. Blackwell 1998). Abbildung 2.1 zeigt mögliche Abhängigkeiten zwischen einigen kognitiven Dimensionen. So führt die Einführung von Abstraktionen zwar zu einer Verringerung der Viskosität (= weniger Schritte notwendig, um Änderung zu bewirken), erhöht damit aber gleichzeitig den Grad an versteckten Abhängigkeiten. Daraus folgt die Einsicht, dass das Profil eines Systems durch das Schließen von Kompromissen sich dem Idealprofil zwar annähern, es aber nicht zwingend erreichen kann.

### 2.3.2.3 Vorteile

Systeme als notationelle Systeme zu begreifen und mithilfe von kognitiven Dimensionen zu diskutieren, hat gegenüber anderen Usability-Evaluationsmethoden eine Reihe von Vorteilen (Blackwell u. Green 2003; Green 1989, 1996; Green u. Blackwell 1998; Green u. Petre 1995):

- Die Diskussionsergebnisse sind verständlich und allgemein. Die Diskutanten ersticken nicht in Details.
- Das CDF verwendet Begriffe, die von Nicht-Spezialisten verstanden werden können und sie damit in Diskussionen nicht ausschließt. Der regelmäßige gemeinsame Gebrauch von CDs etabliert ein gemeinsames Vokabular für Systemeigenschaften, die zuvor umständlich, missverständlich oder schlicht ungenau beschrieben wurden.

Beispielsweise müsste man mit dem CDF nicht mehr umständlich erklären, dass eine Textverarbeitungssoftware es nicht erlaubt, einfach alle Paragrafen, Überschriften, Tabellen und Abbildungen einheitlich zu ändern, sondern würde sagen, dass die Software mangels ausreichender *Abstraktion* (hier: Formatvorlagen) zu *viskos* (hier: jeder Textbaustein muss einzeln angefasst werden) ist.

- Das CDF lässt sich nicht nur auf interaktive Systeme, sondern auch auf papierbasierte und nicht interaktive Systeme anwenden.
- Das CDF unterscheidet zwischen verschiedenen Typen von Aktivitäten. So stellen beispielsweise die Aktivitäten Programmieren und Debuggen unterschiedliche Anforderungen an das verwendete notationelle System. Green et al. (1991) verwenden den Begriff *Superlativismus* (engl. superlativism) für die Beschreibung des Irrglaubens, dass es eine ultimative Notation gibt, die sich am besten für ein System und aller damit verbundenen Aktivitäten eignet.
- Das Zustandekommen der Ergebnisse mittels CDF-Analyse lenkt den Blick auf Aspekte, die sonst übersehen worden wären. Hauptverantwortlich für diesen Vorteil sind die über Jahre immer besser verstandenen kognitiven Dimensionen, die eine umfassendere Betrachtung verlangen, als man von spontanen Analysen erwarten kann.
- Das CDF kann auch von Nicht-HCI-Experten angewendet werden, wie das häufig bei den Entwicklern eines Systems der Fall ist.

### 2.3.2.4 Anwendung

Das CDF wird üblicherweise wie folgt angewendet (Blackwell u. Green 2003; Green 1996):

1. Kennenlernen des Systems
2. Aufzählung möglicher Aktivitäten mit der Notation
3. Auswahl repräsentativer Aktivitäten
4. Diskussion je Aktivität zu allen kognitiven Dimensionen:
  - (a) Kann der Anwender entscheiden, wo er anfängt?
  - (b) Wie können begangene Fehler behoben / korrigiert werden?
  - (c) Mit welchen Abstraktionen muss der Anwender umgehen?
  - (d) etc.

Die ermittelten Dimensionsausprägungen je Aktivität werden als Profil bezeichnet.

5. Vergleich der erhobenen Profile mit dem Idealprofil für die diskutierte Aktivität

### 2.3.2.5 Fragebogen

Neben der oben beschriebenen klassischen Anwendung des CDFs gibt es eine zweite, die in der direkten Befragung von Nutzern des Systems besteht (Blackwell u. Green 2003, 2000). Die Nutzerschaft zeichnet sich dann zwar nicht durch ihre Expertise in Problemlösungsangelegenheiten bzgl. des Systems aus, dafür aber durch ihre intensive Erfahrung im Gebrauch des Systems selbst.

Basierend auf der Arbeit von Green u. Petre (1995) hat Kadoda (2000) einen Fragebogen zur Evaluation der Benutzerfreundlichkeit entwickelt, der in der Arbeit aber selbst nicht abgebildet ist. Er zeichnet sich dadurch aus, dass er nur die für das System relevanten kognitiven Dimensionen umfasst und diese auf das zu evaluierende System bezieht. Die Beschränkung auf einige kognitive Dimensionen hat allerdings einen entscheidenden Nachteil: Die Evaluation durch den Anwender wird mit der des Experten, der möglicherweise auch noch ein Entwickler des Systems ist, vermischt (Blackwell u. Green 2003).

Blackwell u. Green (2000) haben ebenfalls einen Fragebogen entworfen, der im Gegensatz zum Fragebogen von Kadoda (2000) alle kognitiven Dimensionen umfasst. Ein weiterer Unterschied besteht darin, dass dieser Fragebogen nicht “instanziert”, sondern explizit generisch eingesetzt wird. Die Autoren versprechen sich davon eine besonders einfache und günstige Anwendung. Die Anwendung dieses Fragebogens im Rahmen meiner Forschung hat allerdings gezeigt, dass die dafür notwendige Transferleistung von den Befragten in keinem akzeptablen Zeitrahmen erbracht werden kann und darüber hinaus große Interpretationsspannen zulässt. Unter Abschnitt 3.3.4.2 gehe ich auf meine Erfahrungen näher ein.

Die Gliederung des Fragebogens von Blackwell u. Green (2000) lautet wie folgt:

1. Einführung: Philosophie der CD-Analyse<sup>c</sup>

2. Erfragung von Erfahrung mit den zu evaluierenden und ähnlichen Systemen
3. Definition von CD<sup>G</sup>-Termini
4. Hintergrundfragen zum System
5. Hauptteil: Fragen zur Arbeit des Anwenders mit der wichtigsten Notation  
(Sämtliche Fragen werden unter Anhang B.2 aufgeführt.)
6. Fragen zu Teilgeräten (engl. *sub-devices*, siehe 2.3.2.2, Seite 80)

Innerhalb einer Pilotstudie haben Blackwell u. Green (2000) ihren Fragebogen an einem “breiten Spektrum von Befragten” mit “einem breiten Spektrum von unterschiedlicher Programmiererfahrung” getestet. Genauer gesagt wurden 18 CD-unerfahrene Probanden und 13 Systeme getestet, die von LATEX über computergestütztes Notensetzen bis hin zur C++-Programmierung mittels Emacs reichen. Es wurden in allen CDs Probleme aufgedeckt. Gab es zwei Personen mit dem gleichen zu evaluierenden System, entdeckten diese auch in der Mehrheit der Fälle ähnliche oder gar die gleichen Probleme, was für die Autoren ein Indiz für die Reproduzierbarkeit der Ergebnisse ist. Die Autoren mussten allerdings auch beobachten, dass computerfremde Menschen größere Probleme mit der CD *sub-devices* hatten. Außerdem war das Beziehen der CDs auf sub-devices (wie beispielsweise einem Stift) nicht immer einfach. Die Autoren machten eine besonders interessante Beobachtung: Die befragten Musiker machten bei manchen CDs (insbesondere *role-expressiveness* und *consistency*) kaum Angaben. Die Autoren vermuten, dass dies an der 40-jährigen Unverändertheit der Notensprache liegt und damit außerhalb des Kritikbereichs zu liegen scheint. Blackwell u. Green (2000) konnten bei den Programmierern innerhalb der Probanden häufig feststellen, dass sie die CDs auf den Codeeditor, und nicht auf die Notation (hier also die API) anwendeten. Dies leitet auch auf das größte Problem hin: die Generizität. Nicht jedem Befragten gelingt es, den abstrakten Fragebogen und die darin enthaltenen abstrakten Fragestellungen auf den eigenen Gegenstand zu beziehen. Das fängt bereits bei dem Begriff *Notation* an.

a Was ich hier als System bezeichne, nennen die Autoren “Notation”. Dieser Begriff ist so weit gefasst, dass er sogar über die Grenzen des HCI hinausgeht.

b Die ursprünglichen Aspekte lauten: (1) Größe, (2) syntaktische und strukturelle Komplexität, (3) Unterscheidbarkeit, (4) latente und manifeste Informationen und (5) Werkzeugunterstützung.

c Der Fragebogen “Thinking about Notational Systems” von Blackwell u. Green (2000) beginnt mit den Worten: “This questionnaire collects your views about how easy it is to use some kind of notational system. Our definition of “notational systems” includes many different ways of storing and using information – books, different ways of using pencil and paper, libraries or filing systems, software programs, computers, and smaller electronic devices. The questionnaire includes a series of questions that encourage you to think about the ways you need to use one particular notational system, and whether it helps you to do the things you need.”

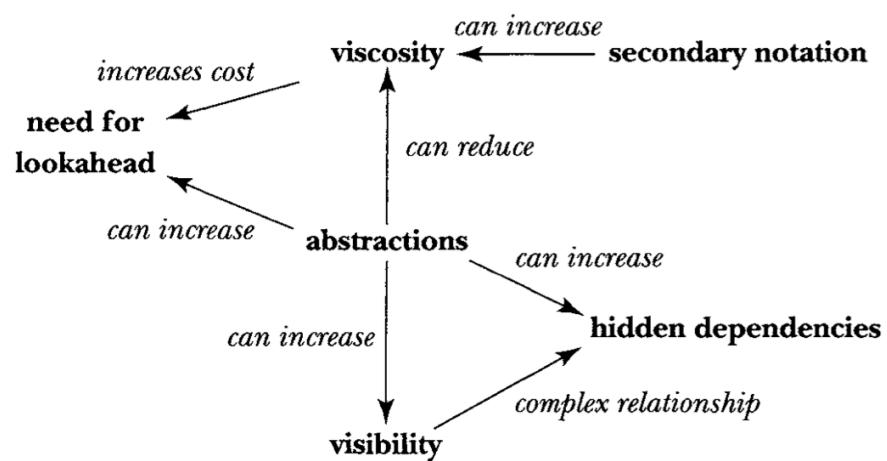


ABBILDUNG 2.1: Typische Abhängigkeiten zwischen kognitiven Dimensionen.  
(Blackwell u. Green 2003)

## API-USABILITY-EVALUATION

Dieser Abschnitt befasst sich mit Usability-Evaluationsmethoden der besonderen Art. Nun besteht die Schnittstelle nicht mehr zwingend aus einer grafischen Benutzeroberfläche, sondern aus einer API<sup>G</sup> API.

Aus diesem Grund spielen die im Abschnitt 1.2 eingeführten Begriffe *API-Entwickler*, *API-Anwender* und *API-Endanwender* im Folgenden eine wichtige Rolle.

### 2.4.1 End-User Programming & End-User Software Engineering (Ko u.a. 2011)

*Diese Arbeit ist relevant für ein korrektes Verständnis der SeqAn-Anwenderschaft, zu denen neben API-Anwendern auch API-Endanwender gehören.*

Ko et al. (2011) haben in ihrer lesenswerten Literaturstudie “The State of the Art in End-User Software Engineering” einen umfassenden Überblick über den Bereich der *Endanwendertechnik* (EUSE) gegeben. Häufiger ist in der Literatur jedoch von *Endanwenderver Programmierung* (engl. *end-user programming*) die Rede. Betrieben wird sie von *Endanwenderver Programmierern* (engl. *end-user programmers*), zu denen Künstler, Architekten, Wissenschaftler, aber auch System-Administratoren gehören können.

Definiert werden Endanwender-Programmierer als Personen, deren Ziel nicht das Programmieren ist, sondern das Programmieren ein “notwendiges Übel” zur Erreichung eines Ziels in ihrer Expertenomäne ist. Die Einordnung von Wissenschaftlern der Bio- oder Wirtschaftsinformatik macht diese lose Definition nicht unbedingt leicht. Daher schlagen Ko et al. (2011) vor, das Unterscheidungskriterium der *Zielgruppe* zu verwenden. Das bedeutet, je mehr eine Person für sich, und damit nicht für andere programmiert, desto eher handelt es sich um einen Endanwender-Programmierer (siehe Abbildung 2.2).

Hat sich die Endanwender-Programmierungsforschung zunächst auf Tabellenkalkulationen und imperative ereignisbasierte Sprachen konzentriert, traten später immer mehr Plattformen, Paradigmen und Softwaretechnikpraktiken in den Fokus der Forschung. Es hat sich gezeigt, dass Endanwender-Programmierer mit allen Bereichen der Softwaretechnik, ob bewusst oder unbewusst, konfrontiert sind. Aus diesem Grund setzt sich immer häufiger der Begriff der *Endanwendertechnik* (EUSE) durch. Die Vorstellung, Endanwender-Programmierer beschränken sich auf die Verwendung von Tabellenkalkulationen, ist überholt. Auch anspruchsvollere Programmiersprachen wie C++ können zum Gebrauch kommen.

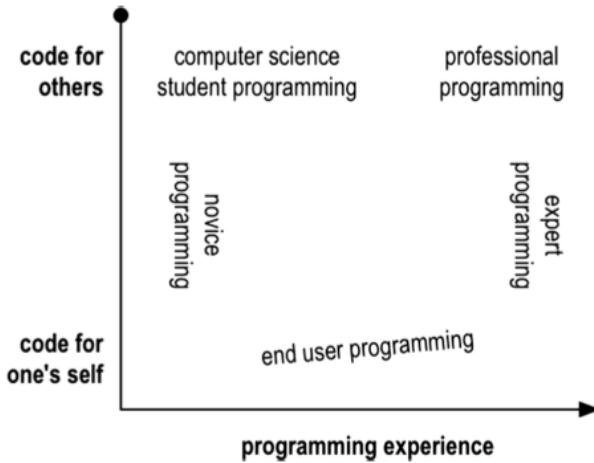


ABBILDUNG 2.2: Einordnung von Endanwender-Programmierern entlang der Dimensionen *Programmiererfahrung* und *Zielgruppe*

Eine besonders extreme, in Tabelle 2.2 dargestellte Beobachtung in Bezug auf Endanwender-Programmierer ist, dass sie sich durch ein opportunistisches (vgl. Abschnitt 2.4.3) und mit einem übersteigerten Selbstbewusstsein geprägtes Verhalten auszeichnet. So werden teils offensichtlich falsche, aber durch ein selbstgeschriebenes Programm berechnete, als korrekte und nicht zu überprüfende Ergebnisse angesehen.

Ein zweite, besonders erwähnenswerte Beobachtung ist, dass für den eigenen Bedarf entwickelte Anwendungen unerwartet oft von Kollegen verwendet und weiterentwickelt werden. Die Anwendung entwickelt sich damit in Richtung Standardsoftware, was durch die unzureichende Beachtung von softwaretechnischen Erkenntnissen und Praktiken wiederum zu Problemen führt (Ko et al. 2011; Letondal 2006). Im Grunde unterscheidet sich die Endanwender-Softwaretechnik von der Softwaretechnik darin, dass die eingesetzten, den Qualitätsaspekten betreffenden Aktivitäten nicht oder nur sekundär verfolgt werden (Ko et al. 2011).

Softwaretechnik-Aktivität	Professionelle technik	Software- technik	Endanwender- Softwaretechnik
Anforderung	explizit		implizit
Spezifikation	explizit		implizit
Wiederverwendung	geplant		ungeplant
Test / Verifikation	vorsichtig		übersteigertes Selbstbewusstsein
Debugging	systematisch		opportunistisch

TABELLE 2.2: Qualitative Unterschiede zwischen professioneller und Endanwender-Softwaretechnik (Ko et al. 2011)

## 2.4.2 Klassische Usability-Evaluation

Beaton et al. (2008b) diskutiert in seiner Arbeit, inwiefern die weiter oben genannten klassischen Usability-Evaluationsmethoden und -techniken zur Evaluation von APIs eingesetzt werden können.

### 2.4.2.1 Think-Aloud-Protokoll

Diese von Beaton et al. (2008b) als “gold standard” bezeichnete HCI-Technik findet häufig Einsatz bei der Evaluation von APIs. Da die Verwender einer API in der Beobachtung beliebige Äußerungen machen können, ist das erfasste Spektrum entsprechend breit und damit die Evaluation schwierig und zeitraubend.

Um dieses Problem zu lösen, schlagen Ellis et al. (2007); Stylos u. Clarke (2007) vor, diese Technik nur bei einfachen Aufgaben anzuwenden. Dies ist zugleich auch ein Nachteil, da Probleme, die nur in komplexen Szenarien auftreten, unentdeckt bleiben. Ein cleveres Verfahren in einer der Arbeiten (Ellis et al. 2007) möchte ich hier unerwähnt lassen: Anstatt Entwickler direkt ein Problem in echten Quellcode lösen zu lassen, sollten sie das Problem zunächst mit Pseudocode lösen. Damit fiel es ihnen viel leichter, auf das mentale Modell der Entwickler zu schließen.

Für quantitative Betrachtungen schlagen Beaton et al. (2008b); Ellis et al. (2007) vor, die Zeit bis Erfüllung der Aufgabe als Maß für die API-Usability zu nehmen.

### 2.4.2.2 Heuristische Evaluation

*Die Heuristische Evaluation wird im Abschnitt 3.2 für eine erste Analyse der SeqAn-API verwendet.*

Nielsen u. Molich (1990) formuliert zehn Heuristiken, die sich dazu eignen, Usability-Probleme durch Experten zu finden. Dieser Ansatz erkennt zwar nicht alle Usability-Probleme, zeichnet sich aber durch seine einfache Handhabung und Kosteneffizienz aus.

Beaton et al. (2008b) schlagen vor, diese Heuristiken auch zur Evaluation der API-Usability zu verwenden. Dabei sollen die Heuristiken *consistency and standards*, *error prevention* und *help and documentation* besonders relevant sein — eine Einschätzung, die ich ebenfalls teile.

Speziell zur Evaluation von API-Usability gibt es nur eine Heuristikaufstellung (Grill et al. 2012), die allerdings konstruktive Mängel aufweist und auf die ich im Ausblick eingehe.

Ein zweiter Vorstoß kommt von Watson (2012), der systematisch drei Heuristiken zur Bewertung von API-Dokumentation entwickelt und validiert hat. Diese lauten Ersteindruck (engl. *initial*

*impression)*, Erlebnis (engl. *experience*) und zusätzliche Daten (engl. *additional data*). Auf diese Heuristiken gehe ich im Abschnitt 4.4.8 genauer ein.

Etwas weniger praktisch anwendbar sind die von Correia et al. (2009) vorgestellten Muster für eine konsistente API-Dokumentation, die in der Forschung leider wenig Beachtung fanden. In den Arbeitsergebnissen sehe ich das Potential, dass sie als Grundlage für Heuristiken für die Evaluation von API-Dokumentationen dienen können.

Abgesehen von den eben genannten Arbeiten konnte ich in der Literatur keine weiteren Forschungsergebnisse zu API-Heuristiken finden.

#### 2.4.2.3 Cognitive Walkthrough

Bei der Anwendung des Cognitive-Walkthrough-Verfahrens (Wharton et al. 1994) werden Annahmen über die idealtypischen Benutzer gemacht, deren Beschreibung sich in Form von Personas (vgl. Abschnitt 2.4.3) anbietet. Dies ist eine Stärke in Bezug auf APIs, wenn die Nutzerschaft unbekannt oder wegen des frühen API-Entwicklungsstandes noch nicht bekannt ist.

Das Cognitive-Walkthrough-Verfahren sieht vor, dass das Evaluatorenteam Anwendungsszenarien beschreibt, deren Erfüllungsgrad durch die Anwendung an Hand von vier Fragen überprüft wird. Diese Fragen werden in ihrer Form von Beaton et al. (2008b) als nicht verwendbar beschrieben, da die Entwicklung mit einer API viel mehr Freiheitsgrade zulässt als bei der Verwendung eines Endanwender-Programms. Daher reformulierten Beaton et al. (2008b) die Fragen wie folgt:

- Welches Ziel hat der Benutzer, wenn ihm dieses Szenario präsentiert wird?
- Welche Optionen werden von dem Benutzer wahrgenommen?
- Welche wahrgenommenen Optionen scheinen am meisten umsetzbar?
- Welches Ergebnis wird der Benutzer erreichen?

#### 2.4.2.4 Cognitive Dimensions Framework

*Dieser Abschnitt ist relevant für den im Abschnitt 3.3 vorgestellten Cognitive-Dimensions-Fragebogen und für den in Kapitel 4 unternommenen Versuch der Validierung meiner Ergebnisse.*

Trotz des engagierten und nicht ganz unerfolgreichen Versuchs von Blackwell u. Green (2000), einen generischen CD-Fragebogen zu erarbeiten, konzentrierten sich andere Arbeiten auf eine Spezialisierung der CDs. Zu den relevantesten Arbeiten gehört “Using the Cognitive Dimensions Framework to evaluate the usability of a class library” von Clarke u. Becker (2003). Wie der Name schon sagt, haben die Autoren in ihrer Arbeit Greens CDs so angepasst, dass sie sich in besonderer

Weise zur Evaluation von APIs eignen. Dies schließt auch etwaige Dinge wie Programmiersprache, Entwicklungsumgebung, Debugging-Werkzeuge und Dokumentation ein.

Die Anpassung der CDs geschah auf zweierlei Weise:

**Terminologie** Clarke und Becker haben die generische Terminologie so deduziert, dass sie von API-Entwicklern verstanden wird. Dabei entfiel insbesondere der Term *Notation*, welcher durch  $API^G$  ersetzt wurde.

**Dimensionen** Abgesehen von der Benennung waren manche Dimensionen auch inhaltlich nicht angemessen für die Evaluation von API. Darüber hinaus liegt es auf der Hand, dass es auch CDs geben kann, die nicht Teil der von Green vorgeschlagenen CDs sind. Letzteres Problem haben die Autoren auf interessante Weise gelöst: Sie haben bereits API-Usability-Tests mittels Videoaufzeichnung und anschließender Auswertung durchgeführt und “signifikante Verbesserungen” erreicht. Teile der Ergebnisse waren allerdings nicht durch die CDs von Green herzuleiten und wurden daher als Grundlage zur Formulierung weiterer CDs verwendet.

Eine prominente neue Dimension heißt *Work-Step Unit*. Sie beschreibt, wie viel Arbeit durch eine typische Codezeile gelöst wird.

Die von Clarke entwickelten 12 CDs lauten wie folgt:

**Abstraktionsebenen (Abstraction Level)** Wie lauten die von der API angebotenen minimalen und maximalen Abstraktionsebenen? Sind sie für die Zielgruppe nutzbar?

**Lernanforderungen (Learning Style)** Welche Lernanforderungen werden durch die API an den API-Anwender gestellt? Welche Lerntypen werden unterstützt?

**Arbeitsgedächtnisanforderungen (Working Framework)** Wie viele konzeptuelle Informationen / Chunks muss der API-Anwender im Kopf behalten, um mit der API effektiv arbeiten zu können?

**Arbeitsschrittgröße / Codedichte (Work-Step Unit)** Welcher Umfang einer Programmieraufgabe wird durch einen einzelnen Schritt (i.d.R. ein Statement) erledigt?

**Fortschreitende Evaluation (Progressive Evaluation)** In welchem Umfang kann teilweise vollständiger Code ausgeführt werden, um Feedback zum Programmverhalten zu erlangen?

**Verfrühter Entscheidungzwang (Premature Commitment)** In welchem Umfang muss der API-Anwender Entscheidungen treffen, bevor ihm alle dafür notwendigen Informationen vorliegen.

**Durchdringbarkeit (Penetrability)** In welchem Umfang wird der API-Anwender unterstützt, die Komponenten der API zu explorieren, zu analysieren und zu verstehen? Wie gelangt der API-Anwender an diese Informationen?

**API-Anpassungnotwendigkeit (API Elaboration)** In welchem Umfang muss die API angepasst werden, um die Bedürfnissen eines API-Anwenders zu erfüllen?

**Anpassungsfreundlichkeit (API Viscosity)** Wie schwierig und aufwändig sind inhärente Änderungen am auf der API basierenden Code?

**Konsistenz (Consistency)** Inwiefern kann über die API erlerntes Wissen auf den Rest der API übertragen werden?

**Rollenerkennbarkeit (Role Expressiveness)** Wie einfach ist die Rolle einer Komponente innerhalb der API als Ganzes zu erkennen?

**Problementsprechung (Domain Correspondence)** Wie deutlich hat eine API-Komponente eine Entsprechung in der Problemdomäne.

Die Autoren betonen, dass diese Aufzählung nicht vollständig ist, da sie aus lediglich einer API-Usability-Studie extrahiert und an Hand von drei Studien validiert wurde. Dennoch finden diese API-CDs bereits regen Gebrauch beim Arbeitgeber der Autoren (Microsoft) und konnten bereits nachweislich API-Verbesserungen verbuchen.

Allerdings sind Clarkes API-CDs nicht vollständig nachvollziehbar. Clarke entfernte beispielsweise die CD<sup>G</sup> *error-proneness*, obwohl diese für die Diskussion von Notationen im Softwarebereich von Blackwell u. Green (1999) als eindeutig hilfreich eingestuft wird. Gleches gilt für die CD<sup>G</sup> *juxtaposability*. Diese CD<sup>G</sup> kann am besten mit Gegenüberstellbarkeit übersetzt werden. In dem Artikel von Blackwell u. Green (1999) wird ein Beispiel genannt, in dem ein Diagramm in Code übersetzt wird (Aktivität: Transkription) und die Anwenderin Diagramm und Code gleichzeitig sehen wollte. An dieser Stelle könnte man das Argument ins Feld führen, dass Blackwell u. Green (1999) von Programmertätigkeiten im Allgemeinen und Clarke u. Becker (2003) konkret von der Arbeit mit API<sup>G</sup>'s sprechen. Ich halte die Kürzung aber für einen Fehler, da man sich für beide CD<sup>G</sup>'s problemlos Beispiele vorstellen kann, die im Kontext von API<sup>G</sup>'s relevant sind.

Ich konnte exakt zwei Anwendungen von Clarkes API-CDs in der Literatur finden:

Die erste Anwendung findet bei *Microsoft* in Form einer API-Usability-Evaluationsmethode statt.

In zwei Nachträgen (Clarke 2003, 2005b) erklärt Clarke, wie dabei verfahren wird:

- Für die Erhebung der CDs einer API werden fünf API-User eingeladen, um API-relevante Aufgaben zu lösen.
- Dabei werden die Teilnehmer verschiedenartig beobachtet. Zu den typischen Beobachtungsverfahren gehören Think-Aloud, Beobachtung durch einen semi-transparenten Spiegel und Videoaufzeichnungen.
- Für die Erledigung der Aufgaben hat jeder Entwickler 2h Zeit.
- Die Beobachtungen werden von Spezialisten ausgewertet und ein CD-Profil wird erstellt.
- Stylos u. Clarke (2007) haben drei Personas entwickelt (*Opportunisten*, *Pragmatiker* und *Systematiker*), auf die ich näher im Abschnitt 2.4.3 eingehe. Für jede Persona existiert ein vorab erstelltes CD-Idealprofil.
- Das erstellte CD-Profil wird mit den Idealprofilen der Personas verglichen, die am ehesten der Zielgruppe entsprechen.
- Je mehr sich die CD-Profile gleichen, desto höher ist die Usability.
- Die CDs selbst werden dann als Diskussionsgrundlage verwendet, um eine zielgerichtete Anpassung des CD-Profil zu erreichen.

Leider halten sich die beiden Arbeiten in den Details auffällig zurück. Es wird weder darauf eingegangen, wie die drei Idealprofile nun konkret aussehen, noch, mit welchem Erfolg die CDs eingesetzt wurden.

Die zweite Anwendung findet sich in der empirischen Arbeit von Piccioni et al.. Unter anderem wurden die Probanden dieser Studie mit einem Fragebogen konfrontiert. Laut der Autoren basiert dieser auf Clarkes API-CDs. Kritisch sehe ich an deren Vorgehen folgende Punkte:

1. Die Autoren zeigen nicht, welche der gestellten Fragen sich auf welche kognitive Dimension stützt. Obwohl ich mich intensiv mit dem CDF auseinandergesetzt habe, gelang es mir nicht, eindeutig diese Bezüge herzustellen.
2. Offensichtlich werden nicht alle CDs durch die Fragen abgedeckt. Unter den Fragen, die ich eindeutig auf die jeweilige CD beziehen konnte, gab es zwei, die auf dieselbe CD abzielten. Da es 12 API-CDs und 12 Fragen gab, muss gemäß dem Taubenschlagprinzip mindestens

auf eine CD verzichtet worden sein. Da die Autoren den Anspruch formulierten, neue problematische API-Usability-Aspekte zu ermitteln, sehe ich es kritisch, die erfragten CDs zu kürzen.

Zusammenfassend stelle ich fest, dass Piccioni et al. das CDF in seinem ursprünglichen Sinne, nämlich als Diskussionswerkzeug, verwenden. Die Verwendung als API-Usability-Evaluationsmethode wird lediglich in Clarke (2003, 2005b) beschrieben — das allerdings unzureichend.

Im Abschnitt 6.1.2.3 gebe ich einen Ausblick auf Weiterentwicklungsmöglichkeiten des CDF im Allgemeinen und seiner Anwendung als Evaluationsmethode für APIs im Speziellen.

### 2.4.3 Personas (Clarke 2007)

*Die von Clarke entwickelten Personas sind relevant für die in Kapitel 4 vorgestellten ● Strategien, die ich bei den SeqAn-Anwendern beobachten konnte.*

Für die Evaluation von Usability und damit auch von API-Usability ist ein Verständnis des Benutzers selbst notwendig (Sarodnick u. Brau 2006). Die Anforderung ist leicht zu erfüllen, wenn es sich um Individualsoftware bzw. eine individuelle API handelt. Sprechen wir aber von Standardsoftware, sind Nutzergruppen mannigfaltig und schwer zu erfassen.

Personas sind Personenbeschreibungen von Repräsentanten verschiedener Nutzergruppen und eignen sich als Alternative zum szenariobasierten Interaktionsdesign. Eine Persona ist facettenreich und ihre Beschreibung umfasst mehr als die Aufzählung ihrer demographischen Werte (Pruitt u. Grudin 2003). Dabei hat die Glaubwürdigkeit einer solchen Beschreibung Vorrang vor Diversität und politischer Korrektheit<sup>a</sup> (Cooper 2004).

Clarke (2007); Stylos u. Clarke (2007) haben die folgenden drei — nicht ganz stereotypfreien — Personas entwickelt, die sich zur Evaluation von Standard-APIs eignen:

**Opportunistische Entwickler** arbeiten *bottom-up* und möchten sich nicht mit Low-Level-Problemen auseinandersetzen. Sie möchten ihren Code schnell zum laufen bringen, ohne ein tieferes Verständnis von der zugrunde liegenden API zu erlangen.

**Pragmatische Entwickler** sind defensiver und lernen “unterwegs”. Sie arbeiten wie opportunistische Entwickler *bottom-up*. Sollte diese Strategie scheitern, wechseln sie zu *top-down*, um ein besseres Verständnis zu erhalten. Sie wägen gerne Einfachheit und Kontrolle ab. Beispielsweise verwenden pragmatische Entwickler häufig gern grafische Code-Editoren. Dennoch möchten sie über die Option verfügen, den automatisch erstellten Code nachbearbeiten zu können. Typischerweise präferierte Sprachen sind Java und C#, die ein ausgewogenes Verhältnis zwischen Einfachheit und Kontrolle bieten.

**Systematische Entwickler** arbeiten *top-down* und versuchen, ein Gesamtverständnis vom System zu erhalten, bevor sie sich auf eine Komponente konzentrieren. Sie programmieren defensiv und machen wenig Annahmen über den Code / der API. Sie misstrauen den Garantien, die eine API<sup>G</sup> macht und testen lieber die Funktionstüchtigkeit in der eigenen Umgebung. Sie möchten nicht nur verstehen, ob, sondern auch warum der Code läuft, welche Annahmen er macht und wann er versagen könnte. Sie ziehen Sprachen wie C++, C oder Assembler vor, die ihnen eine hohe Kontrolle erlauben.

Die Beschreibungen lassen sich mit den Erkenntnissen von Shaft u. Vessey (1998) in Einklang bringen. Zur Erinnerung: Die Arbeit integrierte die Ergebnisse von Brooks (1983) und Pennington (1987), indem sie die Domänenkenntnisse der Entwickler mit in Betracht zog. Demnach beobachtet man bei Entwicklern mit wenig Domänenkenntnissen, dass diese häufig ein Bottom-Up-Vorgehen anwenden. Entwickler, die über viele Domänenkenntnisse verfügen, benutzen häufiger ein Top-Down-Vorgehen. Diese Herangehensweise wird als *flexibel* bezeichnet und entspricht damit der Pragmatischer-Entwickler-Persona.

Entwickler mit einer unflexiblen Herangehensweise hingegen wenden immer das Top-Down- bzw. das Bottom-Up-Vorgehen an, entsprechen also der Systematischer-Entwickler- bzw. Opportunistischer-Entwickler-Persona.

Umgekehrt ausgedrückt hieße das: Bei opportunistischen und systematischen Entwicklern spielen die Domänenkenntnisse keine Rolle bei der Wahl der Herangehensweise — aber für den Erfolg. Pragmatische Entwickler präferieren das Bottom-Up-Verfahren und wechseln — abhängig von auftretenden Problemen und ihren Kenntnissen der Domäne — zum Top-Down-Verfahren.

Schreibt man einer realen Person eine anders definierte Persona zu, muss man damit rechnen, dass diese Zuschreibung nicht immer gilt bzw. sich über die Zeit verändert. Dies ist der Fall, wenn Personas über das Alter, die Erfahrung oder den Beruf definiert sind, was bei Langzeitbeobachtungen problematisch sein kann.

Die drei von Stylos u. Clarke (2007) beschriebenen Personas leiden weniger unter dieser Gefahr, da sie unabhängig von Beruf, Expertise, Ausbildung und Erfahrung gelten (Clarke 2007; Stylos et al. 2006). Für die Sinnhaftigkeit und Anwendbarkeit dieser Personas spricht, dass sie die Beobachtungen von Shaft u. Vessey (1998) ergänzen und in verschiedenen API-Usability-Arbeiten erfolgreich angewandt wurden (Stylos u. Clarke 2007; Stylos et al. 2008; Stylos u. Myers 2008). Clarke (2007) nutzte diese Personas bereits vier bis fünf Jahre vor ihrer Veröffentlichung für seine Forschung.

<sup>a</sup> Cooper (2004) nennt ein amüsantes Beispiel, dass ich Ihnen nicht vorenthalten möchte: Zur Beschreibung eines Computertechnikers zieht er die Persona Nick — ein pickeliges 23jähriges Ex-Mitglied der Technik-AG seines damaligen Gymnasiums — der Persona Hellene — einer klassischen 1,80m großen Schönheit der Beverly Hills High — vor.

## 2.4.4 Metriken (Stylos u. Myers 2007)

Diese Arbeit ist relevant für die in Kapitel 4 vorgenommene Unterteilung von Entwurfsentscheidungen.

Robertson (2007) haben sich im Rahmen ihrer Arbeit Expertenmeinungen, komparative Laborstudien, informelle Onlinediskussionen und Forschung zum Thema objektorientierte Programmierung angesehen und die vielen ungeordneten, sich teils überlappenden oder gar synonymen Qualitätsfaktoren/-merkmale/-eigenschaften verglichen. Die gefundenen Qualitätsfaktoren wurden zu den folgenden drei Kategorien (vgl. Abbildung 2.3) zusammengefasst. Metriken, mit denen sich die Qualitätsfaktoren messen lassen, erläutert die Arbeit nicht.

### Stakeholders

Bestehen aus den Entwicklern und Anwendern der API sowie aus den Verbrauchern der durch die API-Anwender entwickelten Programme.

### Usability

Diese Qualitätsfaktoren beeinflussen das Erzeugen und Debuggen des von API-Anwendern entwickelten Codes.

### Mächtigkeit

Diese Qualitätsfaktoren betreffen Beschränkungen des von API-Anwendern entwickelten Codes.

Für meine Arbeit ist ein anderer Beitrag der Autoren wichtig. Robertson (2007) schlagen zwei Perspektiven vor, unter denen API-Design-Entscheidungen geordnet werden können (Abbildung 2.4). Die erste Perspektive unterscheidet zwischen Design-Entscheidungen auf Architektur- bzw. Sprachebene. Die zweite Perspektive unterscheidet zwischen strukturellen Design-Entscheidungen und solchen auf Klassenebene.

Robertson (2007); Stylos et al. (2006) kritisieren, dass viele in der Literatur gefundenen Entwurfsempfehlungen (z.B. “Returning null versus returning an empty object (i.e., an empty string).”) nie von den jeweiligen Autoren überprüft wurden und teilweise widersprüchlich sind.

Verschiedene Design-Entscheidungen haben unterschiedliche Prioritäten für ihre Behebung — abhängig von ihrem Einfluss auf die Eigenschaften der API. Stylos et al. (2006) nennen drei Dimensionen, über die sich die Priorität herleiten lässt:

- Die *betroffene Partei* - also sind API-Entwickler oder API-(End-)Anwender von der Designentscheidung betroffen.

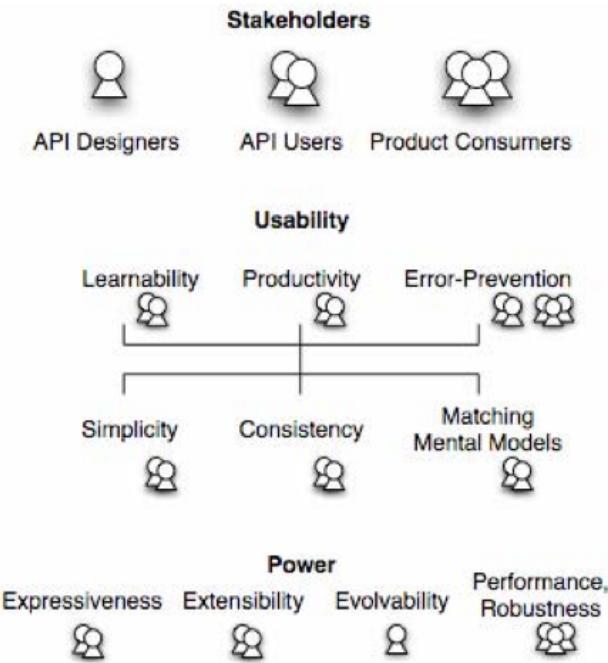


ABBILDUNG 2.3: Qualitätseigenschaften von APIs (Robertson 2007)

- Die *Frequenz*, mit der man von einer Designentscheidung konfrontiert wird.
- Die *Schwierigkeit* des Umgangs mit einer Designentscheidung.

Aus dem besser erforschten Gebiet der klassischen (Nicht-API-)Usability kommt noch eine weitere Dimension dazu, die von Nielsen u. Mack (1994) als *Persistenz* bezeichnet wird. Persistenz beschreibt, ob ein Usability-Problem bei erneutem Auftreten auch immer wieder aufs Neue als störend empfunden wird. Darüber hinaus schlagen Sarodnick u. Brau (2006) *Markteinfluss* als boolesche Eigenschaft vor. Diese Eigenschaft ist lediglich relevant, wenn die Nicht-Behebung eines Problems kommerzielle Folgen, z.B. durch Nicht-Verkauf des Produkts, wahrscheinlich macht.

## 2.4.5 Lernbarrieren (Ko u.a. 2004)

Ko et al. (2004) haben 40 Endanwender-Programmierer (im Sinne von EUSE<sup>G</sup>) beobachtet, die im Begriff waren, die Sprache *Visual Basic.NET* über einen Zeitraum von fünf Wochen zu erlernen. Die Probanden konnten dabei stets auf ein Orakel (einer der Experimentatoren) zurückgreifen, das alle aufgekommenen Fragen beantwortete. Ebenso gaben die Probanden auf Fragen des Experimentators ("Wo hängst du?", etc.) Antwort.

Ziel war es herauszufinden, durch welche Arten von Problemen das Erlernen erschwert wird (vgl. Abbildung 2.5).

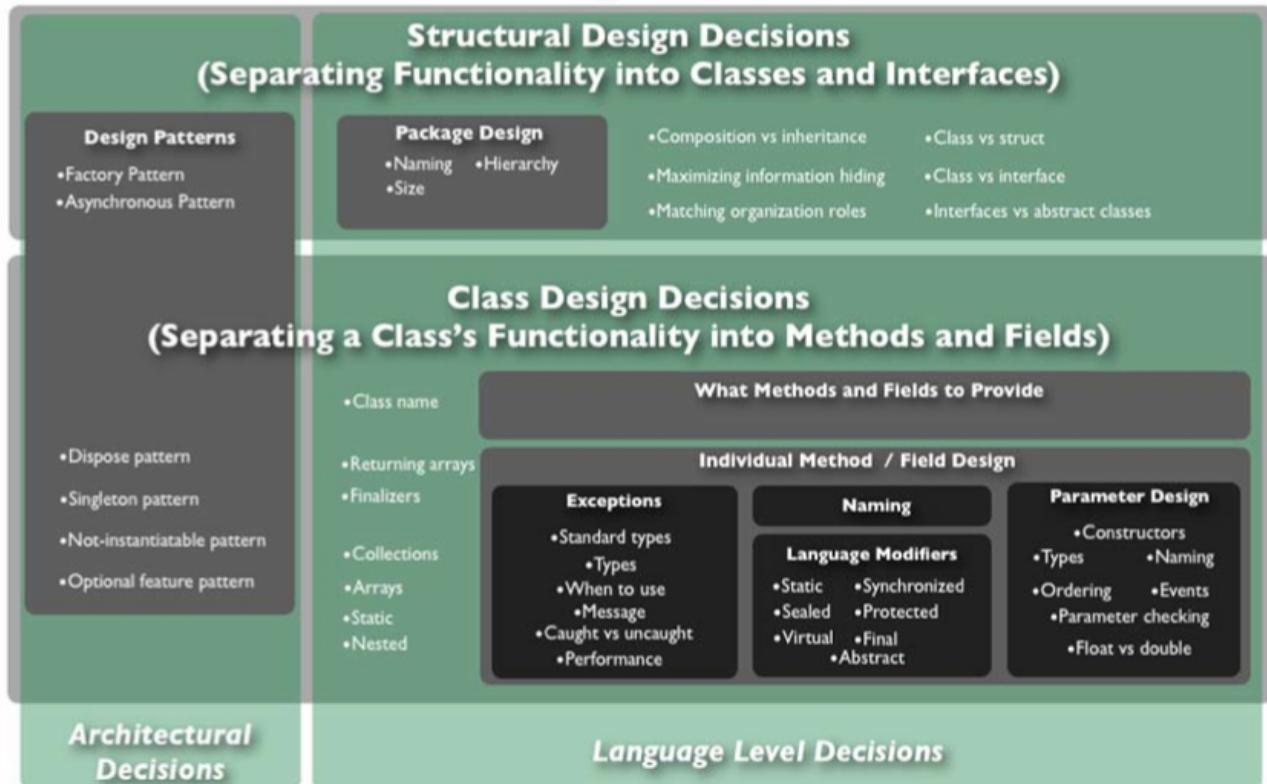


ABBILDUNG 2.4: Kategorisierung von API-Designentscheidungen (Robertson 2007)

Die Autoren konnten 130 Lernbarrieren ermitteln, die sie zu den folgenden sechs Kategorien zusammenfassen:

#### Design Barriers “*I don't know what I want the computer to do...*”

Designbarrieren sind inherent schwierig. Sie zu überwinden, fordert Kreativität. Entwickler des Systems werden von den Autoren aufgefordert, die Kreativität der Endanwender-Programmierer mit Hilfe von Beispielen und anderen Formen zu inspirieren.

#### Selection Barriers “*I think I know what I want the computer to do, but I don't know what to use...*”

Selektionsbarrieren sind schwierig zu überwinden. Bei diesem Problem kann eine Suchfunktionen helfen, die es erlaubt, nach dem Verhalten von Komponenten zu suchen.

#### Coordination Barriers “*I think I know what things to use, but I don't know how to make them work together...*”

Eine wichtige Größe bei diesem Problemen sind unsichtbare Regeln, deren Existenz und Nicht-einhaltung häufig nur durch Fehlerausgaben sichtbar gemacht wird. Dekel (2011) hat eine Möglichkeit vorgestellt, wie dieses Problem weitgehend gelöst werden kann (siehe Abschnitt 2.4.7).

#### Use Barriers “*I think I know what to use, but I don't know how to use it...*”

Gebrauchsbarrieren treten sehr schnell auf. Ursachen sind die mangelhafte Dokumentation des

Interfaces. Besonders wichtig zu dokumentieren sind Feedback und Beschränkungen von Funktionen. Die textuelle Repräsentation von Systemen erschwert Endanwender-Programmierern deren Erlernen.

**Understanding Barriers** *"I thought I knew how to use this, but it didn't do what I expected..."*

Verständnisbarrieren gehen auf fehlende Erklärungen zurück, die Auskunft geben, was das Programm getan bzw. nicht getan an. Nachvollziehbarkeit wird als Hauptursache genannt.

**Information Barriers** *"I think I know why it didn't do what I expected, but I don't know how to check..."*

Nachvollziehbarkeit wird auch für Informationsbarrieren als Hauptursache genannt.

Problematisch werden Lernbarrieren, wenn eine Person falsche Annahmen zur Überwindung der Barriere trifft. Dies kann dazu führen, dass der Anwender ein falsches Verständnis von der zu erlernenden Sprache erwirbt, die ihn ultimativ darin hindert, weitere Barrieren zu überwinden. Diese werden von den Autoren als *unüberwindbar* bezeichnet (siehe Abbildung 2.6).

Ko et al. (2004) argumentieren, dass die ermittelten Lernbarrieren auch auf professionelle Entwickler anwendbar sind. Ich schlussfolgere aus der Top-Down-Theorie von Brooks (1983), dass professionelle Entwickler mit Hilfe der Hypothesenbildungsstrategie und ihrem reicherer Erfahrungsschatz potentiell weniger Probleme bei der Überwindung von Lernbarrieren haben als Endanwender-Programmierer.

Die Anwendbarkeit der Lernbarrieren auf APIs argumentiere ich mit der Vermutung, dass Endanwender-Entwickler nicht hinreichend zwischen Programmiersprache und API unterscheiden. Auch könnte man sagen, dass eine Programmiersprache im Kern eine mögliche Schnittstelle zur Interaktion mit einem Computer darstellt, so wie eine API eine Schnittstelle zur Interaktion mit einer Softwarebibliothek ist. Green (1989) (vgl. Abschnitt 2.3.2) würde in beiden Fällen von einer Notation sprechen, die es zu erlernen gilt.

Für die Anwendbarkeit jedoch spricht am meisten, dass ich noch vor der Lektüre dieser Arbeit alle Informationsbarrieren außer der Koordinationsbarriere in meinen Daten finden konnte<sup>8</sup>.

## 2.4.6 Abstrakte API-Aktivitäten (Stylos 2009)

*Die Dissertation von Stylos ist besonders relevant für die in den Abschnitten 4.3.9 und 4.4.8 vorgestellte Überarbeitung der SeqAn-Dokumentation.*

In seiner Dissertation "Making apis more usable with improved api designs, documentation and tools" fasst Stylos (2009) seine jahrelange Forschung zusammen und stellt sein Verständnis vom Erlernen/Gebrauch von Java-basierten APIs vor. Dabei spielen folgende in Abbildung 2.7 dargestellten Aktivitäten eine Rolle:

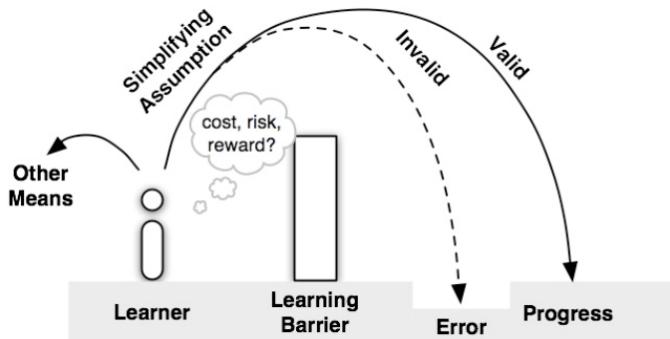


ABBILDUNG 2.5: Beim Überwinden einer Lernbarriere riskieren Erlerner, eine falsche Annahme zu treffen.

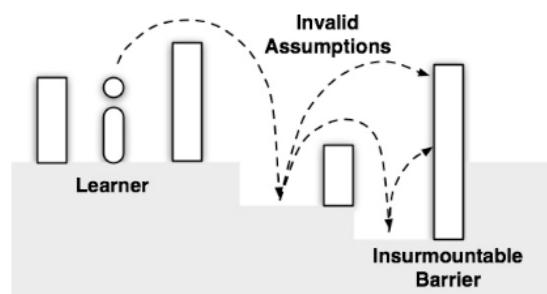


ABBILDUNG 2.6: Treffen Erlerner beim Überwinden von Lernbarrieren falsche Annahmen, führt dies häufig zu unüberwindbaren Barrieren.

#### (A) Initiale Designideen

Bei dieser Aktivität entwickelt der API-Anwender ein erstes Verständnis darüber, wie er ein zuvor formuliertes Problem programmatisch lösen kann.

#### (B) Abstraktes API-Verständnis

In dieser Phase verschafft sich der API-Anwender einen Überblick über existierende und geeignete APIs. Um diese Phase erfolgreich zu absolvieren, muss jede API einen Überblick über sich geben.

Immer häufiger kommen für diese Phase Suchmaschinen wie Google zum Einsatz. Insbesondere für API-Anwender mit wenig Erfahrung stellt dies einen großen Vorteil dar, da sie nicht mit der sonst notwendigen Terminologie vertraut sein müssen. Konkret nennt Stylos (2009) die Suchphrase “creating a form in java” als Beispiel, denn “form” ist in der Java<sup>G</sup>-Entwicklung ein eher unüblicher Begriff und findet in offiziellen Dokumentationen fast keinen Gebrauch.

#### (C) Architektonisches Design

Nachdem Anwender die abstrakten Elemente einer API angefangen haben zu verstehen, entwickelt sich eine Idee, wie eine konkrete API für die eigene Problemlösung genutzt werden kann.

**(D) Auffinden von Methoden**

Diese Phase besteht darin, die Methoden zu finden, die für die Problemlösung gebraucht werden. Erste Hinweise darauf finden Anwender bereits im Zuge von Tutorials oder anderen Artikeln, die auf relevante Methoden/Funktionen verweisen.

Bei Anwendern von Google konnte Stylos (2009) eine höhere Performanz feststellen, denn die Suchergebnisse boten sowohl Code-Beispiele von Dritten als auch Referenzen die offizielle Dokumentation. So bekamen API-Anwender einen schnellen Eindruck über Anwendungsszenarien und konnten dies gleich mit Hilfe der offiziellen Dokumentation verifizieren.

**(E) Auffinden von Beispielen**

Wurden beim Durchlaufen der vorangegangenen Phasen noch keine relevanten Code-Beispiele gefunden, wird die Suche in dieser Phase aufgenommen. Dabei kam wiederum Google häufiger zum Einsatz als die offizielle Dokumentation selbst. Angetrieben wurde diese Suche durch Fragen, wie die folgenden:

- “Wie instanziere ich die zu der Methode gehörende Klasse?”
- “Wie bekomme ich Variablen des Typs, die eine Methode als Parameter verlangt?”
- “An welcher Stelle meines Codes muss ich diese Methode aufrufen?”

**(F) Integration von Beispielen**

Gefundene Beispiele reichten von einer Zeile Code bis hin zu vollständigen Programmen, von denen der gesuchte Teil im besten Fall einen Bruchteil ausmachte. Probleme bei der Integration dieser Beispiele ähneln denen von Fairbanks et al. (2006) (siehe oben). Gelingt es dem Anwender nicht, das Beispiel zu integrieren, sucht er nach einem anderen Beispiel für dieselbe Methode (E), einer anderen Methode (D), ändert seinen Entwurf (C) oder sucht nach einer anderen API (B).

Die Beobachtung, dass Google eine wichtige Rolle in den oben zusammenfassten Phasen spielt, veranlasste Stylos u. Myers (2006), selbst ein Suchwerkzeug zu entwickeln. Dieses verwendet Google selbst und bietet im Rahmen von APIs eine treffsichere Suche. Das Werkzeug wird etwas genauer im Abschnitt 2.7.2 vorgestellt.

## 2.4.7 API-Direktiven (Dekel 2011)

*Diese Arbeit ist relevant für die in den Abschnitten 4.3.9 und 4.4.8 vorgestellte Überarbeitung der SeqAn-Dokumentation.*

Dekel (2011) formulierte “neighbor knowledge awareness”-Probleme als solche, die entstehen, wenn

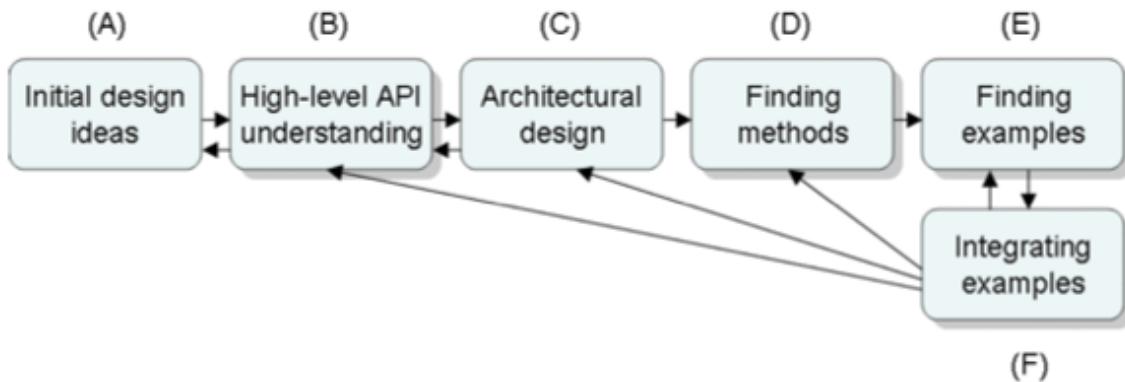


ABBILDUNG 2.7: Abstrakte Programmertätigkeiten (Stylos 2009)

konzeptionell zusammengehörige Entscheidungen und Annahmen über mehrere *Artefakte* (Diagramme, Dokumentation, Quelltext) verteilt sind und später nicht mehr in ihrer Gesamtheit sichtbar gemacht werden.

Beispiel: In einem Objektdiagramm wird die Vererbungshierarchie der Java-Klassen `Widget` und `Button` gezeigt. Dass aber `super.dispose()` aufgerufen werden muss, wenn die `dispose`-Methode der `Button`-Klasse überschrieben wird, findet sich lediglich in der Online-Dokumentation. Wie soll der Entwickler also wissen, ob er über alle relevanten Informationen verfügt? Andererseits würde eine stets vollständige Spezifikation Gefahr laufen, wichtige Details zu verdecken (Dekel 2011).

Natürlichsprachige Aussagen, die solche Entscheidungen, Annahmen, Bedingungen oder Richtlinien einer API betreffen, werden als *API-Direktiven* bezeichnet und haben erst seit kurzem wissenschaftliche Aufmerksamkeit auf sich gezogen (Dekel 2011; Monperrus et al. 2011). Beispiele für API-Direktiven sind “must not be null” und “not being thread-safe”.

Besonders interessant sind überraschende Klauseln in einer Dokumentation wie die der Java-Methode `String.replaceAll(String)`, die den übergebenen String als regulären Ausdruck behandelt, was schnell zu einem unerwarteten Ergebnis führen kann.

Dekel (2011) zeigte in seiner Dissertation, dass das Einblenden von API-Direktiven an der Stelle, wo das Wissen auch benötigt wird, den Entwicklern ein gezielteres Erforschen tangierender Funktionen ermöglicht. Dieser Ansatz wird von Dekel als *Knowledge Pushing* bezeichnet. Er implementierte dazu ein Eclipse-Plugin namens *eMoose* (Abbildung 2.14). Das Plugin reichert unter anderem den gelben Eclipse-Javadoc-Hilfsdialog mit gefundenen API-Direktiven an. Das Plugin wurde lediglich im für die Veröffentlichung notwendigen Umfang implementiert<sup>a</sup> und seitdem nicht weiterentwickelt<sup>b</sup>.

Dekel (2011) und Monperrus et al. (2011) haben folgende Richtlinien entwickelt, um API-Direktiven von der übrigen Dokumentation zu unterscheiden und exakt zu formulieren:

- Direktiven müssen ein klar definiertes API-Element identifizieren. So darf beispielsweise eine Klassen-bezogene Direktive nicht von den Methoden dieser Klasse sprechen. (Monperrus et al. 2011)
- Direktiven müssen genau erklären, in welchen Fällen sie relevant sind. (Monperrus et al. 2011)
- Direktiven müssen eine Aktion erfordern oder implizieren. (Dekel 2011)
- Direktiven dürfen keine vagen Formulierungen wie “sollte” verwenden und stets eine peinlich korrekte Terminologie benutzen. <sup>a</sup> (Monperrus et al. 2011)
- Direktiven sollten nicht trivial, erwartet oder üblich sein. (Dekel 2011)
- Direktiven sollten relevant für die meisten Aufrufer und Szenarien sein. Andernfalls würden sie ihre Wirkung verfehlten und erneut vom Wesentlichen ablenken. (Dekel 2011)
- Direktiven müssen bei Nicht-Beachtung Konsequenzen haben. Je gravierender die Konsequenzen, desto kleiner darf die Menge von Aufrufern und Szenarien sein. (Dekel 2011)

Basierend auf den Dokumentationen wichtiger Java-APIs und den Arbeiten von Dekel (2011) und Bruch et al. (2010) haben Monperrus et al. (2011) eine Taxonomie von 23 API-Direktiv-Typen entwickelt, die in Abbildung 2.8 dargestellt werden.

<sup>a</sup> In der Arbeit von Dekel (2011) musste der Autor die Direktiven manuell erkennen und gruppieren. Außerdem stellte die unzureichende Filterung / Selektion ein Problem dar. Den Anwendern von eMoose wurden stellenweise zu viele Direktiven angezeigt. Neben einfachen Lösungen wie dem Ausblenden einer Direktive, nachdem sie gelesen wurde, macht der Autor auch den Vorschlag, den umgebenden Programmtext mit einzubeziehen.

<sup>b</sup> Stand: 16.03.2015, <https://code.google.com/p/emoose-cmu/>

<sup>c</sup> Beispiel: Man soll von “erweitern” (“extend”) sprechen, wenn `super.function()` aufgerufen wird bzw. werden muss. Andernfalls wird “überschreiben” (“override”) gesprochen.

## 2.4.8 Grundlagen / Psychologie / Strategien

Es fiel mir schwer, für diesen Abschnitt eine gute Überschrift zu finden. Dieser Abschnitt befasst sich mit Beobachtungen, die nicht so recht in die anderen Abschnitte passen wollen, jedoch von durchgängiger Relevanz für meine Arbeit sind.

### 2.4.8.1 Komplizierte Technik (Sarodnick u.a. 2006; Daughtry u.a. 2009a

Daughtry et al. (2009a) haben in ihrer Forschung mehrfach die Beobachtung gemacht bzw. direkt Aussagen dazu gehört, dass Programmieren nun einmal komplizierter ist als der Gebrauch einer grafischen Benutzeroberfläche. Schließlich handele es sich ja um eine Schnittstelle für einen Entwickler und nicht für einen Endanwender.

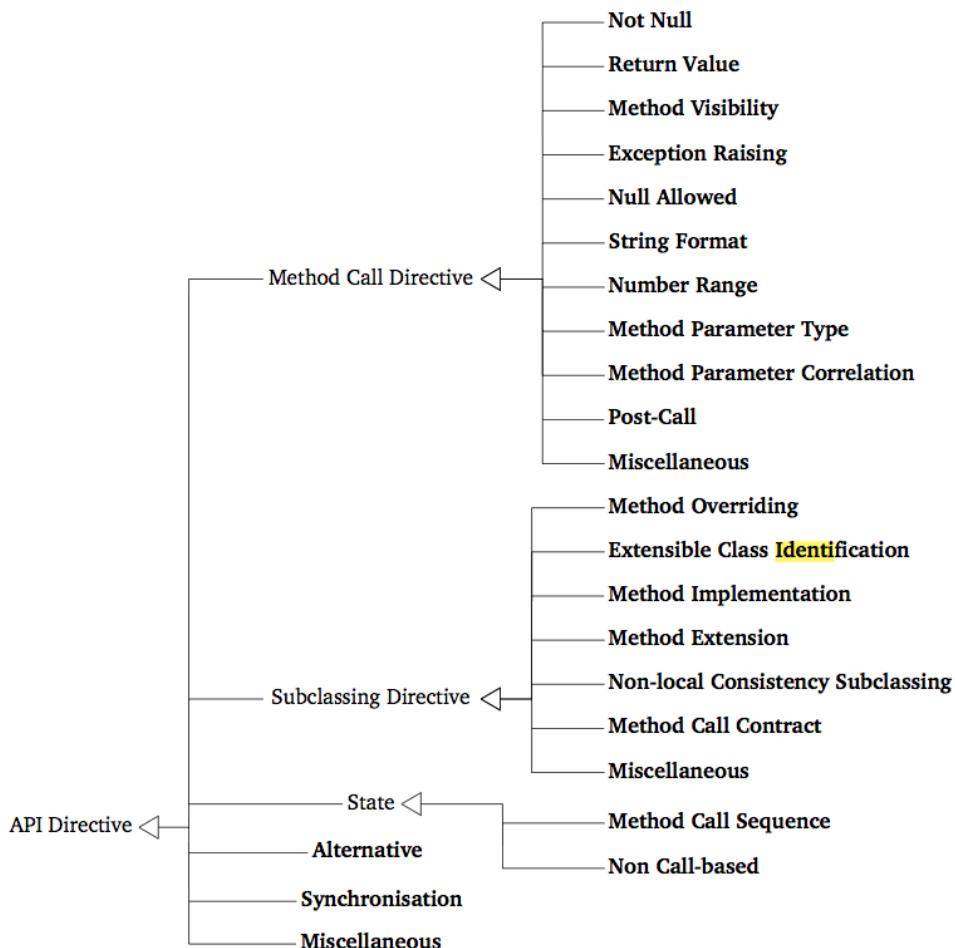


ABBILDUNG 2.8: API-Direktiven-Taxonomie (Monperrus et al. 2011)

Ich konnte eine Steigerung dieser Beobachtungen in der Literatur finden (Sarodnick u. Brau 2006). Dabei geht es darum, dass durch Anwender eines technischen System geäußerte Kritik von den Systementwicklern häufig „technisch wegargumentiert“ wird. Im Umfeld von API-Usability würde es sich um einen Konflikt zwischen API-Entwicklern und API-(End-)Anwendern handeln, den ich konkret während meiner Forschung beobachten konnte (siehe Abschnitte 3.3.3 und 4.2.3).

Das Wissen um dieses Phänomenon ist relevant, wenn der Forscher mit subjektiven Evaluationsmethoden die API-Usability verbessern will.

#### 2.4.8.2 Wiederverwendung

(Lange u. Moher 1989; Rosson u. Carroll 1996; Fairbanks u.a. 2006; Stylos u. Myers 2008)

Wiederverwendung ist nicht nur eines der Grundprinzipien von Softwaretechnik, sondern auch eine Strategie, die durch API-(End-)Anwender genutzt wird und sich auch in meinen Forschungsergebnissen zeigt.

Ich konnte in der Literatur zwei Arten der Wiederverwendung in diesem Sinne finden:

### Implementierungswiederverwendung

Lange u. Moher (1989) haben in ihrer Arbeit einen interessanten Fall beobachtet: Ein erfahrener Entwickler hatte eine existierende Softwarebibliothek, die verschiedene wiederverwendbare Komponenten bereitstellt, um neue Komponenten erweitert. Um dies zu tun, kopierte er häufig eine ähnliche Komponente und passte diese an.

Rosson u. Carroll (1996); Stylos u. Myers (2008) machten ähnliche Beobachtungen. Dabei wurden konkret ganze Klassen kopiert und so lange angepasst, bis die angepasste Klasse ihren neuen Zweck erfüllte.

Obwohl diese Form der Wiederverwendung beobachtet wurde, hat keiner der genannten Autoren ihr jemals einen Namen gegeben. Dieser stammt daher von mir.

### Anwendungswiederverwendung

Rosson u. Carroll (1996) beschreiben die Wiederverwendungsform *reuse of uses*, also das Wiederverwenden von Anwendungen. Die Autoren machten die Beobachtung, wie API-(End-)Anwender vor der Frage standen, wie ein bestimmtes API-Konstrukt (Datenstruktur, Funktion, etc.) denn nun genutzt wird. Dazu suchten manche Anwender nach einem ähnlichen Anwendungskontext (z.B. in Form von Code-Beispielen), die dann häufig als verbindliche Vorlage kopiert und angepasst wurden.

Diese Strategie wird von Fairbanks et al. (2006) kritisch gesehen. Sie nennen drei Schwierigkeiten:

1. Das Auffinden solcher konkreten Anwendungsfälle ist schwierig und zeitaufwändig.
2. Die Bestimmung relevanter Teile ist fehleranfällig.
3. Die Bewahrung der Intention wird sogar als *unmöglich* ("impossible") beschrieben.

Stylos (2009) stellt fest, dass zum Auffinden von Anwendungsfällen immer häufiger die Suchmaschine *Google* zum Einsatz kommt.

#### 2.4.8.3 Verständnisvermeidung (Lange u. Moher 1989; Stylos u. Myers 2008)

Dieses Phänomen / diese Strategie ist eines der interessantesten Dinge, die ich überhaupt finden durfte.

Lange u. Moher (1989); Stylos u. Myers (2008) verstehen darunter die fahrlässige bis aktive Vermeidung, wiederverwendeten Code im Detail zu verstehen. Die Wiederverwendung besteht also im blinden Kopieren und Anpassen vom Code.

Im Sinne von Programmverständnis (vgl. Abschnitt 2.2) erklären Lange u. Moher (1989) damit, dass der Anwender den Code nur *interaktiv*, aber nicht *symbolisch* ausführt.

Ko u. Myers (2005) beobachten diese Strategie häufig im Kontext von EUSE<sup>G</sup>, was, wegen der potentiell schlechteren Softwaretechnikkenntnisse von API-Endanwendern, leicht nachvollziehbar ist.

Bruch et al. (2006) schildert in seiner Arbeit, dass APIs inherent abstrakter und damit schwerer zu verstehen sind. Schließlich werden APIs mit dem Ziel entworfen, flexibel und für ein breites Spektrum von Anwendungen einsetzbar zu sein. Daher reicht es nicht aus, eine einzelne Klasse zu verstehen. Vielmehr muss das Entwurfskonzept verstanden werden.

Ich vermute, dass Verständnisvermeidung durch eine niedrigschwellige und knappe Einführung in den API-Entwurf weniger häufig bzw. stark zu beobachten wäre (vgl. Abschnitt 4.4.8). Auch der folgende Abschnitt deutet darauf hin.

## 2.4.9 Weitere Methoden & Techniken

Aus Platzgründen und aus Gründen der Relevanz möchte ich die andere von mir gesichtete Verfahren und Techniken an dieser Stelle nur knapp beschreiben.

### 2.4.9.1 API-Walkthrough (O'Callaghan 2010)

O'Callaghan (2010) beschreibt dieses Verfahren als im Grunde einer Mischung aus klassischem Cognitive Walkthrough und klassischen Usability-Test. Das heißt, der API-Walkthrough findet in einer Laborumgebung statt. Allerdings erledigen die Probanden keine typischen Aufgaben, sondern arbeiten sich Zeile für Zeile durch den Quellcode durch. An Hand der Beobachtungen können die Experimentatoren auf das mentale Modell der Probanden schließen und Probleme diagnostizieren.

### 2.4.9.2 API (Usability) Peer Review (Farooq u. Zirkler 2009, 2010; Umer Farooq 2010)

Der *API (Usability) Peer Review* (Farooq et al. 2010; Farooq u. Zirkler 2009, 2010) ähnelt methodisch der klassischen Code-Inspektion (Kemerer u. Paulk). Dabei werden Probleme nach ihrem *Typ*, ihrer *Priorität* und *Schwere* sortiert.

Anstatt existierendes Wissen aus der HCI-Forschung auf den Bereich der Softwareentwicklung zu konkretisieren, erfinden die Autoren leider eine Menge neu. Die Idee, Prioritäten entlang der Meilenstein-Planung<sup>13</sup> zu orientieren, macht Sinn. Jedoch halte ich es für äußerst einschränkend, die *Schwere*

13 Mögliche Prioritäten sind beispielsweise: *sofortige Behebung* und *Behebung zum nächsten Meilenstein*.

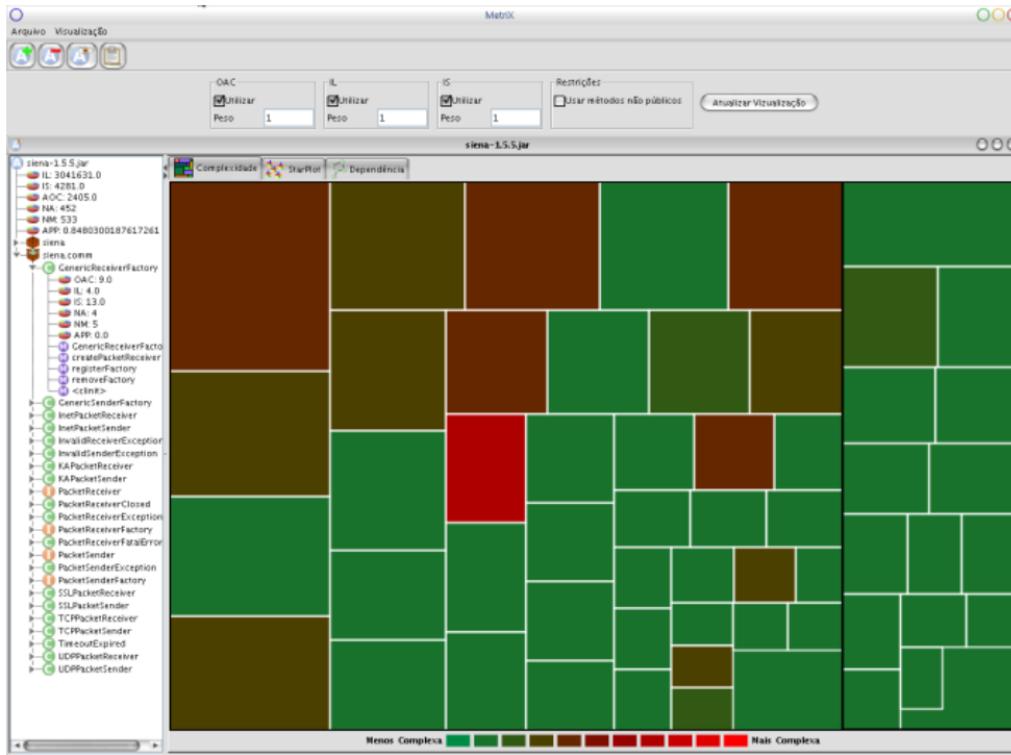


ABBILDUNG 2.9: Visuelle Darstellung der Usability der Siena API, <http://www.sienaproject.com/index.html> (de Souza u. Bentolila 2009)

(*gering, moderat, schwerwiegend* und *unklar*) über den Einfluss auf die Funktionalität zu definieren. Schließlich ist Funktionalität nur ein Teilapekt von Usability. In den Arbeiten von Nielsen (1993), Sarodnick u. Brau (2006) und Kahlert (2011) sind Ordnungssysteme vorgestellt, die das breite Spektrum von Usability erfassen und sich bewährt haben. Konkret werden u.a. die *Frequenz*, der *Einfluss* und die *Persistenz* eines Usability-Problems berücksichtigt.

Dieses Verfahren lässt sich leicht in den Workflow integrieren, wenn ohnehin schon klassische Code Reviews durchgeführt werden. Es ist effizienter als ein Usability-Test, da es mehr Probleme je Zeit findet. Außerdem können mit dieser Art des Reviews konzeptionelle Schwachstellen gefunden werden (Farooq u. Zirkler 2010). Weshalb letzteres der Usability-Test nicht können soll, verschweigen die Autoren jedoch.

#### 2.4.9.3 Metrix (de Souza u. Bentolila 2009)

Die Arbeit von de Souza u. Bentolila (2009) stellt ein objektiv-summatives Verfahren vor. Das dazugehörige Werkzeug macht den Versuch, die Usability einer API automatisch zu analysieren. Das Ergebnis wird grafisch dargestellt (siehe Abbildung 2.9) und soll einen potentiellen API-(End-)Anwender bei der Beurteilen helfen, ob er die API nutzen oder sich nach einer Alternative umschauen sollte.

#### 2.4.9.4 Concept Maps (Gerken u. a. 2011)

Die Verfahren wird von Gerken et al. (2011) als longitudinaler Ansatz zur Bewertung von API Usability beschrieben. Der Terminologie von Sarodnick u. Brau (2006) folgend, handelt es sich um eine empirisch-formativ-subjektive Evaluationsmethode.

Bei dieser Methode werden nicht nur einfache, sondern auch komplexe Problembewältigungen beobachtet, gemeinsam mit den Probanden besprochen und in Form einer auf einem Whiteboard festgehaltenen Concept Map dargestellt. Dazu treffen sich Experimentatoren und API-Anwender in regelmäßigen Abständen. Die API-Anwender werden aufgefordert, vorgegebene API-Konzepte und benutzerdefinierte Prototypkonzepte über benannte Kanten zu verbinden und existierende Verbindungen zu revisionieren. Problematische Bereiche können die API-Anwender rot umkreisen.

Über die Zeit entsteht nicht nur eine Abbildung der mentalen Modelle der API-Anwender, sondern auch eine Dokumentation, wie sich dieses Verständnis über die Zeit ändert. Nachteilig an diesem Verfahren ist, dass eine enge Zusammenarbeit zwischen API-Entwicklern und -Anwendern erforderlich ist. Dies setzt eine geographische Nähe wie auch eine hohe Bereitschaft und Offenheit auf beiden Seiten voraus.

#### 2.4.9.5 Guidelines / Best Practice

Guidelines und Best-Practice-Zusammenstellung, wie die des Buches “Effective C++: 50 Specific Ways to Improve Your Programs and Designs” (Meyers 1992), stellen eine wertvolle Lektüre für API-Entwickler dar. Dennoch beschränken sich diese Ansätze eher auf das *Wie*. Das *Warum* wird allenfalls in Form von Usability-verschlechternden Gegenbeispielen dargestellt. Im Sinne einer API-Usability-Evaluation können derartige Aufzählungen als Checkliste verstanden werden, deren Verstoß vermutlich ein oder mehrere Usability-Probleme hervorrufen können. Für andere Sprachen wie Java (Bloch 2008) gibt es ähnliche Werke.

Robillard u. DeLine (2010) stellt fest, dass die üblichen Guideline-Werke nicht viel mehr als Strukturierungstechniken gegen unbeabsichtigte Komplexität und Hinweise zur Namensgebung geben.

Aus meiner Sicht tragen Guidelines und Best-Practice am meisten dazu bei, einen Konsens zu etablieren und Konsistenz über viele APIs hinweg herzustellen, wodurch sich der Lernaufwand für API-Anwender mit jeder neuen API verringert.

## API-USABILITY-VERBESSERUNG

Dieser Abschnitt befasst sich — im Gegensatz zu Abschnitt 2.4 — nicht mit einzelnen “Zutaten” der API-Usability-Evaluation, sondern mit Arbeiten, die konkrete APIs analysierten und verbesserten.

### **2.5.1 “Participatory Programming: Developing Programmable Bioinformatics Tools for End-Users” (Letondal 2006)**

Die partizipative Feldstudie von Letondal (2006) ist mit Abstand die spezifischste Veröffentlichung, die ich in meinem Forschungskontext finden konnte, denn sie befasst sich mit Bioinformatikern und reicht in die Bereiche EUSE und API-Usability hinein.

Motiviert wird ihre Arbeit durch die Feststellung, dass die Entwicklung anwenderfreundlicher Bioinformatiksoftware nicht mehr mit der schnellen Forschung im Bereich der Bioinformatik Schritt hält. Neue Hypothesen, Algorithmen und immer größere Datenmengen führen dazu, dass bioinformatische Software ihre Anwender zum Programmieren zwingt.

Biologen und Bioinformatiker, die mit dieser Softwareklasse arbeiten wollen, müssen also programmieren und sind damit Endanwender-Programmierer nach der Definition von Ko et al. (2011).

Letondal argumentiert, weshalb existierende Werkzeuge für die neuen Ansprüche der Bioinformatik unzureichend sind. Werkzeuge, die sich unmittelbar an Endanwender-Programmierer wenden, nutzen häufig domänenpezifische Sprachen (DSL) oder gar Endanwender-Programmierer-Sprachen (EUP). Diese Sprachen sind aber in aller Regel nicht Turing-vollständig, wie das die meisten *General Purpose Languages* (GPL) sind. Turing-Vollständigkeit ist aber notwendig, um mit der schnellen Entwicklung Schritt zu halten.

Des Weiteren kritisiert Letondal, dass die Anwender von Bioinformatik-Werkzeugen in deren Entwicklung wenig involviert werden.

Um diese Probleme zu lösen und Bioinformatikern ein adäquates Werkzeug an die Hand zu geben, hat sie im Rahmen ihrer mehrjährigen Arbeit am *Laboratoire d’Informatique Interactive*<sup>14</sup> der *École nationale de l’aviation civile*<sup>15</sup> regelmäßig Interviews, Brainstorming-Sessions und informelle Beobachtungen durchgeführt.

Die Interviews fanden in einem nicht näher spezifizierten Zeitraum statt und wurden teilweise videoaufgezeichnet.

---

<sup>14</sup> <http://lii-enac.fr/en/index.html>

<sup>15</sup> <http://www.enac.fr/en/>

Die Brainstorming-Sessions fanden in den Jahren 1996-2004 mit 5-30 Teilnehmern (vornehmlich Forscher) statt. Die Bioinformatiker wurden also direkt in die Entwicklung dieses Werkzeugs einbezogen.

Letondal stellte fest, dass ihre Probanden immer ihre favorisierten Sprachen (meistens *Perl*) einsetzten, statt eine zu verwenden, die sich für ihre Probleme besser eignet. Sie beobachtete auch, dass ihre Probanden zu einem opportunistischen Arbeitsstil (vgl. Clarke 2007; Ko et al. 2011) neigten und das Programmieren nach Möglichkeiten mieden. Des Weiteren fand Letondal heraus, dass Anwender bereits an den einfachsten Aufgaben scheiterten, wenn das Programm keine expliziten oder einfach zu gebrauchende Funktionen dafür bereitstellte. Probanden äußerten an dieser Stelle häufig das Bedürfnis, die Software anpassen zu wollen.

Letondals Arbeit resultierte in dem Werkzeug *biok*, das eine Mischung aus integrierter Entwicklungsumgebung und Analysewerkzeug darstellt. Anwender dieses Werkzeugs können, abhängig von ihren Fähigkeiten, ausschließlich das Werkzeug nutzen (“programming *with* the user interface”) oder es aber erweitern (“programming *in* the user interface”). Besonders an diesem Ansatz ist die Möglichkeit, das Programm aus sich heraus, nämlich mit Hilfe der eingebauten Entwicklungsumgebung, weiterzuentwickeln. Erweiterungen liegen dabei in einem separaten Verzeichnis und können nicht nur Code hinzufügen, sondern auch beliebigen existierenden Code überschreiben<sup>16</sup>. Werden dabei versehentlich kritische Methoden überschrieben und dadurch *biok* funktionsuntüchtig gemacht, können die Erweiterungen einfach wieder entfernt oder verschoben werden.

Letondal bezeichnet diesen Ansatz als *reflektive Architektur* bzw. *reflektive Umgebung*, welche das Kontinuum zwischen Verwendung und Programmierung abdeckt.

Die Gesamtheit aus partizipatorischem Design (Stiemerling et al. 1997) und Programmierbarkeit des Produkts wird von der Autorin als “partizipatorische Programmierung” bezeichnet. Ein ähnliches Konzept wird von Carroll et al. (1990) behandelt und ist unter der Bezeichnung *Programming In The User Interface* bekannt.

Die Arbeit von Letondal ist sicherlich ein Exot. Das Werkzeug wurde nicht nur unter Einbeziehung der späteren Anwender entwickelt, sondern kann sogar durch sie eigenständig weiterentwickelt werden. Leider geht die Autorin nicht darauf ein, inwiefern die API selbst durch ihre Probanden mitgestaltet wurde. Unabhängig davon ist die erfolgreiche Einbeziehung der zukünftigen Anwender in den Entwicklungsprozess einer erweiterbaren Bioinformatikanwendung eine Erwähnung an dieser Stelle wert.

---

<sup>16</sup> Das Werkzeug basiert auf *XOTcl* (<http://media.wu.ac.at>) und dem *Tk graphical toolkit* (<http://www.tcl.tk>). Bei XOTcl handelt es sich um eine Objektorientierungserweiterung für Tcl. XOTcl zeichnet sich durch seine extreme Erweiterbarkeit aus. So können Klassenmethoden zur Laufzeit überschrieben werden, obwohl es bereits Exemplare der Klasse gibt.

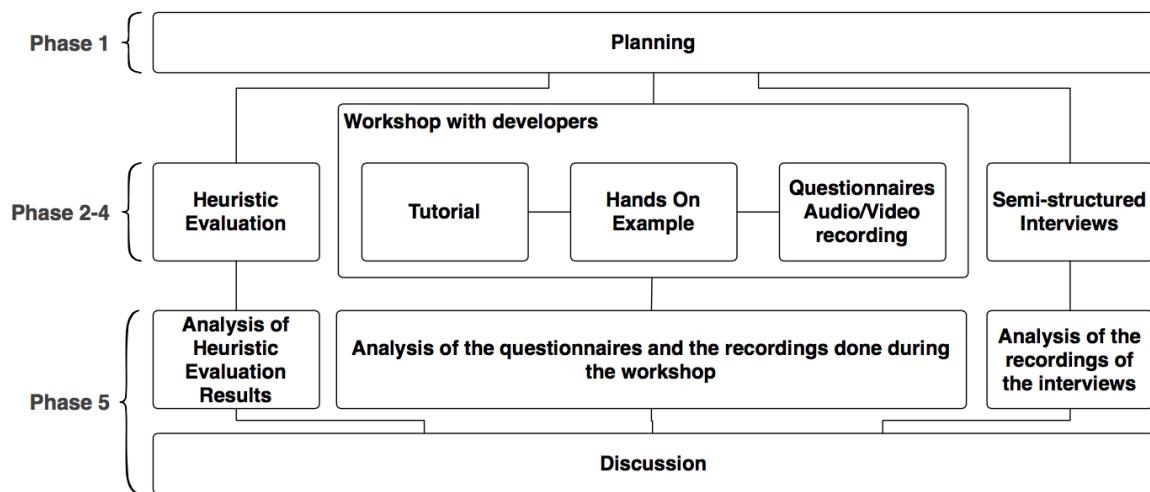


ABBILDUNG 2.10: API-Evaluation nach Grill et al. (2012)

## 2.5.2 “Methods towards API Usability: A Structural Analysis of Usability Problem Categories” (Grill u.a. 2012)

Grill et al. (2012) haben im Rahmen einer Fallstudie eine API-Evaluationsmethode entwickelt, die meinem im nächsten Kapitel vorgestellten Vorgehen sehr ähnelt. Die vorgeschlagene Methode wurde von den Autoren selbst nur einmal angewendet — und im Zuge dessen vermutlich verallgemeinert.

### 2.5.2.1 Methodik

Grill et al. (2012) vorgeschlagenes Verfahren besteht aus fünf Phasen und kombiniert drei verschiedene Methoden, um ein umfassendes Bild zu generieren. Wie in Abbildung 2.10 dargestellt, finden die Phasen zwei, drei und vier zeitnah statt.

#### Phase 1: Planung

In dieser Phase werden die übrigen vier Phasen vorbereitet. Dazu gehört die Bestimmung und Rekrutierung der notwendigen Experten und Entwickler, die Selektion der zu evaluierenden API-Bereiche und eine Zeitplanung.

Im konkreten Anwendungsfall wurde ein Framework<sup>17</sup> evaluiert, das den Entwurf und die Durchführung von Forschungsstudien ermöglicht.

#### Phase 2: Durchführung — HE

In dieser Phase wird eine HE von mehreren geschulten Domänenexperten mit dem Ziel durchgeführt, potentiell problematische API-Bereiche zu finden, damit diese in den weiteren Phasen besondere Beachtung finden.

17 Contextual Interaction Framework, <http://cif.hciunit.org>

Die 16 äußerst knapp formulierten Heuristiken wurden von den 16 API-Design-Guidelines von Zibran (2008) abgeleitet. Zibran stellt in seiner Arbeit die Ergebnisse seiner Literaturforschung im Bereich der API-Usability vor. Beispiele für die Heuristiken sind:

- Benennung: Die Benennung muss selbsterklärend und konsistent sein.
- Dokumentation: Eine Dokumentation und Beispiele müssen bereitgestellt sein.
- Fabrikmethode (engl. *factory pattern*): Die Fabrikmethode darf nur genutzt werden, wenn es unvermeidbar ist.

Untersuchungsgegenstand sind die API selbst und dessen Dokumentation. Im konkreten Anwendungsfall wurden vier Domänenexperten als Evaluatoren eingesetzt.

### Phase 3: Durchführung — Workshops

Workshops sind Veranstaltungen, zu denen API-Anwender eingeladen werden, um die eigentliche Datenerhebung zu ermöglichen. Ein solcher Workshop besteht aus drei Teilen:

1. Einführung in die Anwendungsdomäne
2. Tutorial in Form einer Präsentation mit einfachem Anwendungsbeispiel
3. Praxisteil, bei dem sich die Probanden aus einem vorher zusammengestellten Aufgabenkatalog eine Aufgabe zum selbstständigen Lösen heraussuchen können

Während eines Workshops verfügt jeder Proband über einen Fragebogen, in dem er auftretende Probleme dokumentieren soll. Der Workshop selbst wird videoaufgezeichnet, um Unstimmigkeiten bei der späteren Analyse klären zu können.

Im konkreten Anwendungsfall nahmen acht männliche Akademiker mit mehrheitlich mehrjähriger Berufserfahrung im Alter von Anfang 30 teil.

### Phase 4: Durchführung — Teilstrukturierte Interviews

Diese Phase findet unmittelbar nach einem Workshop oder spätestens am darauf folgenden Tag statt. Dabei wird mit jedem Probanden ein teilstrukturiertes Interview geführt. Das Interview besteht zum einen aus den in Phase 2 verwendeten Heuristiken und zum anderen auf ähnlich geartete Guidelines von Bloch (2006a), von denen ich ein paar im Abschnitt 2.6 nenne.

Außerdem werden während der Interviews individuelle Probleme besprochen. Die Länge eines solchen Interviews ist auf 30 Minuten beschränkt.

### Phase 5: Analyse

In dieser Phase werden die Ergebnisse der HE, die Umfragen und Videoaufzeichnungen der Workshops und die Interviews analysiert.

#### 2.5.2.2 Ergebnisse

Die Verwendung verschiedener Methoden hat sich ausgezahlt, denn von den 169 ermittelten API-Usability-Problemen waren 157 einzigartig. Das bedeutet, sie wurden nur mit einer der verwendeten Methoden gefunden.

Die HE hat dabei die meisten Verstöße gegen die Heuristiken *Dokumentation*, *Komplexität*, *Bennnung* und *Konsistenz / Konventionen* ermittelt. Die Aufzeichnungen des Workshop-Praxisteils haben hingegen die schwerwiegendsten Probleme hervorgebracht. Mit Hilfe der Interviews konnten die Forscher die meisten Benutzererlebnis-Aspekte wie Attraktivität, Emotionen, etc. ermitteln.

Darüber hinaus schlagen Grill et al. (2012) eine mögliche Klassifikation von API-Usability-Problemen vor:

1. Dokumentation
2. Laufzeit (Probleme, die erst bei der Ausführung von Code sichtbar werden)
3. Struktur
4. Benutzererlebnis

### 2.5.2.3 Kritik

Leider sind die Forschungsergebnisse größtenteils quantitativer Natur. Eine intensive qualitative Auseinandersetzung fand nicht statt.

Das vorgeschlagene Verfahren findet in einer Laborumgebung statt, was die Probanden potentiell beeinflusst und die Breite an beobachtbaren API-Usability-Problemen beschränkt. Dieselbe Konsequenz vermute ich beim Einsatz der HE.

Die Verifizierbarkeit der Ergebnisse wurde gezeigt. Die Verallgemeinerbarkeit hingegen nicht; das Verfahren wurde in keiner mir bekannten weiteren Forschung angewendet. Allerdings ist das Bewusstsein für die Wichtigkeit von API-Usability relativ neu (Robillard u. DeLine 2010).

Die verwendeten Heuristiken sind sehr knapp formuliert. Die Autoren geben keine Auskunft darüber, ob den Evaluatoren ausführlichere Beschreibungen zur Verfügung standen. Außerdem bemühen die Autoren das Argument, dass drei bis fünf Experten genügen, um 60% der Usability-Probleme zu entdecken (Farooq et al. 2010). Diese Erkenntnis bezieht sich allerdings auf die Heuristiken von Nielsen (1993). Darüber hinaus wurde in keiner Weise gezeigt, dass die von den Autoren verwendeten Heuristiken vollständig sind.

Die relativ stark quantitative Betrachtung und die Wichtigkeit der Heuristiken schränken die Eignung als Verfahren zur Grundlagenforschung ein.

Der personelle Aufwand für dieses Verfahren ist sehr hoch. Dafür erstreckt es sich aber zumindest über einen viel kürzeren Zeitraum, als dies beispielsweise bei der Concept-Maps-Methode (Gerken

| et al. 2011) der Fall ist.

### 2.5.3 “An Empirical Study of API Usability” (Piccioni u. a. 2013)

Diese Arbeit hatte die Verbesserung der API der in Eiffel geschriebenen Persistenz-Bibliothek *ABEL*<sup>18</sup> zum Ziel. Dabei sollten 25 Probanden mit Hilfe dieser API Probleme lösen. Es handelt sich dabei um die aktuellste Studie, die ich im Bereich der API-Usability-Evaluation finden konnte.

#### 2.5.3.1 Methodik

Methodisch kamen die *kognitiven Dimensionen* von Clarke u. Becker (2003) (siehe Abschnitt 2.4.2.4), *Usability-Tokens* und *Interviews* zum Einsatz. Die fünf verwendeten Usability-Tokens *surprise*, *choice*, *missed*, *incorrect* und *unexpected* werden von den Autoren als orthogonal zu den kognitiven Dimensionen beschrieben, spielen für die Ergebnisse der Arbeit aber nur eine untergeordnete Rolle und werden in dieser Wiedergabe nicht weiter betrachtet.

Die 25 Probanden waren Master- und Promotionsstudenten mit wenigstens einem Jahr Programmiererfahrung in Objektorientierung, durchschnittlich einem Jahr Programmiererfahrung in Eiffel und keinerlei Erfahrung im Umgang mit der ABEL-API.

Das Experiment begann mit einer Einweisung in die wichtigsten Funktionen der IDE. Speziell wurde auf die Möglichkeit hingewiesen, die ABEL-Dokumentation in der IDE zu konsultieren. Außerdem wurde gezeigt, wie die Klassenhierarchie inspiziert werden kann.

Im Anschluss mussten die Probanden Programmieraufgaben mit Hilfe der ABEL-API lösen. Die Arbeitsplätze wurden dabei mit Video aufgezeichnet und die Probanden darum gebeten, ihre Gedanken zu verbalisieren (*think aloud*). Für die Programmieraufgaben gab es kein Zeitlimit. Des Weiteren konnten sich die Probanden an einen der Experimentatoren wenden, wenn sie Probleme hatten.

Unmittelbar nach der Programmieraufgabe wurde mit jedem Probanden ein strukturiertes Interview geführt, das insbesondere zur Erhebung der subjektiven Usability diente. Dazu wurden nachträglich relevante Programmierepisoden mit einem oder mehreren Usability-Tokens klassifiziert. Das Interview selbst basierte auf 12 Fragen, die aus den 12 kognitiven Dimensionen abgeleitet wurden.

#### 2.5.3.2 Ergebnisse

Piccioni et al. schlussfolgerten die folgenden Punkte aus ihren Beobachtungen:

---

<sup>18</sup> <https://svn.eiffel.com/eiffelstudio/branches/eth/eve/Src/library/abel/>

- Die Benennung muss akkurat, knapp und selbsterklärend sein, was in Bezug auf eine Anwendungsdomäne bereits Probleme hervorruft. So erwarteten einige Probanden ein anderes Funktionsspektrum beim Lesen des Klassennamens *REPOSITORY* und hätten den Begriff *DATABASE* für treffender gehalten. Bereits in der Informatik sei eine eindeutige Benennung schwierig. So meinen beispielsweise die Begriffe *method* (Java), *routine* (Eiffel) und *member function* (C++) das selbe Konstrukt.
- Erkenntnisse in Bezug auf den Gebrauch der Fabrikmethode (Ellis et al. 2007) wurden bestätigt. Demnach ist dessen Gebrauch nach Möglichkeit zu vermeiden und unbedingt die Verwendung von Konstruktoren vorzuziehen.
- Typen sind konstanten Strings vorzuziehen.
- Die Dokumentation unbedingt muss akkurat, eindeutig und in sich abgeschlossen sein. Diese Erkenntnis wird mit dem folgenden Zitat untermauert: “bad documentation is a nonstarter”.
- Lösungswege müssen eindeutig sein. Verschiedene Lösungsoptionen sollte es nur geben, wenn sie komplementär sind. Dieser Punkt wird auch von Zibran et al. (2011) bestätigt.

### 2.5.3.3 Kritik

Abgesehen davon, dass der Titel dieser Arbeit in mir höhere Erwartungen geweckt hat, musste ich die folgenden kritischen Beobachtungen machen:

- Die Vorgabe einer IDE gefährdet die Verallgemeinerbarkeit der Erkenntnisse, was in der Arbeit nicht diskutiert wurde.
- Die damit einhergehende Einweisung in die IDE erlaubt keine Aussagen mehr auf die OOBEG<sup>G</sup> (Kahlert 2011). Die Fokussetzung auf bestimmte IDE-Funktionen wirkt sich potentiell auf die Lösungsstrategien der Probanden aus. Beispiel: Die Experimentatoren haben auf die Möglichkeiten hingewiesen, die API-Dokumentation aus der IDE heraus zu konsultieren. Dies lässt eine Beeinflussung vermuten, die der Beobachtung entgegensteht, dass mehr und mehr Google für solche Zwecke verwendet wird (Stylos 2009).
- Die in vier Kategorien gegliederten 12 Interviewfragen wurden aus den 12 kognitiven Dimensionen von Clarke u. Becker (2003) abgeleitet, ohne anzugeben, wie das genau geschah. Mir gelang es trotz meiner intensiven Auseinandersetzung mit dem CDF nicht, jede Frage eindeutig einer kognitiven Dimension zuzuordnen. Bei zwei Fragen jedoch konnte ich dieselbe kognitive Dimension erkennen. Nach dem Taubenschlagprinzip müsste es also mindestens eine nicht erfragte kognitive Dimension gegeben haben. Dies widerspricht dem geäußerten Anspruch der Autoren, neue API-Usability-Aspekte zu finden.



## GRUPPIERTE ERKENNTNISSE BEZÜGLICH DER API-USABILITY-FORSCHUNG

In diesem Abschnitt fasse ich Ergebnisse der API-Usability-Forschung in logische Gruppen zusammen. Ich beschränke mich dabei auf die Arbeiten, die ausreichend verallgemeinerbare Erkenntnisse vorstellen und für meine Forschung von Relevanz sind.

Arbeiten, die ich im Folgenden nicht weiter betrachte, befassen sich mit statischer und dynamischer Typisierung (Mayer et al. 2012), Kapselung (Piccioni et al.; Roberts u. Johnson 1997; Schmidt u. Buschmann; Stylos u. Myers 2008) sowie Sichtbarkeit, Abwärtskompatibilität, Interoperabilität, Nebenläufigkeit und Wartung (Zibran et al. 2011), API-Evolution (Ng et al. 2007) und Speicherverbrauch & Kosten (Henning 2007).

### 2.6.1 Dokumentation

Die meisten (Grill et al. 2012; Zibran et al. 2011) bzw. dramatischsten (Robillard u. DeLine 2010) bzw. fundamentalsten (Bloch 2006a) API-Usability-Probleme röhren von der API-Dokumentation her.

Eine Ursache liegt in der Verwendung verschiedener Bezeichnungen für dasselbe Konzept (Aguiar 2000), was auch als *Vocabulary Problem* bekannt ist (Furnas et al. 1987; Good et al. 1984). Im Kontext von Dokumentationen eignet sich die Unterstützung von Aliassen (Stylos et al. 2009a), also Platzhaltereinträgen, die auf den korrekten Eintrag verweisen.

Weitere Ursachen sind ein inkonsistentes Aussehen, fehlende grafische Darstellungen, sowie schlechte Navigations- und Suchfunktionen. (Jeong et al. 2009)

Eine benutzerfreundliche Dokumentation verfügt über Beispiele, Best-Practise-Beispiele, Erläuterungen der Entwurfsentscheidungen (ebenso Aguiar 2000), einen zielgruppenspezifischen API-Überblick (ebenso Fairbanks et al. 2006; Ko u. Riche 2011; Nykaza et al. 2002; Pugh 2006), Erläuterungen zu komplexen Anwendungsszenarien und eine durchdachte Organisation. (Robillard 2009)

Existenziell wichtige Begriffe müssen erklärt werden — unabhängig von den angenommenen Vorkenntnissen der Anwender. (Jeong et al. 2009; Nykaza et al. 2002)

Eine gute Form der Gruppierung ist die logische Gruppierung. Eine alphabetische Sortierung, wie sie bei JavaDoc zum Einsatz kommt, zwingt den Anwender mental selbst eine Gruppierung vorzunehmen. (Hou et al. 2005)

Der Dokumentationstext selbst muss einen klaren Fokus haben, Direktiven nennen und keine vagen Formulierungen (z.B. “sollte”) verwenden (Monperrus et al. 2011). Außerdem muss er präzise, vollständig und aktuell sein (Parnas 2011; Piccioni et al.; Zibran et al. 2011).

“Die schlechteste Person zum Schreiben der Dokumentation ist der Implementierer und der schlechteste Moment, eine Dokumentation zu schreiben, ist nach der Implementierung.” (Henning 2007)

## 2.6.2 Tutorials

Tutorials stellen eine Spielart des “Fact Findings” (LaToza et al. 2007) dar (siehe Abschnitt 2.2.5) und helfen beim Erlernen einer API (McLellan et al. 1998).

Tutorials stellen eine Möglichkeit dar, den von Robillard u. DeLine (2010) geforderten API-Überblick zu geben (Nykaza et al. 2002). Deren Verlinkung auf der Startseite der Dokumentation ist verpflichtend (Watson 2012).

Tutorials erlauben die Darstellung der von Robillard (2009) geforderten, komplexen Anwendungsszenarien (Hou et al. 2005).

Tutorials erleichtern die Anwendung der *flexiblen Programmverständnisstrategie*.<sup>19</sup> (Shaft u. Vessey 1998)

Ohne die Bereitstellung von Tutorials wird der Einstieg in neue API stark erschwert. (Hou et al. 2005)

## 2.6.3 Beispiele

Beispiele sind notwendiger Bestandteil einer lernförderlichen Dokumentation (Basili et al. 2000; Jeong et al. 2009; McLellan et al. 1998; Nykaza et al. 2002; Tenny 1988). Sie erlauben die Verknüpfung von abstrakten und konkreten API-How-To-Wissen (Shaft u. Vessey 1998).

Jeder nicht-triviale Dokumentationseintrag benötigt ein Beispiel. (Hou et al. 2005)

Es gibt eine Reihe an Werkzeugen (siehe Abschnitt 2.7), welche die Bereitstellung von Beispielen erleichtern.

Je geringfügiger Anwender kopierten Beispielcode an ihren “Anwendungskontext” anpassen, desto größer ist die Wahrscheinlichkeit, dass die Anwender den Code nicht verstanden haben. Wird der Code mittels Debugging angepasst, wird dies als “debugging into existence” bezeichnet. (Rosson u. Carroll 1996)

---

<sup>19</sup> Darunter wird der bedarfsabhängige Wechsel zwischen Top-Down- und Bottom-Up-Vorgehen verstanden.

Für Anwender des Bottom-Up-Vorgehens (siehe Abschnitte 2.2.3 und 2.2.4), sind konkrete Beispiele existenziell. (Rosson u. Carroll 1996)

Bei optimistischen Entwicklern birgt die Bereitstellung von Beispielen die Gefahr der *Verständnisvermeidung* (siehe Abschnitt 2.4.8.3, Rosson u. Carroll 1996; Stylos u. Myers 2008).

## 2.6.4 Benennung

Sinnlose Variablennamen erhöhen die Komplexität des Verständnisprozesses (Shneiderman u. Mayer 1979) und verschlechtern die User Experience<sup>20</sup> (Blinman u. Cockburn 2005).

Sinnvolle Variablen- und Funktionsnamen dienen als *Beacons* (siehe Abschnitt 2.2.4) und beschleunigen den Verständnisprozess. (Gellenbeck u. Cook 1991)

Funktionsnamen sollten die Lösung und nicht die Implementierung beschreiben und bei einer booleschen Rückgabe eine aktive Stimme verwenden. Idealerweise ergibt eine Programmzeile einen Satz.<sup>21</sup> (Cwalina u. Abrams 2008)

Die Beziehungen zwischen Konzepten sollten sich im Namen widerspiegeln. (Cwalina u. Abrams 2008; Dekel u. Herbsleb 2009)

Deskriptive Namen sind abgekürzten Namen vorzuziehen (Zibran et al. 2011), allerdings führen zu lange Namen in der Wahrnehmung zu einem “blinden Fleck” (engl. *blind spot*). Dieser führt wiederum dazu, dass Anwender nur noch Anfang und Ende des Bezeichners wahrnehmen. (Beaton et al. 2008a)

Nicht die abstrakte Klasse, sondern die am häufigsten verwendete Unterklasse sollte den prägnanten und kurzen Namen erhalten<sup>22</sup> (Stylos u. Myers 2008).

Eine Benennung, die nah an der Anwendungsdomäne ist, verbessert das API-Verständnis (Briand et al. 1997; Jeong et al. 2009; Shneiderman u. Mayer 1979; Teasley 1993; Tenny 1988). Dabei muss sie konzeptionell korrekt sein (Zibran et al. 2011).

Umgekehrt wird das Verständnis verschlechtert, wenn die Benennung technisch getrieben ist, also eine Variable beispielsweise `integerArray` heißt (Teasley 1993). Zu einer Verschlechterung kommt es auch, wenn semantisch überladene Begriffe (z.B. “template”<sup>23</sup>) verwendet werden (Gellenbeck u. Cook 1991).

Wissenschaftliche Erkenntnisse zum Thema komplexer Anwendungsdomänen konnte ich nicht finden. Eine den Anwendungsdomänen nahe Benennung ist anspruchsvoll, wenn es spezielle Teildisziplinen

20 In der Studie von Blinman u. Cockburn (2005) mussten die Probanden an Programmen mit einer schlechten Benennung arbeiten. Festgehaltene Aussagen waren “This sucks!”, “I’m getting lost.” und “These are filthy names.”

21 Beispiel: `if(!isValid())` statt `if(!valid())`

22 Negatives Beispiel: Oberklasse = “Message”, Unterklasse: “MimeType”

23 Abhängig von der Anwendungsdomäne könnte es sich um einen fachlichen Begriff handeln. Kommt die Programmiersprache C++ zum Einsatz, würde es zu einer semantischen Überladung kommen, weil nicht eindeutig ist, ob der Begriff fachlich oder technisch gemeint ist.

gibt (z.B. *CRUD*-Operationen im Datenbankbereich), oder das Fach selbst fachübergreifend ist (z.B. Bioinformatik, Wirtschaftsinformatik). Ebenso ergibt sich ein Problem, wenn die Anwenderschaft aus zwei unterschiedlichen Gruppen, z.B. Informatiker und Wirtschaftsexperten, besteht (Jeong et al. 2009). Auf diese Beobachtung werde ich im Abschnitt 4 näher eingehen.

Namen müssen verständlich, konsistent (auch Briand et al. 1997; McLellan et al. 1998; Zibran et al. 2011) und symmetrisch<sup>24</sup> sein. (Bloch 2006a)

Das *Vocabulary Problem* (Furnas et al. 1987; Good et al. 1984) erschwert eine gute Benennung. Das Problem sollte nicht in der API selbst, sondern in dessen Dokumentation gelöst werden (Stylos et al. 2009a). Auf eine technische Lösung gehe ich unter in den Abschnitten 2.7.2 und 4.4.8 ein.

Eine gute Benennung verbessert die API-Anwendung (Bloch 2006b) und verringert die Konsultation der Dokumentation (Blinman u. Cockburn 2005).

Eine schlechte Benennung führt dazu, dass die Verwendung der Dokumentation dramatisch ansteigt. Ist auch die Dokumentation schlecht, sinkt die Performanz. (Blinman u. Cockburn 2005)

Erfahrene Entwickler werden von einer schlechten Benennung weniger beeinflusst als Anfänger (Blinman u. Cockburn 2005; Crosby et al. 2002; Teasley 1993), denn fortgeschrittene Entwickler verfügen über mehr Erfahrung (Teasley 1993) und können ein größeres Spektrum an Beacons erkennen. Dadurch sind sie nicht allein auf die Beacons *naming style* und *documentation* angewiesen (Crosby et al. 2002).

## 2.6.5 Konsistenz

Ein konsistenter Entwurf, der etablierten Konventionen folgt, erhöht die API-Usability. (Zibran et al. 2011)

Immer mehr APIs verwenden das *create-set-call*-Muster als Alternative zu Konstruktoren mit Parametern. Daher sollten Entwickler beim Entwurf einer API das *create-set-call*-Muster vorziehen (Stylos u. Clarke 2007). Konstruktoren, die sich auf die wichtigsten Parameter beschränken, um ein invariantes Objekt zu erzeugen, stellen eine vertretbare Alternative zu parameterlosen Konstruktoren dar Piccioni et al..

Die Benennung innerhalb einer API muss konsistent sein (Bloch 2006a; Briand et al. 1997; McLellan et al. 1998; Zibran et al. 2011).

Je höher der API-interne Konsistenz ist, desto kritischer wirken sich ungewollte Inkonsistenzen aus. Anwender vermuten hinter ihnen häufig bewusste Design-Entscheidungen. (Dekel u. Herbsleb 2009)

---

<sup>24</sup> Diese Eigenschaft ist am besten mit einem Beispiel erklärt: Wenn es die Funktionspaare (`keySet(); hasKey(Object)`) und (`values(); hasValue(Object)`) gibt, muss es auch für die Funktion `entrySet()` das symmetrische Pendant `containsEntry(V, K)` geben. Javas Map-Interface verletzt also diese Forderung.

## 2.6.6 API-Entwurf

Die Anzahl und die Typen von Funktionsparametern haben einen signifikanten Einfluss auf die API-Usability. Zu viele Parameter beeinflussen sie nachteilig. Das Gleiche gilt für Parameter, die schwierig zu konstruieren sind. (Zibran et al. 2011)

Mechanismen, die den kontrollierten Umgang mit Versagenssituationen erlauben, verbessern die Usability. Ebenso verhält es sich, wenn Diagnoseinformationen bereitgestellt werden. (Zibran et al. 2011)

Lösungswege sollten eindeutig sein. Alternative Lösungsoptionen, die das Gleiche tun, führen zu Verwirrung und verschlechtern die API-Usability. Piccioni et al.; Zibran et al. (2011)

### 2.6.6.1 Fabrikmethode

Die Autoren der qualitativen Studie “The Factory Pattern in API Design: A Usability Evaluation” (Ellis et al. 2007) haben mit einem Experiment, bei dem 12 Probanden Programmieraufgaben in Java lösen sollten, die Auswirkungen des Fabrikmusters auf die API-Usability untersucht.

Ellis et al. (2007) fanden heraus, dass die API-Anwender viel mehr Zeit zur Konstruktion eines Objekts mit Hilfe des Fabrikmusters benötigten als mit klassischen Konstruktoren. Sie beobachteten, wie die API-Anwender bei der Abwesenheit von Konstruktoren in der Klasse eher nach Unterklassen mit Konstruktoren suchten. Andere Probanden probierten — meist erfolglos — selbst einen Konstruktor zu schreiben.

Ergebnis der durchgeföhrten Interviews war, dass die Probanden das Fabrikmuster nur dann präferierten, wenn es sich um die Konstruktion von unveränderlichen Objekten ohne Operationen handelt (*opaque objects*).

Die Autoren stellen in ihrer Arbeit außerdem objektive Probleme des Fabrikmusters vor. Dazu gehört die Notwendigkeit einer abstrakten Fabrik, sämtliche konkrete Implementierungen zu kennen. Allein das Finden einer Fabrik ist ebenfalls ein Problem. So ist es in .NET üblich, mit einer Fabrik Objekte verschiedener Typen zu erzeugen, während es bei Java häufig die Suffixregel gibt, bei der die zu einem Typ *Type* gehörige Fabrik *TypeFactory* lautet.

Ellis et al. (2007) schließen aus ihren Ergebnissen, dass sich die Verwendung der Fabrikmethode deutlich negativ auf die API-Usability auswirkt. Alternativen wie der klassische Konstruktor oder das *Class Cluster*<sup>25</sup> sind unbedingt der Fabrikmethode vorzuziehen.

---

<sup>25</sup> <http://www.uow.edu.au/~nabg/ASWEC96/aswec96.html>

### 2.6.6.2 Konstruktoren

Stylos u. Clarke (2007) haben sich mit der Frage befasst, inwiefern sich Konstruktoren, die Parameter verlangen, auf die API-Usability auswirken.

Dazu führten sie ein Experiment durch, bei dem 30 professionelle Entwickler verschiedene Programmieraufgaben in Visual Basic, C# und C++ lösen mussten. Im Anschluss an die Programmieraufgaben wurden Interviews mit den Teilnehmern durchgeführt.

Für die Konstruktion von Objekten gibt es grundsätzlich zwei Ansätze:

**Konstruktoren mit Parametern** Bei dieser Variante wird ein Konstruktor mit einer endlichen Anzahl von Parametern aufgerufen, um ein Objekt zu erzeugen.

**Standard-Konstruktor** Bei dieser Variante wird der parameterlose Standardkonstruktor aufgerufen — gefolgt von endlichen vielen **set**-Aufrufen. Diese Variante wird von den Autoren auch als *create-set-call pattern* bezeichnet.

Die Forscher fanden heraus, dass der Gebrauch des *create-set-call*-Musters zu weniger Fehlern in der Anwendung führte. Dies ist insoweit erstaunlich, als dass sich ein Objekt, das mit einem Standardkonstruktor instanziert wurde, ohne darauf folgende **set**-Aufrufe in einem potentiell inkonsistenten Zustand befinden und zu weiteren Anwendungsschwierigkeiten führen kann.

Die Erkenntnisse wurden nach den Personas von Clarke (Clarke 2007, siehe Abschnitt 2.4.3) aufgeschlüsselt:

**Opportunistischen Entwicklern** kommt das *create-set-call*-Muster am meisten entgegen, denn bei ihnen besteht ein ausgeprägter Fokus auf das Erreichen des Ziels. Die Erklärung ist so einfach wie wichtig. Die analysierten opportunistischen Entwickler wollten zunächst herausfinden, ob sie mit dem korrekten Objekt arbeiten, um dann zu prüfen, welche bereitgestellte Methode die Aufgabe löst (siehe Abbildung 2.11). Für die Exploration der Methoden musst das Objekt jedoch erst konstruiert werden. Bei Konstruktoren mit Parametern konnte dieser Prozess sehr aufwendig sein, da unter Umständen auch diese Parameter über Konstruktoren mit Parametern verfügen, was zu einem rekursiven Konstruktionsprozess führt. Und das “nur”, um die Frage zu klären, ob man es überhaupt mit dem korrekten Objekt zu tun hat.

**Pragmatische Entwickler** empfanden den Gebrauch des *create-set-call*-Musters als weniger störend und geradliniger.

**Systematische Entwickler** favorisierten — zur Überraschung der Forscher — das *create-set-call*-Muster, da es eine höhere Flexibilität bietet. Diese Persona hatte mit keiner der beiden Konstruktionsvarianten Anwendungsschwierigkeiten.

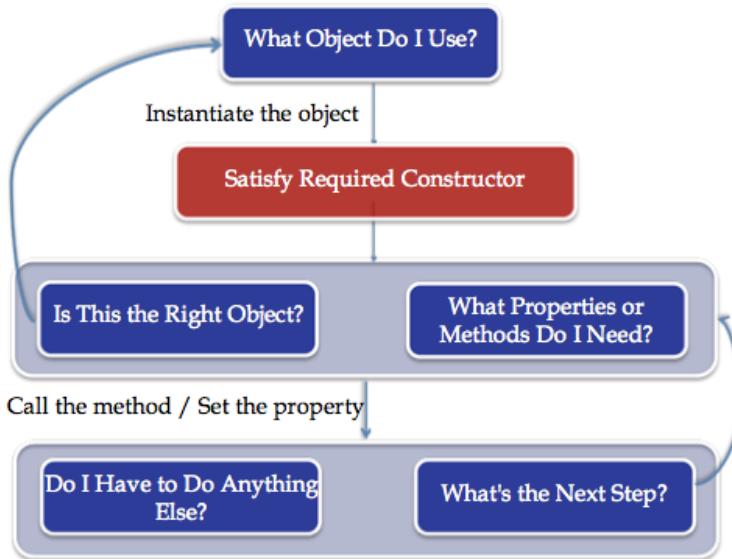


ABBILDUNG 2.11: Wenn Konstruktoren mit Parametern verwendet werden müssen und es dabei zu Problemen kommt (insbesondere IDE-Kompilierfehlern), führt dies zur Unterbrechung des Explorationsprozesses. Der API-Anwender muss den Konstruktor zunächst korrekt verwenden, bevor er fortfahren kann. (Stylos u. Clarke 2007)

Entlang der kognitiven Dimensionen von Clarke (Clarke u. Becker 2003, siehe Abschnitt 2.4.2.4) fassen die Forscher die Ergebnisse der Interviews und ihrer Beobachtungen wie folgt zusammen:

**Work-Step Unit** Die Work-Step Unit<sup>26</sup> wird objektiv durch den Gebrauch von Konstruktoren mit Parametern vergrößert. Das bedeutet, es wird weniger Code für die Erledigung einer konstanten Arbeit benötigt. Umgekehrt benötigt das *create-set-call*-Muster mehr Code, was sich aber laut der Forscher nicht negativ auf die Lesbarkeit des Codes auswirkt.

**Premature Commitment** Diese Dimension wird verstärkt, d.h. der API-Anwender muss zu einem frühen Zeitpunkt Dinge erledigen, die sich nicht auf seinem präferierten Lösungsweg befinden. Das *create-set-call*-Muster erlaubt eine höhere Flexibilität bei der Konstruktion von Objekten.

Ein weiterer Vorteil des *create-set-call*-Musters besteht in der differenzierten Fehlerbehandlung. In diesem Fall kann jede **set**-Methode spezifische Ausnahmen werfen, die bei einem Konstruktor mit Parameter zusammengefasst werden müssten.<sup>27</sup>

Die Forscher resümieren, dass die Implementierung des *create-set-call*-Musters der Implementierung von Konstruktoren mit Parametern vorzuziehen ist.

<sup>26</sup> Inkonsistenter Weise werden in der Arbeit die ursprünglichen kognitiven Dimensionen (Green 1989) mit denen für die Evaluation von APIs (Clarke u. Becker 2003) vermischt, denn die Forscher sprechen sowohl von *Work-Step Unit* als auch von *Diffuseness*. Im Abschnitt 2.4.2.4 habe ich bereits meine Kritik an Clarks kognitiven Dimensionen geschildert.

<sup>27</sup> Dieser Aspekt wird von der Greenschen kognitiven Dimension *Error-Proneness* erfasst, die in Clarks CD fehlt und wahrscheinlich deswegen nicht im Rahmen der kognitiven Dimensionen erläutert wurde.

Piccioni et al. stimmen den Erkenntnissen von Stylos u. Clarke (2007) weitgehend zu, geben dem Argument der Invariante aber ein größeres Gewicht. Sie kommen zu dem Schluss, dass auch Konstruktoren, die sich auf die absolut wichtigsten Parameter beschränken, um einen invarianten Zustand herzustellen, legitim sind.

### 2.6.6.3 Platzierung von Methoden

*Diese Arbeit ist besonders relevant für einige von mir in Kapitel 4 vorgestellte Strategien, die ich bei den SeqAn-Anwendern beobachten konnte.*

Vereinfacht ausgedrückt, hat sich die Arbeit “The implications of method placement on API learnability” von Stylos u. Myers (2008) mit der Frage auseinandergesetzt, wie sich die Entwurfsalternativen `server.send(message)` und `message.send(server)` auf die Erlernbarkeit einer API auswirken.

Dazu wurden zehn Probanden dazu aufgefordert, Programmieraufgaben in Java zu lösen. Zuvor jedoch sollten die Probanden Pseudo-Code schreiben, um auf die Erwartungen der Probanden schließen zu können.

Die Autoren fanden heraus, dass das Zusammenspiel verschiedener Objekte grundsätzlich eine Herausforderung darstellen. Grundsätzlich vermuteten die Probanden die für die Aufgabe notwendige Methode in der so genannten *Anfangsklasse* (engl. *starting class*). Dabei handelt es sich um die Klasse, die nach dem Problemverständnis der Probanden die für die Lösung notwendige Methode bereithält. Die Beziehung zwischen Klasse und Methode wird als *method ownership* bezeichnet.

So selbstverständlich diese Strategie klingen mag, so unerwartet sind die möglichen Probleme, auf welche die API-Anwender treffen:

1. Eine ungünstige Benennung der Anfangsklasse erschwert dessen Auffinden enorm.  
Beispiel: Im Kontext von E-Mails ist es wahrscheinlich, dass die API-Entwickler für die Oberklasse die Bezeichnung “Message”, und für die Unterklassie die Bezeichnung “MimeMessage” gewählt haben. Die Autoren haben jedoch beobachtet, dass die Probanden “Message” als die zu verwendende Klasse vermuteten, obwohl die “MimeMessage”-Unterklassie notwendig war.
2. Findet der API-Anwender keine relevante Funktion in der Anfangsklasse, tritt Verwirrung auf.
3. API-Anwendern fällt es schwer, auf die Notwendigkeit einer zweiten *Helperklasse* zu schließen, da sie davon ausgehen, dass nur eine Klasse notwendig ist.
4. Das Auffinden der benötigten Helperklasse fällt den API-Anwendern schwer.

##### 5. Unsichtbare Abhängigkeiten sind extrem kritisch.

Beispiel: Im Kontext des E-Mailversands in Java wird der Server nicht mit einer entsprechenden `message.setSmtpServer`-Methode gesetzt. Stattdessen muss ein `Options`-Objekt erzeugt werden, das eine solche Methode bereitstellt. Dieses wird dann mittels `message.setOptions(options)` an die Nachricht übergeben.

Es konnte gezeigt werden, dass die korrekte Benennung der Anfangsklasse und die Platzierung der notwendigen Methode in dieser Anfangsklasse die Entwicklung mit der API um ein Vielfaches beschleunigt.

Den Autoren ist bewusst, dass diese beiden Anforderungen nicht immer zu erfüllen sind. Andere wichtige API-Anforderungen wie Performanz oder das *Vocabulary Problem*(Furnas et al. 1987) können der kanonischen Lösung im Weg stehen. Eine verbesserte Dokumentation kann in solchen Fällen helfen. So ist es nicht nur möglich, Aliasse für Klassen, sondern auch Aliasse für Methoden in der Dokumentation zu pflegen. Beispielsweise könnte es einen Aliasseintrag für `Server.send(Message)` geben, der auf `Message.send(Server)` verweist.

Die Auffindbarkeit von Klassen halten die Autoren für existenziell und schlagen vor, für die Darstellung von Klassennamen in der Dokumentation unterschiedliche Schriftgrößen abhängig von ihrer Wichtigkeit zu verwenden.

Für meine Forschung sind die Erkenntnisse von Stylos u. Myers (2008) besonders interessant. Während meiner Forschung zeigte sich, dass praktisch die gesamte Funktionalität von SeqAn durch globale Funktionen bereitgestellt wird. Das Konzept von *method ownership* existiert in SeqAn zwar logisch, aber nicht technisch, was neue Probleme der Auffindbarkeit aufwirft.



## API-WERKZEUGE

Der Literaturüberblick sollte nicht nur einen Eindruck über den Forschungsstand vermitteln, sondern auch die Einsicht erleichtern, dass nicht alle API-Usability-Probleme in dem API<sup>G</sup> selbst gelöst werden können.

Dieser Abschnitt befasst sich mit konkreten Lösungsansätzen, die nicht bei der API selbst, sondern bei den übrigen Einflussfaktoren, wie der Dokumentation, ansetzen. In Anbetracht des Umfangs dieser Literaturübersicht und der Vielzahl existierender Werkzeuge, beschränke ich mich auf eine sehr knappe Beschreibung letzterer.

Ein Vergleich der hier vorgestellten Werkzeuge ist knifflig, da sich auch augenscheinlich ähnliche Werkzeuge im Detail stark unterscheiden. Ich habe mich entschlossen, die Werkzeuge nach ihrer Zielgruppe<sup>28</sup> zu gruppieren und nach dem Aufwand für die API-Entwickler zu sortieren. Abbildung 2.12 ordnet die Werkzeuge entlang dieser beiden Dimensionen ein und zeigt durch den grafischen Marker, welches primäre Ziel von den Werkzeugen verfolgt wird. Diese Einordnung spiegelt meine persönliche Einschätzung wider und ist im besten Falle unscharf<sup>29</sup>.

---

28 Nicht zu verwechseln mit der schlussendlich profitierenden Gruppe, die immer aus API-Anwendern und API-Endanwendern besteht.

29 Insbesondere Werkzeuge, die sich nach meiner Einschätzung für API-Endanwender eignen, wurden von den Autoren in den meisten Fällen nie unter diesem Gesichtspunkt entwickelt. Der Begriff der Zielgruppe trifft also nicht ganz zu.

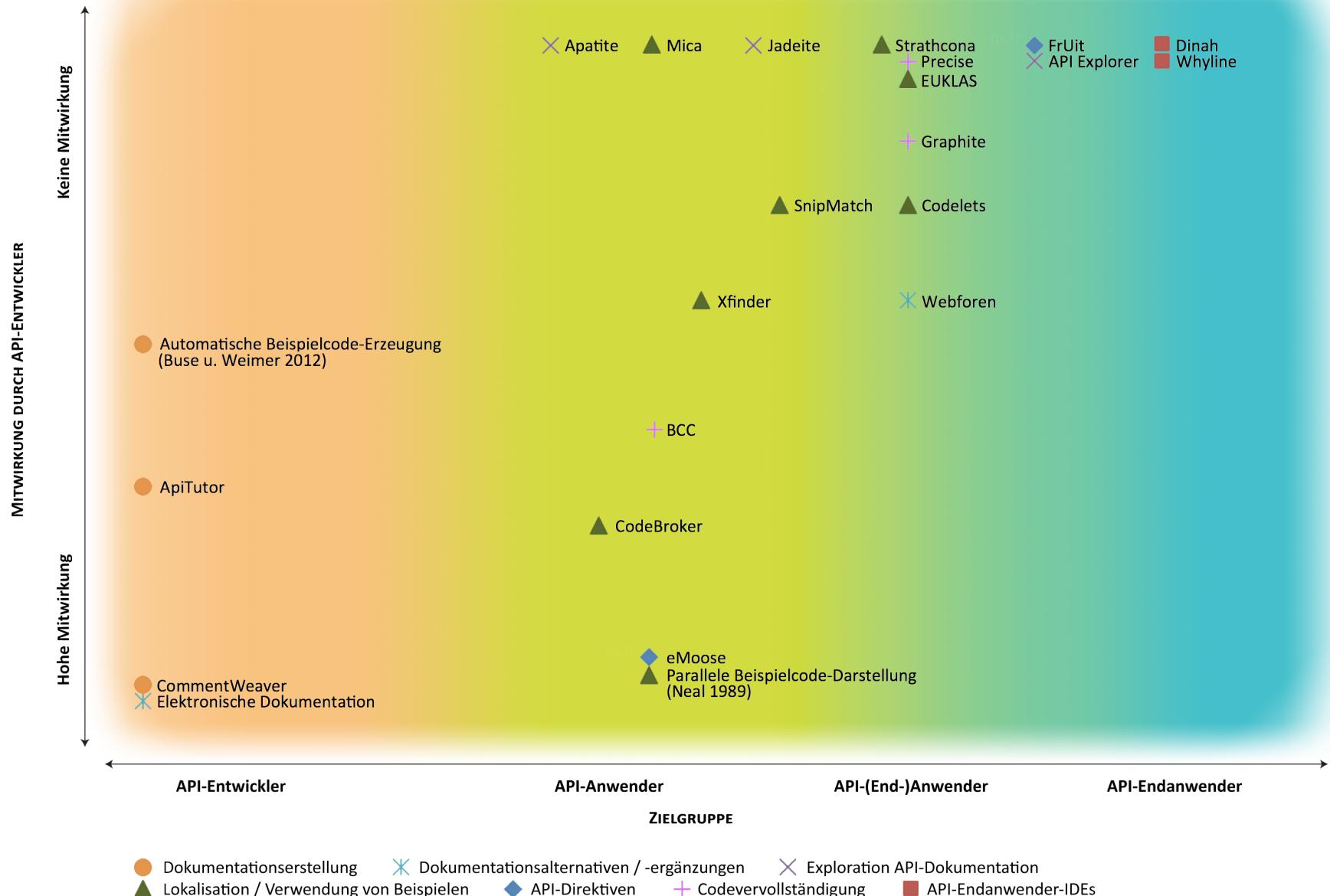


ABBILDUNG 2.12: Grobe Einordnung von API-Werkzeugen entlang der Dimensionen *Zielgruppe* und *Mitwirkung durch API-Entwickler*. Die Ikone stellt dar, welches primäre Ziel vom jeweiligen Werkzeug verfolgt wird.

### 2.7.1 Werkzeuge für API-Entwickler

*CommentWeaver* (Horie u. Chiba 2010) ist eine JavaDoc-Erweiterung, die API-Entwickler bei der Erstellung von API-Dokumentationen unterstützt. Das Werkzeug löst das Problem von verteilt dokumentierten API-Aspekten, indem es verschiedene Mechanismen anbietet, die die modulare Dokumentation einer API erlaubt. Beispielsweise können Dokumentationseinheiten annotiert werden, womit definiert wird, an welcher Stelle dieser Eintrag in der Dokumentation erscheint.

*ApiTutor* (Dahotre et al. 2011) ist ein Werkzeug, das API-Entwickler dabei unterstützen soll, Tutorials für API-Anwender zu entwickeln. In meinen Augen ist der Nutzen allerdings nicht höher als der Aufwand für die API-Entwickler<sup>30</sup>, was durch die ausbleibende Beachtung dieser Arbeit bestätigt wird.

Buse u. Weimer (2012) stellen in ihrer Arbeit einen Algorithmus, der in der Lage ist, mit Hilfe von API-Anwendungscode automatisch menschenlesbare Code-Beispiele für die verschiedenen API-Elemente zu erzeugen (siehe Listung 4). Die Forscher konnten mit einer 154 Probanden umfassenden Studie an Hand von 35 Standard-Java-Klassen zeigen, dass 82% der generierten Beispiele als mindestens gleich gut eingeschätzt wurden, wie von Menschen geschriebene Beispiele.

```

1  FileReader f; //initialized previously
2  BufferedReader br = new BufferedReader(f);
3  while(br.ready()) {
4      String line = br.readLine();
5      //do something with line
6  }
7  br.close();
```

LISTUNG 4: Automatisch generiertes Code-Beispiel Buse u. Weimer (2012)

### 2.7.2 Werkzeuge für API-Anwender

Eine sehr frühe Arbeit auf diesem Gebiet stammt von Neal (1989). Sie prägten den Begriff der Beispielbasierten Programmierung (engl. *example-based programming*), den ich als Teilgebiet der Anwendungswiederverwendung verstehe. Dazu wurde ein existierender Codeeditor um die Funktion erweitert, neben dem bereits vorhandenen Editorbereich einen zweiten neuen Bereich mit Anwendungsbeispielen darzustellen (siehe Abbildung 2.13). Die Beispiele müssen zuvor allerdings manuell erstellt werden.

Das Eclipse-Plugin *eMoose*<sup>31</sup> Dekel (2011) verringert die Fehlerquote von API-Anwendern beim Gebrauch einer API. Dazu werden die im Abschnitt 2.4.7 vorgestellten API-Direktiven im Editor eingeblendet (siehe Abbildung 2.14). In der aktuellen Fassung des seit der Veröffentlichung nicht mehr

30 Der Ablauf besteht in einer integrierten Suchmaschinen-Abfrage und der anschließenden Auswahl von Code-Beispielen, die dann schließlich vom Tutorial-Verfasser didaktisch aufgearbeitet werden.

31 <https://code.google.com/p/emoose-cmu/>

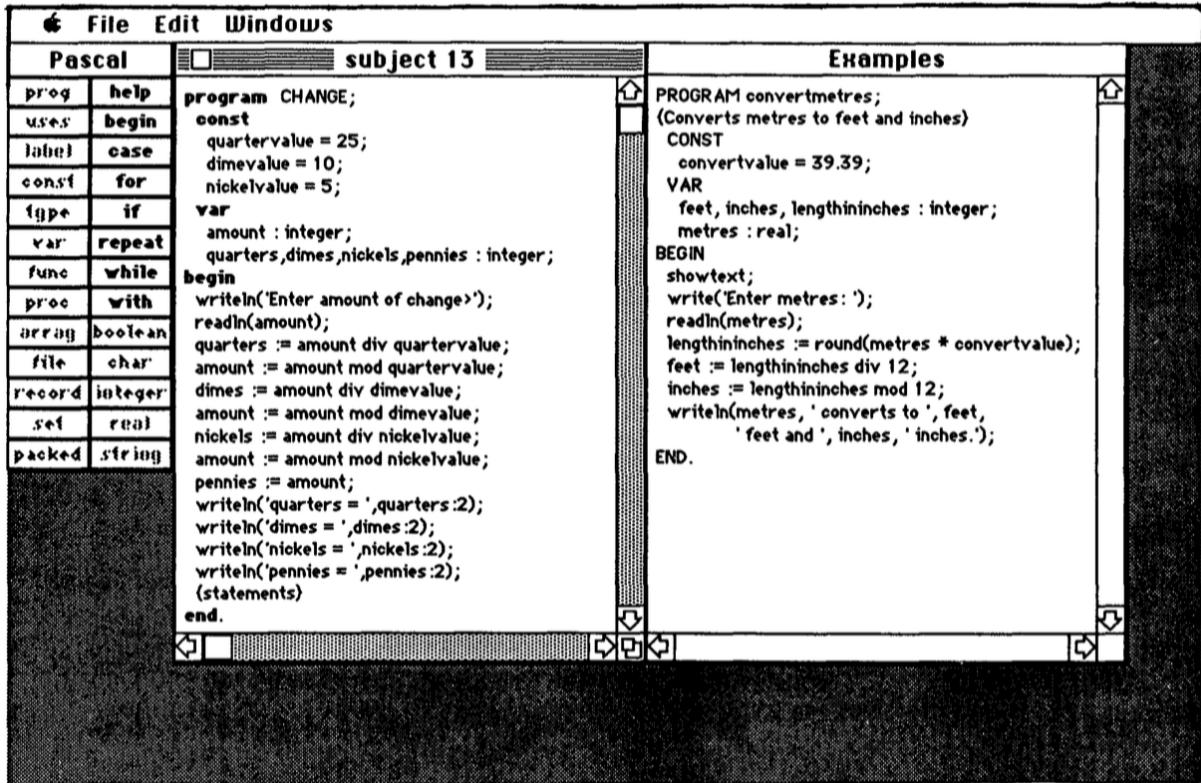


ABBILDUNG 2.13: Codeeditor mit Anwendungsbeispielen (Neal 1989)

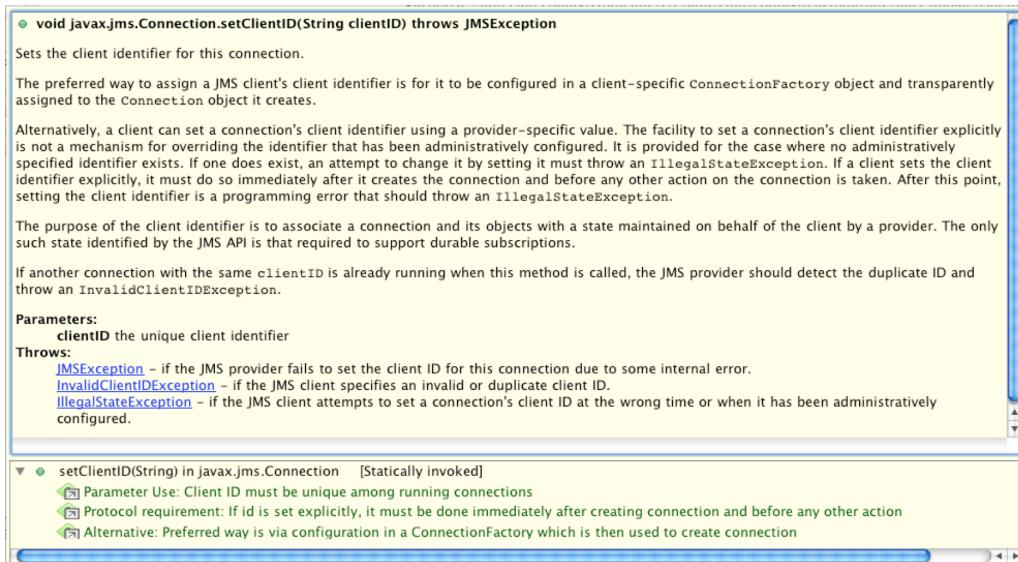


ABBILDUNG 2.14: Beispiel eines Eclipse-JavaDoc-Hilfdialogs für polymorphen Code (Dekel 2011)

aktualisierten Werkzeugs müssen die API-Direkten manuell von den API-Entwicklern beschrieben werden.

*CodeBroker* (Ye u. Fischer 2002) ist ein Emacs<sup>32</sup>-Plugin, das den bereits vom API-Anwender geschriebenen Code mit einem *reuse repository* abgleicht (siehe Abbildung 2.15). Diese Informationen werden

32 <https://www.gnu.org/software/emacs/>

genutzt, um dem API-Anwender wichtige API-Informationen anzuzeigen, von denen das Werkzeug glaubt, dass sie dem API-Anwender unbekannt sind.

The screenshot shows a terminal window titled 'emacs@buddy.cs.colorado.edu' with a menu bar for 'Buffers', 'Files', 'Tools', 'Edit', 'Search', 'Mule', 'JDE', 'Java', and 'Help'. Below the menu is a code editor containing Java code for a 'CardDealer1' class. The code includes comments explaining the class's purpose and a static method 'getRandomNumber'. Below the code editor is a completion dropdown menu with several options starting with 'get'. At the bottom of the window, there is a status bar with the text 'com.objectspace.jgl.util.Randomizer::static int getInt(int lo, int hi)'.

```
emacs@buddy.cs.colorado.edu
Buffers Files Tools Edit Search Mule JDE Java Help
/** This class simulates the process of card dealing. Each card is
 * represented with a number from 0 to 51. And the program produces
 * a list of 52 cards, as it is resulted from a human card dealer */
public class CardDealer1 {
    static int [] cards=new int[52];
    static {
        for (int i=0; i<52; i++) cards[i]=i;
    }
    /** Create a random number between two limits */
    public static int getRandomNumber (int from, int to) {
```

--> CardDealer1.java (JDE)--L10--All--

- 1 0.69 **getInt** Generate a random number using the default generat
- 2 0.64 **getLong** Generate a random number using the default generat
- 3 0.59 **getFloat** Generate a random number using the default generat
- 4 0.59 **getDouble** Generate a random number using the default generat

-1:2\* \*N!-display\* (RequestableComponentInfo)--L1--!cy--

com.objectspace.jgl.util.Randomizer::static int getInt(int lo, int hi)

ABBILDUNG 2.15: CodeBroker (Ye u. Fischer 2002)

*Better Code Completion (BCC)* (Hou u. Pletcher 2010) wurde als Erweiterung der Eclipse-Auto vervollständigung entwickelt und verbessert sie wie folgt:

- BCC unterscheidet zwischen dem `public`-Modifizierer und dem neu eingeführten `API-public`-Modifizierer.
- Methoden werden logisch nach Anwendungskontexten gruppiert.
- Code vervollständigungsvorschläge werden logisch und nach ihrer Popularität sortiert.

Das Werkzeug setzt Zuarbeiten durch die API-Entwickler voraus.

Das Werkzeug *XFinder* (Dagenais u. Ossher 2008) unterstützt API-Anwender ebenfalls durch das automatische Auffinden von Beispielen für konkrete Aufgaben. Im Gegensatz zum weiter oben vorgestellten *Strathcona* benötigt dieses Werkzeug jedoch eine API-Dokumentation und damit die Zuarbeit von API-Entwicklern, um Beispiele aufzufinden. Dafür eignet sich das Werkzeug besser für Aufgaben, bei denen eine Vielzahl von Klassen gemeinsam verwendet werden müssen.

Das Eclipse-Plugin *SnipMatch*<sup>33</sup> (Wightman et al. 2012) hat sich auf den Komfort von Codefragment-Einfügeoperation spezialisiert. So erlaubt es beispielsweise interaktiv, die im Codefragment verwendeten Bezeichner an die des Zielprogramms anzupassen. Als Datengrundlage dienen so genannte Snippets, die entweder von API-Entwicklern bereitgestellt oder von API-Anwendern selbst geschrieben werden müssen.

33 <http://snipmatch.com>

*Mica* (Stylos u. Myers 2006) ist ein prototypische Suchmaschinenerweiterung (siehe Abbildung 2.16), die eine leichtere Auffindbarkeit von API-Klassen und -Methoden unter Zuhilfenahme der Google-Suchmaschine ermöglicht. Mica ordnet dabei die Ergebnisse abhängig von der Verwendungshäufigkeit der API-Klassen und -Methoden in Programmen. Des Weiteren werden Suchtreffer durch Anwendungsbeispiele angereichert. Zwar handelt es sich bei Mica nicht um ein Dokumentationssystem, nutzt letzteres aber indirekt intensiv.



ABBILDUNG 2.16: Mica-Webapplikation (Stylos u. Myers 2006)

*Jadeite<sup>a</sup>* (Stylos et al. 2009a) ist ein an JavaDoc angelehntes Dokumentationssystem (siehe Abbildung 2.17) und steht für “Java API Documentation with Extra Information Tacked-on for Emphasis”. Es basiert auf den Erkenntnissen vorangegangener Arbeiten (insb. Furnas et al. 1987; Stylos u. Clarke 2007; Stylos u. Myers 2008) und verbessert JavaDoc durch die Unterstützung von Aliassen<sup>b</sup>, Konstruktionsbeispielen für Objekte und Hervorhebungen häufig genutzter Klassen und Methoden.

<sup>a</sup> <http://edelstein.pebbles.cs.cmu.edu/jadeite/>

<sup>b</sup> Die Arbeit spricht von *Platzaltern*. Platzhalter kann es für Klassen und Methoden geben. Auf diese Weise erlauben die Autoren dem API-Anwender beispielsweise das Auffinden einer Methode, in einer anderen Anfangsklasse als der von den API-Entwicklern vorgesehenen Klasse.

Das Werkzeug *Apatite* (Eisenberg et al. 2010a, b) ist ein Dokumentationssystem (siehe Abbildung 2.18), das API-Anwendern das Erlernen und Verstehen von komplexen APIs erleichtern soll. Während in klassischen Dokumentationen eine hierarchische Gliederung anzutreffen ist, macht Apatite regen Gebrauch von Assoziationen zwischen den dokumentierten Elementen wie Klassen und Methoden. Den Einstieg in eine Apatite-basierte Dokumentation bildet eine Sucheingabe. Populäre Einträge werden dabei kontextabhängig hervorgehoben. Relationen berechnet das Werkzeug auf der Grundlage von Suchmaschinen und dem Quellcode der Softwarebibliothek.

(a) Shows a tree view of Java classes under the `SSL` package, including `CertPathTrustManagerParameters`, `HandshakeCompletedEvent`, `HttpsURLConnection`, `KeyManagerFactory`, `KeyManagerFactorySpi`, `KeyStoreBuilderParameters`, `SSLContext`, `SSLCertContextSpi`, `SSLEngine`, `SSLEngineResult`, `SSLEngineResultHandshakeStatus`, `SSLEngineResultStatus`, `SSLPermission`, `SSLServerSocket`, `SSLServerSocketFactory`, `SSLSessionBindingEvent`, `SSLSocket`, and `SSLSocketFactory`.

(b) Shows a dialog for "Add a placeholder method" with fields for Method Name, Return Type, Dim Access, Class, Return Type Package, Method Type, and Package, Return Description, Method Description.

(c) Shows a code example for constructing an `SSLSocket` from an `SSLSocketFactory`:

```
SSLSocketFactory factory = ...;
String host = ...;
int port = ...;
SSLSocket socket = (SSLSocket)factory.createSocket(host, port);
```

Based on 38 examples

ABBILDUNG 2.17: Jadeite-Dokumentationssystem (Stylos et al. 2009a)

The screenshot shows four separate search results for different Java classes:

- java.io**: Shows packages like `java.lang`, `java.net`, `java.util`. Under `Classes`, it lists `Object`, `Serializable`, `String`, `File`, `InputStream`, `Serializable`, `Closeable`, `Readable`, `Writable`. Under `Actions`, it lists `add`, `equals`, `hashCode`, `toString`, `remove`, `size`. Under `Properties`, it lists `Class`, `Empty`.
- java.io**: Shows packages like `(No results)`. Under `Classes`, it lists `File`, `InputStream`, `Serializable`, `Closeable`, `Readable`, `Writable`. Under `Actions`, it lists `close`, `list`, `read`, `write`. Under `Properties`, it lists `AbsolutePath`, `Directory`, `Name`, `Path`.
- java.io**: Shows packages like `java.sql`, `java.util.concurrent.locks`, `javax.imageio.stream`. Under `Classes`, it lists `DataInputStream`, `ObjectInputStream`, `RandomAccessFile`. Under `Actions`, it lists `close`, `list`, `read`, `readInt`, `readLine`, `readObject`. Under `Properties`, it lists `list`, `reset`, `skip`.
- java.io**: Shows packages like `javax.imageio.stream`. Under `Classes`, it lists `ObjectInputStream`. Under `Methods`, it lists `close`, `list`, `read`, `write`. Under `Actions`, it lists `read`. Under `Properties`, it lists `(No results)`.

ABBILDUNG 2.18: Apatite-Dokumentationssystem (Eisenberg et al. 2010b)

### 2.7.3 Werkzeuge für API-Anwender und -Endanwender

Wie bereits im Abschnitt 2.4.1 beschrieben, stellen API-Endanwender besondere Anforderungen an eine benutzerfreundliche API. Die meisten hier vorgestellten Werkzeuge richten sich an API-Anwender, obwohl sie sich auch besonders gut für API-Endanwender eignen.

Webforen<sup>a</sup> haben sich als nützliche Möglichkeit herausgestellt, eine andauernde Kommunikation zwischen API-Entwicklern und -Anwendern zu ermöglichen. Sie stellen eine exzellente Ressource für die Themen Debugging, Bugs und Entwurfsfragen dar. (Hou et al. 2005)

Darüber hinaus bergen Webforen das Potential, API-Endanwendern das Wissen und die Hilfe von erfahrenen API-Anwendern zugänglich zu machen. (Ko u. Myers 2005)

<sup>a</sup> auch als Internet- oder Diskussionsforum bezeichnet

Ein zu SnipMatch ähnlichen Ansatz verfolgen Oney u. Brandt (2012). Sie führen den Begriff von *Codelets* ein. Darunter verstehen sie semantisch abgeschlossene Stücke Beispielcode, die, nachdem sie in den Editor eingefügt wurden, auch später noch grafisch bearbeitet werden können. Indem eingefügter Code auch nach Programmierfortschritten vom Editor immer noch als Codelet-Instanzen erkannt werden (siehe Abbildung 2.19), eignen sie sich hervorragend für API-Endanwender. Jedoch müssen die Codelets zuvor von professionellen Entwicklern implementiert worden sein. Eine prototypische IDE-Integration wurde für den *Ajax.org Cloud9 Editor*<sup>34</sup> entwickelt.

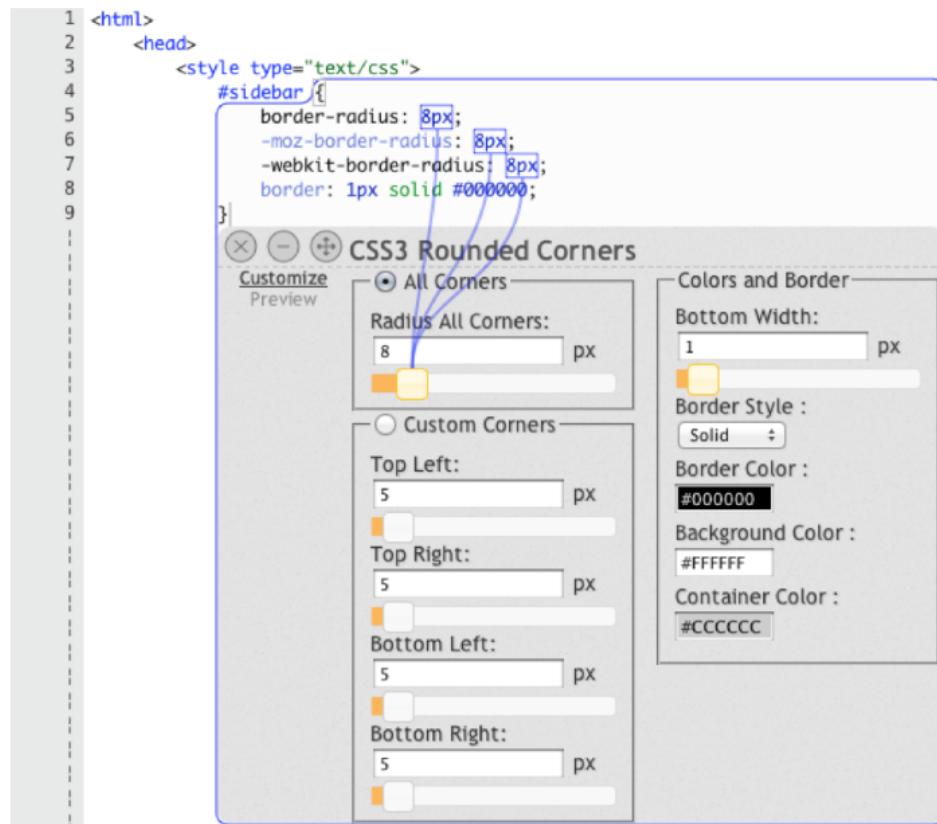


ABBILDUNG 2.19: Codelets-Integration in Webeditor (Oney u. Brandt 2012)

*Graphite* (Omar et al. 2012) ist eine Erweiterung des Eclipse-Auto vervollständigungsassistenten, die den Vervollständigungsdialog mit Hilfe von *Paletten* ersetzt. Schreibt der API-Anwender beispielsweise `return` in einer Methode, die ein Objekt vom Typ `Color` zurückgibt, öffnet sich eine grafische Farbauswahl, die nach der Farbauswahl den korrekten Code für die Instanziierung der gewählten Farbe generiert.

<sup>34</sup> <http://ace.c9.io>

*Precise* (Zhang et al.) ist ein Eclipse-Plugin, dass die Auto vervollständigung erweitert. Im Gegensatz zu den anderen Lösungen vervollständigt dieses Werkzeug einzelne Methoden-Parameter (siehe Abbildung 2.20).

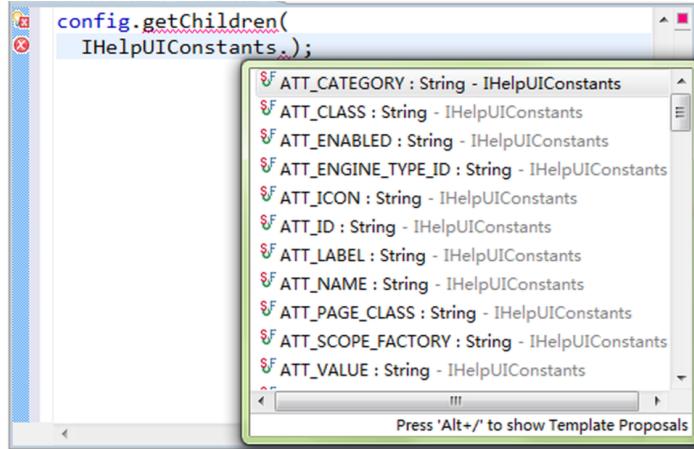


ABBILDUNG 2.20: Precise-Eclipse-Plugin (Zhang et al.)

*EUKLAS* (Eclipse Users' Keystrokes Lessened by Attaching from Samples; Dohnert et al. 2014) ist ein Eclipse-Plugin, das sich voll und ganz der Wiederverwendung von Beispielen verschrieben hat. Es ist behilflich bei typischen Probleme, die beim Einfügen von dritten Beispielcode auftreten. Leider ist die Erweiterung entgegen ihres generischen Namens auf die prototypische Programmiersprache JavaScript beschränkt.

*Strathcona* (Holmes u. Murphy 2005) ist ein Eclipse-Plugin, das es API-Anwendern ermöglicht, nach passenden Beispielcode zu suchen (siehe Abbildung 2.21) und zu übernehmen (siehe Abbildung 2.22). Neuartig an diesem Werkzeug ist, dass es den strukturellen Kontext für die Suche nutzt und die Code-Beispiele selbstständig heuristisch erzeugt.

*FrUIT* (Framework Understanding Tool integrated into Eclipse; Bruch et al. 2006) ist ein Eclipse-Plugin, das *data-mining*-Techniken mit der von Holmes u. Murphy (2005) verwendeten Kontextsensitivität kombiniert (siehe Abbildung 2.23). Es unterstützt den Anwender dabei, typische API-Aufgaben zu lösen, indem es probabilistische Vorschläge für die nächsten Schritte unterbreitet. Dies zwingt den API-Anwender weit weniger, sich intensiv mit der API auseinandersetzen zu müssen, bevor er mit ihr arbeiten kann, was opportunistischen Entwicklern besonders entgegenkommt.

*API Explorer* (Duala-Ekoko u. Robillard 2011) ist ein Eclipse-Plugin, dass sich primär der Auffindbarkeit relevanter API-Elemente wie Klassen und Methoden verschrieben hat. Während Jadeite (Stylos et al. 2009a) "nur" Beispiele für die Erzeugung von Instanzen darstellt, kann API Explorer seine gemachten Vorschläge auch direkt in den Codeeditor einfügen (siehe Abbildung 2.24). Die zweite und wichtigere Funktion besteht darin, für eine Aufgabe relevante Klassen und Methoden zu finden und deren Anwendung ebenfalls direkt in den Codeeditor zu übernehmen (siehe Abbildung 2.25).

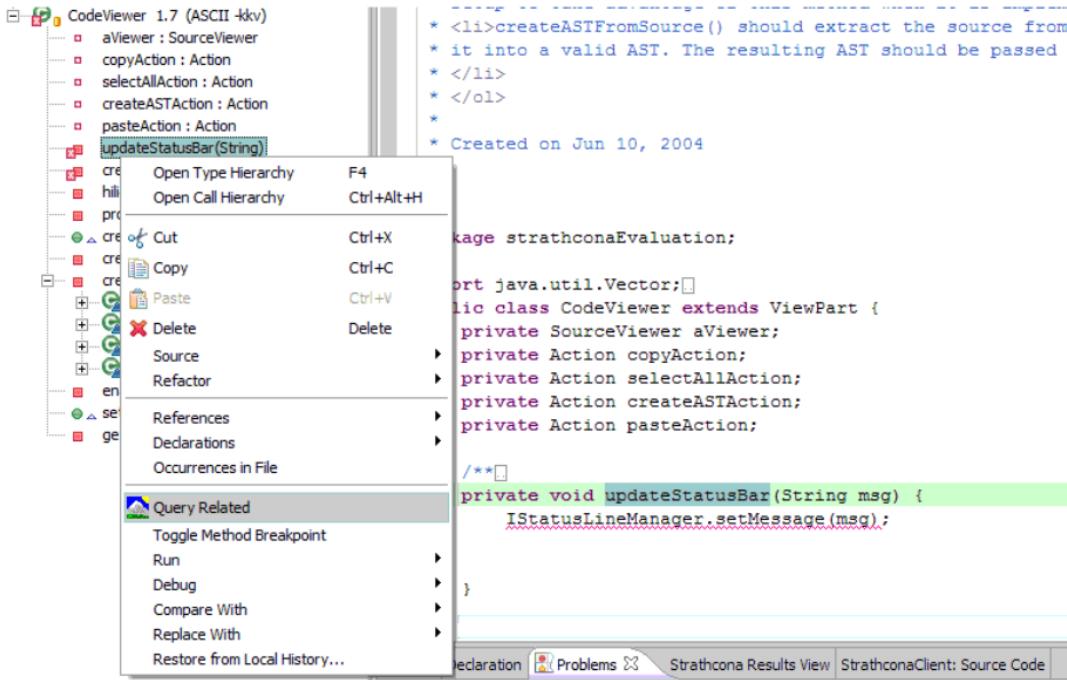


ABBILDUNG 2.21: Strathcona: Formulierung einer Beispiel-Suchanfrage (Holmes u. Murphy 2005)

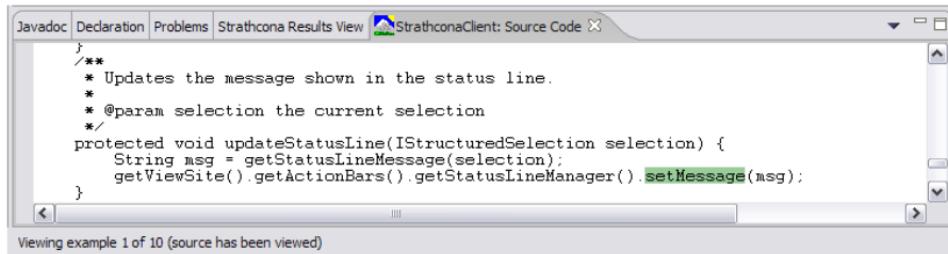


ABBILDUNG 2.22: Strathcona: Einfügen des gewählten Beispielcodes (Holmes u. Murphy 2005)

## 2.7.4 Werkzeuge ausschließlich für API-Endanwender

Zwei Vertreter der Werkzeuge, die sich ausschließlich an API-Endanwender richten, nenne ich hier nur aus Gründen der Vollständigkeit und des Überblicks über die enorme Spannbreite an Hilfestellungen. EUSE ist ein wichtiges Forschungsgebiet, das allerdings nicht im Fokus meiner Arbeit steht.

*Whyline* (Ko u. Myers 2004) ist eine Debugger-Erweiterung für die Endanwender-Programmierer-Entwicklungsumgebung *Alice*<sup>35</sup>, die auf die — ebenfalls in der Arbeit veröffentlichten, empirisch belegten — Anforderungen dieser Anwendergruppe in Bezug auf Debugging eingeht. Die Forscher haben sich mit der Frage beschäftigt, wie ein Werkzeug Endanwender-Programmierer bei Beantwortung von Warum- und Warum-nicht-Fragen behilflich sein kann.

Abschließend sei die Entwicklungsumgebung *Dinah* (Gross et al. 2011) genannt. Für meine Forschung interessant sind die zugrunde liegenden Forschungsergebnisse. Sie zeigen, dass für

35 <http://www.alice.org/index.php>

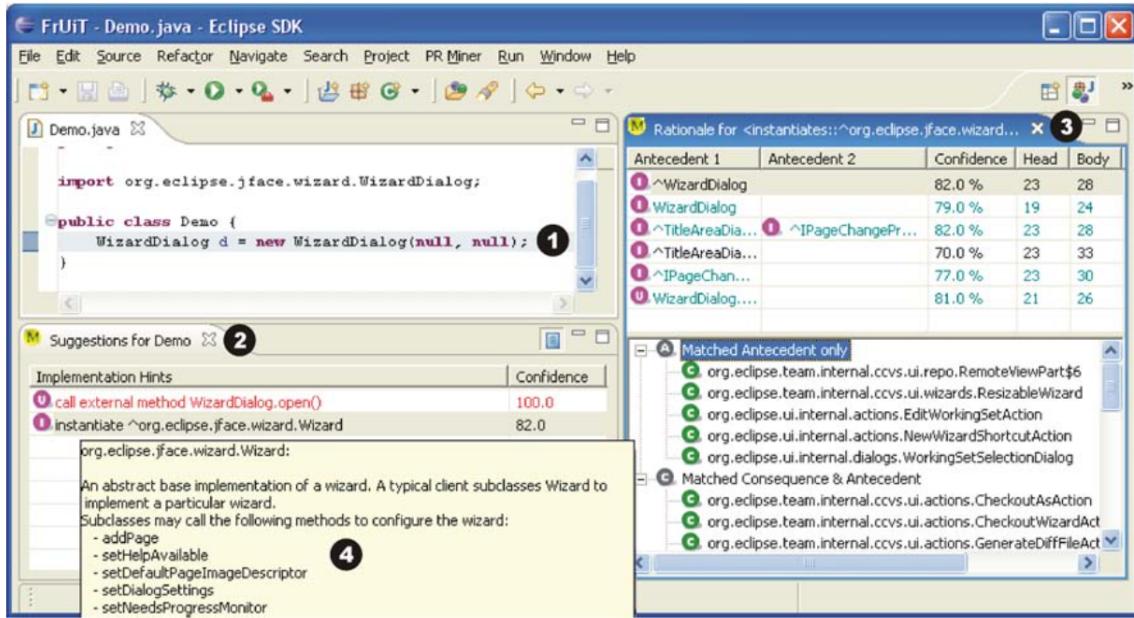


ABBILDUNG 2.23: FrUIT-Eclipse-Plugin (Bruch et al. 2006)

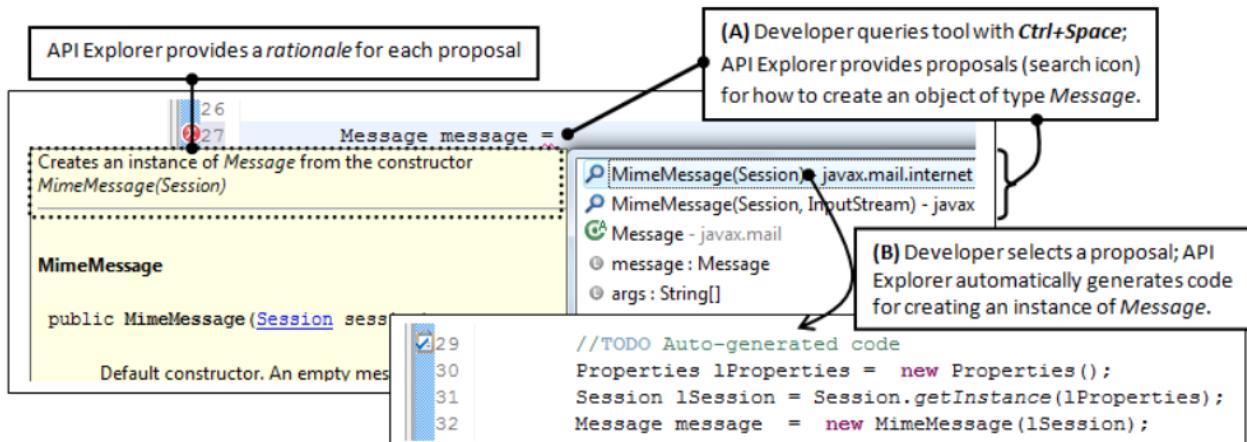


ABBILDUNG 2.24: API-Explorer: Konstruktion einer Instanz (Duala-Ekoko u. Robillard 2011)

Endanwender-Programmierer Beispiel-Code schätzen, aber mit dem *Auswahl-Problem* konfrontiert sind (Gross u. Kelleher 2010). Es besteht darin, dass (1) Endanwender häufig ihre Aufgabe zu abstrakt formulieren, um ein Beispiel zu finden, (2) der für die Lösung notwendige Code zu sehr verteilt ist, oder (3) die wichtigen von den weniger wichtigen Codezeilen des gefundenen Beispiels nicht unterschieden werden können.

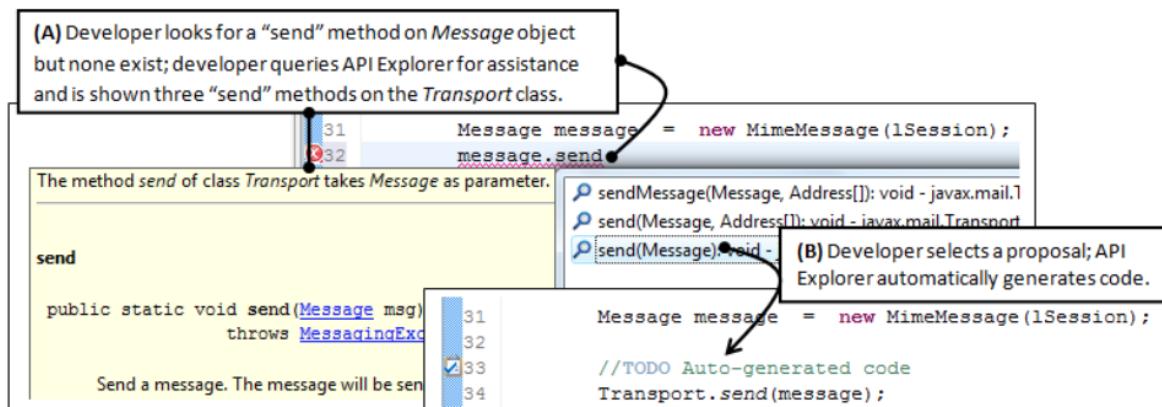


ABBILDUNG 2.25: API-Explorer: Auffinden der zur *Message*-Klasse in Beziehung stehenden *Transport*-Klasse (Duala-Ekoko u. Robillard 2011)

## KAPITEL

### 3

## FORSCHUNG

In diesem Kapitel stelle ich die Erforschung der API-Usability von SeqAn vor. Dazu gebe ich zunächst eine Übersicht zu den Rahmenbedingungen, dem geplanten und tatsächlichen Verlauf meiner Forschung sowie den Schwierigkeiten meines Forschungsvorhabens.

Der übrige Teil dieses Kapitels bespricht detailliert meine Vorgehen und die dabei generierten Zwischenergebnisse. Dabei gehe ich auf diverse Datenerhebungsverfahren (Online-Umfrage, Interviews, Feedback, Gruppendiskussion, Cognitive-Dimensions-Fragebogen und Programmierfortschritte) sowie zwei Forschungsmethoden (Heuristische Evaluation<sup>G</sup> und Methode der Grounded Theory<sup>G</sup>) ein.

Gegliedert ist dieses Kapitel in vier Phasen:

- In Phase 1 bespreche ich die Beseitigung grober Usability-Probleme, die der eigentlichen Forschung vorausging. Dabei kamen die Datenerhebungsverfahren Online-Umfrage, Interviews, Feedback und die Evaluationsmethode HE<sup>G</sup> zum Einsatz.
- In Phase 2 beschäftigte ich mich mit der Planung und Durchführung der für meine Forschung notwendigen Datenerhebung. Erhoben wurden dabei Daten mit Hilfe einer Gruppendiskussion und einem selbst entwickelten Cognitive-Dimensions-Fragebogen.
- Die spezielle Art der erhobenen objektiven Daten machte die Entwicklung eines eigenen Datenanalysewerkzeugs notwendig, welche ich in Phase 3 vorstelle. Bestandteil dieser Phase ist auch eine Gegenüberstellung zum etablierten Datenanalysewerkzeug *ATLAS.ti*.
- Schließlich stelle ich in Phase 4 die Analyse der erhobenen Daten mit Hilfe der GTM<sup>G</sup> und meinem Datenanalysewerkzeugs vor.

In dem nächsten Kapitel präsentiere ich meine eigentlichen Forschungsergebnisse.

## ÜBERSICHT

Bevor ich auf meine Forschung im Detail eingehe, stelle ich in diesem Abschnitt zunächst vor, in welchem Kontext meine Arbeit eingebettet war. Aus diesem Kontext heraus hat sich eine etwas native Planung für das Forschungsvorhaben ergeben. Diese Planung stelle ich in diesem Abschnitt genauso vor, wie deren Abweichungen. Abschließend gehe ich auf Schwierigkeiten meiner Forschung ein, die u.a. die Abweichungen zur ursprünglichen Planung erklären.

### 3.1.1 Rahmenbedingungen

Diese Dissertation entstand im Rahmen meiner Tätigkeit im *BioStore*-Projekt.

Das in der Arbeitsgruppe *Algorithmische Bioinformatik*<sup>1</sup> von Prof. Dr. Knut Reinert angesiedelte BioStore-Projekt wurde durch das *Bundesministerium für Bildung und Forschung (BMBF)*<sup>2</sup> im Rahmen des Programms *VIP — Validierung des Innovationspotenzials wissenschaftlicher Forschung*<sup>3</sup> für einen Zeitraum von drei Jahren bis einschließlich Juli 2014 gefördert.



ABBILDUNG 3.1: Logo des BioStore-Projekts

Das BioStore-Projekt hatte das Ziel zu untersuchen, „wie Computerprogramme zur Anwendung einer neuen Generation von Genomsequenz-Daten effizient entwickelt und vertrieben werden können“ (Reinert et al. 2014) und dies mittels eines App-Stores für bioinformatische standardisierte Werkzeuge — „BioStore“ genannt — zu demonstrieren. Ein ebenfalls bereitgestelltes Workflow-Modul sollte, ähnlich zu Endanwender-Entwicklungsumgebungen, dem Anwender erlauben, bioinformatische Komponenten aus dem BioStore zu beziehen und zu Workflows zu komponieren. (Reinert 2011; Reinert et al. 2014)

Ich besetzte innerhalb dieses Projekts die Stelle des Usability-Spezialisten, der für die Usability-Verbesserung der Softwarebibliothek SeqAn zuständig war. SeqAn spielte im Rahmen von BioStore eine primäre Rolle, da praktisch alle bereitgestellten Bioinformatik-Werkzeuge auf SeqAn basierten (siehe Abschnitt 1.3.3.5). Der zweite Schwerpunkt meiner Projektaktivität bestand in der Mitarbeit am BioStore selbst und am Workflow-Modul. Aus ökonomischen Gründen musste der Umfang des

1 <https://www.mi.fu-berlin.de/en/inf/groups/abi/index.html>

2 <http://www.bmbf.de>

3 <http://www.validierungsfoerderung.de/mediathek/vip-projektfilm-biostore>

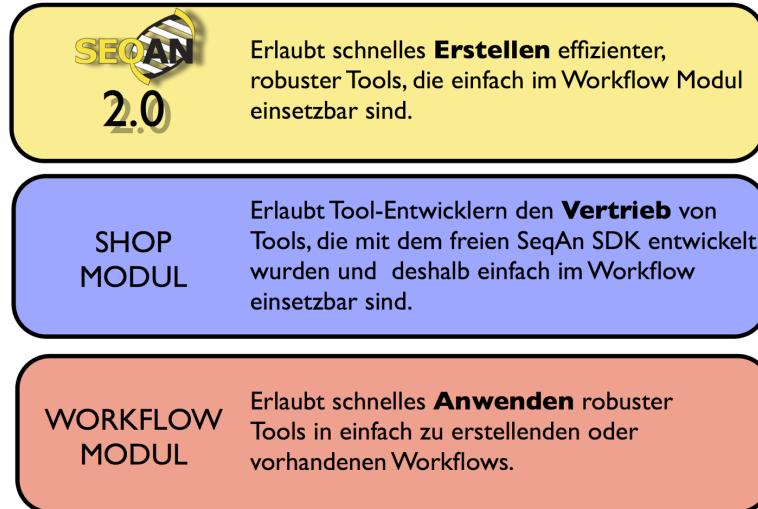


ABBILDUNG 3.2: Komponenten des BioStore-Projekts (Reinert 2011)

Projekts auf Seiten der Arbeitsgruppe gekürzt werden. Anstelle eines eigenen Shops und eines eigenen Workflow-Moduls wurde die Workflow-Engine *KNIME Analytics Platform*<sup>4</sup> (kurz: KNIME) verwendet und erweitert. Die Arbeiten an KNIME fielen somit unter anderem in meinen Zuständigkeitsbereich.

### 3.1.2 Planung

Im Rahmen des dreijährigen BioStore-Projekts sollten drei Workshops stattfinden, die sich an Interessierte aus der Bioinformatik und verwandten Wissenschaften richteten<sup>5</sup>. Ziel der Workshops war es, den Anwendern SeqAn theoretisch wie auch praktisch vorzustellen und den Teilnehmern die Möglichkeit zu geben, ihre auf SeqAn-basierten Projekte vorzustellen. Für den praktischen Teil wurden vornehmlich die online-verfüglichen SeqAn-Tutorials<sup>6</sup> gemeinsam mit den Workshop-Teilnehmer interaktiv bearbeitet. Ein solcher Workshop bot also die ideale Möglichkeit, Forschungsdaten zur Analyse und Verbesserung der API-Usability von SeqAn im Rahmen einer explorativen empirischen Fallstudie zu erheben.

Die ursprüngliche Planung bestand aus den folgenden Phasen:

#### 1. Datenerhebung

Es war vorgesehen, objektive Daten zu erheben. Diese Daten sollten die Entwicklungsschritte der von den SeqAn-Workshop-Teilnehmern entwickelten Programme dokumentieren. Diese Datenquelle wird im Folgenden als *Programmierfortschritte*-Datenquelle bezeichnet und im Abschnitt 3.3.5 genauer vorgestellt.

#### 2. Datenanalyse

Die erhobenen Daten sollten mit Hilfe der GTM analysiert werden. Dieser Schritt sollte die

<sup>4</sup> <http://www.knime.org/knime>

<sup>5</sup> <http://www.seqan-biostore.de/wp/seqan-workshops/>

<sup>6</sup> <http://seqan.readthedocs.org/en/master/Tutorial.html>

vorhandenen API-Usability-Probleme nicht nur aufdecken, sondern auch ein grundlegendes Verständnis verschaffen.

### 3. API-Usability-Verbesserung

Auf der Grundlage der Analyseergebnisse sollten Verbesserungsvorschläge für die API von SeqAn formuliert und umgesetzt werden.

### 4. Validierung

Mit einer weiteren Datenerhebung und -analyse mittels GTM sollte die Effektivität der zuvor umgesetzten Usability-Verbesserungen validiert werden. Die Verbesserungen mussten also spätestens bis zum dritten und letzten Workshop umgesetzt worden sein, um eben diesen Workshop als letzte Möglichkeit der Datenerhebung für die Validierung nutzen zu können.

#### 3.1.3 Tatsächlicher Verlauf

Der zeitliche Verlauf wichen stark von der ursprünglichen Planung ab. Dies ist im Grunde nicht weiter erstaunlich, ist die von mir verwendete GTM doch ein sehr offener — weil explorativer — Forschungsansatz.

Neben den SeqAn-Workshops bot sich eine weitere Datenquelle an, nämlich das jährlich stattfindende Bioinformatik-Praktikum *Projektmanagement im Softwarebereich (PMSB)*<sup>7</sup> des Fachbereichs Mathematik und Informatik der Freien Universität Berlin. Innerhalb dieses Praktikums erhalten die teilnehmenden Studenten eine 4-tägige Einführung in SeqAn, bei der die von den Workshops bekannten Tutorials eingesetzt werden. Der Einführung schließt sich eine einmonatige Projektarbeit an, bei der SeqAn zum Einsatz kommt. Drei dieser Veranstaltungen lagen in dem ursprünglich geplanten Zeitraum für diese Arbeit und boten sich so ebenfalls für die Datenerhebung an.

Die vier größten Schwierigkeiten bei der Einhaltung der ursprünglichen Planung waren:

**Organisatorische Restriktionen** Abgesehen von möglichen Langzeitbeobachtungen<sup>8</sup> gab es nur die SeqAn-Workshops und die hinzugekommenen PMSB-Praktika, deren zeitliche Planung sich an anderen Faktoren orientierte als meiner Forschung. Dieser Umstand führte dazu, dass jede Möglichkeit zur Datenerhebung genutzt und so umfassend wie möglich sein musste. Schließlich erfordert die korrekte Anwendung der GTM für die Klärung von Theorielücken weitere Datenerhebungen (*theoretisches Sampling*) durchzuführen.

**Unübliches Datenformat** Die Art der erhobenen Daten machte die Entwicklung eines Werkzeugs zur Datenvisualisierung notwendig. Jedoch erforderte der Einsatz der GTM eine entsprechende technische Unterstützung. Diese Erkenntnis machte den Ausbau des Datenvisualisierungswerkzeugs zu einem qualitativen Datenanalysewerkzeug notwendig, dessen Entwicklung weit mehr

<sup>7</sup> <http://www.mi.fu-berlin.de/w/ABI/LectureWiki>

<sup>8</sup> Diese Datenquelle war die einzige, bei der ich hohe organisatorische Freiheitsgrad bzgl. der Durchführung hatte. Darauf gehe ich genauer im Abschnitt 3.3.5.2 auf Seite 188 ein.

Zeit in Anspruch nahm, als ich annahm. Dieses Werkzeug mit dem Namen API Usability Analyzer<sup>G</sup> stelle ich im Abschnitt 3.4 vor.

**Unreife Usability** Bereits bei der Vorbereitung und Durchführung der ersten Datenerhebung und spätestens bei der Analyse dieser Daten wurde klar, dass SeqAn unter teils offensichtlichen und in der Literatur längst bekannten API-Usability-Problemen litt. Diese Probleme dominierten die Daten so stark, dass die interessanteren und fundamentaleren Probleme kaum noch zu beobachten waren oder gar nicht erst auftraten. Darum mussten die groben API-Usability-Probleme zunächst zeitraubend beseitigt werden.

Retrospektiv betrachtet ergab sich der folgende, vereinfacht dargestellte Verlauf (vgl. Abbildung 3.3):

1. Erste Datenerhebung (Workshop'11)<sup>9</sup>
2. Implementierung eines Werkzeugs zur Datenvisualisierung / -exploration
3. Zweite Datenerhebung (PMSB'12)
4. Datenanalyse (Workshop'11 und PMSB'12)  
und parallele Weiterentwicklung des Datenvisualisierungswerkzeugs zu einem Datenanalysewerkzeug, das später den Namen *APIUA*<sup>G</sup> erhielt
5. Behebung grober API-Usability-Probleme
6. Dritte Datenerhebung (Workshop'12)
7. Literaturforschung
8. Datenanalyse (Workshop'11 und PMSB'12)  
und bedarfsgetriebene parallele Weiterentwicklung von APIUA
9. Vierte Datenerhebung (PMSB'13)
10. Abschluss der Behebung grober API-Usability-Probleme  
(insbesondere Dokumentation)
11. Fünfte Datenerhebung (Workshop'13)
12. Datenanalyse (Workshop'13-Fragebögen)  
und bedarfsgetriebene parallele Weiterentwicklung von APIUA
13. Datenanalyse (Workshop'12-Gruppendiskussion)
14. Verifikation der Erkenntnisse an Hand der Datenquelle *Programmierfortschritte*
15. Synthese der Forschungsergebnisse
16. Formulierung von API-Usability-Verbesserungsvorschlägen

---

<sup>9</sup> Ich verwende die Notation 'xx für die Darstellung des Jahres, in dem eine Veranstaltung stattfand. '11 steht beispielsweise für das Jahr 2011.

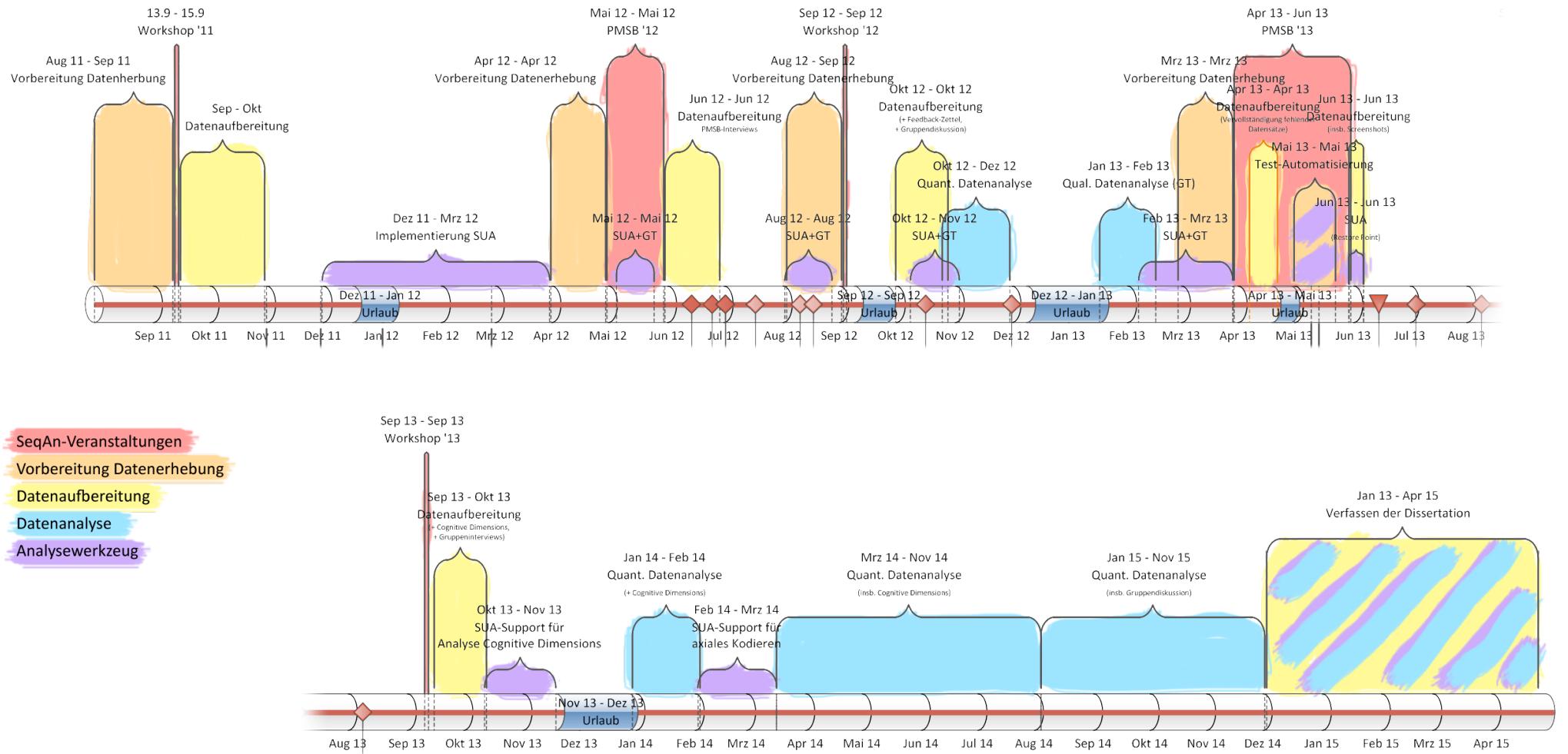


ABBILDUNG 3.3: Zeitlicher Verlauf dieser Arbeit

Datenerhebungen sind rot, Datenerhebungsvorbereitungen orange, Datenerhebungsnachbereitungen gelb, Datenanalysen blau und Arbeiten am Datenanalysewerkzeug violett dargestellt.

Die zwei wichtigsten Auffälligkeiten sind einerseits die zusätzlichen Datenerhebungen während zwei PMSB-Praktika und andererseits der große Zeitbedarf der APIUA-Entwicklung.

Im Verlauf meiner Forschung stellte sich schnell heraus, dass meine Unerfahrenheit im Umgang mit der GTM, gepaart mit der speziellen Gestalt der erhobenen Programmierfortschritte-Daten zu zeitintensiv sein würde, um das Ziel der Verbesserung der API von SeqAn zu erreichen. Ich musste zunächst Erfahrungen im Gebrauch der GTM sammeln und meine *theoretische Sensibilität* schärfen. Aus diesen Gründen entschloss ich mich, zunächst subjektive Datenquellen wie Fragebögen, Interviews und Gruppendiskussionen für die Datenerhebung zu verwenden und zu analysieren. Dieser Methodenmix hatte den Vorteil, dass ich so auch APU-Usability-Probleme aufdecken konnte, die nur in einer der Datenquellen zu finden waren. Außerdem wurde dieses Vorgehen dem Gütekriterium *Triangulierung* (Mayring 2002) gerechter.

Aus diversen, im nächsten Abschnitt beschriebenen Schwierigkeiten musste ich von der Planung Abstand nehmen, eine empirische Validierung der API-Usability-Verbesserungen vorzunehmen.

### 3.1.4 Schwierigkeiten

#### 3.1.4.1 Wissenschaftliche und methodische Schwierigkeiten

Die Einarbeitung in das Gebiet der API-Usability war unerwartet aufwändig, da es keine umfassenden, über “den Tellerrand” schauenden, wissenschaftlichen Literaturstudien gab. Eine solche Literaturstudie musste ich also zunächst erarbeiten.

Diese im vorangegangen Kapitel vorgestellte Literaturstudie — insbesondere Eisenberg et al. (2010b); Ellis et al. (2007); Robillard u. DeLine (2010); Stylos u. Myers (2008); Stylos et al. (2009b) — zeigt, dass sich die Forschung häufig nur auf kleine definierte API-Usability-Aspekte konzentrierte. Entsprechend sinnvoll war die Verwendung der sich für explorative Studien besonders geeigneten GTM.

Die Verwendung der GTM ist für einen Informatiker jedoch nicht einfach und selbst empirische Forschung findet in der Informatik wenig statt. In einer Literaturstudie von 1995-1999 fanden Glass et al. (2002) heraus, dass nur ein Bruchteil der veröffentlichten Studien empirisch zu Stande gekommen ist und gerade einmal 2% der Arbeiten Bezug auf andere Disziplinen genommen haben. Außerdem habe ich im Abschnitt 1.4.1 dargestellt, wie wenig Studien überhaupt die GTM einsetzen. Unter den mit bekannten GTM-Studien befindet sich nicht eine, die tatsächlich eine GT<sup>G</sup> vorgestellt hat. Bei der Hälfte wurde die GTM nur in unzureichender Weise angewandt.

Für die Erforschung der API-Usability-Forschung mittels GTM gibt es keine Studie, an der ich mich methodisch orientieren konnte. Entsprechend eigenständig musste ich also die Datenerhebung und -analyse individuell planen und realisieren.

Forscher, die ein gutes Verständnis von API-Usability und von ihrem fachlichen Forschungsgegenstand haben, sind rar (Hou et al. 2005). Dies trifft auch auf Informatik und Bioinformatik zu (Tisdall 2001). In Bezug auf Bioinformatiker als Anwendergruppe gibt es kaum Forschung (Letondal 2006). Das spezielle

Datenerhebungsverfahren und der ihm geschuldeten Bau eines Datenanalysewerkzeugs gestaltete das Vorhaben nicht einfacher.

Die oben genannten Gründe und Zeitprobleme zwangen mich zu einer kostengünstigen Validierung meiner Arbeit. Diese scheiterte jedoch an der, nicht ohne weiteres ersichtlichen Unreife des Cognitive Dimension Frameworks (siehe Abschnitt 4.5.2.1), welches ich für diesen Zweck verwenden wollte. Darum musste ich mich gegen eine empirische und für eine argumentative Validierung entschließen.

Auf Seiten der Bioinformatik-Arbeitsgruppe gab es andere Vorstellungen von meiner Arbeit im BioStore-Projekt. Mein Promotionsziel und den API-Usability-Forschungsstand übersehend, wurden quantitative, zeitnahe und unmittelbar umsetzbare Forschungsergebnisse erwartet. Ich führe dies auf den quantitativen Forschungsschwerpunkt der Bioinformatik, einer gewissen Unerfahrenheit mit qualitativer Forschung<sup>10</sup> und auf die zeitliche Begrenzung des BioStore-Projekts zurück.

### 3.1.4.2 Organisatorische Schwierigkeiten

**Aufgaben** Häufig sind Forscher, die ein Promotionsziel verfolgen, mit der Tatsache konfrontiert, dass auch andere Aufgaben in ihren Arbeitsbereich fallen. In meinem Fall gehörten zu diesen Aufgaben die Arbeit an der Workflow-Engine KNIME, die im ersten Jahr mehrere Monate umfasste.

Die geforderten quantitativ-konstruktiven Forschungsergebnisse belasteten mich jedoch mehr, denn sie waren nicht nur zeitraubend und für meine Promotion von nur geringer Relevanz, sondern dienten auch dem Projekt nur als kosmetischer Balsam für die Entwickler-Ehre und weniger zur grundlegenden Usability-Verbesserung der SeqAn-API.

Ich habe die These, dass Teile der SeqAn-Entwickler von der Usability der Softwarebibliothek bereits überzeugt waren. Starke Indizien dafür sind *Bauchgefühl-Usability*<sup>11</sup> und *technisches Wegargumentieren*<sup>12</sup>.

Es boten sich sechs Datenerhebungsmöglichkeiten (drei SeqAn-Workshops, drei PMSB-Praktika). Eine einzelne Datenerhebung forderte einen erheblichen Vor- und Nachbereitungsaufwand. Noch weitaus schwerer wog der Aufwand für die Analyse eines solchen Datensatzes. Auch wenn es illusorisch war, alle Datensätze zu analysieren, erntete ich mit meiner Entscheidung, die letzte Datenerhebungsmöglichkeit aus zeitlichen und inhaltlichen Gründen nicht mehr zu nutzen, größtenteils Unverständnis.

Meine Stellung in der Bioinformatik-Arbeitsgruppe (AGABI) war durch meine wissenschaftliche Zugehörigkeit zur Arbeitsgruppe *Software Engineering* (AGSE) diffus, was sich selbst in einer Diskussion zu

<sup>10</sup> Äußerungen der Mitglieder der Bioinformatik-Arbeitsgruppe in persönlichen Gesprächen, wie auch bei der Präsentation von Zwischenergebnissen gaben Aufschluss über die existierende Skepsis gegenüber und Unerfahrenheit mit qualitativer Forschung.

<sup>11</sup> Damit bezeichne ich die *intuitive* (im Gegensatz zur *argumentativen* oder *empirischen*) Entscheidungsfindung in Bezug auf Usability-relevante Entwurfsentscheidungen. Im Abschnitt 4.1.2 beschreibe ich dieses Konzept ausführlicher.

<sup>12</sup> Dieses Konzept wird von Sarodnick u. Brau (2006) anekdotisch beschrieben. Dabei tendieren Softwareentwickler stark dazu, Eindrücken ihrer Anwenderschaft — berechtigt oder nicht — mit technischen Argumenten zu begegnen. Diese Erfahrung machte ich selbst bereits bei der ersten Feedback-Runde während des ersten SeqAn-Workshops. Im Abschnitt 3.3.3 bespreche ich die daraus gezogenen Konsequenzen für meine Datenerhebung.

den Schließrechten der entsprechenden Räumlichkeiten zeigte. Dies führte in Verbindung mit meiner, eben geschilderten Unzufriedenheit zu einer unbewussten Entfernung von der AGABI, was aber auch in einen geringeren Informationsfluss mündete.

**Projekt** Auf Projekt-Ebene gestaltete sich die Erarbeitung dieser Dissertation schwierig, denn das BioStore-Projekt war auf drei Jahre beschränkt — und damit auch die Bezahlung des Personals. Dies führte zu finanziellen Problemen meinerseits. Meine Erfahrung mit anderen Kollegen zeigte mir, dass der erfolgreiche Abschluss der Promotion nur in Vollzeit möglich ist. Diese Monate musste ich mir mit meinen knappen Ersparnissen finanzieren und mich finanziell auf das Nötigste reduzieren.

Die inhaltliche Arbeit erschwerte sich durch den Projektende-bedingten Wegfall wichtiger Projektmitglieder, zu denen insbesondere zwei SeqAn-Hauptentwickler gehörten. Dadurch rutschte die Umsetzung wichtiger API-Usability-Verbesserungen in weite Ferne — beziehungsweise in den Ausblick dieser Arbeit. Diese Entwicklung machte die empirische Validierung dieser Verbesserungsvorschläge unmöglich.

**Technik** Die personelle Dynamik setzte sich in der eigentlichen SeqAn-Softwarebibliothek fort, denn sie wurde natürlich während meiner Arbeit stetig im Sinne des BioStore-Projekts funktionell (z.B. IO-Modul, Parallelisierung) und strukturell (z.B. Beseitigung von Forward-Declarations) weiterentwickelt. Zu diesen Änderungen gehörten allerdings auch eigenmächtige Bauchgefühl-Usability-Verbesserungen wie dem Wechsel der Tutorial-Dokumentationsplattform von *trac*<sup>13</sup>, hin zur auf *Sphinx*<sup>14</sup>-basierten *Read the Docs*-Plattform<sup>15</sup>.

Darüber hinaus hat sich sogar die, SeqAn zu Grunde liegende Programmiersprache C++ weiterentwickelt. Anpassungen an den C++11-Sprachstandard wurden in SeqAn Mitte 2014 angefangen und mittlerweile zum Abschluss gebracht.

Im Folgenden werde ich eine — von den oben genannten Schwierigkeiten weitestgehend bereinigte — Darstellung meiner Forschung vornehmen und sie nicht rein zeitlich, sondern vornehmlich inhaltlich gliedern. Um den Lesefluss und das Verständnis dieser Arbeit nicht unnötig zu erschweren, nehme ich nur an Stellen Bezug auf diese Schwierigkeiten, wenn es der Nachvollziehbarkeit dienlich ist.

---

13 <http://trac.edgewall.org>

14 <http://sphinx-doc.org>

15 <http://seqan.readthedocs.org>

## PHASE 1: BEHEBUNG GROBER API-USABILITY-PROBLEME

Die erste Einarbeitung in SeqAn und spätestens der erste GTM-Analyseversuch der unter 3.3.5 erläuterten *Programmierfortschritte*-Datenquelle haben gezeigt, dass SeqAn teils offenkundige, schwere Usability-Probleme aufweist. Diese galt es in dieser ersten Phase zu beheben. Es bestand begründete die Befürchtung, dass derartige Probleme die Sicht auf die interessanten und tief schürfenden API-Usability-Probleme verdecken. Insbesondere Dokumentationsprobleme, wie veraltete Installationsanleitungen, fehlende, kritische Dokumentationseinträge<sup>16</sup> und unzureichende Anwendungsbeispiele stellen starke Indizien für diese Befürchtung dar. Es ist davon auszugehen, dass derartige Lernhindernisse interessante Probleme gar nicht erst eintreten lassen. Es hat sich beispielsweise in der späteren Analyse gezeigt, dass manche Anwender bei zu großen Anfangsproblemen nicht weiter mit SeqAn arbeiten (siehe Seite 261). Die genauen Ergebnisse dieser ersten Phase stelle ich weiter unten ab Seite 154 vor.

Um diese groben Probleme überhaupt systematisch aufzudecken, analysierte ich die Artefakte, auf die vermutlich ein Anwender trifft, wenn er das erste Mal SeqAn verwendet. Ich habe also die *OOBE<sup>G</sup>* (Fouts 2000) von SeqAn mit Hilfe einer vereinfachten *Heuristischen Evaluation (HE)* analysiert.

Darüber hinaus habe ich während der ersten drei Datenerhebungsmöglichkeiten aus Triangulierungsgründen Befragungen unterschiedlicher Form durchgeführt (*Workshop'11*: Online-Umfrage, *PMSB'12*: Online-Umfrage und Interviews, *Workshop'12*: Feedback-Zettel).

Die Datenanalyse diente, neben der Aufdeckung grober Usability-Probleme, auch der Verbesserung der SeqAn-Workshops selbst. Ein weiterer Nutzen war die Verbesserung meiner, für die Anwendung der GTM notwendigen *theoretischen Sensibilität*.

### 3.2.1 Datenerhebung

#### 3.2.1.1 Out-Of-Box-Experience relevante Artefakte

“Die Out-Of-Box-Experience — kurz OOBE — beschreibt die ersten Erfahrungen, die ein Anwender mit einem Produkt macht. Häufig hat der Anwender dabei — abhängig von der Art des Produkts — Kontakt mit den folgenden Artefakten: Produktverpackung und -handbuch, Installationsprozedur, Konfigurationsassistent, usw. (Fouts 2000)”. (Kahlert 2011)

Für die relevanten OOBE-Artefakte von SeqAn habe ich die Installationsanweisungen (*Getting Started*), die drei Anfänger-Tutorials (*Basics*, *Sequences*, *Iterators*) und die den SeqAn-Entwurf erklärenden Tutorials (insb. *Metafunctions* und *Template Subclassing*) betrachtet.

---

<sup>16</sup> U.a. waren die Konstruktoren der wohl wichtigsten Klasse — nämlich der `String`-Klasse — nicht dokumentiert.

### 3.2.1.2 Online-Umfrage

Die Online-Umfrage hatte folgende Ziele:

**Vorerfahrung** Die Anwenderschaft von SeqAn sollte besser verstanden werden. Interessant waren allgemeine Programmervorkenntnisse und spezielle Programmierkenntnisse in Bezug auf die in SeqAn eingesetzten Techniken (siehe Abschnitt 1.3.1).

**Installation** Probleme bei der Installation eines Produkts sind inhärenter Bestandteil der OOSE und entsprechend auch für SeqAn von hoher Relevanz.

**Tutorials** Auch die Tutorials sind wichtige OOSE-relevante Artefakte. Schließlich handelt es sich bei den Tutorials um *die* Lernressource für SeqAn-Anfänger.

**Bewertung** Persönliche Einschätzungen von SeqAn und dessen Gebrauch waren ebenfalls von Interesse für mich.

Der vollständige Fragebogen befindet sich im Anhang E.2. Er enthält auch Fragen zur Organisation des SeqAn-Workshops, die aber nicht Gegenstand dieser Arbeit sind.

Der Fragebogen wurde unter Berücksichtigung von Mayring (2002) entwickelt und mit Hilfe von *LimeSurvey*<sup>17</sup> implementiert und bereitgestellt.

Die Online-Umfrage wurden im Anschluss an den praktischen Teil des Workshops'11 und nach der SeqAn-Einführung des PMSB'12-Praktikums eingesetzt.

Insgesamt nahmen 18 Teilnehmer an der Umfrage teil.

### 3.2.1.3 Interviews

Neben der Online-Umfrage habe ich während des PMSB'12-Praktikums mit vier Teilnehmern jeweils ein offenes Interview (Mayring 2002) geführt. Dabei interessierten mich Probleme, auf die die Studenten beim Gebrauch von SeqAn stießen.

### 3.2.1.4 Feedback

Bei der Durchführung der Online-Umfrage musste ich feststellen, dass eine nicht kleine Anzahl Workshop-Teilnehmer sich nicht an der Online-Umfrage beteiligte, was auf Teilnehmerseite nachvollziehbar, aber für meine Forschung nachteilig war.

Basierend auf den Erfahrungen der beiden Online-Befragungen habe ich einen Feedback-Zettel erstellt, der um weitere Fragen ergänzt wurde. Die neuen Fragen (u.a. Motivation für Bearbeitung eines Tutorials; Anwendungsform von SeqAn) bezweckten ein besseres Verständnis der SeqAn-Anwendergruppe.

---

<sup>17</sup> <https://www.limesurvey.org>

Des Weiteren habe ich den Befragungsmodus geändert. Statt eine Befragung am Ende des gesamten dreitägigen Workshops durchzuführen, sollte je ein Fragebogen am Anfang der des Workshops, im Anschluss an jedes Tutorial und am Ende des Workshops ausgefüllt werden.

Die drei Feedback-Fragebogen-Varianten (Einstieg, Tutorial, Abschluss) befinden sich vollständig im Anhang E.3.

Insgesamt wurden 210 Feedback-Zettel von max. 58 Entitäten ausgefüllt. Die genaue Anzahl der Beteiligten kann nicht bestimmt werden, da nicht jeder Teilnehmer seine Feedback-Zettel mit einem selbst gewählten Pseudonym markiert hat (Details siehe Abschnitt 3.3.5.1). Dadurch konnten die verschiedenen Feedback-Zettel nicht einer Person zugeordnet werden und mussten separat analysiert werden.

### 3.2.2 Datenanalyse

Für die Analyse der eben beschriebenen Daten habe ich, neben einfachen quantitativen Mitteln (vgl. Abschnitt 3.1.4), die Heuristische Evaluation (HE) eingesetzt.

Die HE wurde erstmalig von Nielsen u. Molich (1990) und praxisbezogener von Nielsen (1993); Nielsen u. Mack (1994) beschrieben. Sie wird dem *Discount-Usability-Engineering* zugesprochen (Sarodnick u. Brau 2006); stellt also ein kostengünstiges Verfahren zur Usability-Evaluation dar. Das Verfahren sieht vor, dass Usability-Experten Artefakte eines Softwaresystems mit der Perspektive des Anwenders gedanklich verarbeiten und dabei Verstöße gegen die vorher definierten Heuristiken aufdecken. Heuristiken haben ihren Namen von der Tatsache, dass ein Verstoß nur auf ein Usability-Problem hindeutet, es jedoch nicht beweist und die Heuristiken auch nicht alle existierenden Probleme aufdecken.

Dem ursprünglichen Verfahren liegen zehn Heuristiken zu Grunde, die ich im Anhang B.1 aufführe.

Beim Versuch, die originären Heuristiken anzuwenden, stellte ich fest, dass sie sich nicht für die Evaluation meiner Artefakte eignen. Das betrifft insbesondere die Heuristiken *Sichtbarmachung des Systemstatus*, *Benutzerkontrolle und -freiheit*, *Wiedererkennen, statt sich erinnern*, *Flexibilität und Effizienz der Benutzung* und *Ästhetik und minimalistisches Design*. Der Grund: Diese Heuristiken haben einen starken Bezug auf grafische Benutzeroberflächen, die es im Falle von SeqAn nicht gibt. Natürlich kann man Elemente von APIs beispielsweise unter ästhetischen Gesichtspunkten betrachten. Dass eine geringe Ästhetik jedoch ein hinreichend sicher auf ein Usability-Problem hindeutet, ist nicht empirisch gezeigt und stellt damit auch keine praktikable Heuristik dar.

An dieser Stelle stand ich also vor der Wahl, speziell für die Evaluation von APIs geeignete Heuristiken herzuleiten oder lediglich die verbliebenen Heuristiken zu verwenden und mich auf meine HE-Anwendungserfahrung (vgl. Kahlert 2011) zu verlassen. Ich habe mich für die zweite Variante aus den folgenden Gründen entschieden:

1. Die theoretische bzw. literaturbasierte Herleitung von Heuristiken entspricht nicht meiner Vorstellung von explorativer Forschung. Ich befürchtete, dass eine zu intensive Auseinandersetzung

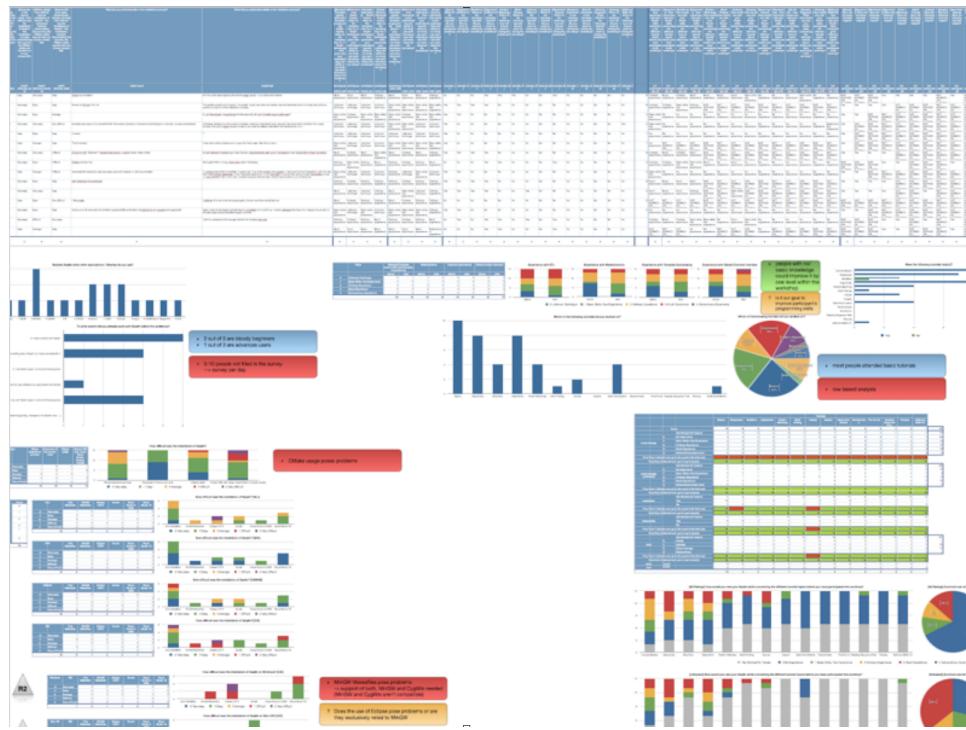


ABBILDUNG 3.4: Erste Analyse in Apple Keynote

mit API-relevanten Heuristiken meine GTM-Forschung — insbesondere beim offenen Kodieren — verzerren und die in den Daten verankerte Theorie nicht mehr korrekt wiedergeben würde.

2. Für einen speziellen Untersuchungsgegenstand individuelle Heuristiken zu entwickeln, ist nicht trivial (Nielsen 1993). Die von Grill et al. (2012) entwickelten API-Heuristiken wurden erst nach meiner Analyse veröffentlicht. Allerdings sehe ich das Zustandekommen dieser Heuristiken kritisch (siehe Abschnitt 2.5.2.3).
3. Die Bereinigung der groben Usability-Probleme stellte “nur” ein notwendiges und vor allem unvorhergesehenes Übel dar. Da ich mich mit der HE gut auskenne und es nicht um die Kategorisierung, sondern um die Beseitigung von Problemen ging, habe ich mir die mühsame Problem-Klassifikation erspart.

### 3.2.3 Ergebnisse

Die Ergebnisse meiner HE sind vollständig im Anhang D beschrieben. Insgesamt habe ich 59 Usability-Probleme gefunden, von den 32 schwer oder katastrophal<sup>18</sup> waren. Im Folgenden beschränke ich mich auf die Darstellung der wichtigsten Ergebnisse.

Die Auswertung habe ich ursprünglich in *Apple Numbers* vorgenommen (siehe Abbildung 3.4) und später auf *Wolfram Mathematica* umgestellt.

<sup>18</sup> Nielsen u. Mack (1994) formulieren die folgenden Fatalitäten:

0 = Kein Usability-Problem, 1 = Kosmetisch, 2 = Gering, 3 = Schwer, 4 = Katastrophal.

### 3.2.3.1 Anwender

Die Anwendergruppen habe ich qualitativ und quantitativ analysiert. Die Anzahl der betrachteten Anwender beträgt 35. Wegen dieser geringen Anzahl, dem Workshop-Format und der subjektiven Einschätzung der Anwender selbst, kann man nicht davon ausgehen, dass die Ergebnisse verallgemeinerbar sind. Dennoch geben sie einen plausiblen Anhaltspunkt für die Charakterisierung der Anwenderschaft.

- In gleichen Teilen waren Studenten und berufstätige Wissenschaftler vertreten.
- Jeweils knapp die Hälfte der Teilnehmer kam aus der Bioinformatik und Informatik. Aus der Biotechnologie, der Molekularbiologie und der Physik kamen jeweils einer der 35 Teilnehmer.
- Häufig wurden “Effizienz”, “Geschwindigkeit” und “Performance” als Motivation zur Auseinandersetzung mit SeqAn formuliert.
- Etwa die Hälfte der Anwender nutzt *Microsoft Windows*, ein Drittel *Linux* und die übrigen *Mac OS X*.
- Die Hälfte der Anwender nutzte eine integrierte Entwicklungsumgebung. Die andere Hälfte nutzte Makefiles.
- Mehr als zwei Drittel der Teilnehmer verfügt über fortgeschrittene Kenntnisse im Gebrauch von objektorientierter Programmierung in den Sprachen C++ oder Java.
- C++-Kenntnisse sind gleich verteilt (jeweils ein Drittel Anfänger, Fortgeschrittene und Experten).
- Erfahrungen in Bezug auf den von SeqAn eingesetzten Techniken waren wenig vorhanden (Beispiel *Metafunktionen*: nur 20% fortgeschrittene Kenntnisse oder besser).
- Einige Anwender gaben an, dass sie von der SeqAn-Installation abgeschreckt waren. Hätten sie nicht am Workshop teilgenommen, hätten sie die Installation abgebrochen und sich nach einer anderen API umgesehen. Diese Beobachtungen machten auch Sunshine et al. (2014).

### 3.2.3.2 Dokumentation

- 2 von 3 Befragten hielten die Dokumentation für mangelhaft beschrieben (vgl. Abbildung 3.5).
- Den Befragten fehlten vor allem eine Einführung, Beispiele und Verlinkungen zu den Tutorials.
- Eine Begründung und Motivation für die Entscheidung, Templatemetaprogrammierung einzusetzen, fehlt.
- Es fehlen Best-Practise-Beschreibungen (z.B. `++var` oder `var++?`).
- Es fehlt die Beschreibung von Benennungsregeln/-konventionen für Variablen, Funktionen, etc.

Metaprogramming

## Infix

Infix sequence type.

`Infix<T>::Type`

## Include Headers

---

`seqan/sequence.h`

## Parameters

---

T	A sequence type. Types: String
---	-----------------------------------

## Return Values

---

Type	The infix type.
------	-----------------

## Remarks

---

Note that an infix of a `Segment` object is an `InfixSegment` object having the same host type.

## See Also

---

[Prefix](#), [Suffix](#), [InfixSegment](#)

ABBILDUNG 3.5: Mangelhafter Dokumentationseintrag zur Metafunktion `Infix`,  
Stand: 02.07.2012

### 3.2.3.3 Tutorials

- Die Tutorials haben einen zu hohen Anspruch.
- Die Tutorials sind von geringer didaktischer Qualität.
- Die Tutorials verfügen über keinerlei explizite Meta-Angaben wie Zielgruppe, Schwierigkeitsgrad, etc.
- Die Qualität der Tutorials variiert eklatant.

### 3.2.3.4 Installation

- Die Installation wurde mehrheitlich als schwierig beschrieben.
- Hauptgrund 1: Mängel in der Installationsanleitung (inkonsistent, fehlerhaft)
- Hauptgrund 2: SeqAn hat sich als Framework und nicht als Softwarebibliothek entpuppt. Dieser Punkt wird ausführlicher im Abschnitt 4.1.2 erläutert.

### 3.2.3.5 Softwarebibliothek

- Den Anwendern war nicht klar, weshalb SeqAn die “komplizierte” Templatemetaprogrammierung verwendet. Die Mehrheit der Teilnehmer erwartete, dass SeqAn auf Objektorientierung basiert. Ein Teilnehmer bezeichnete SeqAn sogar als “Vergewaltigung der OO-Programmierung”<sup>8</sup>. Dieser Punkt hat sich als äußerst relevant herausgestellt und wird u.a. im Abschnitt 4.1.2 besprochen.
- Die `length`-Funktion gibt alle Eingaben, für die sie nicht explizit entwickelt wurde, 1 zurück.
- Es wurden mehrere funktionale Schwächen gefunden. Beispiel: Die Konkatenierung eines SeqAn-Strings und eines C++-Strings war nicht mit dem +-Operator möglich.
- Funktionen sind nur schwer aufzufinden, denn sie gehören keiner Klasse an.
- Mehrfach wurde das Fehlen der `substring`-Funktion zur Erzeugung von Teilstrings, bemängelt.

All die hier genannten Punkte sind von großer Relevanz, wie die spätere GTM-Analyse gezeigt hat.

### 3.2.3.6 Zusammenfassung der Ergebnisse

Mit meiner Analyse der OOB-E-Artefakte und den, in zwei SeqAn-Workshops und einem PMSB-Praktikum erhobenen Daten konnte ich zu einer besseren Charakterisierung der SeqAn-Anwendergruppe beitragen und teilweise schwerwiegende Probleme in der Dokumentation, den Tutorials, sowie bei der Einrichtung und bei dem Gebrauch von SeqAn aufdecken.

Als größtes Usability-Problem hat sich die Erlernbarkeit von SeqAn herausgestellt. Es gibt Indizien dafür, dass diese Anwender von der Verwendung von SeqAn abschrecken. Das Usability-Problem hat zwei Ursachen:

1. Die Dokumentation ist von vergleichsweise geringer Qualität, die bei den verschiedenen Dokumentationseinträgen variiert.
2. SeqAn setzt auf das Programmierparadigma Templatemetaprogrammierung. Dies stellt Anwender mit C++-Vorerfahrung vor das Problem, dass dieser Ansatz sich stark von der C++ Standard (Template) Library unterscheidet. Anwender mit Java-Vorerfahrung vermissen die Ähnlichkeit zur objektorientierten Softwareentwicklung.

Das Problem der Erlernbarkeit spielt eine wichtige Rolle im späteren Teil dieser Arbeit. Die Behebung vieler anderer Probleme wird im folgenden Abschnitt vorgestellt.

### 3.2.4 Verbesserungen

Jedes Usability-Problem isoliert zu lösen, ist aus praktischer Sicht weder möglich noch effizient. Viel mehr Sinn macht es, Maßnahmen zu formulieren, die eine ganze Gruppe von Usability-Problemen beheben.

Für die Behebung der gefundenen Usability-Probleme, habe ich Maßnahmen definiert und eine Maßnahmen-Probleme-Zuordnung vorgenommen. Den Aufwand einer jeden Maßnahme habe ich in Stunden geschätzt. Der Nutzen einer Maßnahme wiederum, ergibt sich aus der Anzahl und der *Fatalität* der durch sie behobenen Usability-Probleme.

Zu den wichtigsten formulierten Maßnahmen gehören:

- Vollständige Überarbeitung und Vereinheitlichung der Installationsanleitungen
- Definition von Anforderungen für Tutorials
- Bereitstellung einer Vorlage für Tutorials
- Anpassung sämtlicher Tutorials an Anforderungen und Vorlage
- Erstellung eines neuen Anfänger-Tutorials
- Einführung von Aliassen in der Dokumentation (Auffindbarkeit von Einträgen durch Synonyme)

Die vollständige Zuordnung, samt der Kosten-/Nutzen-Schätzungen, befindet sich im Anhang D. Es wurden vornehmlich die Arbeitspakete umgesetzt, die nicht die Softwarebibliothek im engeren Sinne selbst betreffen. Für die Verbesserung der Softwarebibliothek selbst sollte die, im Abschnitt 3.5 beschriebene Phase 4 dienen.

Im Folgenden stelle ich die tatsächlichen Änderungen vor. Meine organisatorischen und inhaltlichen Verbesserungen der SeqAn-Workshops sind nicht Gegenstand dieser Arbeit und werden daher nicht vorgestellt.

#### 3.2.4.1 Prozessverbesserungen

**Commit-Nachrichten** Die SeqAn-API-Entwickler haben ihre Commit-Nachrichten für ihr Versionsverwaltungssystem nach Belieben formuliert. Diese erschwerte die Nachvollziehbarkeit von Code-Änderungen für die Entwicklerkollegen. Für mich war dieses Format ebenso wichtig, da ich für meine Analyse darauf angewiesen war, wichtige Änderungen des SeqAn-Codes zu erfahren (vgl. Abschnitt 3.1.4). Allein für den Versionssprung von SeqAn 1.3 auf 1.4 gab es rund 3.500 Commits.

## Commit Messages Format

[trac.seqan.de/wiki/HowTo/WriteCommitMessages](http://trac.seqan.de/wiki/HowTo/WriteCommitMessages)

[TAGS] Short message description  
 <empty line>  
 Longer descriptive text.



### API

Changes in the API, breaking backward compatibility.  
 E.g. renaming of function names, function parameter order changes.

### INTERNAL

Changes in the implementation, no influence on public API. E.g. renaming of variable names, simplification of code.

### FEATURE

A user-visible feature. E.g. extension of an interface, measurable performance improvement. *Use both tags FEATURE and API if the change also breaks the API.*

### FIX

Bug removal. Use [FIX-#7,#35] when fixing bugs from tracker, where #7 and #35 are ticket numbers.

### TEST

Addition/changes of tests and test data.

### NOP

Only whitespace changes (spaces, line breaks).

### DOC

Changes in the user documentation (DDDoc, README, etc.)

### COMMENT

Changes in the source code documentation for developers, includes // TODO(\${NAME}):

### CLI (for users)

Change to the command line interface of a program. E.g. to the command line arguments or user-targeted messages.

### LOG (for developers)

Change of output for developers or advanced users, meant for debugging or detailed introspection.

ABBILDUNG 3.6: Standardisiertes Format für Commit-Nachrichten

Zu Vereinheitlichung haben mein Kollege Manuel Holtgrewe und ich ein Format entwickelt, das ausführlich online<sup>19</sup> beschrieben wird. Darüber hinaus habe ich einen positiv angenommenen “Spickzettel” (siehe Abbildung 3.6) erarbeitet, auf den API-Entwickler zurückgreifen können.

**Umstellung Subversion auf Git** In der Bioinformatik-Arbeitsgruppe arbeiten viele Mitarbeiter an einer einzigen SeqAn-Anwendung im Rahmen ihrer Tätigkeit. Dies führte durch den Gebrauch des zentralen Versionsverwaltungssystems *Subversion* dazu, dass SeqAn nach dem Commit von Codeänderungen nicht mehr kompilierte.

Um SeqAn-Entwicklern eine größere Freiheit und Sicherheit zu geben, indem sie Änderungen lokal revisionieren können, wurde eine Umstellung auf *Git* vorgenommen. Im Zuge dieser Umstellung wurden die Kollegen geschult und ein SeqAn-Git-Workflow<sup>20</sup> basierend auf dem prominenten Gitflow<sup>21</sup> formuliert und etabliert.

**Code-Reviews** Die Entwicklung von SeqAn-Code unterlag keiner praktischen Qualitätssicherung. Aus diesem Grund habe ich angeregt, Codeinspektionen (*Code-Reviews*) durchzuführen.

Diese Qualitätssicherungsmaßnahme wurde schließlich in den Commit-Prozess als Prä-Commit-Review integriert. Durch die spürbare Verlangsamung des Entwicklungsprozesses, haben die SeqAn-Entwickler, im Zuge der Git-Umstellung, auf das Post-Commit-Review gewechselt. Das heißt, Commits werden nun erst inspiziert, wenn sie bereits in den Code integriert wurden. Für externe Entwickler gilt weiterhin ein Prä-Commit-Verfahren, das durch die Arbeitsweise von Git leicht zu implementieren war.

19 <http://seqan.readthedocs.org/en/master/HowTo/WriteCommitMessages.html>

20 <http://seqan.readthedocs.org/en/master/Infrastructure/SeqAnWorkflow.html>

21 <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>

### 3.2.4.2 Argument-Parser

Für die sich vornehmlich an Wissenschaftler richtende Workflow-Engine KNIME wurde basierend auf einer Vorarbeit der Universität Tübingen<sup>22</sup> ein generischer Wrapper für Konsolenanwendungen zur Bereitstellung von SeqAn-Awendungen in KNIME entwickelt<sup>23</sup>.

Dieser, unter dem Namen *GenericWorkflowNodes* firmierende Wrapper nutzt ein XML-basiertes Datenformat zur Beschreibung seiner Ein- und Ausgabeschnittstelle. Konsolenanwendungen verfügen selbst bereits auch schon über eine beschriebene Schnittstelle, auch wenn diese nur — möglicherweise über das ganze Programm verstreut — programmatisch beschrieben ist.

Um eine redundante Schnittstellenbeschreibung — nämlich einmal im Programm und einmal in der KNIME-Knotenbeschreibung — zu vermeiden, wurde eine neue Komponente zum Parsen von Argumenten geschrieben. Diese kann sowohl die Hilfebeschreibung einer Konsolenanwendung mittels des Parameters `-help` bzw. `-h`, eine Manpage, eine HTML-Dokumentation, als auch eine KNIME-Knotenbeschreibungsdatei ausgeben.

Dieser Argument-Parser wurde in der ersten Version von mir und später von meinem Kollegen Stephan Aiche weiterentwickelt und perfektioniert. Der Parser unterstützt ein breites Spektrum an Funktionen, das von zahlreichen Parametertypen (*flag options*, *value options*, *positional options*, ...) bis hin zu Restriktionen (Typisierung, Wertebereiche, Datentypen, ...) reicht.

Sämtliche SeqAn-Anwendungen wurden an den neuen Argument-Parser angepasst. Dessen Verwendung stellt eine API-Usability-Verbesserung sowohl für API-Entwickler als auch für API-Anwender dar.

---

22 [http://www.knime.org/files/ugm2013\\_talks/knime\\_ugm\\_2013\\_knutreinert\\_final.pdf](http://www.knime.org/files/ugm2013_talks/knime_ugm_2013_knutreinert_final.pdf)

23 <https://github.com/genericworkflownodes>

API-Entwickler profitieren von einer einfachen, mächtigen Komponente zur Beschreibung der Schnittstelle (siehe Listung 5) und dem Beziehen von Parametern.

```
1 seqan::ArgumentParser parser("Argument-Parser Demo");
2
3 setDescription(parser, "Basic functionality of the Argument-Parser");
4 setVersion(parser, "0.1");
5 setDate(parser, "2013-09-18");
6
7 addUsageLine(parser, "[OPTIONS] \"TEXT\"");
8 addDescription(parser, "This program allows simple string repetition by i times.");
9
10 addSection(parser, "Demo Options");
11 addOption(parser, seqan::ArgParseOption("i", "times", "Number of repetitions.",
12   seqan::ArgParseArgument::INTEGER, "INT"));
13
14 setDefaultValue(parser, "times", 1);
15 addTextSection(parser, "Examples");
16 addListItem(parser, "modify_string -i 5 \"text\"", "Print \"text\" 5 times");
```

LISTUNG 5: Argument-Parser: Beispielhafte Schnittstellenbeschreibung in C++

API-Anwendern stehen nun einheitlich und ausführlich beschriebene Hilfeseiten zur Verfügung (siehe Listung 6). Fehlerhafte Eingabedaten werden besser erkannt und ausführlicher zurückgemeldet (siehe Listung 7).

**Argument-Parser Demo - Basic functionality of the Argument-Parser**

---

#### SYNOPSIS

```
demo [OPTIONS] "TEXT"
```

#### DESCRIPTION

This program allows simple string repetition by i times.

-h, --help

Displays this help message.

--version

Display version information

#### Demo Options:

-i, --times INT

Number of repetitions. In range [1..100]. Default: 1.

-O, --output-file OUT

Path to the output file Valid filetype is: txt.

#### EXAMPLES

```
modify_string -i 5 "text"
```

Print "text" 5 times

```
modify_string "text" --output-file out.txt
```

Print "text" once in file out.txt

#### VERSION

Argument-Parser Demo version: 0.1

Last update 2012-08-30

#### LISTUNG 6: Argument-Parser: Beispielhafte Hilfeseite

```
demo$ ./demo -i no_int
```

Argument-Parser Demo: the given value 'no\_int' cannot be casted to integer

#### LISTUNG 7: Argument-Parser: Beispielhafte Fehlerausgabe

### 3.2.4.3 Installationsanleitungen

Die Installationsanleitungen waren fehlerhaft, uneinheitlich, unstrukturiert und verfügten über zu wenig Beispiele, um den Anleitungen folgen zu können.

Ich habe eine inhaltliche und grafische Vorlage für die plattformabhängigen Installationsanleitungen (*Linux — Makefiles, Linux — Eclipse<sup>G</sup>, Mac OS X — Makefiles, Mac OS X — Xcode und Windows — Visual Studio*) erstellt.

Die von mir formulierte Gliederung lautet:

1. Prerequisites — Was bereits installiert sein muss + Verlinkungen
2. Install — Die eigentliche SeqAn-Installation
3. A First Build — Überprüfung, ob Installation korrekt verlief
4. Hello World! — Skelett für erste SeqAn-Anwendung
5. Further Steps — Verlinkungen auf Dokumentation und Tutorials

Sämtliche Installationsanleitungen wurden korrigiert, vereinheitlicht und zentral verlinkt<sup>24</sup>. Abbildung 3.7 zeigt einen Ausschnitt aus der Installationsanleitung für Windows.

### 3.2.4.4 Dokumentation

Die Dokumentation wurde über mehrere Iterationen hinweg verbessert. In die Verbesserung flossen neben den hier besprochenen Ergebnissen, insbesondere die im Abschnitt 4.1 vorgestellten Ergebnisse der GTM-Analyse. Daher wird die überarbeitete Dokumentation ausführlich im Abschnitt 4.4.8 vorgestellt.

Die Abbildungen 3.8a und 3.8b geben einen Eindruck über den Grad der Verbesserung.

### 3.2.4.5 Tutorials

Basierend auf einem etablierten (u.a. Aggarwal 2009; Reardon 2008) Lernphasenmodell (Gagné 1985), den Analyseergebnissen der OUBE-Ressourcen, der Workshops '11 und '12, der PMSB'12-Veranstaltung und einem intensiven Gespräch mit meinen Kollegen am 05.07.2012 — also knapp ein Jahr nach Beginn der Arbeit — habe ich die Struktur der Tutorials überarbeitet und Qualitätskriterien formuliert.

Das Ergebnis habe ich in dem Dokument “Writing Tutorials”<sup>25</sup> zusammengefasst. Es richtet sich an die Autoren von SeqAn-Tutorials und umfasst alle notwendigen Informationen zum Verfassen eines qualitativen Tutorials.

24 <http://seqan.readthedocs.org/en/master/Tutorial/GettingStarted.html>

25 <http://seqan.readthedocs.org/en/master/HowTo/WriteTutorials.html>

The screenshot shows a web browser displaying the SeqAn documentation. The URL is [seqan.readthedocs.org/en/master/Tutorial/GettingStarted/WindowsVisualStudio.html#tutorial-getting-started-windows-visual-studio](http://seqan.readthedocs.org/en/master/Tutorial/GettingStarted/WindowsVisualStudio.html#tutorial-getting-started-windows-visual-studio). The page title is "Getting Started With SeqAn On Windows Using Visual Studio". The left sidebar has a "Tutorial" section with a "Getting Started" heading and a long list of topics. The main content area contains the "Getting Started With SeqAn On Windows Using Visual Studio" article, which includes a "Prerequisites" section, a "Warning" box, and an "Important" box. A "ToC" sidebar on the right lists the contents of the article.

ABBILDUNG 3.7: Ausschnitt aus der verbesserten SeqAn-Installationsanleitung für Windows

Das eben genannte Dokument “Writing Tutorials” verfügt über die folgende Struktur:

## 1. Konventionen

Dieser Abschnitt beschreibt, innerhalb eines Tutorials, einzuhaltende Vorgaben.

### 1. Wiki-Konventionen

Anforderungen an Wiki-Syntax

### 2. Namens-Konventionen

Anforderungen an Groß- und Kleinschreibung, Benennung des Tutorials, etc.

Looking for a different entry? Unhide the navigation bar and start your search.

## Spec

# Infix Segment

An infix of a sequence.

**Extends** Segment  
 All Segment  
 Extended  
 All Impl'd AssignableConcept, ContainerConcept,  
 DestruktibleConcept, ForwardContainerConcept,  
 RandomAccessContainerConcept,  
 ReversibleContainerConcept, SegmentableConcept  
 Defined <seqan/sequence.h>  
 in  
 Signature template <typename THost>  
 class Segment<THost, InfixSegment>;

**TEMPLATE PARAMETERS**

**THost** The underlying sequence type.

**MEMBER FUNCTION OVERVIEW**

**Member Functions Inherited From AssignableConcept**  
`operator=`

**Member Functions Inherited From RandomAccessContainerConcept**  
`operator[]`

**INTERFACE FUNCTION OVERVIEW**

**Interface Functions Inherited From Segment**  
`beginPosition endPosition`

**Interface Functions Inherited From AssignableConcept**  
`assign set move`

**Interface Functions Inherited From ContainerConcept**  
`getObjectId moveValue append appendValue shrinkToFit begin  
 end length empty swap writeValue write directionIterator`

**Interface Functions Inherited From RandomAccessContainerConcept**  
`value assignValue getValue`

**Interface Functions Inherited From SegmentableConcept**  
`prefix infixWithLength infix suffix`

**INTERFACE METAFUNCTION OVERVIEW**

**Interface Metafunctions Inherited From ContainerConcept**  
`DefaultGetIteratorSpec DefaultIteratorSpec Difference  
 DirectionIterator GetValue Iterator Position Reference Size  
 Value`

**Interface Metafunctions Inherited From SegmentableConcept**  
`Infix Prefix Suffix`

**DETAILED DESCRIPTION**

Also known as: substring

**SEE ALSO**

[Spec PrefixSegment](#) [Spec SuffixSegment](#)

TOP HOME

- > TEMPLATE PARAMETERS
- > MEMBER FUNCTION OVERVIEW
- > INTERFACE FUNCTION OVERVIEW
- > INTERFACE METAFUNCTION OVERVIEW
- > DETAILED DESCRIPTION
- > SEE ALSO

(B) Neu, Stand: 10.04.2015

ABBILDUNG 3.8: Vergleich Dokumentationseintrag Infix

### 3. Design- und Layout-Konventionen

Anforderung an die Hervorhebung von Schlüsselkonzepten, Verweisen, Programmeingaben und -ausgaben, etc.

## 2. Struktur

Die Struktur sieht vor, dass ein Tutorial aus einem Meta-Informationen-Block, einer Einführung, inhaltlichen Abschnitten und weiterführenden Links besteht.

### 1. Meta-Informationen

Angabe von Lernziel, Schwierigkeitsgrad, voraussichtliche Bearbeitungsdauer und Voraussetzungen mit entsprechenden Links

### 2. Einführung

Tutorial-Inhalt, Relevanz/Wichtigkeit, praktische Anwendungsgebiete und erworbenes Wissen nach Tutorial-Bearbeitung

### 3. Abschnitte

Jeder inhaltliche Abschnitt bespricht einen logischen Lernschritt bestehend aus einem schriftlichen Ausführungen, Beispielen und einer Übungsaufgabe.

#### (a) Einführung

Abschnittsinhalt, Nennung wichtiger Konzepte, Lernziel

#### (b) Erklärung

Eigentlicher Inhalt

#### (c) Beispiele

Beispiele, die die Erklärung veranschaulichen und die Umsetzung in SeqAn demonstrieren

#### (d) Übungsaufgaben

Wiederholung bzw. Anwendung des erworbenen Wissens

## 3. Didaktik

Dieser Abschnitt soll die Autoren von Tutorials für eine benutzerfreundliche, anwender-zentrische Schreibweise sensibilisieren und motivieren.

### 1. Übungsaufgaben-Typ

Erklärung der verschiedenen Typen von Übungsaufgaben

### 2. Zeitbedarf

Schätzung des Zeitbedarfs von Tutorials und Aufgaben

### 3. Sprache

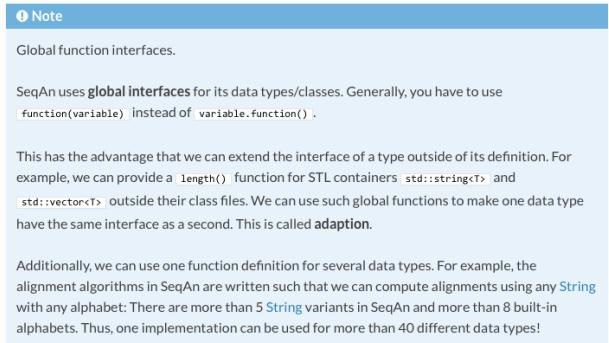
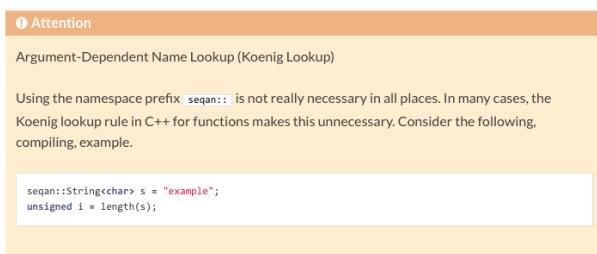
Gebrauch einer einfachen verständlichen Sprache

### 4. Mentales Modell

Einnahme der Leserperspektive

## 4. Integration

Dieser Abschnitt erläutert technische Fragestellungen wie Orte, an denen ein Tutorial verlinkt werden muss.



(A) Box für wichtige Inhalte

(B) Box für weiterführende Inhalte

ABBILDUNG 3.9: Boxen zur Auszeichnung von Inhaltstypen

## 5. Vorlage

Dieser Abschnitt stellt eine Vorlage bereit, die über erklärende und zu ersetzende Platzhalter verfügt. So soll sichergestellt werden, dass die Grundstruktur aller neuen Tutorials konsistent bleibt und den qualitativen Anforderungen genügt.

Während manche Tutorials sehr kurz waren, waren andere außerordentlich ausschweifend formuliert. Bei der Bearbeitung Letzterer, gingen wichtige Informationen und Schlüsselkonzepte in der Masse unter. Aus diesem Grund wurden Elemente eingeführt, die explizit den Typ einer Information grafisch mittels einer farbigen Box hervorheben. Es existieren Boxen für wichtige (orange) und optionale (blau) Inhalte (siehe Abbildungen 3.9a und 3.9b). Darüber hinaus existieren Boxen für Beispiele (grau), Code-Ausschnitte (ebenfalls grau), Programmausgaben (schwarz) und Übungsaufgaben (beige). Außerdem wurden Schlüsselkonzepte fett hervorgehoben und sämtliche genannten Entitäten, wie Funktionen, mit den entsprechenden Dokumentationseinträgen verlinkt.

Die Übungsaufgaben waren vor der Verbesserung von unterschiedlicher Qualität und verfügten über didaktische Schwächen. Diese Probleme sollten u.a. durch die enge Kopplung an einen inhaltlichen Abschnitt gelöst werden. Wichtiger jedoch ist, dass Übungsaufgaben nun von einer der drei folgenden Typen sein müssen: *Review*, *Application*, *Transfer*.

*Review*-Übungen beschränken sich auf die reine Wiederholung von Inhalten und sollen lediglich die Verwendung von SeqAn üben. *Application*-Übungen umfassen Aufgaben, bei denen Variationen vorgenommen werden müssen (z.B. Belegung eines optionalen Parameters). *Transfer*-Übungen sind die anspruchsvollsten und sollen verschiedene Anwendungsmöglichkeiten für das erworbene Wissen aufzeigen.

Eine Forderung an die Übungen ist, dass sie nur in aufsteigender Schwierigkeitsreihenfolge auftauchen dürfen und sich aufeinander beziehen müssen. Beispielsweise darf keine *Review*-Übung auf eine *Application*-Übung folgen. **Gäbe es tatsächlich für solch einen Fall einen Bedarf, ist das Tutorial wahrscheinlich zu umfassend und muss aufgetrennt werden.**

The screenshot shows a section titled "Assignment 5" with a yellow background. It contains several sections: "Type" (Application), "Objective" (Provide a generic print function which is used when the input type is not `String<int>`), "Hint" (Keep your current implementation and add a second function. Don't forget to make both template functions. Include `seqan/score.h` as well.), and "Solution" (with a "more..." link). Below this is a section titled "Tags in SeqAn" with a yellow background. It includes a code snippet: `globalAlignment(align, seqan::MyersHirschberg())`. A note explains that this is a default constructor call within a function call. It also mentions that `MyersHirschberg()` is a tag for determining the specialization of `globalAlignment`. There is a "more..." link for tags. At the bottom, there are download links for "Plain Text" and "Download in other formats". The footer includes the Trac logo, the version "0.12.2", the author "Powered by Edgewall Software", and the URL "www.seqan.de".

ABBILDUNG 3.10: Erste Verbesserung des neuen Anfänger-Tutorials

Es wurden mehrere neue Tutorials durch das Auftrennen existierender Tutorials verfasst. Besonders erwähnenswert ist dabei das neue Anfänger-Tutorial “A First Example”<sup>26</sup>, das eine besonders geringe Einstiegshürde aufweist. Dazu führt es in einfachste Konzepte von SeqAn ein und wird der Beobachtung gerecht, dass viele Anwender einen OOP-Hintergrund haben.

Die Tutorials wurden sukzessive über eine längere Diskussionsphase hinweg gemeinsam von mir und den jeweiligen Autoren verbessert (siehe Abbildungen 3.10, 3.11 und 3.12). Zum Workshop’12 waren die wichtigsten und zum Workshop’13 alle Tutorials überarbeitet.

### 3.2.5 Validierung

In diesem Abschnitt bespreche ich, wie ich die eben vorgestellten Verbesserungen validiert habe.

#### 3.2.5.1 Prozessverbesserungen

**Commit-Nachrichten** Die Vorstellung des neuen Commit-Nachrichtenformats wurde während einer Vorstellung vom SeqAn-Team positiv angenommen. Fast alle darauffolgenden Commit-Nachrichten hielten sich an das Format und vielen deutlicher genauer aus. Als Beispiel habe ich zwei Commit-Nachrichten ausgewählt und jeweils eine inhaltlich passende spätere Commit-Nachricht ermittelt.

#### Beispiel 1

26 <http://seqan.readthedocs.org/en/master/Tutorial/FirstStepsInSeqAn.html>

Note that we do not have to close the file manually: The `SequenceStream` object will automatically close any open files when it goes out of scope and it is destructed. If you want to force a file to be closed, you can use the function `close`.

## Adding Error Handling

Now, a new FASTA file named `example.fasta` in a directory of your choice with the following content:

```
>seq1
CCCCCCCCCCCC
>seq2
CGATCGATC
>seq3
TTTTTT
```

Then, copy the program above into new application `basic_seq_io_example`, adjust the path "`example.fasta`" to the just created FASTA file, compile the program, and run it. You should see the following output:

```
# basic_seq_io
seq1 CCCCCCCCCCCCCC
```

**Assignment 1**

Type: **6** Reproduction

**Objective**

Adjust **11** program above to use the first command line parameter `argv[1]`, i.e. the first argument. Check that there actually is such an argument (`argc >= 2`) and let `main()` return 1 otherwise.

**Solution**

Click [more...](#) to see the solution.

[more...](#)

**13** Our program is very simple but there is one large problem: Anything can go wrong during file I/O and we have not used any means to handle such errors. Let us add some error handling.

We can use the Function `isGood` to check whether the creation of the object, this function indicates whether the `readRecord` return **15** value that indicates whether the different value otherwise.

**12** Björn had this to say, **14** reading. After the creation of the object, the function `isGood` returns 0, and a

The program will now read as follows:

```
#include <iostream>
#include <seqan/basic_seq_io.h>
```

ABBILDUNG 3.11: Revision der ersten Verbesserung des neuen Anfänger-Tutorials

## Strings

In this section, we will have a detailed look at the SeqAn class `String`. You will learn how to build and expand strings as well as how to compare and convert them.

### Building Strings

Let's first have a look at an example on how to define a `String`. The type of the contained value is specified by the first template argument, e.g. `char` or `int`.

```
String<char> myText; // A string of characters.
String<int> myNumbers; // A string of integers.
```

Any type that provides a default constructor, a copy constructor and an assignment operator can be used as the alphabet / contained type of a `String`. This includes the C++ POD types, e.g. `char`, `int`, `double` etc., but you can use more complex types, e.g. `String`s, too.

```
String<String<char>> myList; // A string of character strings.
```

### Hint

Nested Sequences (aka "Strings of Strings")

A set of sequences can either be stored in a sequence of sequences, for example in a `String< String<char>>`, or in `StringSet`. See the tutorial `String Sets` for more information about the class `StringSet`.

ABBILDUNG 3.12: Zweite Verbesserung des neuen Anfänger-Tutorials

**Vorher** “RazerS3: Fixing mate pair modus, deferred compaction.”

**Nachher** “[FIX] Always adding option --thread-count for RazerS 3, but hiding it in sequential mode.”

## Beispiel 2

**Vorher** “Disabling parallel STL in fiona on MinGW.”

**Nachher** “[API] removed assertions from arg\_parse value accession methods, changed behavior of getValue methods

- previously, an assertion ensured that no unset value was requested from the Argument-Parser
- now, the method will just not alter the passed reference or will return empty values respectively”

**Umstellung Subversion auf Git** Die Umstellung auf Git wurde mit dem SeqAn-Team besprochen und eine Einführung wurde gegeben. Alle SeqAn-Entwickler arbeiten zentral und lokal mit Git und können nun lokale Revisionen erstellen, ohne das zentrale Repository zu kompromittieren. Zuvor kam es immer wieder zu Commits, die das Bauen von SeqAn verhinderte.

**Code-Reviews** Die Code-Reviews wurden gut angenommen. Im späteren Verlauf wurde das Prä-Commit-Verfahren auf ein Post-Commit-Verfahren umgestellt. Für Externe gilt weiterhin das Prä-Commit-Verfahren. Ich habe die Codequalität von SeqAn nicht weiter verfolgt, weil ich dieser Verbesserung in Anbetracht meiner eigentlichen GTM-Forschung keinen weiteren Raum einräumen wollte. Die gute Annahme seitens der SeqAn-Entwickler deutet aber darauf hin, dass sich die Code-Qualität verbessert hat.

### 3.2.5.2 Argument-Parser

Sämtliche SeqAn-Anwendungen wurden auf den neuen Argument-Parser umgestellt. Bei der Vorstellung dieses Parsers auf den folgenden Workshops gab es durchweg positives Feedback. Ein Workshop-Teilnehmer sah darin sogar einen Grund, SeqAn allein aus diesem Grund zu verwenden.

### 3.2.5.3 Installation

Die Installationsanleitungen wurden vollständig überarbeiteten und angeglichenen. Konkrete Hürden bei der Installation von SeqAn (insbesondere unter Windows), wurden auf technischer Seite beseitigt.

Während der Workshops gab es durchweg positives Feedback. Ein Anwender war über die Befragung zu den Installationsanleitungen sogar überrascht und bezeichnete sie als “vollkommen problemlos”. Ein PMSB’13-Teilnehmer beurteilte die Installationsanleitung als “idiotensicher”.

Vor der Verbesserung bewertete die Hälfte der Workshop-Teilnehmer die Installation als “einfach” oder “sehr einfach”. Nach der Verbesserung waren es 85%. Unter den Windows- und Mac-Anwendern kam es zu der deutlichsten Verbesserung.

### 3.2.5.4 Dokumentation

Die Verbesserung der Dokumentation wird im Abschnitt 4.4.8 und die dazugehörige Validierung im Abschnitt 4.5 besprochen.

### 3.2.5.5 Tutorials

Die Tutorials sind neben der Dokumentation die wichtigsten Lernressourcen für SeqAn-Anwender. Daher habe ich die Validierung sowohl argumentativ, als auch empirisch vorgenommen.

**Argumentative Validierung** Bei der Aufgabe des Lernens werden acht sequentielle Phasen durchlaufen (Aggarwal 2009; Gagné 1985). Reardon (2008) formuliert Instruktionen, die diese Phasen unterstützen und in den SeqAn-Tutorials wie folgt umgesetzt wurden:

#### 1. Motivationsphase

Diese Phase wird durch die Tutorial-Metainformationen und -Einführung bedient. Der Anwender kann auf der Grundlage dieser Informationen entscheiden, ob und in welchem Umfang ihn das Tutorial helfen kann. Das gleiche trifft auf die Einführungen der inhaltlichen Abschnitte zu.

#### 2. Wahrnehmungsphase

Durch den Einsatz von typisierten Informationen (z.B. blaue Box für optionale Inhalte) werden wesentliche, von wichtigen und weiterführenden Informationen unterschieden. Durch die unterschiedliche visuelle Darstellung, kann der Anwender schnell diese Informationstypen wahrnehmen.

#### 3. Akquisitionsphase

Diese Phase wird durch Beispiele und Übungsaufgaben des Typs *Review* unterstützt, indem eben wahrgenommenes Wissen verfestigt wird.

#### 4. Retentionsphase

Diese Phase beschreibt die Speicherung von Informationen im Gehirn des Lernenden und kann nicht beeinflusst werden.

#### 5. Abrupphase

Diese Phase wird durch Übungsaufgaben des Typs *Application* unterstützt, indem gespeicherte Wissen, erneut abgerufen und damit verfestigt wird.

## **6. Generalisierungsphase**

Diese Phase wird durch Beispiele und Übungsaufgaben des Typs *Transfer* unterstützt, indem gespeichertes Wissen auf verwandte Aufgabenstellungen angewendet werden muss.

## **7. Durchführungsphase**

Diese Phase wird geringfügig durch die Angabe der voraussichtlichen Bearbeitungsdauer einer jeden Übungsaufgabe unterstützt.

## **8. Rückmeldungsphase**

Diese Phase wird durch die Bereitstellung vollständiger und kompilierbarer Teil- und Gesamtlösungen zu den Übungsaufgaben unterstützt.

Darüber hinaus wird der Anwender durch die Angabe weiterführender Tutorials inspiriert und über weitere Anwendungsgebiete von SeqAn informiert.

**Empirische Validierung** Das neue Einführungs-Tutorial “A First Example” wurde sehr positiv von den Workshop’12- und PMSB’12-Teilnehmern angenommen. 80% der Befragten bewerten dieses Tutorial als überdurchschnittlich (40%) bzw. außergewöhnlich hilfreich (40%).

Wurden die Tutorials in ihrer Gesamtheit während des Workshops’11 nur von 24% als gut oder besser bewertet, waren das nach der Verbesserung zum Workshop’12 ganze 67%.

Bei der PMSB’12-Befragung gab es durchweg sehr gute Bewertungen, die aber durch die enge Zusammenarbeit mit den SeqAn-Entwicklern verzerrt sein könnte und damit nicht für die Validierung verwendet werden können.

## PHASE 2: PLANUNG UND DURCHFÜHRUNG DER DATENERHEBUNG

Da nun die größten Usability-Probleme — insbesondere bzgl. der Out-Of-Of-Experience — beseitigt wurden, kann mit der Erforschung der API im engeren Sinne fortgefahren werden. Dazu habe ich für die Analyse, der in diesem Abschnitt vorgestellten erhobenen Daten, die GTM verwendet, welche ich bereits im Abschnitt 1.4 vorgestellt habe.

Um eine für die Analysezwecke sinnvolle Datenerhebung zu planen, lohnt sich zunächst der Vergleich mit anderen Studien.

### **3.3.1 Vergleich mit anderen Studien**

In der Arbeit von de Souza et al. (2004) wurde eine nicht-partizipative Feldstudie über einen Zeitraum von 11 Wochen durchgeführt. Dabei wurden Beobachtungen auf nicht weitere definierte Weise festgehalten und semi-strukturierte Interviews durchgeführt.

Bei der fachlich-nahen Arbeit von Letondal (2006) wurden für die Entwicklung eines Anwendungsentwicklungsumgebung-Hybriden (Ansatz: *Programming In The User Interface*), über mehrere Iterationen hinweg, größtenteils Brainstorming-Sessions und teilweise videoaufgezeichnete Interviews eingesetzt (Details siehe Abschnitt 2.5.1).

In der Clarkeschen Forschung (u.a. Clarke 2005b) werden API-Anwender bei der Lösung gestellter Programmieraufgaben videoaufgezeichnet.

LaToza et al. (2007) beobachtete bei seiner Arbeit ebenfalls Entwickler und zeichnete diese mit Video auf. Des Weiteren wurden die Entwickler instruiert, lautes Denken (engl. *Think Aloud*) anzuwenden. Am außergewöhnlichsten ist jedoch die Instrumentalisierung der Eclipse-Entwicklungsumgebung, mit deren Hilfe die Autoren diverse Ereignisse innerhalb der Entwicklungsumgebung mitgeschnitten haben. Details zur dieser Arbeit finden sich im Abschnitt 2.2.5.

Bei der im Abschnitt 2.4.9.4 näher beschriebenen Concept-Maps-Methode (Gerken et al. 2011) werden informelle Gruppendiskussionen mit den API-Entwicklern und -Anwendern geführt und die Ergebnisse an einem Flipchart, für alle zugänglich, festgehalten.

Grill et al. (2012) stellen in ihrer Fallstudie ein Verfahren zur API-Usability-Evaluation vor und nutzen dabei Workshops, die den beim BioStore-Projekt eingesetzten, ähneln. Während dieser Workshops werden API-Anwender bei ihrer Arbeit videoaufgezeichnet und im Anschluss interviewt. Weitere Details finden sich im Abschnitt 2.5.2.

Piccioni et al. verwendeten zur Verbesserung einer Persistenz-Bibliothek ebenfalls Videoaufzeichnungen von Problem-lösenden API-Anwendern und anschließenden Interviews (Details siehe Abschnitt 2.5.3).

### 3.3.1.1 Kombination von Methoden

Die meisten genannten Arbeiten verwenden eine Kombination verschiedener Datenerhebungs- und Analysemethoden. Für die Kombination der, in der klassischen Usability-Evaluation gebräuchlichen Methoden *Heuristische Evaluation* (Nielsen u. Molich 1990) und *Usability-Test* (Faulkner 2003) konnte bereits gezeigt werden, dass diese verschiedenartige Usability-Probleme aufdecken und ideal kombiniert werden können (Fu et al. 2002).

Eine ähnliche Beobachtung konnten Grill et al. (2012) bei der Kombination von semi-strukturierten Interviews und Videoaufzeichnungen machen. Ihre Analyse beider Quellen förderte 168 API-Usability-Probleme zu Tage, von denen 157 in nur einer der beiden Datenquellen zu finden waren. Sie stellten beispielsweise fest, dass sich Laufzeit-Probleme nicht mit einer HE auffinden konnten. Außerdem bemerkten sie, dass sie zwar die meisten Probleme mit Hilfe der Interviews fanden, die schwerwiegendsten Probleme jedoch in den Videoaufzeichnungen verborgen waren. Ähnliche Erfahrungen haben auch Piccioni et al. gemacht.

### 3.3.1.2 Subjektive und objektive Daten

Es gibt in der API-Usability-Forschung einen Trend zur Betonung subjektiver Daten (Robillard u. DeLine 2010; Rosson u. Carroll 2001; Stylos et al. 2008), dem auch alle mir bekannten API-Evaluationsstudien folgen. Auch wenn Videoaufzeichnungen, technisch betrachtet, objektive Daten darstellen<sup>27</sup>, werden diese meist nur verwendet, um Erklärungslücken bei der Analyse von Interviews zu klären (vgl. Gerken et al. 2011; Grill et al. 2012; Letondal 2006) oder quantitative Betrachtungen vorzunehmen (vgl. Piccioni et al.). Ausschließlich objektive Daten verwenden nach meiner Kenntnis nur zwei API-Evaluationsstudien (de Souza u. Bentolila 2009; Watson 2009).

Tatsächlich sind aber beide Datentypen<sup>28</sup> wichtig:

**Subjektive Daten** erhalten viele relevante Informationen zu Usability-Problemen (Hou et al. 2005; Robillard u. DeLine 2010; Rosson u. Carroll 2001; Stylos et al. 2008), z.B. in Bezug auf die Anforderungen an das Softwaresystem (Eagan u. Stasko 2008). Allerdings besteht die Gefahr, dass Befragte manchen für irrelevant gehaltenen (Daughtry et al. 2009a) oder kritischen Punkt in Interviews (*soziale Erwünschtheit*, Hartmann 1991) bzw. in Gruppendiskussionen (*Schweigespirale*, Noelle-Neumann 1989) nicht äußern. Ebenso schwer wiegt, dass Probanden auch nur Äußerungen zu Punkten machen können, die ihnen bewusst sind (Ko et al. 2011).

<sup>27</sup> Abschnitt 2.3 befasst sich mit den Klassen der Erhebungs- und Analyseverfahren.

<sup>28</sup> Subjektive Verfahren erheben subjektive Meinungen/Ansichten/Darlegungen der Benutzer, wohingegen objektive Verfahren direkt beobachtbare Daten erfassen. Eine ausführlichere Differenzierung beschreiben Sarodnick u. Brau (2006).

**Objektive Daten** In objektiven Daten hingegen manifestieren sich Probleme, die dem Anwender möglicherweise nicht bewusst sind oder nicht korrekt erfragt wurden (Ko et al. 2011). Gegebenenfalls kann der Befragte ein Problem nicht verbalisieren, weil er es bereits gelöst hat, bevor er jemals danach gefragt wurde (Sunshine et al. 2014).

Die Mischung beider Datentypen vereint die jeweiligen Vorteile (Sarodnick u. Brau 2006). Im Laufe dieser Arbeit habe ich die Erfahrung gemacht, dass die qualitative Analyse objektiver Daten anspruchsvoller ist, als die subjektiver Daten. Objektive Daten enthalten häufig weniger Hinweise darauf, "wo die Musik spielt".

### 3.3.1.3 Studienformen

Die meisten mir bekannten API-Usability relevanten Studien sind Labor- oder Fallstudien. Zu den wenigen partizipativen Feldstudien gehören Letondal (2006) und Gerken et al. (2011). Die einzige mir bekannte nicht-partizipative Feldstudie ist die von de Souza et al. (2004).

Langzeitstudien sind rar. Zwei Studien (Gerken et al. 2011; de Souza et al. 2004) machten Feldbeobachtungen über jeweils elf Wochen hinweg. Die Arbeit von Letondal (2006) fußt auf Datenerhebungen, die über einen Zeitraum von acht Jahren angefertigt wurden.

## 3.3.2 Planung der Datenerhebung

### 3.3.2.1 Ziele

Die von mir geplante Datenerhebung, verfolgte zwei primäre Ziele:

1. Die Datenerhebung sollte so reichhaltig wie möglich sein, denn die möglichen Zeitpunkte zur Datenerhebung waren begrenzt.
2. Die Datenerhebung sollte so wenig wie möglich, die Arbeit der SeqAn-API-Anwender beeinflussen, um verallgemeinerbare Aussagen treffen zu können.

### 3.3.2.2 Anforderungen

Für die Datenerhebung musste ich ein Verfahren entwickeln, das die eben beschriebenen primären Ziele und die daraus folgenden Anforderungen erfüllt:

- Die Daten müssen auf den individuellen Arbeitsplätzen der Probanden erhoben werden. Die Probanden sollen also nicht an einer speziell für die Datenerhebung vorbereiteten Arbeitsstation arbeiten, was anderenfalls zu Verfälschungen führt (McKeogh u. Exton 2004).

- Die Datenerhebung muss einfach einzurichten sein, um eine möglichst hohe Bereitschaft zur Datenerhebung auf Seite der Probanden zu erhalten.
- Die Datenerhebung muss unauffällig sein und darf den Proband in seiner Arbeit nicht stören.
- Die Datenerhebung muss sich für Langzeitbeobachtungen eignen, um auch API-Usability-Probleme zu entdecken, die erst nach längerem Gebrauch einer API auftreten (Ellis et al. 2007; Stylos u. Clarke 2007) bzw. nicht in einem einzelnen Messpunkt zu beobachten sind (Gerken et al. 2011; Grill et al. 2012).
- Die Datenerhebung muss datenschutzrechtlichen Anforderungen genügen. Dazu gehört, neben einem einzuholenden Einverständnis, die Möglichkeit, in die erhobenen Daten Einsicht zu nehmen und die Datenerhebung deaktivieren zu können.
- Die Datenerhebung muss sowohl subjektive als auch objektive Daten erheben, um einerseits reichhaltige Erkenntnisse zu ermöglichen und andererseits den begrenzten Möglichkeiten zur Datenerhebung Rechnung zu tragen.
  - Für meine Forschung setze ich die GTM ein, was mögliche weitere Datenerhebungen erfordert (*theoretisches Sampling*). Die besondere Reichhaltigkeit der Daten soll die Wahrscheinlichkeit senken, weitere Daten erheben zu müssen.
  - Die Reichhaltigkeit der Daten wird durch die Verwendung subjektiver und objektiver Datenerhebungen erreicht (siehe Abschnitt 3.3.1.2).
- Die Datenerhebung muss Vertreter der tatsächlichen Anwendergruppe, der zu analysierenden API, umfassen. Nur so lassen sich relevante API-Usability-Probleme erheben (Clarke 2004; Henning 2007).

### 3.3.2.3 Mögliche Datenquellen

Für die Datenerhebung eigneten sich, die im Rahmen des BioStore-Projekts durchgeführten drei SeqAn-Workshops. Darüber hinaus boten sich die ebenfalls jährlich stattfindenden PMSB-Praktika an. Beide Formate wurde bereits in den Abschnitten 3.1.2 und 3.1.3 vorgestellt.

### 3.3.2.4 Konzept

Um die eben beschriebenen Anforderungen, unter Beachtung der möglichen Datenquellen, zu erfüllen, habe ich mich entschieden, eine nicht-partizipative Feldstudie unter Verwendung von zwei Typen von Datenerhebungen durchzuführen — zwei subjektive und ein objektives Datenerhebungsverfahren. Die Kombination unterschiedlicher Erhebungsverfahren hat sich bewährt (vgl. Grill et al. 2012; LaToza et al. 2007; Letondal 2006; Piccioni et al.; de Souza et al. 2004).

Als subjektive Datenerhebungsverfahren sollten ein *Cognitive-Dimensions-Fragebogen* und eine *Gruppendiskussion* zum Einsatz kommen. Für das objektive Verfahren sollen Aufzeichnungen der Fortschritte dienen, die SeqAn-Anwender bei der Entwicklung von, auf SeqAn-basierenden Programmen entwickeln. Letzteres bezeichne ich als *Programmierfortschritte*-Erhebung. Alle drei Verfahren werden in folgenden Abschnitten vollständig beschrieben.

In der ersten Analysephase habe ich die Anwenderschaft von SeqAn charakterisiert (siehe Abschnitt 3.2.3.1). Zu dieser gehören berufstätige, nationale und internationale Wissenschaftler aus den Bereichen Informatik, Bioinformatik und der Physik. Ich betrachte diese Personen als geeignete Probanden für meine Forschung. Der Studentenanteil stellt keine Probleme dar, da er einen beachtlichen Teil zur bioinformatischen Arbeit beiträgt (Letondal 2006) und damit mindestens zur zukünftigen Anwendergruppe gerechnet werden kann.

### 3.3.3 Gruppendiskussion

Als erste Datenquelle habe ich eine Gruppendiskussion nach Mayring (2002) konzeptioniert und durchgeführt. Die Gruppendiskussion ist ein Verfahren, das von keiner mir bekannten Studie verwendet wurde. Lediglich bei Gerken et al. (2011) werden informelle Gespräche in der Gruppe geführt.

Interviews wurden zwar in einigen Studien angewandt (vgl. Grill et al. 2012; Letondal 2006; Piccioni et al.; de Souza et al. 2004), haben jedoch gegenüber der Gruppendiskussion Nachteile, „denn die Erfahrungen zeigen, dass in gut geführten Gruppendiskussionen Rationalisierungen, psychische Sperren durchbrochen werden können und die Beteiligten dann die Einstellungen offen legen, die auch im Alltag ihr Denken, Fühlen und Handeln bestimmen.“ (Mayring 2002, S. 77)

Als subjektive Datenquelle hatte die Gruppendiskussion in meiner Arbeit den Vorteil, mich für die von den Anwendern empfundenen Usability-Probleme in SeqAn sensibler zu machen — ohne ihnen im Vorhinein eine wegweisende Wichtigkeit zu geben, wie das bei der Usability-Evaluation nach Grill et al. (2012) der Fall ist.

#### 3.3.3.1 Planung und Vorbereitung

Nach Mayring (2002) besteht eine Gruppendiskussion aus den folgenden vier Phasen:

##### 1. Darbietung Grundreiz

Diese Phase dient dazu, das Thema der Diskussion knapp vorzustellen.

##### 2. Freie Diskussion

In dieser Phase findet die eigentliche Diskussion statt.

##### 3. Weitere Reizargumente

Kommt die Diskussion ins Stocken, können weitere Reizargumente vorgestellt werden.

#### 4. Metadiskussion

Diese Phase dient der Diskussion über die geführte Diskussion und soll eine letzte Reflektion anregen.

Die Analyse aus Phase 1 (siehe Abschnitt 3.2) hat gezeigt, dass SeqAns größtes Usability-Problem im Bereich des Programmierparadigmas Templatemetaprogrammierung verortet ist. Darum habe ich dieses Thema als Grundreiz gewählt.

Die Gruppendiskussion sollte am letzten Tag des Workshop'12 stattfinden. Für die Behebung grober Usability-Probleme habe ich bei diesem Workshop ebenfalls, die bereits im Abschnitt 3.2.1.4 vorgestellten Feedback-Zettel eingesetzt. Die auf den Zetteln geäußerten Punkte und die Ergebnisse meiner Heuristischen Evaluation dienten mir als Reizargument.

Die Reizargumente (die Quelle steht in Klammern) lauten konkret:

- STL-Konformität (Feedback)
- Operatorenüberladungen (Feedback, HE)
- Lesbarkeit von Compilerfehlern (Feedback, HE)
- Online-Dokumentation (Feedback, HE)
- Konkrete API-Probleme, insbesondere `Shape`, `Hash` und `Index` (Feedback)
- `Iterators`, `StringSet` (Feedback, HE)
- Namenskonventionen (Feedback, HE)
- Rückmeldung von Lesefehlern (Feedback, HE)

Für die Diskussion habe ich 30 Minuten vorgesehen — mit der Verlängerungsoption um weitere 30 Minuten. Die Diskussion sollte audioaufgezeichnet werden. Für eine maximale Ausdrucksfreiheit, sollte ein Whiteboard zu Verfügung gestellt werden.

Aus dem SeqAn-Team habe ich ein Mitglied für die technische<sup>29</sup> und drei Mitglieder für die fachliche Beobachtung<sup>30</sup> ausgewählt (Details siehe Mayring 2002). Alle Beobachter wurden von mir vor der Gruppendiskussion schriftlich und mündlich eingewiesen.

Die übrigen Team-Mitglieder habe ich der Diskussion, wegen meiner negativen Erfahrungen bei der Feedback-Runde des Workshops'11 (vgl. Abschnitt 3.1.4), ausgeschlossen. Dies habe ich getan, um das Phänomen des *technischen Wegargumentierens* zu vermeiden. Damit sollte verhindert werden, dass die Hemmschwelle zur freien Meinungsäußerung angehoben wird, der Diskussionsverlauf durch anderen SeqAn-Entwickler beeinflusst wird oder gar “Revierkämpfe” ausgefochten werden.

---

<sup>29</sup> Der technische Beobachter hält Mimik und Gestik von Diskussionsteilnehmern schriftlich fest.

<sup>30</sup> Die fachlichen Beobachter halten inhaltlich relevante Äußerungen schriftlich fest.

Ich war mir bewusst, dass der Ausschluss wichtiger SeqAn-Entwickler das Verständnis detailliert vorgetragener, technischer Kritik erschweren kann. Darum sollten sich die Beobachter in der zweiten Hälfte der Gruppendiskussion selbst einbringen dürfen. Dabei wurden die, nun als SeqAn-Experten dienenden Beobachter darum gebeten, eine gewaltfreie Sprache (vgl. *klientenzentrierte Gesprächsführung* nach Carl R. Rogers in Wingchen 2014) zu verwenden und unbedingt Äußerungen wie “Dummer Vorschlag!”, “Geht nicht, weil...” oder “Wozu?!” zu vermeiden.

### 3.3.3.2 Durchführung und Fazit

Die Gruppendiskussion fand am 06.09.2012 — also am letzten Tag des Workshop’12 – statt. Die Durchführung entsprach der Planung und verlief sehr gut. Der Diskussionsbedarf war so groß, dass die volle Zeit von einer Stunde ausgeschöpft wurde und aus terminlichen Gründen abgebrochen werden musste.

Die von Mayring (2002) beschriebene erleichterte Offenlegung von, und Reflektion über tatsächliche Alltagsrätselrisse, konnte ich klar feststellen. Bei der Darlegung des Grundreizes *Templatemetaprogrammierung* und der Reizargumente *Namenskonventionen* und *Rückmeldung von Lesefehlern* entwickelte sich eine ausgeprägte Dynamik. Das führt mich auf die gute inhaltliche Vorbereitung und auf den Ausschluss der meisten SeqAn-Entwickler zurück. Dennoch war vereinzelt, gegen Ende der Diskussion *technisches Wegargumentieren* zu beobachten.

Für die Metadiskussion verblieben aus Zeitgründen leider nur wenige Minuten. Die Gruppendiskussion wurde von zwei Dritteln der Teilnehmer als sehr positiv bewertet und als notwendig erachtet. Einzelne Teilnehmer sagten, dass diese Diskussion eine einmalige Erfahrung für sie war.

Die pseudonymisierte Transkript befindet sich im Anhang C.4. Die Gruppendiskussion wurde mit Hilfe der GTM analysiert. Die Ergebnisse werden in den Abschnitten 3.5 und 4 vorgestellt.

### 3.3.4 Cognitive-Dimensions-Fragebogen

Das CDF<sup>G</sup> ist ein Diskussionswerkzeug für die Bewertung der Benutzerfreundlichkeit von so genannten *Notationen* (siehe Abschnitt 2.3.2). Um die, im Framework formulierten, kognitiven Dimensionen (CD) zu ermitteln und damit Einsichten in den zu untersuchenden Gegenstand zu erhalten, gibt es die Möglichkeit, einen generischen Fragebogen (Blackwell u. Green 2000) einzusetzen (siehe Abschnitt 2.3.2.5). Alternativ hat Kadoda (2000) selbst einen Fragebogen entwickelt, der allerdings nur relevant erscheinende CDs umfasst und selbst nicht veröffentlicht wurde.

Für die Evaluation eignet sich allerdings auch nicht der generische Fragebogen. Seine größte Schwäche ist seine Generizität, die von den Autoren als seine größte Stärke herausgestellt wird. Tatsächlich stellten aber Selbige im Rahmen ihrer Pilotstudie fest, dass Anwender Probleme beim Verständnis der generischen Fragen hatten.

### 3.3.4.1 Planung und Vorbereitung

Um das Problem der Generizität zu lösen, habe ich selbst einen Fragebogen entwickelt, der nicht mehr allgemein von einer “Notation”, sondern konkret von “SeqAn” spricht. Das heißt, ich habe den generischen Fragebogen auf SeqAn instanziert/spezialisiert.

Genauer gesagt habe ich unter anderem folgende Anpassungen vorgenommen:

- Ersetzung produktrelevanter Begriffe durch “SeqAn” bzw. “API”

- Entfernung des zweiten Abschnitts

Dieser Abschnitt umfasst, bei Schriftgröße 10, eine ganze A4-Seite und dient nur dazu, die generischen Termini *Product*, *Notation*, *Helper Device*, *Redefinition Device* und *Sub-Device* so zu erklären, dass der Befragte den Fragebogen auf den Sachgegenstand beziehen kann.

- Umformulierung der generischen Aktivitäten zu API-relevanten Aktivitäten, wie “Implementieren von Code” oder “Umstrukturieren von Code”
- Verwendung einer Mischung aus den originären CDs und denen von Clarke u. Becker (2003)
- Formulierung von Fragen für die neuen 12 CDs

Im Abschnitt 2.4.2.4 beschreibe ich den Wert der Arbeit von Clarke u. Becker (2003), zeige aber auch Unzulänglichkeiten auf. So wurden beispielsweise die originären CDs *Fehleranfälligkeit* und *Vorläufigkeit* entfernt, ohne dass diese durch die anderen CDs abgedeckt wären. Auf der Grundlage der 14 CDs von Blackwell u. Green (2000) und den 12 CDs von Clarke u. Becker (2003) habe ich eigene 12 CDs entwickelt, die auch die beiden entfernten CDs umfassen.

Der Fragebogen wurde als Online-Formular entwickelt. Abbildung 3.13 zeigt einen Ausschnitt. Alle von mir verwendeten kognitiven Dimensionen, die dazugehörigen Fragen und der vollständige Fragebogen befinden sich im Anhang E.4.

Gemeinsam mit meinen Arbeitskollegen und den Autoren des Cognitive Dimension Frameworks Alan Blackwell und Thomas Green habe ich den Fragebogen mehrfach revisioniert. In einer Mail vom 21.03.2013 hat Alan Blackwell den Fragebogen abschließend mit “looks nice” beurteilt.

### 3.3.4.2 Durchführung und Fazit

Die Befragung mit Hilfe des Fragebogens fand im Rahmen des Workshops’13 statt.

Der von mir investierte Aufwand zeigt, dass es vollkommen unverhältnismäßig gewesen wäre, jedem Befragten diese Transfer-Leistung aufzubürden. An dieser Stelle kann man auch die Argumentation von Ellis et al. (2007); Henning (2007) anwenden: Fragebögen werden von viel mehr Menschen ausgefüllt als entworfen. Es ist also ökonomisch absolut sinnvoll, die Instanziierung/Spezialisierung eines Fragebogens durch deren Autoren durchführen zu lassen.

SeqAn Usability Fragebogen    #1 Hintergrundinformationen    #2 Zeitangaben    #3 API    #4 Persönliche Meinung

Dieser Fragebogen erfragt, wie leicht/schwer dir die Arbeit mit der SeqAn-Softwarebibliothek fällt.  
Er enthält eine Reihe von Fragen, die dich dazu motivieren sollen, über Möglichkeiten nachzudenken, wie du SeqAn benötigst und ob dir SeqAn hilft, die Dinge zu tun, die du benötigst.

## Teil 1 Hintergrundinformationen

**Was studierst du / hast du studiert?**  
Informatik, Bioinformatik, Biotechnologie, ...

**Was beschreibt deinen Arbeitsstil am besten?**  
Systematisch    Pragmatisch    Opportunistisch

**Wie lange hast du bereits mit SeqAn gearbeitet?**  
6 Tage Wochen Monate Jahre

ABBILDUNG 3.13: Ausschnitt aus dem deutschsprachigen Cognitive-Dimensions-Fragebogen

Insgesamt betrachtet, bleibt ein Cognitive-Dimensions-Fragebogen anspruchsvoll. Bei drei Probanden kam es bei jeweils einer Frage zu Missverständnissen<sup>31</sup>. Eine Reihe von Fragen wurden von manchen Probanden nicht beantwortet. Die Begeisterung für den Fragebogen war geteilt. Ein Proband bezeichnete ihn als “anstrengend”<sup>32</sup>. Andere Probanden hingegen bewerteten den Fragebogen als “Super”<sup>33</sup> oder “Klasse!”<sup>34</sup>.

Die aufgefüllten Fragebögen wurden mit Hilfe der GTM analysiert. Die Ergebnisse werden in den Abschnitten 3.5 und 4 vorgestellt.

### 3.3.5 Programmierfortschritte-Erhebung

Bei dieser Datenerhebungsmethode handelt es sich um eine, die ich selbst entwickelt habe und in keiner mir bekannten Studie angewendet wurde. Sie besteht in der Erhebung von Daten, die den Entwicklungsprozess von Anwendungen, die von SeqAn-Anwendern entwickelt werden, nachvollziehbar machen sollen.

#### 3.3.5.1 Planung und Vorbereitung

Um die Programmierfortschritte der verschiedenen SeqAn-Anwender dokumentieren zu können, benötigte ich die erstellten und veränderten Quellcode-Dateien, die bei der Programmentwicklung anfallen.

<sup>31</sup> Beispiel: In einer Frage zu der CD *Hard Mental Operations* wurde die Formulierung “when combining things” verwendet. Die Frage zielte auf die Verarbeitung verschiedener Informationen im Arbeitsgedächtnis abgezielt. Ein Proband jedoch bezog sie auf die Verknüpfung zwischen Funktion und dessen Dokumentationseintrag (siehe `apiua://survey/cd/2013-09-18T17:41:31.929+02:00/hardMentalOperations`).

Diese sollten bei jedem Versuch, den eigenen Programmcode zu kompilieren, erhoben werden, da davon auszugehen ist, dass dann ein irgendwie gearteter Arbeitsschritt abgeschlossen ist.

Um den Verständnisprozess des Anwenders besser verstehen zu können, interessierten mich auch die Zugriffe auf die Online-Dokumentation. Es war abzusehen, dass zwischen zwei Kompilierversuchen mehrere Minuten vergehen können. Die Zugriffe auf die Dokumentation sollten helfen, diese Lücken besser zu verstehen — beispielsweise, ob ein SeqAn-Anwender gerade einen Dokumentationseintrag liest. Des Weiteren konnte ich so die Online-Dokumentation im Gebrauch durch den Anwender untersuchen. Dies zu können ist essentiell, weil die Dokumentation selbst Bestandteil der API im weiteren Sinne ist und eine wichtige Ressource zum Erlernen einer API darstellt (Robillard 2009; Robillard u. DeLine 2010).

Zu guter Letzt interessierten mich Daten zur Arbeitsumgebung, wie der verwendeten Entwicklungsumgebung.

Für die Datenerhebung selbst habe ich eine technische Lösung geplant und unter Verwendung einer Client-Server-Architektur implementiert.

**Client** Die Datenerhebung auf den individuellen Arbeitsplätzen sollte so transparent laufen und einzurichten sein, wie möglich.

SqAn verwendet *CMake*<sup>32</sup> als plattformübergreifendes Build-System. Als SeqAn-Anwender muss man bei der Installation, zunächst SeqAn herunterladen und dann, mit Hilfe von CMake, die notwendigen Projektdateien für die eigene Entwicklungsumgebung erstellen. Anschließend kann man mit SeqAn in der Entwicklungsumgebung seiner Wahl arbeiten.

Für die Datenerhebung habe ich die CMake-Dateien von SeqAn — und damit den Build-Prozess — so verändert, dass nach jedem Kompilierversuch ein Script ausgeführt wird, das Änderungen im SeqAn-Verzeichnis erkennt. Die betroffenen Dateien werden bei diesem Prozess in ein ZIP-Archiv gepackt und auf einen Datenerhebungsserver asynchron hochgeladen.

Der von mir entwickelte Datenerhebungsclient ist auf GitHub gehostet<sup>33</sup>.

**Server** Die Server-Komponente dient der Sammlung der, von den Clients hochgeladenen Daten. Außerdem stellt er ein selbst entwickeltes JavaScript bereit, das in jede beliebige Webseite integriert werden kann und auf minutiöse Weise eine große Spannbreite an Ereignissen auf den instrumentierten Webseiten mitschneidet.

Die Server-Komponente habe ich in Form einer Java EE Web-Anwendung implementiert und auf GitHub bereitgestellt<sup>34</sup>.

---

32 <http://www.cmake.org>

33 <https://github.com/bkahlert/api-usability-analyzer-client-python>

34 <https://github.com/bkahlert/api-usability-analyzer-server-java-ee>

Das JavaScript zur Webseiten-Überwachung ist Bestandteil des Servers und verwendet ein interessantes Verfahren zur Umgehung der so genannten *Same-Origin-Policy* (siehe grauer Kasten).

### Website-Überwachung im Detail

Um eine Webseite (z.B. `client.com`) mit Hilfe des Datenerhebungsservers zu überwachen, muss zunächst eine JavaScript-Datei in die zu überwachende Seite eingebunden werden. Dies geschieht mit folgendem HTML-Code:

```
<script src="https://srv.tld/SUAsrv/static/js/SUAcjt.js"></script>
```

Der kanonische Weg zur Übermittlung von Ereignissen, wie dem Laden oder Verlassen einer Seite, wäre die Absetzung einer Ajax-Anfrage<sup>a</sup>. Dabei handelt es sich inhaltlich nicht um eine Anfrage sondern um eine Übermittlung von Daten (hier: Nutzeraktivitäten).

Ajax-Anfragen unterliegen jedoch der Same-Origin-Policy<sup>b</sup>, die den Zweck hat, Datendiebstahl und andere Angriffsformen zu verhindern. Die Übermittlung an einzelne Domains (hier: `srv.tld`) kann zwar erlaubt werden, wird allerdings nicht von älteren Browsern unterstützt<sup>c</sup>.

Um die Datenübermittlung auch bei älteren Browsern zu unterstützen, werden keine Ajax-Anfragen, sondern JSONP-Anfragen<sup>d</sup> abgeschickt. Technisch gesehen, wird dabei ein Script mit Hilfe des `<script>`-Tags in das *Document Object Model* der überwachten Seite eingefügt, das ein Script lädt und dabei Parameter übergibt. Dabei dienen die belegten Parameter der Datenübermittlung. Das zurückgegebene Script enthält optional Rückgabedaten, die in diesem Fall aber irrelevant sind. Diese Art der Datenübermittlung unterliegt nicht der Same-Origin-Policy und erlaubt so die Übertragung der Nutzeraktivitäten.

<sup>a</sup> <http://www.w3.org/2007/06/mobile-ajax/>

<sup>b</sup> <http://www.ibm.com/developerworks/web/library/wa-crossdomaincomm/index.html?ca=drs-#N1019B>

<sup>c</sup> [https://developer.mozilla.org/en-US/docs/Web/HTTP/Access\\_control\\_CORS](https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS)

<sup>d</sup> <http://bob.ippoli.to/archives/2005/12/05/remote-json-jsonp/>

**Datenquelle: Arbeitsumgebung** Für die Analyse der Programmierfortschritte kann es hilfreich sein, Daten zur Arbeitsumgebung des API-Anwenders zu kennen.

Dazu übermittelt der, weiter oben beschriebene Client bei der ersten Einrichtung der Entwicklungs-umgebung Daten zum Betriebssystem und zur Entwicklungsumgebung an den Server. Listung 8 zeigt Inhalt und Struktur der übermittelten, JSON-formatierten Daten.

```
{
  "machine": {
    "machine": "x86_64",
    "processor": "i386",
    "architecture": "64bit"
  },
  "devenv": {
    "CMAKE_CXX_COMPILER": "/usr/bin/g++",
    "CMAKE_GENERATOR": "Xcode"
  },
  "os": {
    "platform": "Darwin-11.1.0-64bit"
  }
}
```

LISTUNG 8: Beispiel: Daten zur Arbeitsumgebung

**Datenquelle: Quelldateien** Bei jedem Kompilierversuch werden alle geänderten Dateien im SeqAn-Arbeitsverzeichnis vom Client in Form eines ZIP-Archivs an den Server übermittelt. Das Senden findet asynchron statt, um den Kompilierprozess nicht zu verlangsamen.

Das ZIP-Archiv erhält dabei einen Namen, der folgende Informationen enthält:

**ID** des Probanden. Diese ID identifiziert jeden Probanden eindeutig.

**Hash-Wert** des SeqAn-Arbeitsverzeichnis-Pfades. Existieren mehrere SeqAn-Installationen auf einem Rechner, können diese so unterschieden werden.

**Zeitpunkt** des Kompilierversuchs. Für den Zeitpunkt wird eine abgewandelte Form des, in der ISO 8601<sup>35</sup> festgelegten Formats zur Darstellung von Datum und Zeit, verwendet. Diese Anpassung war notwendig, da beispielsweise der Doppelpunkt auf den meisten Dateisystemformaten nicht erlaubt ist.

Ein beispielhafter Name für ein übermitteltes ZIP-Archiv lautet:  
**6ndbc4zuiueuaiyv\_b9dc\_2013-04-09T10-31-07.379654+0200.diff.zip**.

**Datenquelle: Onlinedokumentation** Um die Verwendung einer Webseite durch einen Anwender nachvollziehen zu können, reicht es nicht, das Laden der entsprechenden Seite zu protokollieren.

Stattdessen müssen weitaus differenziertere Ereignisse protokolliert werden:

---

<sup>35</sup> <http://www.iso.org/iso/home/standards/iso8601.htm>

**READY** Die Seite wurde vollständig geladen.

**UNLOAD** Die Seite wurde geschlossen.

**FOCUS** Die Seite (bzw. der Browser-Tab) hat den Fokus erhalten.

**BLUR** Die Seite (bzw. der Browser-Tab) hat den Fokus verloren.

**RESIZE** Die Größe der Anzeigefläche wurde verändert. Üblicherweise passiert dies durch die Veränderung der Fenstergröße. Aber auch eingeblendete Browser-Menüs können dieses Ereignis provozieren.

**SCROLL** Der Anwender hat innerhalb der Seite gescrollt.

**TYPING** Der Anwender hat eine Text-Eingabe vorgenommen.

**LINK** Der Anwender hat auf einen Link geklickt.

Bei jedem Ereignis werden, durch das vom Datenerhebungsserver ausgelieferte Script, eine Reihe von Daten asynchron übertragen: Zeitstempel, Ereignistyp, URI<sup>G</sup> der Seite, IP, Proxy-IP, Scrollposition und Größe der Anzeigefläche. Darüber hinaus werden Ereignis-abhängige Daten übermittelt, wie beim **TYPING**-Ereignis die eigentliche Eingabe und der Name des Eingabefeldes.

Im Abschnitt 3.4.3 auf Seite 199 stelle ich das Datenformat genauer vor. Dort befindet sich auch ein Auszug aus den so erfassten Daten (siehe Listung 10).

**Identifikation und Datenschutz** Eben habe ich drei Datenquellen beschrieben, die in ihrer Gesamtheit die Programmierfortschritte eines Anwenders protokollieren. Allerdings stellt sich die Frage, wie die Datenquellen den Probanden zugeordnet werden können.

Die Identifikation eines Anwenders wird wie folgt realisiert:

- Der Anwender muss die Datei `.APIUA` in seinem Benutzerverzeichnis anlegen. Damit weiß SeqAns Build-Prozess, dass eine Datenerhebung erlaubt ist. Darüber hinaus muss der Proband eine *Einverständniserklärung zur Datenerhebung* (siehe Anhang E.1) unterschrieben haben, die ihn über den Umfang der Datenerhebung aufklärt.
- Bei der Einrichtung der SeqAn-Installation erfolgt durch den Anwender ein *CMake*-Aufruf. Der von mir veränderte SeqAn-Build-Prozess prüft das Vorhandensein der `.APIUA`-Datei und aktiviert ggf. die Datenerhebung.
- Es wird eine zufällige 16-stellige alphanumerische ID erzeugt und in der Datei `.APIUA-ID` abgelegt — ist sie bereits vorhanden, wird die darin enthaltene ID wiederverwendet.
- Der Standard-Browser wird geöffnet und eine spezielle Seite auf dem Datenerhebungsserver aufgerufen. Dabei wird die generierte ID an den Server übermittelt. Dem Anwender erscheint eine Seite, die sich für die Bereitschaft zur Datenübermittlung bedankt, die ID des Anwenders zeigt und das Verfahren zur Datenübermittlung erklärt (siehe Abbildung 3.14).

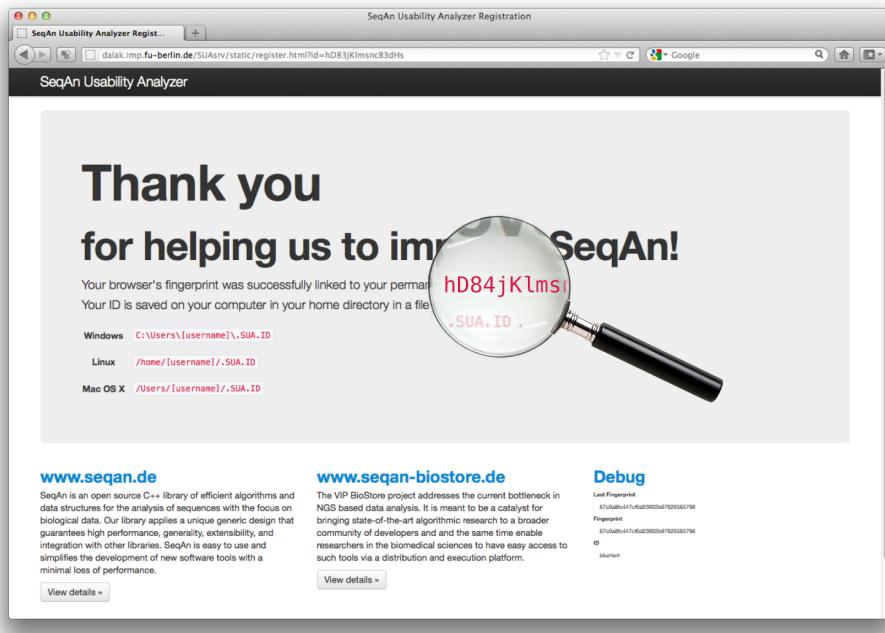


ABBILDUNG 3.14: Bestätigung der Aktivierung der Datenerhebung

- Die, auf dem Client erhobenen Arbeitsumgebungsdaten und Quelldateien enthalten in ihrem Dateinamen diese ID.
- Zur Identifikation des Browsers wird ein sogenannter *Browser-Fingerprint* verwendet. Dabei wird ein Hash über diverse Datenpunkte (Browser, Betriebssystem, Zeitzone, etc.) berechnet.
- Durch den Aufruf der Bestätigungsseite erfährt der Server sowohl die übermittelte ID als auch den jeweils mit übermittelten Browser-Fingerprint. Dieses Tupel wird von dem Server gespeichert.
- Zukünftige Aufrufe von beobachteten Seiten übermitteln nun Nutzeraktivitäten an den Server. Mit Hilfe des Browser-Fingerprints, können diese Datenübermittlungen der ID des Anwenders zugeordnet werden.
- Der Browser-Fingerprint ist nicht dauerhaft stabil und kann sich, beispielsweise durch die Installation von Updates, verändern. Ich habe ein Verfahren entwickelt, dass überaus stabil ist und selbst das Löschen von Cookies überlebt. Mehr Details befinden sich im grauen Kasten.
- Die im Abschnitt 3.2.1.4 beschriebenen Feedback-Zettel aus Phase 1, konnten von den Befragten ebenfalls mit ihrer ID versehen werden, um ein ganzheitliches Bild der Probanden zu erhalten (siehe Abbildung 3.15).

Durch die ausschließliche Verwendung einer ID, setze ich das Prinzip der Pseudonymisierung um. Ein Rückschluss auf die Identität des Probanden ist so, stark erschwert. Ich hatte bei der Datenanalyse selbst mehrere Male das Bedürfnis, einen Probanden zu kontaktieren. Selbst mit dieser starken Motivation ist es mir nicht gelungen, die korrekte Person zu identifizieren.

Die Datenerhebung kann jederzeit vom Proband durch Löschen der Datei .APIUA eigenständig beendet werden.

### Stabiles Browser-Fingerprinting

Browser-Fingerprinting erfreut sich in der Online-Werbebranche großer Beliebtheit (Bager 2013) und verwendet diverse Merkmale, die der Browser (a) in seiner HTTP-Anfrage mitschickt oder (b) sich mit JavaScript auslesen lassen. Durch installierte Plugins, wie Flash oder Java, steigt der verwendbare Merkmalumfang weiter an. Ein Großteil, der weltweit eingesetzten Browser-Installationen, ist durch diese Merkmale eindeutig identifizierbar<sup>a</sup>.

Je mehr Merkmale für den Browser-Fingerprint verwendet werden, desto wahrscheinlicher ist eine eindeutige Erkennbarkeit. Allerdings steigt so auch die Wahrscheinlichkeit einer Veränderung des Browser-Fingerprints.

Um einen Browser dauerhaft zu identifizieren, benötigt man einen Mechanismus, der es einem erlaubt, den alten und den neuen Browser-Fingerprint in Erfahrung zu bringen.

Dazu muss zunächst der alte Browser-Fingerprint im Browser — z.B. in Form eines Cookies — gespeichert werden. Cookies (und alle anderen lokalen Speicherformen) setzen die Same-Origin-Policy um. Das heißt, dass ein, auf einer beliebigen Seite *example.com* eingefügtes Datenerhebungsscript von *server.com*, nicht die Cookies der jeweils anderen Domain auslesen kann. Das Datenerhebungsscript arbeitet im Kontext von *example.com*, muss aber Cookies im Kontext von *server.com* ablegen, damit es Anwender, über mehrere Domains hinweg, verfolgen kann.

Um das zu erreichen, erzeugt mein Datenerhebungsscript im *Document Object Model* von *example.com* ein unsichtbares *IFrame* und lädt darin eine speziell präparierte HTML-Seite von *server.com*. Diese spezielle Seite verfügt selbst über ein JavaScript, das nun Cookies im Kontext von *server.com* anlegen und so den alten Fingerprint im Browser speichern kann. Verändert sich nun der Browser-Fingerprint, kann *server.com* den alten Fingerprint aus dem Cookie auslesen und die, unter dem alten Fingerprint gespeicherten Daten auf den neuen Fingerprint umschreiben.

Tatsächlich ist das Problem noch ein ganzes Stück komplexer. Für die Kommunikation zwischen der Seite und dem IFrame übergebe ich Daten über das `window.name`-Objekt<sup>b</sup> und für die Datenablage verwende ich alle erdenklichen Speichermöglichkeiten (*local storage*, *flash cookies*, etc.), wie es das *evercookie* (Kamkar 2010) vormacht. Löscht der Anwender nicht sämtliche Speicherorte auf

## Kick-Off Survey

**1**

Please give us the first  
4 characters of your ID: **hD84** . . . . .

**2**

How much experience do you have with the shell and the following languages?

	No Experience	Basic Skills, Few Experience	Ordinary Experience	Much Experience	Extraordinary Experience
Shell	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
C	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
C++	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

ABBILDUNG 3.15: Eintragung der ID im Feedback-Zettel

einmal, stellt *evercookie* sicher, dass die gespeicherten Informationen wieder mit allen Speicherorten synchronisiert werden. Der Anwender kann die, in seinem Browser gespeicherten Informationen praktisch nicht löschen<sup>c</sup>.

- a <http://m.heise.de/newstickermeldung/Fingerprinting-Viele-Browser-sind-ohne-Cookies-identifizierbar-1982976.html>
- b <https://panopticlick.eff.org/browser-uniqueness.pdf>
- c [http://www.heise.de/newstickermeldung/User-Tracking-Werbefirmen-setzen-bereits-haeufig-nicht-loeschbare-Cookie-Nachfolger-ein-2264381.html?wt\\_mc=rss.ho.beitrag.atom](http://www.heise.de/newstickermeldung/User-Tracking-Werbefirmen-setzen-bereits-haeufig-nicht-loeschbare-Cookie-Nachfolger-ein-2264381.html?wt_mc=rss.ho.beitrag.atom)

### 3.3.5.2 Durchführung und Fazit

Aktivitäten wurden auf den Webseiten für die Online-Dokumentation und für die Tutorials protokolliert.

Die Aufzeichnung der Programmierfortschritte fand bei allen Workshops ('11, '12 und '13), sowie bei zwei von drei PMSB-Praktika ('12, '13) statt. Des Weiteren konnte ich einen Langzeitprobanden gewinnen. Dieser Datensatz umfasst mehr als 1.200 Entwicklungsschritte. Insgesamt liegen Datensätze zu über 50 Probanden vor.

Zur Analyse der Datensätze kam die GTM zum Einsatz. Wegen des hohen Aufwand zur Analyse dieser Daten, wurden nur wenige Datensätze betrachtet. Die Ergebnisse werden in den Abschnitten 3.5 und 4 vorgestellt.

Die hier präsentierte Datenerhebungsmethode unterscheidet sich von den, üblicherweise gebräuchlichen Videoaufzeichnungen (vgl. Clarke 2005b; Grill et al. 2012; LaToza et al. 2007; Piccioni et al.). Es arbeitet Plattform- und Entwicklungsumgebung-übergreifend, erfordert keine Anpassungen am Arbeitsplatz der Probanden und eignet sich für Langzeitstudien, was die Beobachtung von Usability-Problemen erlaubt, die nicht in einem einzelnen Messpunkt zu finden sind oder erst nach einem längeren Gebrauch

einer API auftreten. Technisch gesehen, könnte dieses Verfahren sogar vollkommen transparent, automatisiert und ohne jedes anwenderseitige Zutun eingesetzt werden.

Vergleichbare, manuell einzurichtende Datenerhebungen haben LaToza et al. (2007); Layman et al. (2008) implementiert. Dabei wurde jedoch Eclipse instrumentalisiert, was andere Entwicklungsumgebungen ausschließt. Der Vorteil besteht allerdings darin, dass mit einem Eclipse-Plugin auch feingranularere Ereignisse erfasst werden können. Browserzugriffe werden hingegen nicht von deren Datenerhebungen erfasst.

Die Datenerhebungsmethode wurde im Verlauf dieser Studie mehrfach verbessert. Während der ersten Datenerfassung wurden Schwächen deutlich, die beseitigt werden mussten. Diese Behebungen waren teilweise sehr aufwändig, da die bereits erhobenen Daten immer wieder auf aktualisierte Datenformate umgestellt werden mussten. Die folgende Auflistung nennt die wichtigsten Verbesserungen:

- Manche Anwender verwendeten mehrere SeqAn-Installationen oder mehrere Entwicklungsumgebungen. Die entsprechende Unterstützung musste implementiert werden, um unterschiedliche Programmentwicklungsverläufe unterscheiden zu können.
- Bei den ersten Datenerhebungen wurde die Zeitzone nicht mit erfasst. Dies führte bei einem Teilnehmer zu Problemen, da der Server eine andere Zeitzone verwendete und damit die verschiedenen Datenquellen zeitlich verschoben waren. Die Datenerhebung wurde um eine entsprechende Zeitzonen-Unterstützung ergänzt.
- Ursprünglich wurden nicht die geänderten Dateien, sondern nur das Dateien-Diff<sup>36</sup> übertragen. Bei fehlerhaften Datenerhebungen führte das dazu, dass die späteren Dateistände nur noch händisch oder gar nicht mehr rekonstruiert werden konnten.
- Fehler bei der Datenerhebung wurden nicht protokolliert. Eine entsprechende Protokollierung wurde implementiert. Auftretende Fehler werden nun an eine zuvor hinterlegte E-Mail-Adresse weitergeleitet.
- Die große Bandbreite an möglichen Konfigurationen auf Probandenseite, machte automatisierte Tests notwendig. Dieses Tests umfassen alle hier vorgestellten Datenquellen. Dabei kam das Code-Coverage-Tool *EclEmma*<sup>37</sup> zum Einsatz. Für das Testen der verschiedenen Browser verwendete ich *Selenium*<sup>38</sup>.
- Die Unterstützung von SSL-basierten Webseiten wurde implementiert, weil einige Webseiten sowohl über HTTP, als auch über HTTPS verfügbar sind.
- Die Datenerhebung wurde — wo es möglich war — parallelisiert und asynchron implementiert.
- Die Webseiten-Ereignisse *FOCUS* und *BLUR* wurden hinzugefügt, um besser nachvollziehen zu können, ob und welche Webseite bzw. Browser-Tab gerade gelesen wird.

36 <https://www.gnu.org/software/diffutils/>

37 <http://www.eclemma.org>

38 <http://www.seleniumhq.org>

Ein Problem konnte nicht gelöst werden: Der Erfolg oder Misserfolg von Kompilierversuchen konnte nicht protokolliert werden. Dies ist ohne größeren Aufwand nicht Plattform- und Entwicklungsumgebung-übergreifend möglich. Eine Option besteht darin, die verschiedenen Dateistände nachträglich zu kompilieren. Eine zuverlässige Rekonstruktion würde jedoch die Bereitstellung der entsprechenden Arbeitsumgebungskonfigurationen erfordern, was seine eigenen Probleme mit sich bringt.

### 3.3.6 Zusammenfassung

In diesem Abschnitt habe ich ein umfassendes Datenerhebungsverfahren vorgestellt.

Durch die Erfassung von Programmierfortschritten erlaubt dieses Datenerhebungsverfahren, im Gegensatz zu fast allen mir bekannten Arbeiten, Langzeitstudien durchzuführen. Die erhobenen Daten werden nicht durch die Erzwingung der Arbeit an einem, für eine Datenerhebung vorbereiteten Arbeitsplatz verfälscht. Darüber hinaus wird die Datenerhebung durch das einfache Anlegen bzw. Löschen einer Datei aktiviert bzw. deaktiviert und der Anwender nicht abgelenkt.

Die objektiven Daten werden durch subjektive Daten ideal ergänzt, da sie teilweise verschiedenartige Usability-Probleme beherbergen. Subjektive Daten können außerdem Orientierung für die Analyse der objektiven Daten geben. Sie werden mit Hilfe einer Gruppendiskussion und dem hier vorgestellten Cognitive-Dimensions-Fragebogen erhoben.

Das Datenerhebungsverfahren wird meinen speziellen Anforderungen gerecht. Durch die nicht beeinflussbaren Datenerhebungsmöglichkeiten, war eine besonders hohe Reichhaltigkeit vonnöten, denn weitere Datenerhebungen im Sinne des *theoretischen Samplings* waren nicht ohne weiteres möglich. Dieser Ansatz ist tolerabel, denn einerseits umfassen die Daten bereits verschiedene Datenquellen, was dem Triangulierungsgütekriterium entgegenkommt und andererseits erlaubt die Reichhaltigkeit der Daten weitere Analysen unter einem anderen Betrachtungswinkel.

Die Teilnehmer der Workshops und PMSB-Praktika repräsentieren hinreichend die SeqAn-Anwendergruppe.

Die verschiedenen Datenerhebungen wurden bei den folgenden Veranstaltungen wie folgt durchgeführt:

#### Workshops

**Workshop'11** 13.-15.09.2011

**Programmierfortschritte** von 13 Teilnehmern

**Workshop'12** 04.-06.09.2012

**Gruppendiskussion** von rund 20 Teilnehmern

**Programmierfortschritte** von 14 Teilnehmern

**Workshop'13** 17.-19.09.2013

**Cognitive-Dimensions-Fragebogen** von 10 Teilnehmer

**Programmierfortschritte** von 16 Teilnehmern

## PMSB

**PMSB'12 05.2012**

**Programmierfortschritte** von 6 Teilnehmern

**PMSB'13 04.2013**

**Fragebögen** von xx Teilnehmer

**Programmierfortschritte** von 10 Teilnehmern

**PMSB'14 04.2014**

**Programmierfortschritte** nicht erhoben

## Langzeitprobanden

**Programmierfortschritte** von einem Teilnehmer

Da nun die Daten erhoben werden konnten, dürfte der Analyse Selbiger mit Hilfe der GTM nichts im Weg stehen. Allerdings unterstützte kein gängiges Datenanalysewerkzeug die qualitative Analyse dieser hoch-strukturierten Daten. Aus diesem Grund musste ich selbst ein Werkzeug entwickeln, das ich im folgenden Abschnitt vorstelle.

In Phase 4 (Abschnitt 3.5) werde ich schließlich die eigentliche Forschung mit Hilfe der GTM und meinem Datenanalysewerkzeug besprechen.



## PHASE 3: ENTWICKLUNG DES API USABILITY ANALYZERS

Der *API Usability Analyzer (APIUA)* (siehe Abbildungen 3.16 und 3.17) ist das Werkzeug, mit dessen Hilfe ich die von mir erhobenen und im vorangegangenen Abschnitt beschriebenen Daten analysiert und die im nächsten Kapitel vorgestellten Forschungsergebnisse erarbeitet habe.

Dabei war das Werkzeug zunächst als reine Visualisierungslösung für die spezielle Form der *Programmierfortschritte*-Datenquelle gedacht. Erst durch die Notwendigkeit, die GTM direkt dieser Software anwenden zu können, kam sukzessive die Unterstützung der verschiedenen Aktivitäten dieser Forschungsmethode, wie die verschiedenen Kodierungsarten, hinzu. Eine Einführung in die GTM findet sich im Abschnitt 1.4. Die Entwicklung von APIUA im zeitlichen Verlauf ist im Abschnitt 3.1.3 dargestellt.

Der API Usability Analyzer besteht aus den folgenden drei Komponenten:

**API Usability Analyzer** wird vom Forscher für die eigentliche Forschungsarbeit verwendet.<sup>39</sup>

**API Usability Analyzer Client** läuft auf den Arbeitsplätzen der Probanden und ist für die klientenseitige Datenerhebung zuständig. Diese Komponente wird genauer online<sup>40</sup> und im Abschnitt 3.3.5.1 beschrieben.

**API Usability Analyzer Server** sammelt die durch Clients erhobenen objektiven Rohdaten, die dann, mit Hilfe des Datenanalysewerkzeugs, verarbeitet werden können. Diese Komponente wird genauer online<sup>41</sup> und im Abschnitt 3.3.5.1 beschrieben.

Dieser Abschnitt beschränkt sich auf das von mir entwickelte Datenanalysewerkzeug selbst. Anhang A ergänzt diese Beschreibung, durch eine nähere Erläuterung einzelner Komponenten.



ABBILDUNG 3.16: Der Startbildschirm von APIUA

39 <https://github.com/bkahlert/api-usability-analyzer>

40 <https://github.com/bkahlert/api-usability-analyzer-client-python>

41 <https://github.com/bkahlert/api-usability-analyzer-server-java-ee>

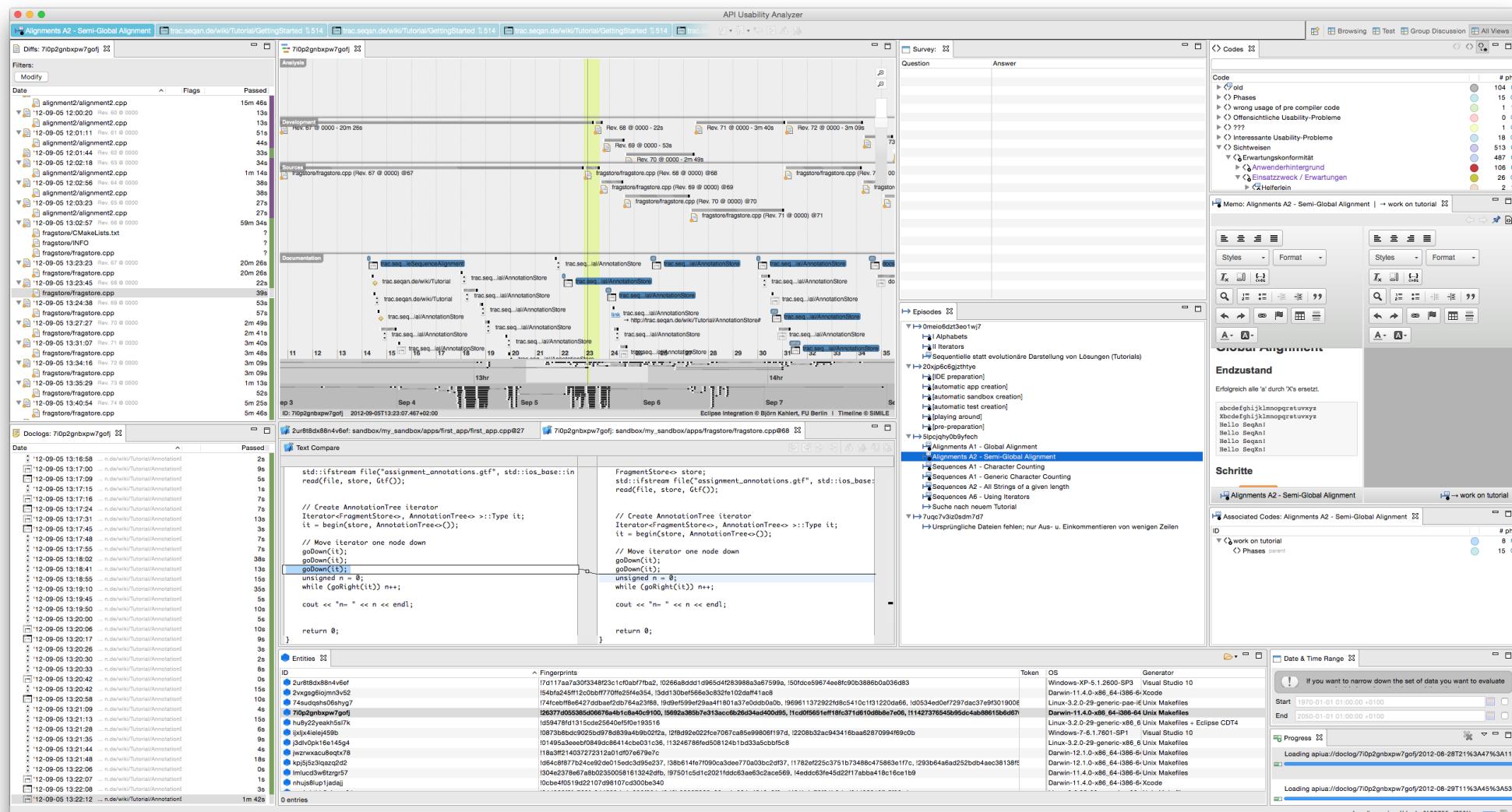


ABBILDUNG 3.17: Dieser Screenshot von APIUA zeigt eine typische Sitzung mit Fokus auf offenes Kodieren. In diesem Fall werden Programmierfortschritte des Probanden mit der ID **7i0p2gnbxpw7gofj** analysiert.

### 3.4.1 Motivation

Je umfassender ein zu analysierendes Datenmaterial ist, desto mehr zwängt sich ein computergestütztes Verfahren auf. Auf dem Markt gibt es verschiedene Anwendungen, die sich grundsätzlich für die Verwendung der GTM eignen. Zu den führenden CAQDAS<sup>42</sup>-Vertretern gehören *MAXQDA*<sup>43</sup> und *ATLAS.ti*<sup>44</sup> (Lewins u. Silver 2007). Beide Programme dienen dem Zweck der qualitativen Datenanalyse und eignen sich für die GTM (Dresing 2006; Lewins u. Silver 2007; Wolf 2014).

Bedauerlicherweise haben alle großen CAQDAS-Vertreter nicht zu vernachlässigende Schwächen in Bezug auf ihren Einsatz als GTM-Software (Lewins u. Silver 2007). Axiales Kodieren wird von *MAXQDA* überhaupt nicht (Wolf 2014) und von anderen Werkzeugen nur im Umfang einer Zeichenfläche unterstützt (Lewins u. Silver 2007). Im letzteren Fall verlieren, auf eine Zeichenfläche positionierte Kodes sogar die Information, woher sie kommen und werden — im Falle von Kode-Änderungen — nicht mehr aktualisiert. Einzig *ATLAS.ti* speichert die Referenz auf eine verwendete Relation in ihrem Netzwerk-Editor (Lewins u. Silver 2007). Beziiglich *ATLAS.ti* schildern meine Arbeitsgruppenkollegen Schenk (2014) und Zieris (2014) die folgenden Probleme bei ihrer Forschungsarbeit:

#### Funktionale Probleme

- Offenes Kodieren wird erschwert, weil Kodes nur sehr eingeschränkt<sup>45</sup> gruppiert und gar nicht sortiert werden können. Zudem sind Referenzen innerhalb von Memos nicht möglich. Im Zusammenhang mit den unten genannten Defekten resümiert Zieris (2014), dass es “kein[en] Überblick beim Open Coding” gibt. Weiterhin fehlt die Möglichkeit, zwei Akteure separat zu kodieren, was den Kodierprozess bei der Analyse von Paarprogrammierungssitzungen verkompliziert. Anfang und Ende einer Quotation können nicht verändert werden.
- Axiales Kodieren ist nur eingeschränkt möglich, da es für jeden Kode maximal eine axiale Kodierung bzw. ein axiales Kodiermodell geben darf. Eine Übersicht über alle existierenden axialen Kodierungen und axialen Kodiermodellen gibt es nicht. Die Bedienung des Editors selbst gestaltet sich schwierig. So ist es nicht möglich, als Pfeilverbindungen dargestellte Relationen auszublenden oder auf eine andere Weise in ihrer Gestalt zu verändern. Die beiden Befragten nutzen aus diesen Gründen die entsprechende Funktionalität von *ATLAS.ti* nicht bzw. nur in seltenen Fällen und weichen auf andere Programme aus (*OmniGraffle*<sup>46</sup> bzw. *Microsoft Visio*<sup>47</sup>). In *ATLAS.ti* gibt es die Möglichkeit Kodes zu verankern. Diese Möglichkeit besteht aber nicht für Relationen zwischen Kodes.

#### Bedienschwierigkeiten

42 Computer-unterstützte qualitative Datenanalysesoftware  
(computer assisted/aided qualitative data analysis software)

43 <http://www.MAXQDA.de>

44 <http://atlasti.com/de/>

45 Mit Hilfe von Superkodes, Familien und Superfamilien gibt es eine Möglichkeit der Gruppierung (Mühlmeyer-Mentzel 2011), die aber von den beiden Kollegen wegen ihrer schwierigen Handhabung gemieden wird. Beispielsweise können Familien nicht während der axialen Kodierung verwendet werden.

46 <https://www.omnigroup.com/omnigraffle>

47 <http://products.office.com/de-de/visio>

- Es ist schwer, den Fokus auf das gerade betrachtete Datum zu behalten, denn verschiedene Operationen führen dazu, dass sich unbeabsichtigte Änderungen der Ansicht ergeben. Dazu gehört beispielsweise das bloße Klicken auf eine Quotation, das die Zeitleiste springen lässt.
- Die Arbeit ist schwerfällig, weil viele Ansichten exklusiv sind und der Wechsel zwischen ihnen “zeitraubend” ist. Dies behindert insbesondere das *ständige Vergleichen*. Beispiel: Entweder kodiert man sein Videomaterial oder man interessiert sich für die Eigenschaften eines Codes. Die dazu jeweils notwendigen Ansichten können nicht parallel geöffnet sein.
- Die Suchfunktion wird nur eingeschränkt wegen ihrer komplizierten Syntax genutzt.
- Der Zustand der Arbeitsumgebung wird nicht vollständig gespeichert. Nach einem Neustart der Anwendung muss — bevor die Arbeit fortgesetzt werden kann — der alte Zustand wiederhergestellt werden.

### Defekte / Sonstiges

- Die Identifikatoren von Kodes werden von *ATLAS.ti* wiederverwendet. Wird ein Kode gelöscht und ein neuer Kode erzeugt, kann es passieren, dass der neue Kode den Identifikator des alten Kodes erhält. Verweise auf den alten Kode zeigen dann fälschlicherweise auf den neuen Kode.
- Existieren auf einem Abschnitt auf der Zeitliste zu viele Quotations, werden diese nicht mehr vollständig sichtbar aufgezählt sondern abgeschnitten.

Die Verwendung eines der existierenden Produkte empfiehlt sich auch aus einem weiteren wichtigen Grund nicht: Die im Rahmen dieser Arbeit erhobenen Daten sind hochstrukturiert. Ihr Informationsgehalt ist für einen Menschen nur schwer zu erfassen. Warum das so ist, sollen die beiden folgenden Beispiele veranschaulichen.

### 3.4.2 Beispiel: Diff-Dateien

Eine Diff-Datei beschreibt sämtliche vom Probanden gemachten Dateiänderungen zwischen zwei Komplierversuchen innerhalb des SeqAn-Verzeichnisses.

Oder einfacher ausgedrückt: Wenn ein Proband in der Datei `first_app.cpp` eine Zeile Code hinzufügt und anschließend in seiner Entwicklungsumgebung die Anwendung ausführen will, wird die folgende Datei aufgezeichnet:

```

1 diff -u -r -N -x '*.o' -x Thumbs.db -x .DS_Store -x CMakeCache.txt -x
   → misc/seqan_instrumentation/userdata/id.txt -x
   → C:/Software/SeqAn/seqan/misc/seqan_instrumentation/userdata/id.txt -x
   → misc/seqan_instrumentation/userdata/2ur8t8dx88n4v6ef_stats.txt -x
   → C:/Software/SeqAn/seqan/misc/seqan_instrumentation/userdata/
   → 2ur8t8dx88n4v6ef_stats.txt -x .svn -x bin -x build -x util -x misc -x docs -x
   → docs2 -x extras -x core -x misc/seqan_instrumentation/bin -x
   → C:/Software/SeqAn/seqan/misc/seqan_instrumentation/bin -x
   → misc/seqan_instrumentation/last_revision_copy -x
   → C:/Software/SeqAn/seqan/misc/seqan_instrumentation/last_revision_copy -x
   → misc/seqan_instrumentation/last_revision_copy -x
   → C:/Software/SeqAn/seqan/misc/seqan_instrumentation/last_revision_copy -x
   → misc/seqan_instrumentation/userdata -x
   → C:/Software/SeqAn/seqan/misc/seqan_instrumentation/userdata -x
   → misc/seqan_instrumentation/userdata -x
   → C:/Software/SeqAn/seqan/misc/seqan_instrumentation/userdata
   → ./misc/seqan_instrumentation/last_revision_copy/sandbox/my_sandbox/
   → apps/first_app/first_app.cpp ./sandbox/my_sandbox/apps/first_app/first_app.cpp
2 --- ./misc/seqan_instrumentation/last_revision_copy/sandbox/my_sandbox/
   → apps/first_app/first_app.cpp 2012-09-04 13:32:34.281250000 +0200
3 +++ ./sandbox/my_sandbox/apps/first_app/first_app.cpp 2012-09-04
   → 13:33:05.968750000 +0200
4 @@ -13,6 +13,8 @@
5
6         readRecord(id, seq, seqStream);
7
8 +
9 +     std::cout << id << '\t' << seq << '\n';
10 +
11     seqan::CharString mySeqanString = "Done.";
12     std::cout << mySeqanString << std::endl;
13     return 1;

```

LISTUNG 9: Einfache Diff-Datei, die zwei hinzugefügte Code-Zeilen dokumentiert

Welche Informationen können wir der Diff-Datei 2ur8t8dx88n4v6ef\_2b2f\_2012-09-04T13-33-07+0200.diff entnehmen?

Information	Fundort	Wert
ID des Probanden	Diff-Dateiname	2ur8t8dx88n4v6ef
Hash der Entwicklungsumgebung	Diff-Dateiname	2b2f
Zeitpunkt des Kompilierversuchs	Diff-Dateiname	04.09.2012, 13:33:07
Zeitpunkt der letzten Änderung an <code>first_app.cpp</code>	Zeile 3	04.09.2012, 13:33:06
Für die Änderung beanspruchte Zeit	Differenz Zeilen 2 & 3	rund 32s
Code-Änderungen	Zeilen 8-9	Ausgabe und Leerzeile
Kompiliererfolg	Infrastruktur nachstellen und selbst kompilieren	Die Datei kompiliert erfolgreich.

TABELLE 3.1: In einer Diff-Datei enthaltene Informationen

Tabelle 3.1 beschreibt, welcher Informationsgehalt in einer Diff-Datei steckt. Die Tabelle und die Diff-Datei veranschaulichen aber auch, wie schwer diese Informationen, bereits bei einem kleinen Beispiel, fehlerfrei zu extrahieren sind.

Damit nicht genug: Die Information *für die Änderung beanspruchte Zeit* ist falsch berechnet. Richtig wäre es, anstelle der vorletzten Bearbeitung von `first_app.cpp` den vorangegangen Kompilierversuch zu verwenden. Es soll nicht erfasst werden, wie viel Zeit die letzte Bearbeitung einer Datei zurücklag, sondern wie viel Zeit der Anwender (vermutlich) innerhalb einer Iteration auf eine Datei verwendete. Diese Zeitspanne ist also maximal so lang, wie die Iteration selbst. Nie länger.

Für die korrekte Berechnung dieses Maßes wird also neben der betrachteten Diff-Datei auch ihr Vorgänger benötigt. Ähnlich verhält es sich mit dem Umstand, dass eine Diff-Datei nur die geänderten und deren Nachbarzeilen enthält. Möchte man den gesamten Inhalt zum Zeitpunkt des Kompilierversuchs wissen, muss man die lückenlose Historie aller Änderungen betrachten. Auch dann ist noch nichts darüber gesagt, ob die Datei tatsächlich kompiliert und wenn nicht, welcher Grund dafür verantwortlich ist.<sup>48</sup>

Noch unerwähnt blieben Konstellationen, die potentiell selten auftreten, dann aber besonders relevant sind, leicht übersehen werden und zu Fehlinterpretationen führen können. In einem Fall hatte eine vom Probanden veränderte Datei ein Modifikationsdatum, das weit zurücklag — weiter als das zuvor dokumentierte Datum. Ein Vergleich aller vorangegangenen Dateiänderungen zeigte, dass der Proband eine alte Version der Datei wiederhergestellt hatte. Eine Operation, die man durch das bloße Betrachten des Dateiinhalts wahrscheinlich nicht erkannt hätte.

<sup>48</sup> Inzwischen werden zur Datenerhebung die vollständigen, geänderten Dateien und nicht mehr deren Diffs übertragen. Dennoch veranschaulicht dieses Beispiel, wie komplex das Lesen von Rohdaten ohne Werkzeugunterstützung sein kann.

The screenshot shows the APIUA software interface. On the left, there's a tree view of file differences with columns for Date, Flags, and Passed? (with a question mark icon). The right side shows a text editor with two panes. The left pane contains the original code, and the right pane shows the modified code with specific changes highlighted in blue.

```

#include <iostream>
#include <seqan/sequence.h> // CharString, ...
#include <seqan/file.h> // to stream a CharString into cout

#include <seqan/seq.io.h>

int main(int, char const **)
{
    seqan::SequenceStream seqStream("C:\\temp\\example.fasta");

    seqan::CharString id;
    seqan::DnaString seq;
    readRecord(id, seq, seqStream);

    seqan::CharString mySeqanString = "Done.";
    std::cout << mySeqanString << std::endl;
    return 1;
}

```

ABBILDUNG 3.18: Dieser APIUA-Bildschirmausschnitt zeigt den Probanden 2ur8t8dx88n4v6ef. Zu sehen sind im linken Drittel die ID des Probanden (im Tab), seine Kompilierversuche (Knoten erster Ordnung), alle bearbeiteten Dateien (Kindknoten), deren Kompiliererfolg (grüne, gelbe oder rote Icon-Annotation) und die auf sie verwendete Zeit.

Im rechten Teil des Ausschnitts kann man die aktuell betrachtete Datei vollständig in ihrem vorangegangenen und aktuellen Zustand sehen. Die Codeänderungen sind grafisch hervorgehoben.

Zu den allgemein anerkannten Grundprinzipien / Praktiken der GTM gehört das *ständige Vergleichen* (Corbin u. Strauss 2014; Glaser u. Strauss 1967; Salinger 2013; Strauss u. Corbin 1990)<sup>49</sup>. Es ist zu befürchten, dass die GTM-Praktiken — durch eine aufwändige, händische und damit fehlerträchtige Datenaufbereitung — weniger sorgfältig angewendet werden und so die Qualität der GT sinkt.

Durch technische Unterstützung können alle genannten Informationen automatisch aus einem Datum extrahiert und so aufbereitet werden, dass sie durch den Forscher einfacher verarbeitet werden können. Abbildung 3.18 zeigt, wie alle, in Tabelle 3.1 aufgelisteten Informationen visuell in APIUA dargestellt werden.

### 3.4.3 Beispiel: Doclog-Datei

Eine Doclog<sup>50</sup>-Datei dokumentiert alle Zugriffe/Ereignisse auf die, im Internet bereitgestellte SeqAn-Dokumentation. Diese Dokumentation besteht hauptsächlich aus der Dokumentation selbst und den Tutorials.

Die Erfassung der Dokumentationszugriffe soll schwer nachvollziehbare Entwicklungsschritte zwischen zwei Kompilierversuchen vereinfachen. So können größere Code-Einfügungen bedeuten, dass dem Probanden ein Licht aufgegangen ist. Sie können aber auch bedeuten, dass der eingefügte Code schlicht von der Online-Dokumentation kopiert wurde. Letztere Interpretation kann aber nur zuverlässig vollzogen werden, wenn man dafür trifftige Indizien hat.

<sup>49</sup> Der Vollständigkeit halber sei erwähnt, dass Mey u. Mruck (2007) eine “Akzentverlagerung” von der “constant comparison method” hin zur “Dimensionalisierung” bei Strauss (und Corbin) beobachten.

<sup>50</sup> documentation log

---

```

481 2012-09-04T14:14:23.729+02:00 READY http://trac.seqan.de/wiki/Tutorial/
    ↳ GettingStarted 160.45.233.238 - 0 0 1350 612
482 2012-09-04T14:14:26.774+02:00 SCROLL http://trac.seqan.de/wiki/Tutorial/
    ↳ GettingStarted 160.45.233.238 - 0 228 1350 612
483 2012-09-04T14:14:27.463+02:00 LINK-http://trac.seqan.de/wiki/Tutorial/
    ↳ GettingStarted/WindowsVisualStudio http://trac.seqan.de/wiki/Tutorial/
    ↳ GettingStarted 160.45.233.238 - 0 228 1350 612
484 2012-09-04T14:14:27.784+02:00 UNLOAD http://trac.seqan.de/wiki/Tutorial/
    ↳ GettingStarted 160.45.233.238 - 0 228 1350 612
485 2012-09-04T14:14:28.804+02:00 READY http://trac.seqan.de/wiki/Tutorial/
    ↳ GettingStarted/WindowsVisualStudio 160.45.233.238 - 0 147 1350 612
486 2012-09-04T14:14:35.196+02:00 SCROLL http://trac.seqan.de/wiki/Tutorial/
    ↳ GettingStarted/WindowsVisualStudio 160.45.233.238 - 0 3420 1350 612
487 2012-09-04T14:14:49.711+02:00 SCROLL http://trac.seqan.de/wiki/Tutorial/
    ↳ GettingStarted/WindowsVisualStudio 160.45.233.238 - 0 4104 1350 612
488 2012-09-04T14:14:53.414+02:00 SCROLL http://trac.seqan.de/wiki/Tutorial/
    ↳ GettingStarted/WindowsVisualStudio 160.45.233.238 - 0 3876 1350 612
489 2012-09-04T14:15:00.302+02:00 SCROLL http://trac.seqan.de/wiki/Tutorial/
    ↳ GettingStarted/WindowsVisualStudio 160.45.233.238 - 0 3648 1350 612
490 2012-09-04T14:15:06.149+02:00 BLUR http://trac.seqan.de/wiki/Tutorial/
    ↳ GettingStarted/WindowsVisualStudio 160.45.233.238 - 0 3648 1350 612
491 2012-09-04T14:15:21.071+02:00 FOCUS http://trac.seqan.de/wiki/Tutorial/
    ↳ GettingStarted/WindowsVisualStudio 160.45.233.238 - 0 3648 1350 612

```

LISTUNG 10: Auszug aus einer Doclog-Datei

Die Doclog-Datei ist eine Tabulator-separierte Liste. Ihre Spalten sind:

**DateTime** Zeitstempel in ISO8601

**Action** Ereignis (READY, UNLOAD, FOCUS, LINK, SCROLL, etc.)

**URI** Adresse der geöffneten Seite

**IP** IP des Anwenders

**ProxyIP** Proxy-IP des vom Anwender verwendeten HTTP-Proxies

**ScrollX** Scrollposition X-Achse in Pixeln (px) vom oberen Rand des Dokuments

**ScrollY** Scrollposition Y-Achse in Pixeln (px) vom linken Rand des Dokuments

**Width** Breite des Browser-Ansichtsbereichs (Viewport) in Pixeln (px)

**Height** Höhe des Browser-Ansichtsbereichs (Viewport) in Pixeln (px)

Die in der Doclog-Datei dokumentierte Nutzung der Online-Dokumentation durch den Probanden zeigt also, dass er das Tutorial *Getting Started* um 14:14 Uhr öffnete (Zeile 1) und anschließend vertikal um 228px nach unten scrollte (Zeile 2). Zeile 3 zeigt, dass er den Link auf die SeqAn-Installationsanleitung für Visual Studio öffnete und dabei die, eben noch geöffnete Seite schloss (Zeile 4). Anschließend gab es mehrere Scrollaktionen auf der eben geöffneten Seite. In Zeile 10 verlor die Seite den Fokus, um ihn in Zeile 11, etwa 15s später, wiederzuerlangen.

Mit ein wenig Übung kann man das Protokoll gut lesen, allerdings kaum gewinnbringend verwenden. Die Daten wurden dokumentiert, um den Gedankengang des Probanden bei seiner Arbeit zwischen zwei Kompilierversuchen besser nachvollziehen zu können. Die Zeitstempel der Kompilierversuche und die der Aktionen auf der Online-Dokumentation müssen also verknüpft werden. Darüber hinaus ist noch nichts darüber gesagt, was genau der Anwender auf der Online-Dokumentation gesehen hat. Dazu müsste man selbst die Dokumentation öffnen, die Größe des Browsers korrekt anpassen und an dieselbe Position scrollen. Spätestens, wenn eine Dokumentationsseite durch einen SeqAn-Entwickler verändert wurde, lässt sich die Information nicht oder nur unverhältnismäßig aufwändig wiederbeschaffen.

In APIUA muss nicht das Rohformat vom Forscher analysiert werden, sondern dessen Aufbereitung. Dazu werden intern, für noch nicht prozessierte Doclog-Einträge, Browser-Instanzen erzeugt, die die jeweilige URI laden, den Anzeigebereich auf die korrekte Größe setzen, an die dokumentierte Scrollposition springen, eventuelle Texteingaben vornehmen und einen Screenshot erzeugen. Diese können dann bei der Analyse von Diff-Dateien zu Rate gezogen werden. Abbildung 3.19 zeigt eine geöffnete Datei mit einem schwebenden Fenster in der rechten Bildschirmhälfte, das den Screenshot für die, in diesem Moment vom Probanden geöffneten Seite zeigt.

### 3.4.4 Herausforderungen für ein GTM-Datenanalysewerkzeug

Die beiden Beispiele sollten zeigen, dass

1. hochstrukturierte Daten viele Informationen enthalten,
2. die teilweise nur sehr aufwändig und
3. fehleranfällig extrahiert werden können und
4. damit potentiell die Sorgfalt bei der Anwendung der GTM schmälert.

Die Aufgabe eines GTM-Datenanalysewerkzeugs darf also nicht nur darin bestehen, die Analyse technisch irgendwie zu ermöglichen. Sie besteht auch darin, zeitraubende und fehlerträchtige Routinearbeiten adäquat zu automatisieren und die Phasen und Praktiken der GTM reibungslos zu erlauben. Die durch die GTM ohnehin schon stark geforderte Sorgfalt und Disziplin des Forschers, dürfen durch das Datenanalysewerkzeug nicht unnötig strapaziert werden, da sonst die Qualität der generierten GT gefährdet wäre.

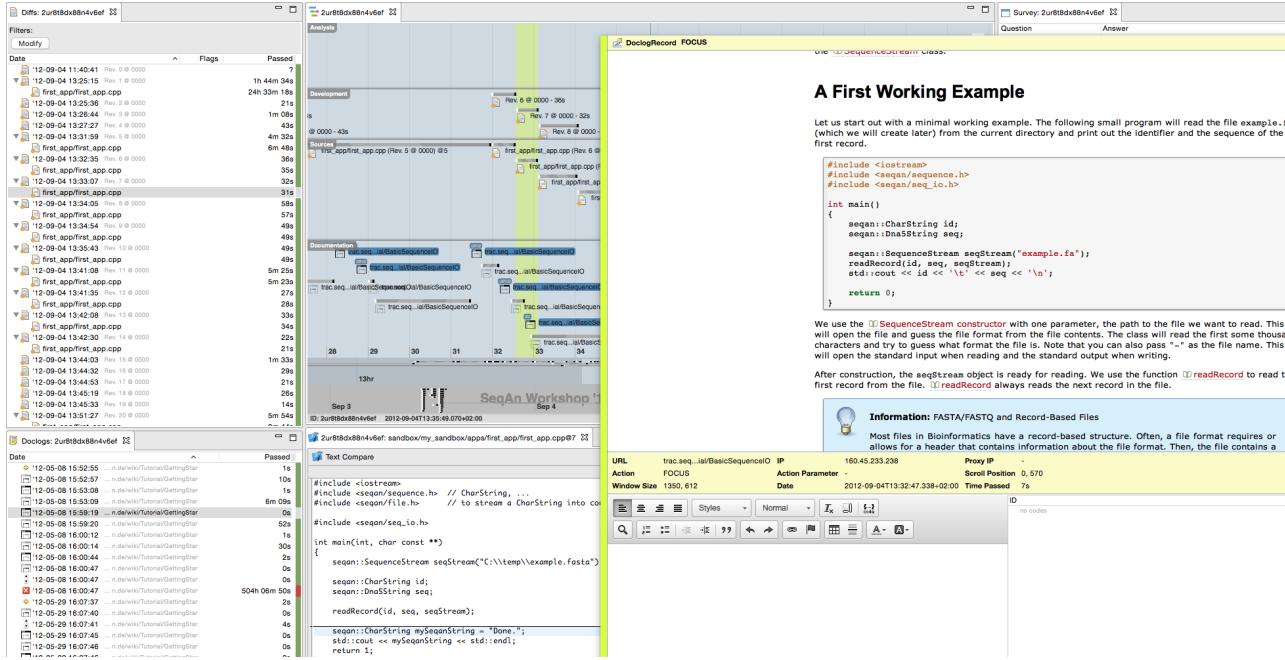


ABBILDUNG 3.19: Dieser APIUA-Bildschirmausschnitt zeigt den Probanden 2ur8t8dx88n4v6ef. Die Beschreibung des linken Bildbereichs kann Abbildung 3.18 entnommen werden. Der blaue Bereich in der Mitte zeigt einen Abschnitt der Zeitleiste, auf der sich sämtliche Datenpunkte zum Probanden befinden. In Neon ist die Zeitspanne hervorgehoben, die zwischen dem vergangenen und dem aktuellen Kompilierversuch liegt. In der rechten Hälfte ist ein schwebender Dialog dargestellt, der Informationen zu dem Element darstellt, über dem gerade die Maus schwebt. Konkret handelt es sich um einen Doclog-Eintrag bestehend aus dem Screenshot, darunter den Metainformationen und der (leeren) Memo.

Als Forscher, der sich selbst in die GTM einarbeitete, sogar das für die eigene Datenanalyse zugehörige Analysewerkzeug selbst entwickelte und im ständigen Austausch mit seiner GTM-affinen Forschungsgruppe<sup>51</sup> stand, konnte ich eine Reihe von, in Tabelle 3.2 zusammengefassten Anforderungen formulieren.

<sup>51</sup> Allein diese Ausrichtung scheint äußerst selten zu sein. Zumindest wenn man sich eine Stichprobe der in führenden Softwaretechnik-Zeitschriften der Jahre 1995-1999 veröffentlichten Artikel ansieht: Demnach wurde die GTM in weniger als 1% der Arbeiten verwendet. (Glass et al. 2002)

Anforderung	Beschreibung	Erfüllung durch APIUA
Große Datenmengen	Die Verarbeitung von mehreren Gigabyte Datenmaterial muss unterstützt sein.	Auf einem durchschnittlichen Computer konnten 24GB Datenmaterial verarbeitet werden.
Datenformate	Videos, Audioaufnahmen, Textdokumente, aber auch etablierte Datenaustauschformate wie XML und JSON müssen unterstützt werden. Die unterstützten Datenformate müssen erweiterbar sein.	Unterstützt werden lediglich die im Rahmen dieser Arbeit erfassten Datenformate. Ein Plugin <sup>G</sup> -Mechanismus erlaubt die Erweiterung um beliebige weitere Datenformate (siehe Abbildung 3.22).
Offenes Kodieren	Die Phase des offenen Kodierens muss optimal (Kategorien, Eigenschaften, etc.) unterstützt werden.	Offenes Kodieren wird umfänglich unterstützt (siehe Abbildung 3.31). Kategorien werden in Form von Kode-Hierarchien ermöglicht (siehe Abbildung 3.23). Eigenschaften werden auf Kode-Ebene definiert; entsprechende Wertebelegungen auf Kodeinstanz-Ebene erlaubt (siehe Abbildung 3.24).
Axiales Kodieren	Die Phase des axialen Kodierens muss optimal (Relationen, etc.) unterstützt werden. Es muss Funktionen geben, die dabei helfen, axiale Kodiermodelle <sup>G</sup> zu abstrahieren.	Axiales Kodieren wird umfänglich unterstützt. Relationen werden auf Kode-Ebene definiert und können verankert werden. Aus den vorhandenen Relationen können automatisch axiale Kodierungen <sup>G</sup> und axiale Kodiermodelle erzeugt werden (siehe Abbildung 3.33). Abstrahierungsfunktionen existieren rudimentär in Form fest implementierter Interferenzregeln.

Fortsetzung auf nächster Seite

Tabelle 3.2 – Fortsetzung

Anforderung	Beschreibung	Erfüllung durch APIUA
Selektives Kodieren	Die Phase des selektiven Kodierens muss stark unterstützt werden. Dazu werden Funktionen benötigt, welche die eine weiter gehende Abstraktion / Generalisierung von Kodes, Relationen und schließlich von axialen Kodiermodellen erlauben.	Selektives Kodieren wird teilweise durch APIUAs Interferenzmöglichkeiten und Verallgemeinerungs- und Zusammenfassungsmöglichkeiten von Relationen unterstützt.
Implizites Wissen	In den von dem Anwender gefunden Verankerungen verbirgt sich in manchen Fällen implizites Wissen. Das Werkzeug sollte derartiges Wissen sichtbar machen. Weitere Details siehe weiter unten.	Implizite Verankerungen und Relationen werden sichtbar gemacht.
Vorwissen	Es muss die Möglichkeit geben Verankerungen, die auf Vorwissen des Forschers beruhen, vorzunehmen.	Vorwissen kann durch die Verwendung von <code>bibtex:-</code> und <code>file:-URI<sup>G</sup></code> s verankert werden.
Umstrukturierungsoperationen <sup>52</sup>	Es muss Möglichkeiten geben, Änderungen an der Modellierung vorzunehmen, die der GT zu Grunde liegt. Beispiele sind Kode-Umbenennung, Verallgemeinerung oder Spezialisierung von Relationen, Verschieben von Kode-Eigenschaften und Zusammenfassung oder Auf trennen von Kodes.	Einige Strukturänderungsoperationen werden unterstützt. Dazu gehören einfache Operationen wie die Umordnung des Kode-Baumes und die Verallgemeinerungen bzw. Zusammenfassung von Relationen (siehe Abbildung 3.20). Außerdem können während der Zuweisung von Phänomenen zu Kodes in einem speziellen Dialog (siehe Abbildung 3.21) direkt Eigenschaftswerte zugewiesen werden.

Fortsetzung auf nächster Seite

<sup>52</sup> Auch wenn man verführt ist, die Bezeichnung *Refactoring* zu verwenden, wäre die von Fowler (1999) formulierte Anforderung verletzt, dass die Semantik bei einer derartigen Operation nicht verändert wird. Eine kanonische Übertragung dieser Eigenschaft auf Theorien ist nicht möglich, da die Grenzen einer Theorie nicht hinreichend scharf sind. Während man sich noch streiten kann, ob einfache Operationen, wie eine Kode-Umbenennung, bereits die Semantik einer Theorie ändern, ist die Frage bei komplexeren Operationen wie die Verallgemeinerung von Relationen eindeutig mit „ja“ zu beantworten.

Tabelle 3.2 – Fortsetzung

Anforderung	Beschreibung	Erfüllung durch APIUA
Weitergehende Analysefunktionen	Beispielsweise könnte eine Funktion, die anzeigen, welche Kodes in welchen Datenquellen/-erhebungen verankert sind, bei der Bewertung der Frage helfen, ob und in welche Richtung eine weitere Datenerhebung gehen kann/soll. Diese Funktion würde also den Forscher beim <i>theoretischen Sampling</i> unterstützen.	Weitere Analysefunktionen wurden aus zeitlichen Gründen nicht implementiert.
Interoperabilität	Die Forschungsergebnisse müssen in einem Format gespeichert werden, das sich zur Weiterverarbeitung durch dritte Programme eignet. Darüber hinaus erlaubt die Verwendung standardisierter Datenformate die Bereitstellung der Forschungsdaten im Rahmen von <i>Open Science</i> — also in diesem Fall, dem freien Zugang zu wissenschaftlichen Ergebnissen.	Die Forschungsergebnisse werden in XML abgelegt. Memos werden in Form von HTML-Dateien abgelegt, deren Name der URI <sup>G</sup> des beschriebenen Datenpunktes entspricht. Axiale Kodiermodelle liegen in Form von JSON-Dateien vor. Datenpunkte werden innerhalb der HTML- und JSON-Dateien einheitlich mit deren URI referenziert, was jede Möglichkeit von technischer Datenredundanz ausschließt.

Fortsetzung auf nächster Seite

Tabelle 3.2 – *Fortsetzung*

<b>Anforderung</b>	<b>Beschreibung</b>	<b>Erfüllung durch APIUA</b>
Usability	Die Usability des Werkzeugs selbst muss hoch sein, um die Qualität der erarbeiteten GT nicht zu schmälern.	Der Zustand der Arbeitsumgebung wird vollständig gesichert, d.h. die Anordnung der verschiedenen Bereiche, deren Anzeigeoptionen, deren Inhalt und viele weitere Informationen, wie zuletzt verwendete Kodes, werden nach einem Neustart der Anwendung wiederhergestellt. Zur Erfüllung einer Aufgabe werden verschiedene Möglichkeiten angeboten. Zuletzt verwendete Elemente (Kodes, Phänomene, etc.) werden in einer Historie festgehalten. Kodes werden automatisch mit sinnvollen Farben versehen.
Plattformabhängigkeit	—	APIUA ist lauffähig unter Windows, Mac OS und Linux.

TABELLE 3.2: Anforderungen an GTM-Datenanalysewerkzeuge

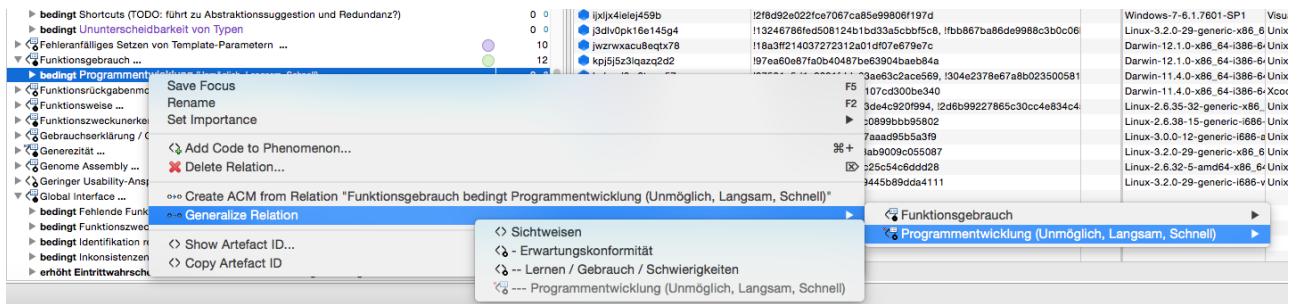


ABBILDUNG 3.20: Dieser Screenshot von APIUA zeigt das die Möglichkeit zur Verallgemeinerung von Relationen.

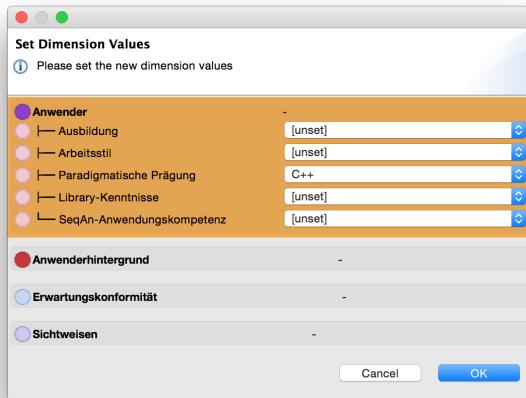


ABBILDUNG 3.21: Dieser Screenshot von APIUA zeigt den Dialog zur Zuweisung von Eigenschaftswerten bei Umstrukturierungen.

Im Abschnitt 3.4.6 auf Seite 212 gehe ich genauer auf einige der Anforderungen und ihrer Erfüllung durch APIUA ein. Der folgende Abschnitt soll zunächst den groben technischen Aufbau von APIUA vorstellen.

### 3.4.5 Entwurf

In diesem Abschnitt werden die wichtigsten APIUA-Entwurfsentscheidungen vorgestellt. Die vollständigen Quellen von APIUA sind unter GitHub<sup>53</sup> veröffentlicht und werden dort gepflegt.

APIUA basiert auf der *RCP*<sup>54</sup> (Version 3), die wiederum die Grundlage für die bekannte Entwicklungsumgebung *Eclipse* darstellt.

Architektonisch verwendet APIUA einen Hybrid aus einer serviceorientierten und einer schichtenbasierten Architektur (siehe Abbildung 3.25).

53 <https://github.com>

54 [http://wiki.eclipse.org/index.php/Rich\\_Client\\_Platform](http://wiki.eclipse.org/index.php/Rich_Client_Platform)

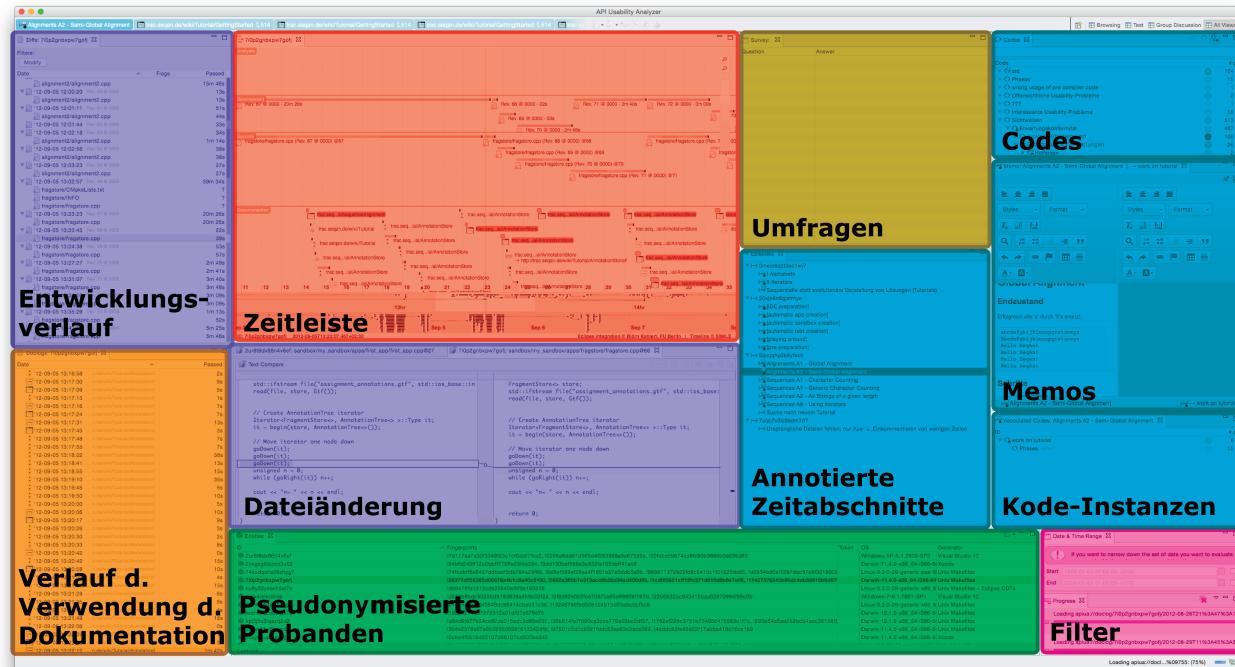


ABBILDUNG 3.22: Dieser Screenshot von APIUA in der Perspektive für offenes Kodieren zeigt die von den verschiedenen Plugins der Schicht 3 beigesteuerten Eclipse-Views.

V.l.n.r.: Violett: Diff-Plugin; Rot: Timeline-Plugin; Gelb: Stats-Plugin; Blau: GTM-Plugin; Orange: Doclog-Plugin; Grün: Entity-Plugin; Pink: Core-Plugin

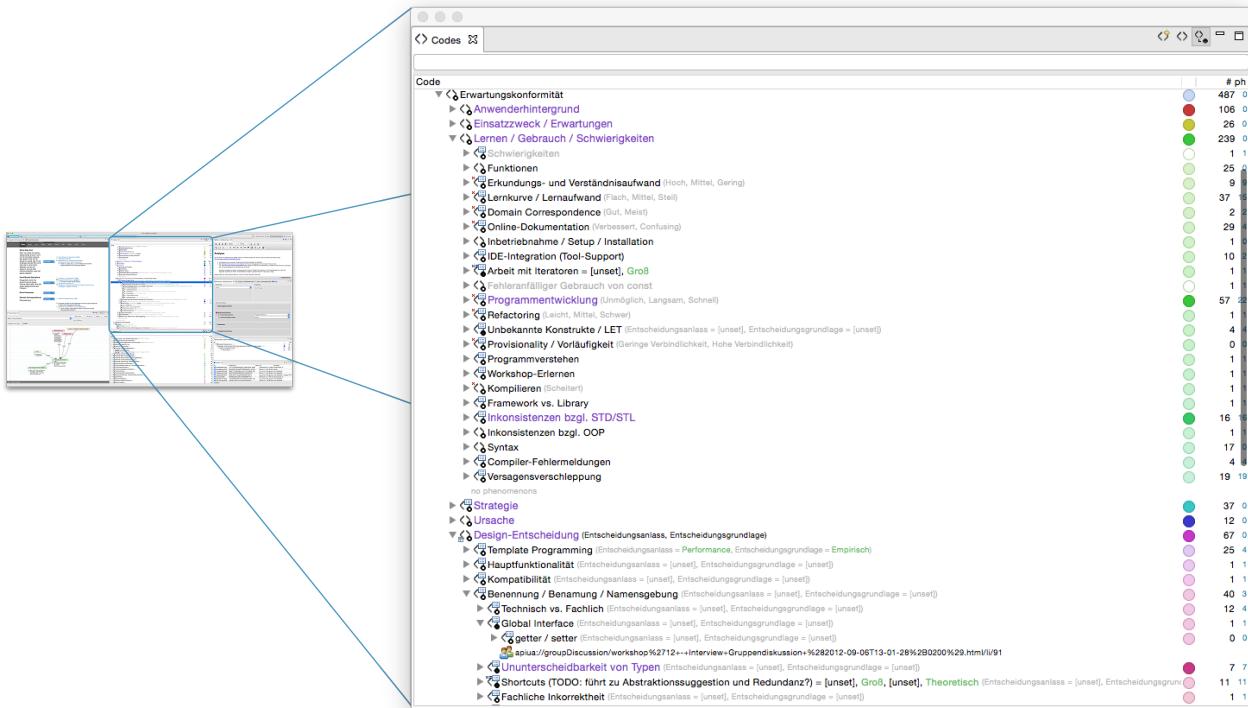


ABBILDUNG 3.23: Dieser Screenshot von APIUA zeigt das Eclipse-View “Codes”, das alle verankerten Kodes darstellt.

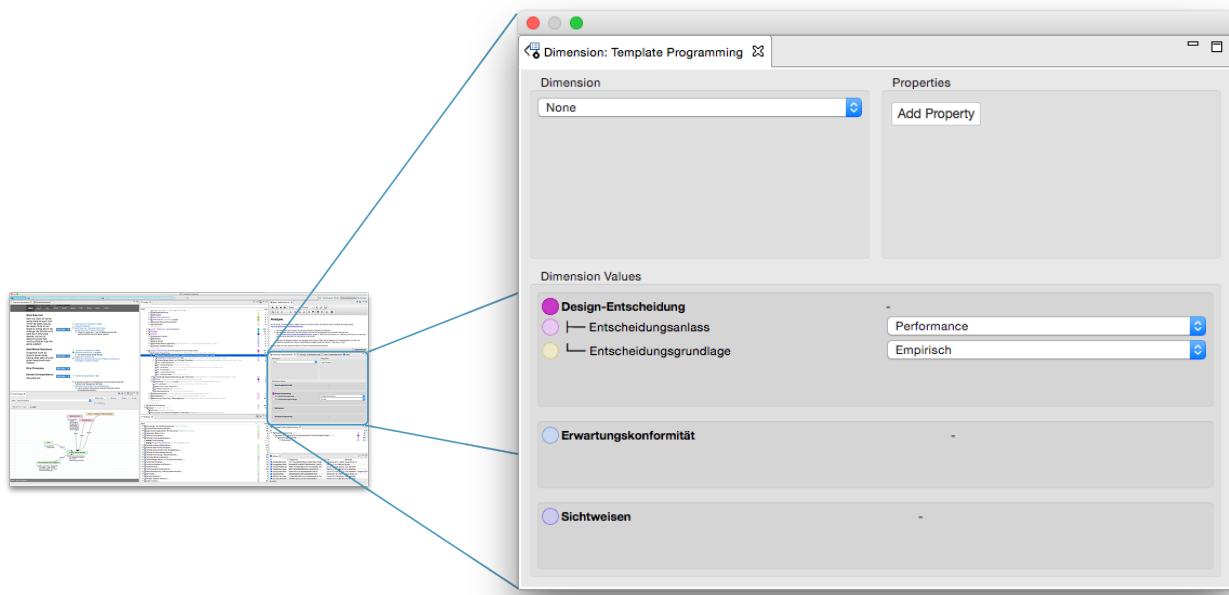


ABBILDUNG 3.24: Dieser Screenshot von APIUA zeigt das Eclipse-View “Dimension”, das die Dimension, alle Eigenschaften und Wertebelegungen für das aktuell selektierte Element darstellt.

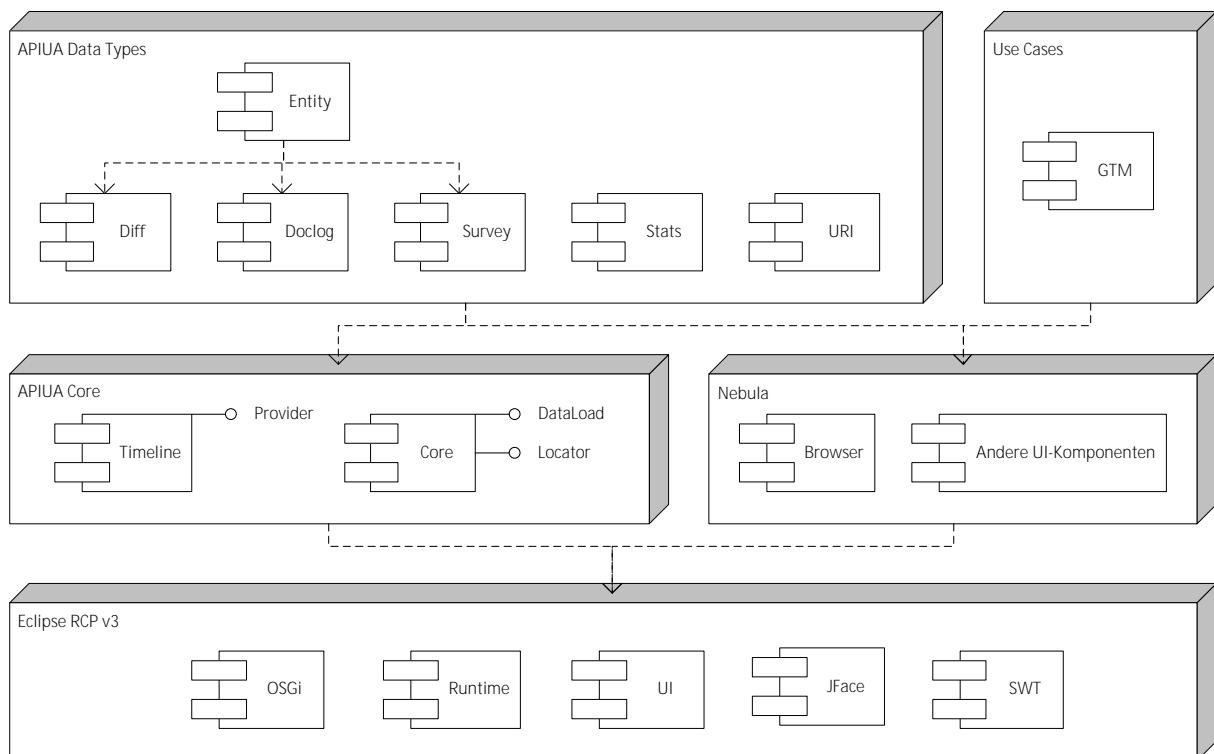


ABBILDUNG 3.25: Dieses UML-Diagramm beschreibt die Architektur von APIUA. Sie besteht aus den folgenden drei Schichten: RCP-Schicht, Infrastrukturerweiterungsschicht (Core und Nebula), Komponentenschicht (Data Types, Use Cases).

### 3.4.5.1 Schicht 1: Eclipse Rich Client Platform

Die unterste Schicht bildet die RCP-Schicht und stellt das Fundament von APIUA dar.

Die Komponenten *OSGi<sup>G</sup>* und *Runtime* stellen den, von Eclipse bekannten Plugin-Mechanismus bereit und erlauben es, die Kopplung zwischen Modulen (hier: Plugins) erheblich zu verringern. Die *SWT<sup>G</sup>*-Komponente erlaubt den plattformübergreifend einheitlichen Zugriff auf plattformabhängige UI-Elemente. *JFace* und *UI* ermöglichen die, von Eclipse bekannte, Organisation des Anwendungsfensters mit Hilfe von Perspektiven, Editoren und Ansichten (engl. *views*).

### 3.4.5.2 Schicht 2: Infrastrukturerweiterung

Die zweite Schicht besteht aus zwei Teilen, die die RCP-Schicht um weitere Infrastruktur-Funktionalitäten erweitert.

Den ersten Teil stellt das *Nebula*-Plugin dar, das ich im Rahmen meiner Masterarbeit entwickelt (Kahlert 2011) habe. Es stellt wiederverwendbare SWT-Komponenten bereit, die nach meiner Ansicht Bestandteil des SWT-Plugins selbst hätten sein sollen. Die Komponenten wurden im Rahmen dieser Arbeit von mir um eine Browser-Komponente ergänzt, welche ausführlich im Anhang A.2 erläutert wird. Kurzfassung: Viele Teile der *UI<sup>G</sup>* von APIUA sind programmatisch sehr anspruchsvoll. Um den Entwicklungsaufwand gering zu halten, traf ich die Entscheidung, die umfangreiche Funktionalität moderner Webbrowser wiederzuverwenden und UI-Elemente basierend auf den Websprachen HTML, CSS/Less<sup>55</sup> und JavaScript zu entwickeln. Der *Nebula*-Browser ist dafür zuständig, die so entwickelten UI-Elemente darzustellen.

Der zweite Teil ist das *Core*-Plugin. In Form von Diensten stellt es Funktionalität zur Datenorganisation bereit. Dazu gehören die Datenpersistierung, sowie die Möglichkeit, Forschungsdaten zu laden. Erweiterungspunkte (in Eclipse heißen sie *extension points*) und Dienste machen das *Core*-Plugin zu einem Bussystem, über das Plugins anderen Plugins (insbesondere denen der Schicht 3) Daten zur Verfügung stellen können. Produzenten und Konsumenten werden dadurch entkoppelt. Weitere Details zu den bereitgestellten Diensten werden im Anhang A.1 beschrieben.

Im *Core*-Plugin manifestiert sich außerdem eine grundlegende Entwurfsentscheidung. Sie besteht darin, jede Art von Datum mittels eines URI adressierbar zu machen. Nicht nur die Datenhaltung sondern alle Komponenten verwenden URIs zum Lokalisieren von Daten. Die Größe der im Rahmen dieser Arbeit erfassten Daten beträgt mehr als 24GB<sup>56</sup>. Diese Datenmenge kann nicht problemlos bei jedem

---

<sup>55</sup> Less (<http://lesscss.org>) stellt gemeinsam mit Sass (<http://sass-lang.com>) die populärste Spracherweiterung für CSS dar. Sie erlauben beispielsweise die Verwendung von Schleifen und Variablen. Die gleichnamigen Präprozessoren sind dafür zuständig, `.less`- bzw. `.sass`-Dateien nach CSS zu kompilieren.

<sup>56</sup> Diese Angabe bezieht sich auf die expandierten Daten. Das heißt, die Daten wurden so aufbereitet, dass sie effizient lesend verarbeitet werden können. Zur Aufbereitung gehören unter anderem die Erstellung von Screenshots und die Berechnung der effektiven Quellcode-Dateien auf Grundlage vorliegender Diff-Dateien.

Programmstart in den Arbeitsspeicher geladen werden. Darum stellt dieses Plugin den *LocatorService* bereit. Dieser erlaubt den intelligenten und speicherschonenden Zugriff<sup>57</sup> auf Datenobjekte — unabhängig davon, ob sie gerade geladen sind oder nicht.

Mit wenigen Ausnahmen haben URIs folgenden Aufbau:

`apiua://datatype/id/detail`

**datatype** bezeichnet den Datentyp bzw. das Datenformat. Beispiele sind *diff* und *entity*.

**id** bezeichnet einen Identifikator. Beispiel: 2ur8t8dx88n4v6ef

**detail** erlaubt eine feingliedrigere Lokalisierung. Dieser Teil ist sehr von dem Datentyp abhängig.

#### Beispiel 1:

`apiua://code/526` verweist auf einen Kode mit der ID 526.

#### Beispiel 2:

`apiua://diff/2ur8t8dx88n4v6ef/5/%2Fmy_sandbox/%2Ffirst_app.cpp` beschreibt die von Proband 2ur8t8dx88n4v6ef in der 6. Iteration editierte Datei `first_app.cpp`, die sich im Ordner `my_sandbox` befindet.

Auch wenn ich diese Funktion nicht implementiert habe, könnte man, für das sich im Einsatz befindliche Betriebssystem, einen *url scheme handler* programmieren, der URIs mit dem Schema `apiua` öffnet, um die Interoperabilität zwischen APIUA und Drittsoftware zu verbessern.

### 3.4.5.3 Schicht 3: Komponentenschicht

Diese Schicht besteht wiederum aus zwei Teilen, die die für den Anwender sichtbare Funktionalität implementieren.

Der erste Teil wird als *data types* bezeichnet und folgt einer Komponenten-basierten Architektur. Jede Komponente / jedes Plugin innerhalb dieses Teils

- ist ausschließlich abhängig von unteren Architekturniveaus<sup>58</sup>,
- stellt einen bestimmten Datentyp/-format über den *Core-Plugin-Bus* allen Konsumenten bereit,
- kann Daten dieses Datentyps/-formats lesen,
- stellt Eclipse-Views bereit, die die geladenen Daten visualisieren (vgl. Abbildung 3.22) und

---

<sup>57</sup> Dieser Dienst setzt einen Cache ein, der Einträge auf Basis von Zugriffshäufigkeiten verdrängt.

<sup>58</sup> Die Komponente *Entity* bildet eine Ausnahme, denn sie synthetisiert die geladenen Daten anderer Komponenten (siehe Abschnitt 3.3).

- stellt Visualisierungsinformationen (Bezeichnung, Ikone, Meta- und Detailinformationen) für Daten des Datentyps/-formats über den *Core-Plugin-Bus* bereit.

Die Anforderungen werden, wie bereits weiter oben beschrieben, ausschließlich über Plugin-Erweiterungen und *Core-Plugin-Dienste* realisiert. Die Architektur und die damit einhergehende geringe Kopplung machen das Werkzeug APIUA besonders wart- und erweiterbar. Unterstützung für weitere Datentypen/-formate kann problemlos in Form von Plugins geschaffen werden.

Der zweite Teil stellt die *Use Cases* (Anwendungsfälle) implementierenden Komponenten dar. Tatsächlich gibt es nur die GTM-Komponente, die es erlaubt, Kodes, Relationen und deren Eigenschaften auf der Grundlage der, durch die Datentyp-Komponenten bereitgestellten Daten zu modellieren. Da jedes Datum eine URI besitzt und das *Core-Plugin* Visualisierungsinformationen zwischen den Plugins vermittelt, kann auch dieses Plugin mit nur wenigen Abhängigkeiten arbeiten.

### 3.4.6 Funktionsweise von APIUA und Implementierung der GTM

Ursprünglich wollte ich ein Datenanalysewerkzeug schaffen, das exakt meine Bedürfnisse erfüllt. Das bedeutet, es muss meine Interpretation der GTM und meine speziellen Daten unterstützen. Ich habe dabei bewusst den Aufbau anderer Datenanalysewerkzeuge ignoriert, um eine möglichst hohe Spezialisierung auf meine Anforderungen zu erreichen. Umso erstaunlicher ist es, dass die Entwicklung am Ende mehr als 18 Monate “verschlingen” sollte und den etablierten Werkzeugen am Ende ähnlicher war, als ich erwartet hatte.

**Terminologie** Die terminologische Ähnlichkeit zu *ATLAS.ti* zeigt Tabelle 3.3. Am meisten fällt auf, dass mein verwendetes Vokabular eher technisch getrieben ist. Ein Beispiel dafür ist die Kodeinstanz, die in der GTM als Konzeptualisierung bezeichnet wird. Als zweites Beispiel soll das GTM-Phänomen dienen, welches in APIUA *Locatable* genannt wird. Es handelt sich dabei um das Interface, das von jedem Datenformatstyp (siehe Abschnitt 3.4.5.3) implementiert wird und damit durch das GTM-Plugin verarbeitet werden kann.

**Organisation des Arbeitsbereichs** In *ATLAS.ti* gibt es verschiedene Ansichten, die exklusiv sind, d.h. niemals parallel geöffnet sein können. APIUA hingegen verwendet die erprobte Organisation der Benutzeroberfläche nach Perspektiven. Perspektiven können frei definiert werden. Eine Perspektive besteht aus einer frei wählbaren Anordnung gewünschter Ansichten. Perspektiven für die Phasen des offenen und axialen Kodierens sind bereits vordefiniert (siehe Abbildungen 3.17, 3.31, 3.32 und 3.33). Abbildung 3.30 zeigt die Perspektive für die Exploration des gesammelten Datenmaterials.

GTM	ATLAS.ti	APIUA
Phänomen ( <i>Phenomenon</i> )	<i>Quotation</i> <sup>a</sup>	<i>Locatable</i> <sup>b</sup>
Konzeptualisierung ( <i>Conceptualization</i> )	<i>Annotation</i>	<i>Kodeinstanz</i> ( <i>Code Instance</i> )
Konzept ( <i>Concept</i> )	<i>Code</i>	<i>Kode</i> ( <i>Code</i> )
Eigenschaft ( <i>Property</i> )	<i>Code</i>	<i>Eigenschaft</i> <sup>d</sup> ( <i>Property</i> )
Kategorie ( <i>Category</i> )	<i>Code</i> <sup>c</sup>	<i>Kode</i> ( <i>Code</i> ) <sup>e</sup>
Beziehung ( <i>Relationship</i> )	<i>Relationship/Relation</i>	<i>Relation</i>

<sup>a</sup> In *ATLAS.ti* handelt es sich um Datenausschnitte.

<sup>b</sup> In APIUA handelt es sich um Datenausschnitte, deren Granularität — in Abhängigkeit vom Datentyp — programmatisch vorgegeben ist.

<sup>c</sup> Technisch handelt es sich um einen dimensionalisierten Kode, der als Eigenschaft eines anderen Kodes deklariert wird.

<sup>d</sup> In *ATLAS.ti* können zur Modellierung *Code Families* verwendet werden.

<sup>e</sup> In APIUA sind Kodes hierarchisch angeordnet. Eine Kategorie ist ein Kode mit Unterkodes.

TABELLE 3.3: Gegenüberstellung der in der GTM, in *ATLAS.ti* und in APIUA verwendeten Begriffe.  
Die Angaben zur GTM und *ATLAS.ti* entstammen Salinger (2013).



ABBILDUNG 3.26: Vergleich Kode-Darstellung; links: APIUA; rechts: *ATLAS.ti*

### 3.4.6.1 Offenes Kodieren

Das Kodieren von Datenmaterial ähnelt sich in den typischen CAQDAS-Werkzeugen und besteht im Grunde darin, einem referenzierbaren Datenausschnitt (GTM: Phänomen) einen Kode zuzuordnen, der entweder schon existiert oder innerhalb dieses Prozesses erstellt wird. Typischerweise kommt dabei Drag'n'Drop zum Einsatz. Erstellte Kodes und annotierte Datenausschnitte (GTM: Konzeptualisierung) werden listenartig dargestellt. Die Abbildungen 3.26 und 3.27 zeigen, wie diese Kode-Darstellung in APIUA und *ATLAS.ti* realisiert wird.

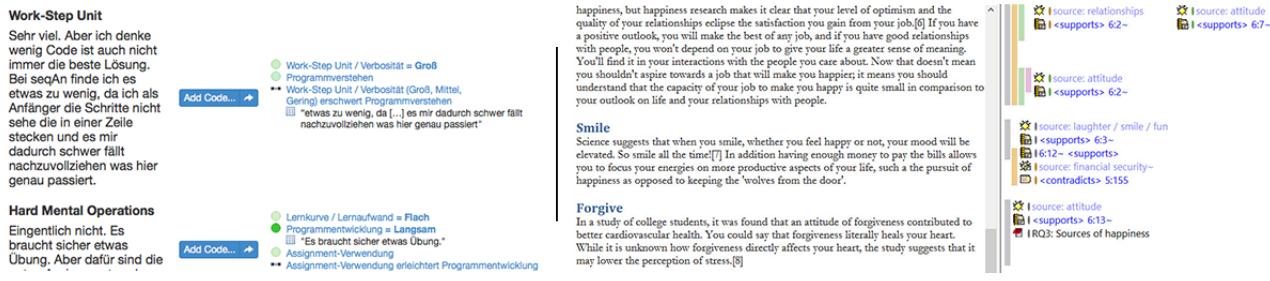


ABBILDUNG 3.27: Vergleich Kodeinstanz-Darstellung; links: APIUA; rechts: ATLAS.ti

**Organisation und Sortierung von Kodes** Im Gegensatz zur *ATLAS.ti* können in APIUA Kodes gefiltert, sortiert und hierarchisch angeordnet werden (siehe Abbildung 3.26, links). Die Semantik der hierarchischen Modellierung wird von dem Werkzeug wie folgt behandelt: "Wenn Kode ● A<sup>59</sup> der Elter des Kodes ● A.1 ist, ist ● A.1 eine Ausprägung von ● A, oder: ● A manifestiert sich in ● A.1."<sup>60</sup>

An dieser Stelle ist *ATLAS.ti* scheinbar differenzierter. Dort können Beziehungen zwischen Elter und Kind von einer der drei Arten [*Kategorie/generelles Konzept – Kode*], [*Oberbegriff – Unterbegriff*] und [*Das Ganze – Bestandteile*] sein (Mühlmeyer-Mentzel 2011). Aber: Die Semantik dieser Beziehungen existiert nur in der Vorstellung des Forschers. Technisch handelt es sich lediglich um vordefinierte, benannte Relationstypen, die beliebig editiert werden können und von *ATLAS.ti* selbst nicht weiter interpretiert werden. Im Gegensatz zu APIUA wird der Relationstyp in *ATLAS.ti* nicht verwendet, um eine weitergehende Werkzeugunterstützung zu ermöglichen.

Mühlmeyer-Mentzel (2011) räumt ein, dass die Strukturierungsmöglichkeiten in *ATLAS.ti* beschränkt sind. Insbesondere können Kodes nicht sortiert werden. Die Möglichkeit der hierarchischen Organisation von Kodes verändert den Arbeitsstil. Entschließt sich ein Forscher jedoch dazu, Kodes hierarchisch anzugeordnen, kann er dies in *ATLAS.ti* nur über die Definition von individuellen Relationen simulieren, die von der Anwendung aber nicht weiter berücksichtigt und beispielsweise nicht zur Darstellung eines Kode-Baums verwendet wird.

In APIUA kann jedes Datum, das *Locatable* implementiert, auch kodiert werden. Da Kodes auch *Locatable* implementieren, können Kodes andere Kodes kodieren. Um die Modellierungskomplexität nicht unnötig zu erhöhen, habe ich mich dazu entschlossen, diese Semantik mit der Eltern-Kind-Beziehung gleichzusetzen. Ist also ● A.1 ein Kind von ● A, wird dies von APIUA genauso gehandhabt, als wäre ● A.1 mit ● A kodiert worden.

Die eben beschriebene Beziehung gilt auch transitiv. Hat ● A.1 neben seinem Elternkod ● A auch noch den Unterkode ● A.1.1, so gilt: "● A.1 ist mit ● A.1 kodiert" und "● A.1 ist mit ● A kodiert".

**Memos** werden in APIUA umfassend unterstützt. Jedes Datum, das das *Locatable*-Interface implementiert, kann ein Memo besitzen. Das Memo des aktuell fokussierten Elements, erscheint im Memo-Editor (siehe Abbildung 3.28). Ist das Fokus habende Element ein Kode oder eine Relation, werden

<sup>59</sup> Zur Erinnerung: Es handelt sich hierbei um die Darstellung eines Kodes. Details zu dieser Stellung können im Abschnitt 1.4.5.2 auf Seite 54 nachgelesen werden.

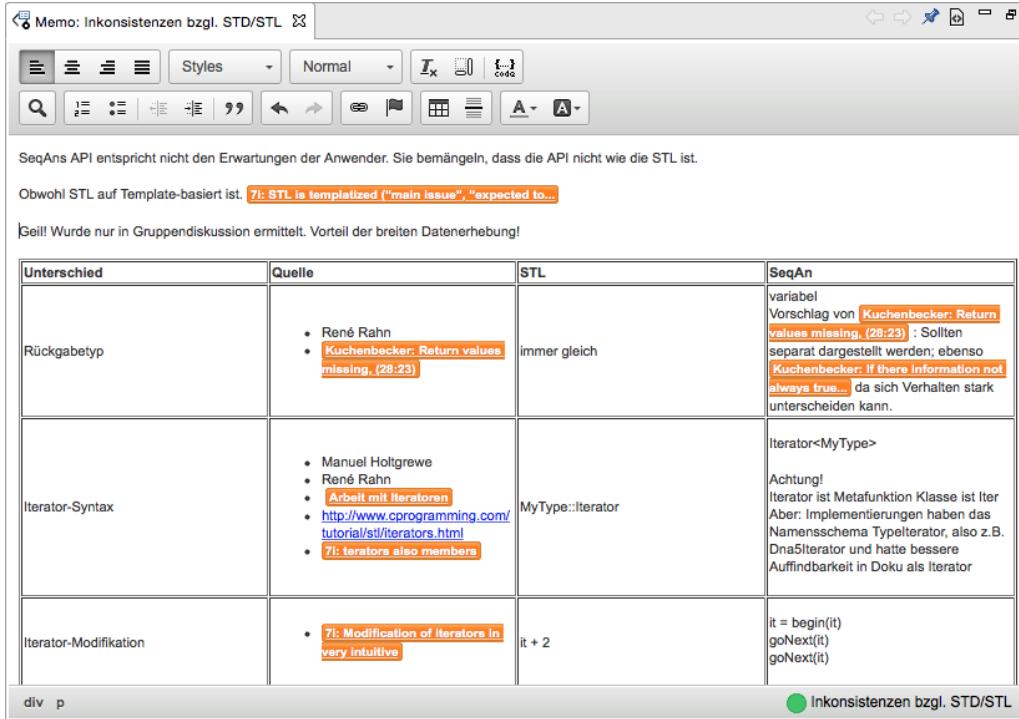


ABBILDUNG 3.28: Dieser Screenshot zeigt den auf dem *Nebula*-Browser basierenden Memo-Editor samt Inhalt für den Kode `apiua://code/-9223372036854775633`.

optional auch sämtliche Memos der entsprechenden Kode- bzw. Relationsinstanzen gezeigt. Der Memo-Editor ist ein erweiterter, auf dem *Nebula*-Browser basierender Rich-Text-Editor, der sowohl interne als auch externe Verknüpfungen zu anderen Daten erlaubt. Eine interne Verknüpfung bezieht sich auf eine URI mit dem Schema *apiua*. Klickt der Anwender auf einen solchen Link, öffnet dieser den Datenpunkt entsprechend seines Typs: Memo wird im Memo-Editor geladen, Kode wird in Kode-Ansicht hervorgehoben, Quellcode-Datei wird in der Vergleichsansicht geöffnet, etc. Verweilt der Anwender mit seiner Maus hingegen für einen kurzen Moment über einem Link (ohne ihn anzuklicken), so öffnet sich ein gelbes Informationsfenster, das wichtige — wiederum typabhängige — Informationen zu diesem Datenpunkt anzeigt (siehe Abbildung 3.30). Dieses Informationsfenster kann von anderen Plugins erweitert werden.

**Visualisierung** Wie in *ATLAS.ti* auch, können Kodes mit Farben versehen werden. Allerdings wird die Bedienung dieser Funktion von Zieris (2014) als umständlich und nicht für große Datenmengen geeignet, beschrieben. In APIUA kann der Anwender die Farbe eines spezifischen Kodes, eines Teilbaums oder des gesamten Kode-Bestands automatisch färben lassen. Die zweite und dritte Option skalieren auch für große Datenmengen gut. Dazu werden die Farben der Nachbarknoten berücksichtigt und ein konfliktfreier Farbraum bestimmt. Dieser wird dann rekursiv auf die Kindkodes gleich verteilt. Dadurch erhalten Kodes, die sich nah sind (gleicher Teilbaum, wenig Abstand zum Nachbarkode) eine ähnlichere Farbe als solche, die weiter voneinander entfernt sind. Abbildung 3.29 zeigt ein Beispiel. Diese Funktion hat sich als sehr nützlich für das axiale Kodieren erwiesen, da ich so ein besseren Überblick über verwandte Konzepte hatte.



ABBILDUNG 3.29: Dieser Screenshot von APIUA zeigt einen Kode-Teilbaum, der automatisch eingefärbt wurde. Auf den Elternkodes ● A, ● B und ● C wird das gesamte Farbspektrum von Rot über Grün bis hin zu Violett verwendet. Die Kindkodes erhalten Farben, die links und rechts von der Elternfarbe liegen ohne den Bereich eines Nachbar teilbaums zu verletzen. Beispiel: Die Farbe von ● B.1 liegt links von ● B; die von ● B.2 rechts davon.

**Eigenschaften** In APIUA kann jeder Kode eine Dimension haben. Die zugrunde liegende Skala kann aktuell nominal oder ordinal sein<sup>60</sup>. Dimensionalisierte Kodes können als Eigenschaften anderer Kodes dienen. Ist ein Phänomen mit einem Kode kodiert worden, können dem Phänomen, in Bezug auf die Dimension und Eigenschaften des Kodes, Werte zugewiesen werden (siehe Abbildung 3.24). Beispiel: Der Kode ● Auto verfügt über eine Eigenschaft ● Farbe mit der nominalen Skala (*Rot, Gelb, Grün, Blau*). Wird nun ein Phänomen ● X mit ● Auto kodiert, kann ● X nun ein Wert für die Farbe zugewiesen werden, denn ● X ist fortan eine Instanz/Ausprägung/Phänomen für ● Auto.

Diese Art der Implementierung der GTM erlaubt sogar *Subdimensionalisierung* (Strauss 1987), also die Modellierung von Eigenschaften, die selbst wiederum Eigenschaften besitzen. Am Beispiel der Autofarbe könnte eine Untereigenschaft von Farbe die Eigenschaft *Lackierung (matt, glänzend)* sein.

**Schwierigkeiten der Modellierung** von Analyseergebnissen ergeben sich bei der Verwendung hierarchischer Kodes und Eigenschaften:

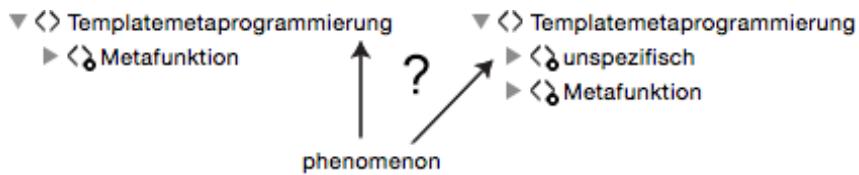
### Spezifität

Während der Analyse meiner subjektiven Daten musste ich feststellen, dass die getroffenen Aussagen inhaltlich derart in die Breite gingen, dass ich für jeden generierten Kode kaum eine nennenswerte Zahl Verankerungen hatte. In solch einem Fall könnte man *theoretischen Samplings* betreiben, indem man weitere Daten erhebt, um diese Kodes besser verstehen zu können.

60 Die zur Verfügung stehenden Skalentypen können durch Plugins erweitert werden.

Allerdings kann eine solche Datenerhebung sehr teuer sein und ein Mittelweg muss gefunden werden.

Konkretes Beispiel: In meinen Daten schildert<sup>8</sup> ein Proband seine Frustration von dem Templatemetaprogrammiungs-Aspekt *Metafunktion*, was ich mit dem Kode ● Metafunktionen kodierte. Dieser Kode ist selbst ein Unterkode von ● Templatemetaprogrammierung. An anderer Stelle gibt es eine weniger spezifische Aussage<sup>9</sup>, in der nur noch allgemein von *Templatemetaprogramming* die Rede ist.



Hier stellt sich nun die Frage, ob die allgemeinere Aussage mit ● Templatemetaprogrammierung kodiert werden sollte. Oder man führt einen “unspezifischen” Unterkode von ● Templatemetaprogrammierung ein und verwendet diesen dann als “Sammelbecken” für Phänomene mit geringerem Aussagegehalt.

## Vererbung

Welche Auswirkungen haben Eigenschaften eines Kodes auf seine Eltern- bzw. Kindkodes? Denkt man an Objektorientierung, hieße die Antwort, dass Eigenschaften nach unten vererbt werden. Aber gilt das auch für Kodeinstanzen? Würden wir uns beim obigen Beispiel für die erste Variante entscheiden, würde das Phänomen nicht nur eine Instanz von ● Templatemetaprogrammierung darstellen, sondern auch von allen Unterkodes von ● Templatemetaprogrammierung, u.a. auch ● Metafunktionen. Ob sich ein Phänomen auf alle Unterformen bezieht oder schlichtweg nur eine geringe Aussagekraft hat, kann man nicht immer zuverlässig sagen. Oder gilt die Vererbung von Kodeinstanzen gar nach oben? Immerhin sollten sich Aussagen über ● Templatemetaprogrammierung treffen lassen, wenn man konkretere Phänomene zu ● Metafunktionen gesehen hat.

Meinen Lösungsvorschlag für diese Problemfragen gebe ich weiter unten.

Nach der GTM von Corbin u. Strauss (2014); Strauss u. Corbin (1990) “darf” auch das Vorwissen des Forschers in den Forschungsprozess eingebracht werden. Die *URI*-Komponente von APIUA erlaubt das Erfassen URI-referenzierbarer Quellen. Dies ermöglicht es dem Forscher Konzepte in beliebigen Dokumenten zu verankern, auf das sein Vorwissen fußt. In meiner Forschung habe ich Verankerungen ganz konkret auf Basis von Literatur mit Hilfe von `bibtex`-URIs und verschiedenartigste Dokumente mit Hilfe von `file`-URIs vorgenommen. Dieses Vorgehen ist aus meiner Sicht unerlässlich, um die Gütekriterien *Argumentative Interpretationsabsicherung* und *Nähe zum Gegenstand* zu erfüllen. Keines der mir bekannten CAQDAS-Anwendungen erfüllt diese Anforderungen, ohne dass die referenzierten Dokumente auch direkt in das Datenanalysewerkzeug importiert und damit auch von dem Werkzeug unterstützt werden müssten.

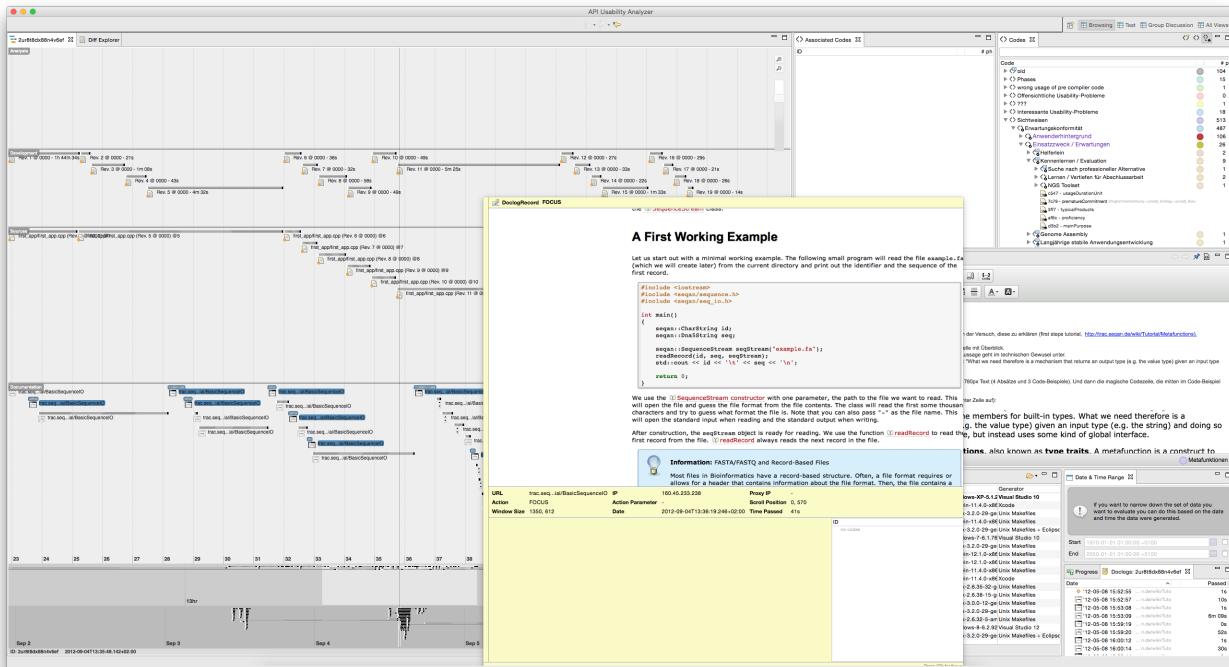


ABBILDUNG 3.30: Dieser Screenshot von APIUA zeigt eine typische Analysesitzung, bei der sich der Forscher einen Überblick über die Daten verschafft. Dazu nutzt er die Zeitleistenansicht.

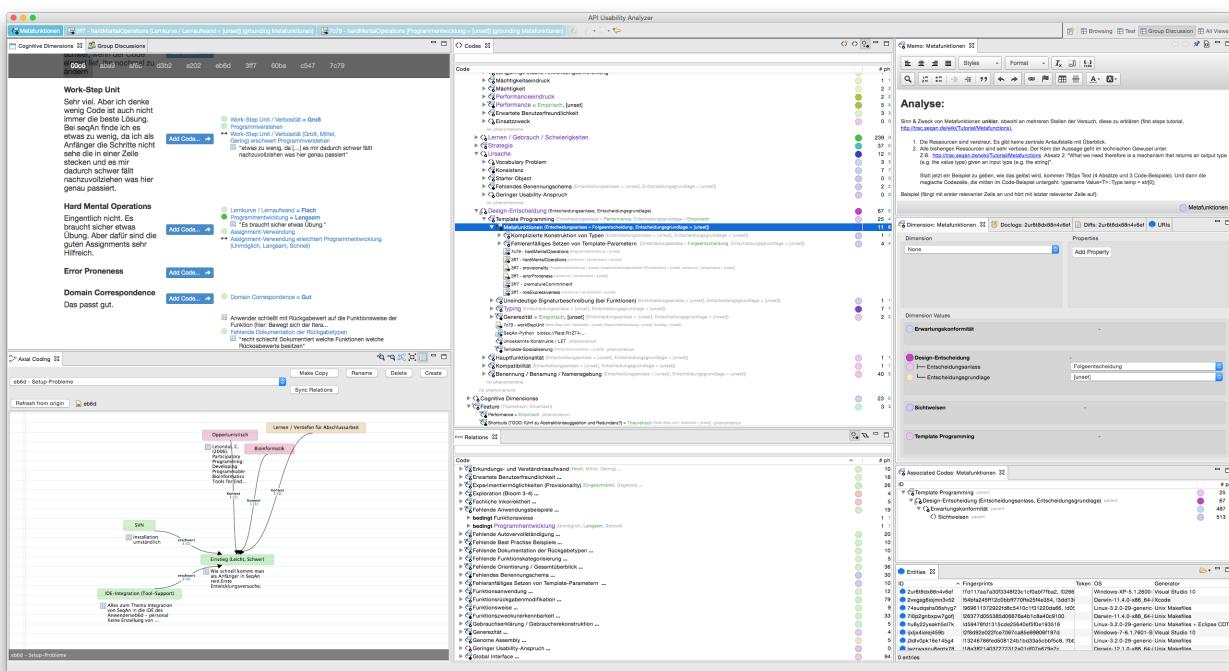


ABBILDUNG 3.31: Dieser Screenshot von APIUA zeigt eine typische Sitzung mit Fokus auf offenes Kodieren. In diesem Fall werden die Cognitive-Dimensions-Fragebögen (oben links) analysiert.

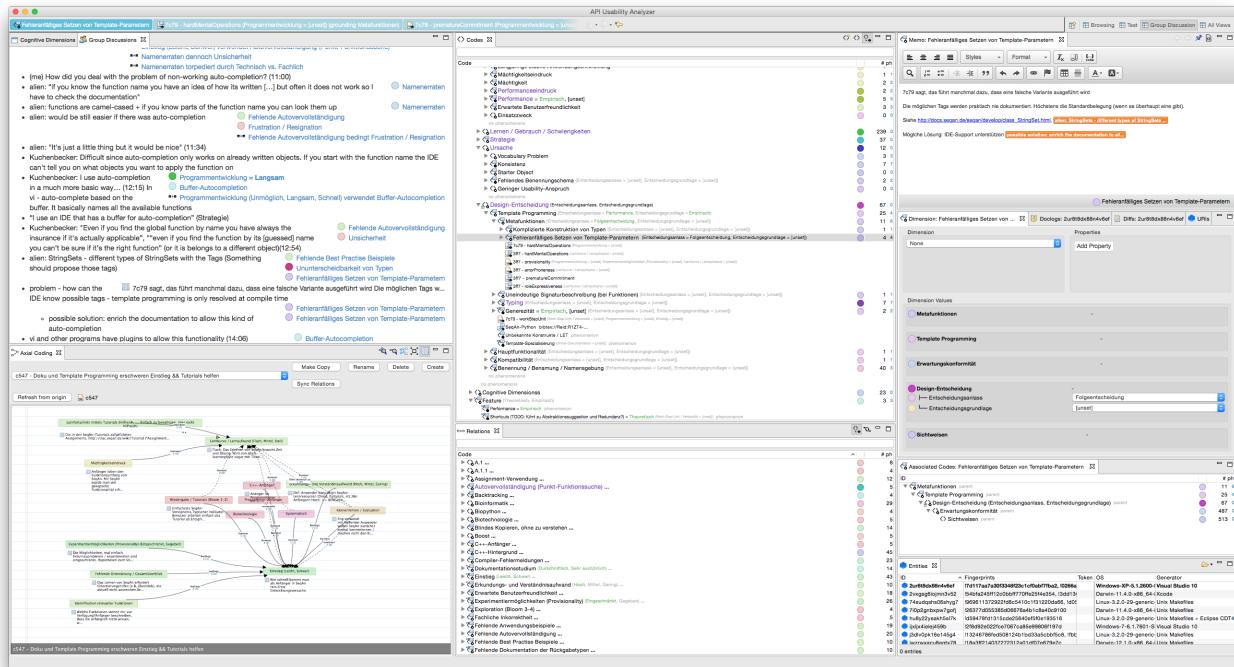


ABBILDUNG 3.32: Dieser Screenshot von APIUA zeigt eine typische Sitzung mit Fokus auf offenes Kodieren. In diesem Fall wird eine Gruppendiskussion (oben links) betrachtet.

### 3.4.6.2 Axiales Kodieren

Die Möglichkeit axial kodieren zu können, ist ein integraler Bestandteil von APIUA. Eine Relation in APIUA ist wie ein benannter gerichteter Graph zwischen zwei Kodes und wird in dieser Arbeit wie folgt notiert: ● Kode 1  $\xrightarrow{\text{Bezeichner}}$  ● Kode 2. Abbildung 3.33 zeigt die entsprechende Ansicht in APIUA. So wie Kodes mit Hilfe von Kodeinstanzen verankert werden, können auch Relationen mittels Relationsinstanzen verankert werden.

*ATLAS.ti* verwendet zwar für Kodes ein vergleichbares Datenmodell (Mühlmeyer-Mentzel 2011), kann aber Relationen selbst nicht verankern (Zieris 2014). Möchte der Forscher also nachschauen, auf welchen Datenpunkten eine Relation überhaupt fußt, besteht für ihn nur die Möglichkeit, alle Stellen zu ermitteln, die als Verankerungen für die beiden, durch die Relation in Beziehung gesetzten Kodes dienen und in Gedanken neu zu kodieren.

Kurzum: Wenn der Forscher das Verankerungsproblem nicht auf andere Weise gelöst hat, können Verankerungen von Relationen durch Dritte nur eingeschränkt nachvollzogen werden, was aus meiner Sicht die Verifizierbarkeit von mit *ATLAS.ti* erstellten Relationen potentiell einschränkt.

Eine weitere Einschränkung in *ATLAS.ti* besteht darin, dass es zwischen zwei Kodes nur eine Relation geben kann. APIUA unterstützt beliebig viele — wenn der Forscher es für notwendig hält auch gleich benannte — Relationen.

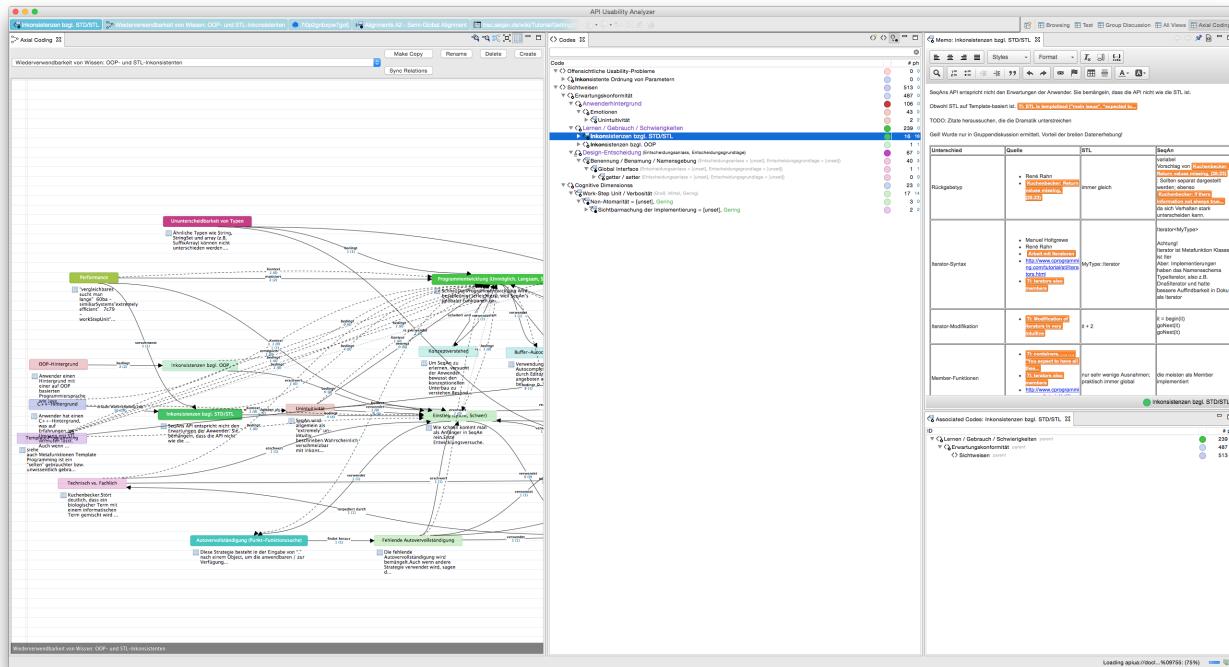


ABBILDUNG 3.33: Dieser Screenshot von APIUA zeigt eine typische Sitzung mit Fokus auf axiales Kodieren. In diesem Fall wird das axiale Kodiermodell “OOP- u. STL-Inkonsistenzen” — links dargestellt — bearbeitet.

Der Forscher hat zwei verschiedene Möglichkeiten axiale Kodierungen bzw. axiale Kodiermodelle<sup>61</sup> zu erstellen. Entweder beginnt er “from scratch”, entwickelt also eines von Grund auf oder er erstellt eines auf der Grundlage eines existierenden Kodes, einer existierenden Relation oder einer jeweiligen Instanz. In beiden Fällen hält der Editor den Graphen bei Änderungen (Umbenennungen, Farbänderungen, Hierarchieänderungen, etc.) aktuell. Im zweiten Fall wird darüber hinaus der Graph selbst dynamisch erzeugt, initial formatiert und ebenfalls aktuell gehalten. Der so erstellte Graph enthält alle Kodes bzw. Kodeinstanzen, mit denen der ursprünglich ausgewählte Kode bzw. die Kodeinstanz über Relationen verbunden ist. Gibt es also  $\textcolor{pink}{\bullet} A \rightarrow \textcolor{teal}{\circ} B$  und basiert der Graph auf  $\textcolor{pink}{\bullet} A$ , enthält der Graph neben  $\textcolor{pink}{\bullet} A$  auch  $\textcolor{teal}{\circ} B$  und  $\textcolor{pink}{\bullet} A \rightarrow \textcolor{teal}{\circ} B$ . Für Elemente, die für den Aussagekern des axialen Kodiermodells unwichtig sind, besteht die Möglichkeit, sie auszublenden.

Zurück zum obigen Beispiel: Ein Proband machte eine Aussage<sup>62</sup> zu Metafunktionen. Genauer: Die Notwendigkeit, sich mit Metafunktionen auseinanderzusetzen, wird als frustrierend beschrieben. Daraus ergibt sich  $\textcolor{pink}{\bullet} \text{Metafunktionen} \xrightarrow{\text{bedingt}} \textcolor{pink}{\bullet} \text{Frustration/Resignation}$ . Eine andere Aussage<sup>63</sup> ist ähnlich, aber weniger spezifisch:  $\textcolor{pink}{\bullet} \text{Templatemetaprogrammierung} \xrightarrow{\text{bedingt}} \textcolor{pink}{\bullet} \text{Frustration/Resignation}$ .

Durch die Unterstützung von Unter- und Überkodes könnte man nun definieren, dass  $\textcolor{pink}{\bullet} \text{Metafunktionen} \xrightarrow{\text{bedingt}} \textcolor{pink}{\bullet} \text{Frustration/Resignation}$  eine Unterrelation von  $\textcolor{pink}{\bullet} \text{Templatemetaprogrammierung} \xrightarrow{\text{bedingt}} \textcolor{pink}{\bullet} \text{Frustration/Resignation}$  ist. Oder allgemeiner: Ist  $\textcolor{pink}{\bullet} A.1$  Unterkode von  $\textcolor{pink}{\bullet} A$  und  $\textcolor{teal}{\circ} B.1$  Unterkode von  $\textcolor{teal}{\circ} B$ , wären  $\textcolor{pink}{\bullet} A.1 \xrightarrow{\text{?}} \textcolor{teal}{\circ} B$ ,  $\textcolor{pink}{\bullet} A \xrightarrow{\text{?}} \textcolor{teal}{\circ} B.1$  und

61 Zur Erinnerung: Eine axiale Kodierung stellt Relationen auf Kodeinstanz-Ebene dar, wohingegen ein axiales Kodiermodell Relationen auf Kode-Ebene beschreibt.

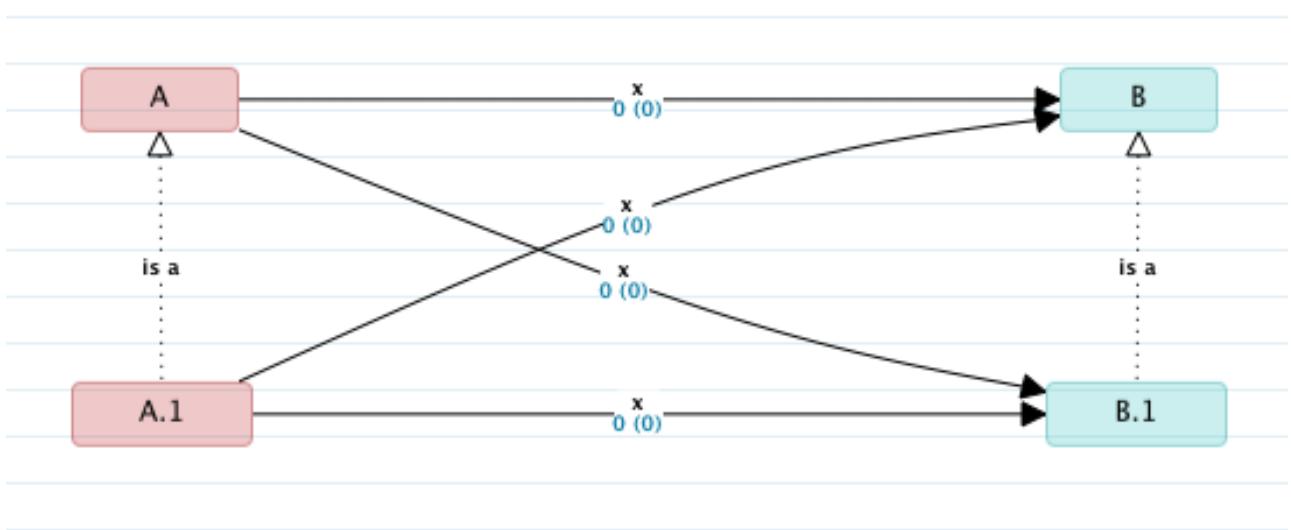


ABBILDUNG 3.34: Die Relationen  $\text{A.1} \xrightarrow{x} \text{B}$ ,  $\text{A} \xrightarrow{x} \text{B.1}$  und  $\text{A.1} \xrightarrow{x} \text{B.1}$  sind Unterrelationen von  $\text{A} \xrightarrow{x} \text{B}$ .

$\text{A.1} \xrightarrow{x} \text{B.1}$  Unterrelationen von  $\text{A} \xrightarrow{x} \text{B}$  ( $x$  ist ein beliebiger gemeinsamer Bezeichner; siehe Abbildung 3.34).

Auch diese Konstruktion ist wie bei den Kodes transitiv. Hätte  $\text{A.1}$  den Unterkode  $\text{A.1.1}$  und  $\text{B.1}$  den Unterkode  $\text{B.1.1}$  und gäbe es die Relationen  $\text{A} \xrightarrow{x} \text{B}$ ,  $\text{A.1} \xrightarrow{x} \text{B.1}$  und  $\text{A.1.1} \xrightarrow{x} \text{B.1.1}$  so wäre nicht nur  $\text{A.1} \xrightarrow{x} \text{B.1}$  Unterrelation von  $\text{A} \xrightarrow{x} \text{B}$  sondern auch  $\text{A.1} \xrightarrow{x} \text{B.1}$  Unterrelation von  $\text{A} \xrightarrow{x} \text{B}$ .

So wie Kodes Kodeinstanzen haben können, können Relationen auch Relationsinstanzen besitzen. Hier stellt sich dieselbe Vererbungsfrage wie bei Kodes: Erben Unterrelationen die Instanzen ihrer Überrelation (oder anders herum)? Und wie kodiere ich unspezifische Aussagen?

### 3.4.6.3 Selektives Kodieren

In *ATLAS.ti* gibt es keine explizite Unterstützung für selektives Kodieren. Von dem Forscher wird erwartet, dass er dazu die vorhandene Funktionalität der Software nutzt. Dazu zählen insbesondere die für das axiale Kodieren bereitgestellten Möglichkeiten.

Ähnlich verfährt auch APIUA — mit zwei Ausnahmen:

1. APIUA implementiert in Ansätzen, die Möglichkeit, Interferenzregeln auf die erarbeitete theoretische Modellierung anzuwenden, was das selektive Kodieren unterstützen kann. Dabei handelt es sich um eine Funktion, die keine der mir bekannten Konkurrenzprodukte unterstützt.
2. Darüber hinaus gibt es die Möglichkeit, einem Kode eine Hervorhebungsgruppe zuzuordnen. Standardmäßig werden Kodes nicht hervorgehoben. Der Anwender hat die Möglichkeit die Hervorhebung abzusenken — die schwarze Kodebeschriftung wird dann plattformweit grau und die

Kodefarbe blass dargestellt — oder sie anzuheben — dann wird die Kodebeschriftung **fett und violett** und die Kodefarbe kräftiger dargestellt (siehe Abbildung 3.31). Auf diese Weise kann der Forscher in jeder Phase seiner Forschung Kandidaten für selektives Kodieren festhalten.

### 3.4.6.4 Erkenntnisperspektive

In den vorangegangenen Abschnitten habe ich Fragen aufgeworfen, die die konkrete Modellierung einer Grounded Theory (GT) innerhalb eines Datenanalysewerkzeugs betreffen. Sie betreffen einerseits den Umgang mit Aussagen geringerer Spezifität und andererseits die Vererbung von Erkenntnissen. Diese Fragen stellten sich mir bei dem Versuch, automatisch generierte GTMs zu verallgemeinern und ihre wesentliche Aussage herauszuarbeiten (siehe Abschnitt 3.5.3.2).

Als *Erkenntnisperspektive* bezeichne ich, ob der Forscher entweder gerade eine synthetische (abduktiv oder induktiv – also “Erkenntnis erweiternd”) oder deduktive (Rehfus 2003) Sichtweise einnimmt. Während der Entwicklung einer GT nimmt der Forscher häufig die synthetische Erkenntnisperspektive ein, denn in ihr besteht ja gerade der Erkenntnisgewinn. Genauso häufig aber wechselt er in die deduktive Erkenntnisperspektive, um seine Hypothesen zu überprüfen (Kelle 1994). Dieser Wechsel zwischen Induktion und Deduktion findet während des gesamten Forschungszeitraums statt (siehe Abschnitt 1.4 und Strauss 1987).

Mit einer deduktiven Erkenntnisperspektive sollte es sich wie bei der Objektorientierung verhalten: Die Vererbung findet von oben nach unten statt. Aussagen, die ich für Überkodes bzw. Überrelationen treffen kann, müssen — wenn die GT funktioniert — auch für deren Unterkodes bzw. Unterrelationen gelten<sup>62</sup>.

Mit einer synthetischen Erkenntnisperspektive hingegen kann man eine hypothetische Vererbung nach oben in Erwägung ziehen. So könnten Aussagen, die für Unterkodes gelten möglicherweise auch für den Überkode gelten.

APIUA soll den bereits von Glaser u. Strauss (1967) als erforderlich beschriebenen Wechsel zwischen induktivem und deduktivem Denken — insbesondere beim axialen Kodieren (Strauss u. Corbin 1990) — Werkzeug-seitig unterstützen. Darum ist APIUA dem Forscher beim Einnehmen der synthetischen Erkenntnisperspektive bereits jetzt schon behilflich, indem es zum Beispiel zwischen expliziten und impliziten Relationen unterscheidet. Eine explizite Relation  $\vec{R}$  ist eine, die vom Forscher modelliert wurde. Eine implizite Relation hingegen basiert auf einer anderen explizit definierten Relation  $\vec{R}_-$ , die selbst Unterrelation von  $\vec{R}$  ist und damit  $\vec{R}$  impliziert (siehe Abbildung 3.35). Existiert  $\vec{R}$  allerdings (noch) nicht, fungiert  $\vec{R}_-$  als hypothetische Relation für ein mögliches  $\vec{R}$ .

---

<sup>62</sup> Ganz so streng ist die Prämisse nicht, und soll auch so nicht verstanden werden. Die theoretische Konstruktion verfügt lediglich über eine “Aura der Objektivität” (“aura of objectivity”) und darf nicht für ein “erzwungenes Rahmenwerk” (“forced framework”) gehalten werden. (Charmaz 2006)  
Eine GT, die tatsächlich jede Beobachtung vollumfänglich erklären kann, würde Gefahr laufen, sehr deskriptiv zu werden und sich nicht mehr dazu eignen, verallgemeinerbare Aussagen zu treffen.

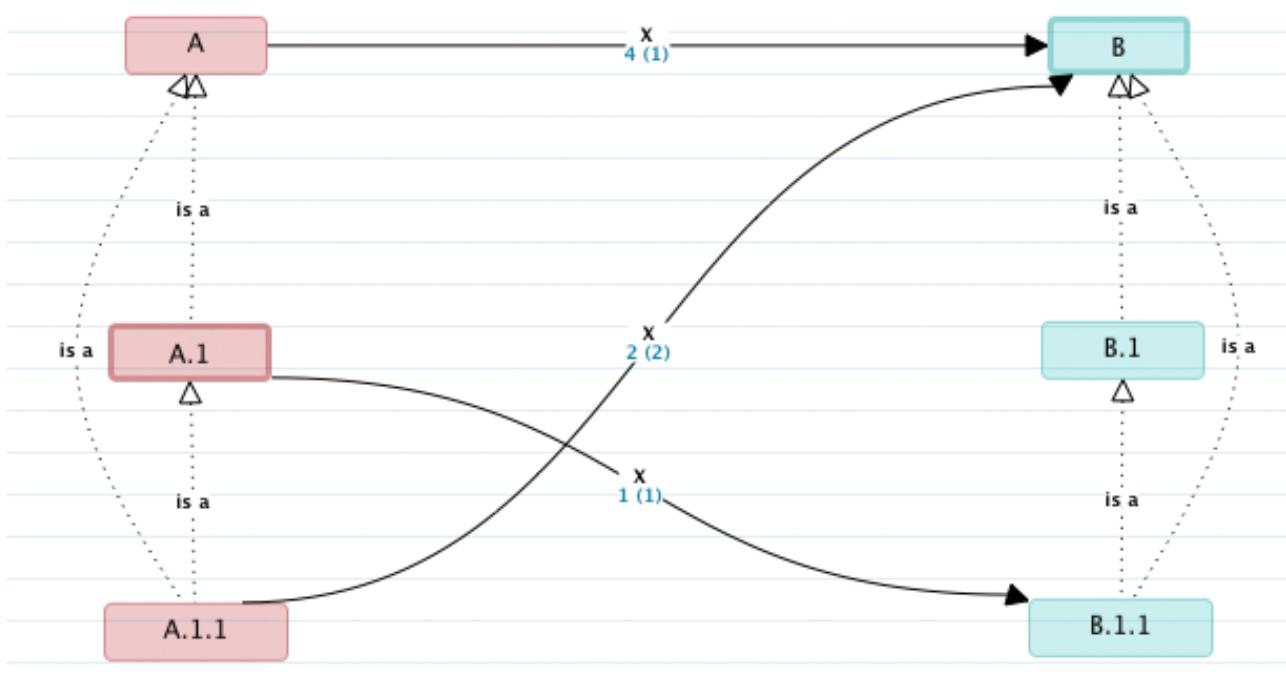


ABBILDUNG 3.35: Die Relationen  $\text{A.1} \xrightarrow{x} \text{B.1.1}$ , sowie  $\text{A.1.1} \xrightarrow{x} \text{B}$  sind implizit für  $\text{A} \xrightarrow{x} \text{B}$ . Demnach erhöht sich die Verankerung von  $\text{A} \xrightarrow{x} \text{B}$  um die Summe der Verankerungen der impliziten Relationen ( $1 + 2 = 3$ ).  $\text{A} \xrightarrow{x} \text{B}$  konnte also einmal direkt und dreimal indirekt verankert werden.

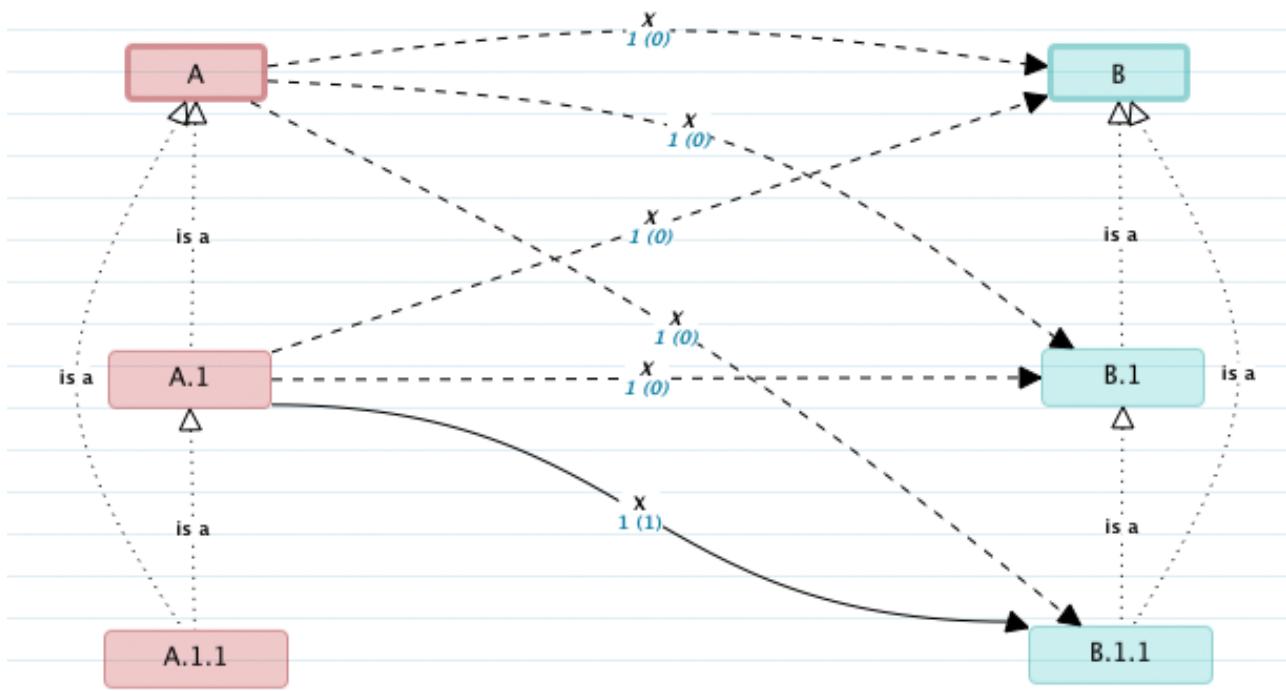


ABBILDUNG 3.36: Die Relation  $\text{A.1} \xrightarrow{x} \text{B.1.1}$  ist in diesem ACM die einzige explizite Relation. Die übrigen mit gestrichelten Kanten dargestellten Relationen sind Vorschläge, die sich aus  $\text{A.1} \xrightarrow{x} \text{B.1.1}$  ergeben.

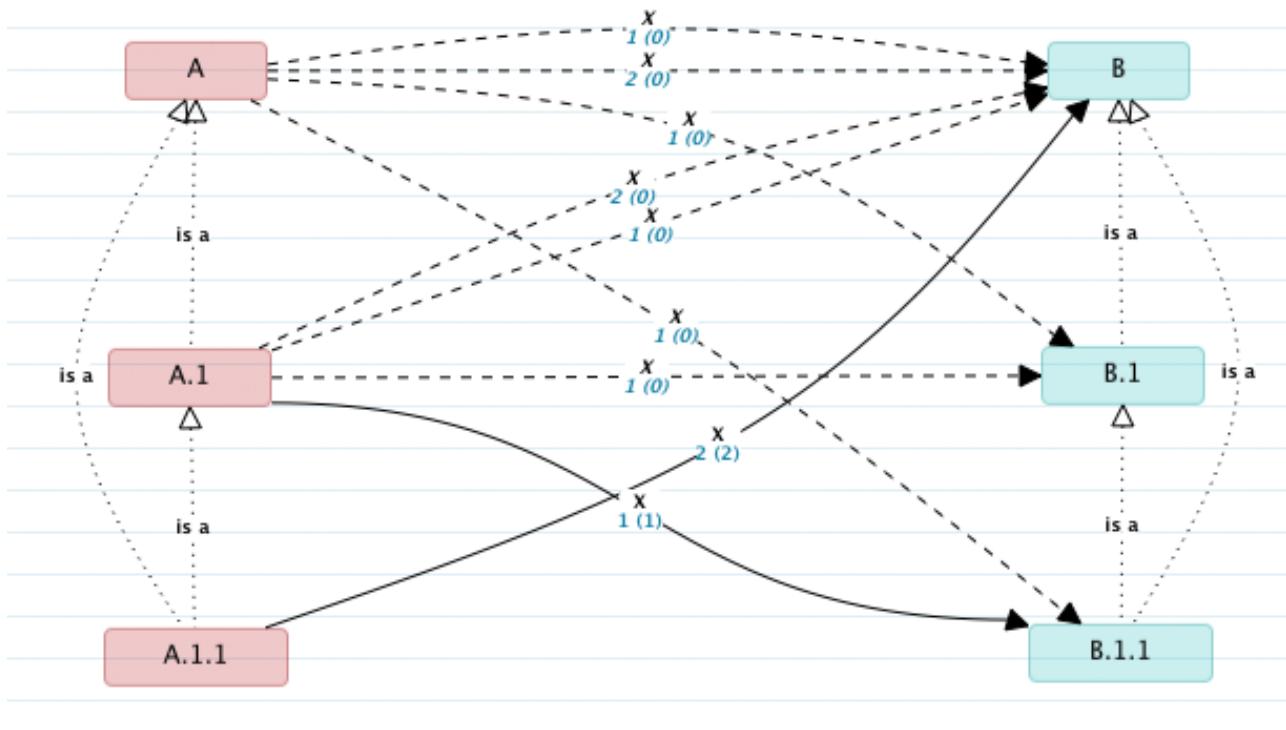


ABBILDUNG 3.37: Dieses ACM unterscheidet sich von 3.36 lediglich darin, dass es neben  $\text{A.1} \xrightarrow{x} \text{B.1}$  auch die Relation  $\text{A.1.1} \xrightarrow{x} \text{B}$  enthält. Daraus ergeben sich mehrere gleich lautende hypothetische Relationen zwischen denselben Konzepten (insb. zwischen  $\text{A}$  und  $\text{B}$ ).

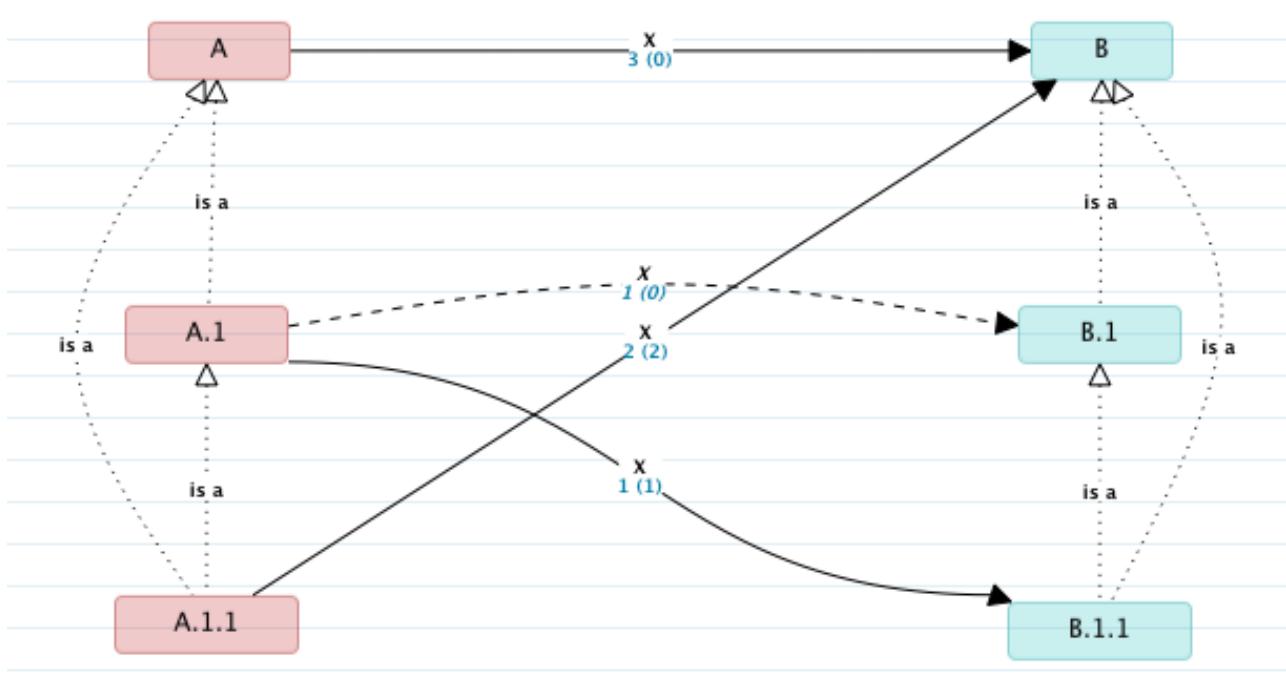


ABBILDUNG 3.38: Im Gegensatz zu 3.37 enthält dieses ACM zwischen  $\text{A}$  und  $\text{B}$  die explizite Relation  $\text{A} \xrightarrow{x} \text{B}$ . Dadurch entfallen alle hypothetischen Relationen, die in  $\text{A}$  oder  $\text{B}$  anfangen oder enden. Die einzige gebliebene hypothetische Relation ergibt sich aus  $\text{A.1} \xrightarrow{x} \text{B.1.1}$ .

Informatisch gesehen ist das Produkt der Analyse, das hauptsächlich in den Phasen des offenen und axialen Kodierens entsteht, eine Ontologie — also eine “explizite Spezifikation einer Konzeptualisierung” (Gruber 1993). Denn: Bevor der Forscher seine Grounded Theory in Form einer so genannten *Story* erzählt, liegen lediglich axiale Kodiermodelle vor, die man durch eine Ontologie beschreiben kann. Man könnte also auch vereinfacht(!) sagen, dass eine Grounded Theory auf einer rückwärts mit Hilfe synthetischen Schließens entwickelten Ontologie basiert, die in den Daten verankert ist bzw. ein Model für die Daten darstellt. Meiner Meinung nach eignet sich diese Beobachtung dazu, bessere Datenanalysewerkzeuge zu entwickeln, als es sie heute gibt.

Wenn man einmal von den im Abschnitt 1.4 angesprochenen terminologischen Ungereimtheiten der GTM absieht, gibt es einen abgegrenzten Baukasten von Elementen (Konzepte, Eigenschaften, Kategorien, Relationen, etc.), aus denen die Grundlage der GT besteht. Diese mögliche Anordnung dieser Elemente könnte man in einer Meta-Ontologie beschreiben. Ein GTM geeignetes Datenanalysewerkzeug müsste dann nur noch diese Meta-Ontologie unterstützen und dem Forscher zur Verfügung stellen. Schließlich würde jede Ontologie, die ein Forscher entwickelt, eine Instanz dieser Meta-Ontologie sein.

Trifft man die Annahme, dass jede Theorie gewissen Gesetzmäßigkeiten unterliegt (z.B. Vererbung wie oben beschrieben), könnte man diese Interferenz- und Integritätsregeln ebenfalls in der Meta-Ontologie beschreiben. Auf diese Weise könnte ein Datenanalysewerkzeug Verletzungen der Ontologie an der Meta-Ontologie diagnostizieren und implizites Wissen durch Anwendung der Interferenzregeln explizit machen.

In APIUA sind Interferenzregeln wie hypothetische Relationen aktuell fest kodiert. Ideal wäre es, wenn Interferenzregeln selbst vom Anwender definiert und bedarfsweise, beispielsweise während Erkenntnisperspektivwechseln, (de)aktiviert werden könnten. Auf diese Weise müssten keine Gesetzmäßigkeiten mehr postuliert werden, die für alle Theorien gelten. Stattdessen hätte der Forscher die Möglichkeit, für seine GT individuelle Regeln zu formulieren.

Ein ganz konkreter Anwendungsfall für individuelle Interferenzregeln könnte wie folgt lauten: Für fünf von sechs Unterkodes von Kode ● A gilt die Relation ● A<sub>x</sub> → ● B. Das Werkzeug könnte nun dem Anwender vorschlagen, zu prüfen, ob diese Relation möglicherweise auch für den sechsten Unterkode von ● A zutrifft und damit auf ● A → ● B verallgemeinert werden kann. Das Werkzeug sollte sich darüber hinaus die Grundlage dieser Entscheidung merken. Sollte später ein siebter Unterkode zu ● A hinzukommen, gäbe es zwei Möglichkeiten: (1) Der Forscher prüft selbstständig, ob ● A → ● B auch für ● A<sub>7</sub> → ● B gilt. (2) Das Datenanalysewerkzeug erkennt die veränderte Datenlage und fragt den Anwender aktiv. Die zweite Möglichkeit würde eine Umstrukturierungsoperation im Sinne der in Tabelle 3.2 aufgezählten Anforderungen darstellen.

Die Krux ist, dass der Forscher im Sinne des *ständigen Vergleichens* und der Güte seiner Forschungsergebnisse dazu verpflichtet ist, die im Beispiel skizzierten Überlegungen anzustellen. Ich halte es aber für unwahrscheinlich, dass der Forscher diesem Anspruch bei hinreichend komplexen Abhängigkeiten innerhalb seiner Theorie ohne Werkzeugunterstützung genügen kann.

Diese von mir vorgeschlagenen Verbesserungen sind allerdings mit Vorsicht zu genießen, denn sie sind nicht ausgiebig erprobt. Im Rahmen der in dieser Arbeit präsentierten Forschung haben die implementierten Funktionen (Vererbung von Verankerungen; hypothetische Relationen; siehe Abschnitt 3.5.3.2) zwar das axiale Kodieren deutlich vereinfacht. Dennoch kann man zu bedenken geben, dass allein der von einem Werkzeug stammende Vorschlag zur Änderung des eigenen Theoriemodells den Anwender auf eine Weise beeinflusst, auf die er ohne diese Hilfestellung nicht beeinflusst worden wäre. Ich kann nicht ausschließen, dass der unreflektierte Umgang mit derartigen Funktionen zu einer gewissen Starrheit/Rigidität auf Seiten des Forschers führen kann.

Meine ontologische Auseinandersetzung mit der GTM an dieser Stelle war vornehmlich technisch gemeint. Sie möchte zum einen die Möglichkeit geben — insbesondere im Rahmen von Open Science — Forschungsergebnisse menschen- aber auch maschinenlesbar zur Verfügung zu stellen. Das Potential dieses Vorgehens stellt auch Mühlmeyer-Mentzel (2011) heraus. Zum anderen möchte sie als Vorschlag für den Bau einer besseren Datenanalysesoftware verstanden werden. Die Abschätzung der Konsequenzen für eine derart entwickelte GT kann diese Arbeit nicht leisten. Allein das Feld der Methode der Grounded Theory mit seinen verschiedenen Strömungen (Breckenridge et al. 2012), wie der klassischen GTM von Glaser (1978), der von Strauss u. Corbin (1990) oder der konstruktivistischen GTM von Charmaz (2006) ist dafür zu breit.

### 3.4.7 Zusammenfassung

In diesem Unterkapitel habe ich das für meine Forschung entwickelte Datenanalysewerkzeugs APIUA vorgestellt.

Qualitative Datenanalysewerkzeuge müssen große Datenmengen, verschiedenste Datenformate und alle Phasen bzw. Praktiken der GTM unterstützen, um eine GT von hoher Qualität nicht zu gefährden. Um einen wirklichen Mehrwert zu erzielen, müssen ebenso Umstrukturierungsoperationen und Analysefunktionen angeboten werden. Die Interoperabilität der Forschungsergebnisse erlaubt die Weiterverarbeitung durch andere Akteure der Forschungsgemeinde.

APIUA verwendet zur Erfüllung dieser Anforderungen eine komponentenbasierte Drei-Ebenen-Architektur. Komponenten werden durch den Einsatz der RCP, welche wiederum auf OSGi<sup>G</sup> basiert, unterstützt. Das Werkzeug ist allgemein, aber insbesondere mit Hinblick auf die Unterstützung weiterer Datenformate, außerordentlich erweiterbar. Die GTM-Komponente arbeitet ausschließlich mit URIs, durch die jeder Datenpunkt eindeutig identifiziert wird. Auf diese Weise werden Funktionen wie das Kodieren von Daten oder das Schreiben von Memos einheitlich implementiert. Außerdem ist so das Werkzeug bei der Erfüllung der Gütekriterien *Argumentative Interpretationsabsicherung* und *Nähe zum Gegenstand* behilflich.

APIUA erlaubt die Arbeit mit hoch strukturierten Daten. Die Anwendung wurde mit dem Anspruch entwickelt, selbst benutzerfreundlich zu sein. Dazu gehört unter anderem eine frei konfigurierbare Benutzeroberfläche, die ihren Zustand detailliert über mehrere Forschungssitzungen speichert und nicht — im Gegensatz zu anderen Werkzeugen — immer wieder neu hergestellt werden muss.

Im Unterschied zu *ATLAS.ti* können in APIUA Kodes einfach gefiltert, intuitiv farbkodiert, sortiert und hierarchisch angeordnet werden. Trotz der überschaubaren Funktionalität wird axiales Kodieren in APIUA in einem konkurrenzlosen Umfang unterstützt.

Das Werkzeug unterstützt fest implementierte Interferenzregeln, die besonders beim axialen Kodieren hilfreich sind und bei *ATLAS.ti* in keiner Weise angeboten werden. Eine Möglichkeit, frei konfigurierbare Interferenzregeln zu erlauben, erachte ich für außerordentlich wünschenswert. Eine Möglichkeit, dies zu erlauben, besteht darin, mit einer Meta-Ontologie zu arbeiten. Ich biete die Grundlage, auf der die zukünftige Forschung meine Überlegungen untersuchen kann.

Abgesehen vom Umbenennen unterstützt *ATLAS.ti* keinerlei Umstrukturierungsoperationen. In der aktuellen Version bietet APIUA jedoch einige Funktionen, wie Operationen am Kodebaum und die Verallgemeinerung bzw. Zusammenfassung von Relationen. Umstrukturierungsfunktionen wie das Zusammenfassen oder Auf trennen von Kodes fehlen beiden Werkzeugen.

Weitergehende Analysefunktionen fehlen beiden Anwendungen ebenfalls. Eine Funktion zum Anzeigen von Kodes, die nur in wenigen Datenquellen auftauchen, wäre außerordentlich spannend. Mit dieser Information könnte der Forscher beispielsweise *theoretisches Sampling* besser betreiben — also exakter auswählen ob und welche Daten er noch erheben möchte, was wiederum der Erfüllung des Gütekriteriums *Triangulierung* zuträglich wäre.

Der nächste Abschnitt befasst sich mit der eigentlichen GTM-Analyse unter Verwendung des eben beschriebenen qualitativen Datenanalysewerkzeugs API Usability Analyzer.



## PHASE 4: VERFAHREN DER ANALYSE DER API-USABILITY VON SEQAN MIT HILFE DER METHODE DER GROUNDED THEORY

Wie bereits mehrfach — insbesondere in den Abschnitten 1.4 und 3.1.3 — erläutert, habe ich für meine Forschung die Methode der Grounded Theory (GTM) eingesetzt. Dazu habe ich ein spezielles Datenerhebungsverfahren (siehe Abschnitt 3.3) und ein dazu passendes Datenanalysewerkzeug (siehe Abschnitt 3.4) entwickelt.

Speziell zum Zweck der GTM-basierten Analyse habe ich die Programmierfortschritte von SeqAn-Anwendern erhoben, eine Gruppendiskussion geführt und einen speziell für die Evaluation von APIs entwickelten Cognitive-Dimensions-Fragebogen eingesetzt.

Darüber hinaus habe ich die Ergebnisse aus der Beseitigung grober Usability-Probleme (siehe Abschnitt 3.2) in meine Analyse einbezogen. Zur Erinnerung: Für diese erste Phase kamen eine Heuristische Evaluation (HE) und drei Befragungen (Online-Umfrage, Interviews, Feedback-Zettel) zum Einsatz.

In dieser Phase stelle ich zunächst die Forschungsmethoden anderer Studien vor. Anschließend erläutere ich den Verlauf meiner Forschung an Hand der verschiedenen in der vorherigen Phase erhobenen Daten. Meine Probleme bei der Anwendung der GTM bespreche ich im darauffolgenden Abschnitt. Abschließend demonstriere ich anekdotisch und an konkreten Beispielen meine Forschungsgründlichkeit bevor ich im nächsten Kapitel meine Forschungsergebnisse vorstelle.

### **3.5.1 Vergleich mit anderen Studien**

Die GTM eignet sich besonders gut für die explorative Erforschung schlecht erforschter Gegenstandsbereiche (Mayring 2002). Es handelt sich dabei um einen empirischen Forschungsansatz, dessen Nutzen für die Untersuchung von API-Usability bereits beschrieben wurde (Farooq u. Zirkler 2009). Bereits Brooks (1980) hat die Notwendigkeit erkannt, API-Nutzungsdaten qualitativ und datenverankert zu analysieren und zu interpretieren.

Die Kombination verschiedener Evaluationsmethoden (in dieser Arbeit: HE und GTM) unter Verwendung unterschiedlicher Datenquellen, stellt ein Gütekriterium im Sinne der *Triangulierung* (siehe Abschnitt 1.4 und Mayring 2002) dar und hat sich bereits mehrfach als geeignetes Mittel zur Erlangung eines reichhaltigen Verständnisses bewährt (Boehm et al. 2003; Grill et al. 2012). Hingegen kann der alleinige Gebrauch von klassischen Usability-Evaluationsverfahren (siehe Abschnitt 2.3.1) die Interaktion zwischen API und API-Anwendern nur teilweise erfassen (Gerken et al. 2011). Die Nutzung von Erkenntnissen aus anderen Disziplinen wurde bereits von Shneiderman u. Mayer (1979) gefordert — umso erstaunlicher, dass diesem Ruf die Forschergemeinde kaum nachkam (Glass et al. 2002).

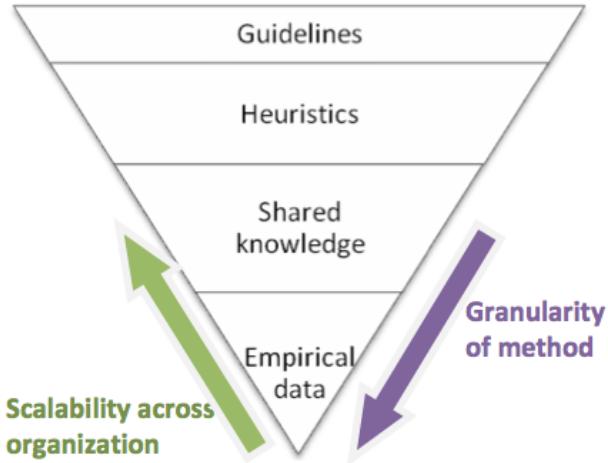


ABBILDUNG 3.39: Skalierbarkeit von API-Evaluationsverfahren (Farooq u. Zirkler 2009)

Empirische Verfahren erlauben sehr tiefe Einsichten, skalieren aber schlecht und werden von Farooq u. Zirkler (2009) nur zur Evaluation eines einzelnen API-Features für geeignet betrachtet (siehe Abbildung 3.39).

Meine Arbeit ist nicht die einzige, bei der der eigentlichen Forschung eine HE vorausging. Eine andere Arbeit (siehe Abschnitt 2.5.2 und Grill et al. 2012) nutzt diesen Ansatz auch — jedoch nicht, um den Untersuchungsgegenstand für die Forschung vorzubereiten, sondern um vorab zu ermitteln, worauf sich die anschließende Erforschung fokussieren sollte. Die von mir durchgeföhrte HE diente jedoch lediglich der Beseitigung grober Usability-Probleme und der nicht-einschränkenden Sensibilisierung für mögliche Probleme.

Die Methode *Concept Maps* (siehe Abschnitt 2.4.9.4 und Gerken et al. 2011) hat sich ebenfalls als geeignet für die explorative Erforschung einer API herausgestellt. Jedoch setzt sie eine sehr hohe Einbeziehung der API-Entwickler voraus, was sich angesichts des im Abschnitt 3.1.4 geschilderten *technischen Wegargumentierens* (Sarodnick u. Brau 2006) als nachteilig herausgestellt haben könnte.

Das *Cognitive Dimensions Framework* wird in der Forschung von Clarke (2005b) zur API-Usability-Evaluation eingesetzt. Wie ich bereits im Abschnitt 2.4.2.4 beschrieben habe, ist die Darstellung des Verfahrens in der Literatur mangelhaft, was sich aber leider erst beim Versuch seiner Anwendung herausgestellt hat. Auch der Versuch, eine kostengünstige Validierung meiner im nächsten Kapitel dargestellten Forschungsergebnisse durchzuführen, ist gescheitert. Dieser Validierungsversuch wird am Ende des nächsten Kapitels (siehe Abschnitt 4.5.2.1) erläutert.

In einer Studie zur Verbesserung einer Persistenz-Bibliothek (siehe Abschnitt 2.5.3 und Piccioni et al.) wurden die kognitiven Dimensionen für APIs von Clarke u. Becker (2003) für die Vorbereitung von Interviewfragen genutzt. Leider ist die Zustandekommen fraglich, weil es nicht nachvollziehbar dargestellt wurde (siehe Abschnitt 3.3.4).

Wie bereits am Anfang dieser Arbeit gezeigt (siehe Abschnitt 1.4.1), wurde die GTM nur in drei mir bekannten und lediglich mittelbar für API-Usability-Forschung relevanten Studien (Yamashita 2012; Yamashita u. Moonen, 2013) korrekt eingesetzt. Dabei kamen “lediglich” die Phasen *offenes Kodieren*

und *axiales Kodieren* zum Einsatz. In der unmittelbar relevanten API-Usability-Forschung setzten Gerken et al. (2011); de Souza et al. (2004); Sunshine et al. (2014) die GTM nur geringfügig und mangelhaft bzw. kaum nachvollziehbar ein.

### 3.5.2 Analyse mit Hilfe der Methode der Grounded Theory

Wie bereits am Anfang dieses Kapitels (siehe Abschnitt 3.1.3) beschrieben, wichen meine Forschung stark von der ursprünglichen Planung ab. Die ohnehin spezielle Art der Datenerhebung und der erhobenen Daten selbst, machte die Entwicklung eines dafür geeigneten qualitativen und GTM-unterstützenden Datenanalysewerkzeugs notwendig. Die Datenanalyse und die daraus gewonnenen Erkenntnisse standen in ständiger Wechselwirkung mit den Datenerhebungen und der Werkzeugentwicklung. Beispielsweise stellte ich bei der Datenanalyse fest, dass ich nicht sehen konnte, welche Begriffe die SeqAn-Anwender in das Suchfeld der Onlinedokumentation eingegeben hatten, woraufhin ich die Datenerhebung anpassen musste (Details siehe Abschnitt 3.3.5.2). Wiederum haben die ständigen Verbesserungen von Datenerhebung und Analysewerkzeug Zeit gekostet. Der so entstandene Zeitdruck wirkte sich auf die Analysetiefe, die Auswahl der zu analysierenden Daten und schließlich auch auf die in Kapitel 4 vorgestellten Ergebnisse aus.

#### 3.5.2.1 Analyse der Programmierfortschritte-Daten

Für den ersten Analyseversuch verwendete ich den API Usability Analyzer (APIUA) in Verbindung mit den während der Workshops'11 erhobenen Programmierfortschritte-Daten. Die Ergebnisse dieses Versuchs sind in Abbildung 3.40 dargestellt. Als damaliger GTM- und API<sup>G</sup>-Usability-Evaluations-Anfänger musste ich nach insgesamt etwa drei Monaten feststellen, dass mich meine Forschung nur wenig voran gebracht hatte. Wie die Abbildung zeigt, habe ich die Programmierfortschritte unter verschiedenen Gesichtspunkten — beispielsweise ● operation und ● purpose — analysiert.

Ganz erkenntnislos war der erste Analyseversuch jedoch nicht. So deuteten die Kodes ● correctly pasted snipped und ● paste code bereits auf das Konzept der ● Wiederverwendung hin.

Während der Analyse vollzog ich typischerweise die folgenden Schritte, die in Abbildung 3.41 veranschaulicht werden. Die folgenden Schritte beziehen sich auf den Kode ● correctly pasted snipped:

1. Zunächst habe ich einen Proband ausgewählt, dessen Programmierfortschritte ich analysieren wollte. Der entsprechende Anzeigebereich führt alle Probanden an Hand ihrer ID und ihrer Browser-Fingerprints (Details siehe Abschnitt 3.3.5.1) auf. In diesem Beispiel wurde Proband *0meio6dz3eo1wj7* geöffnet.
2. Die Programmierfortschritte des Probanden können auf zweierlei Weise erkundet werden.
  - (2a) zeigt sämtliche Aktionen des Probanden auf einer Zeitleiste:
  - (2b) beschränkt sich auf die Darstellung der Fortschritte beim Programmieren.

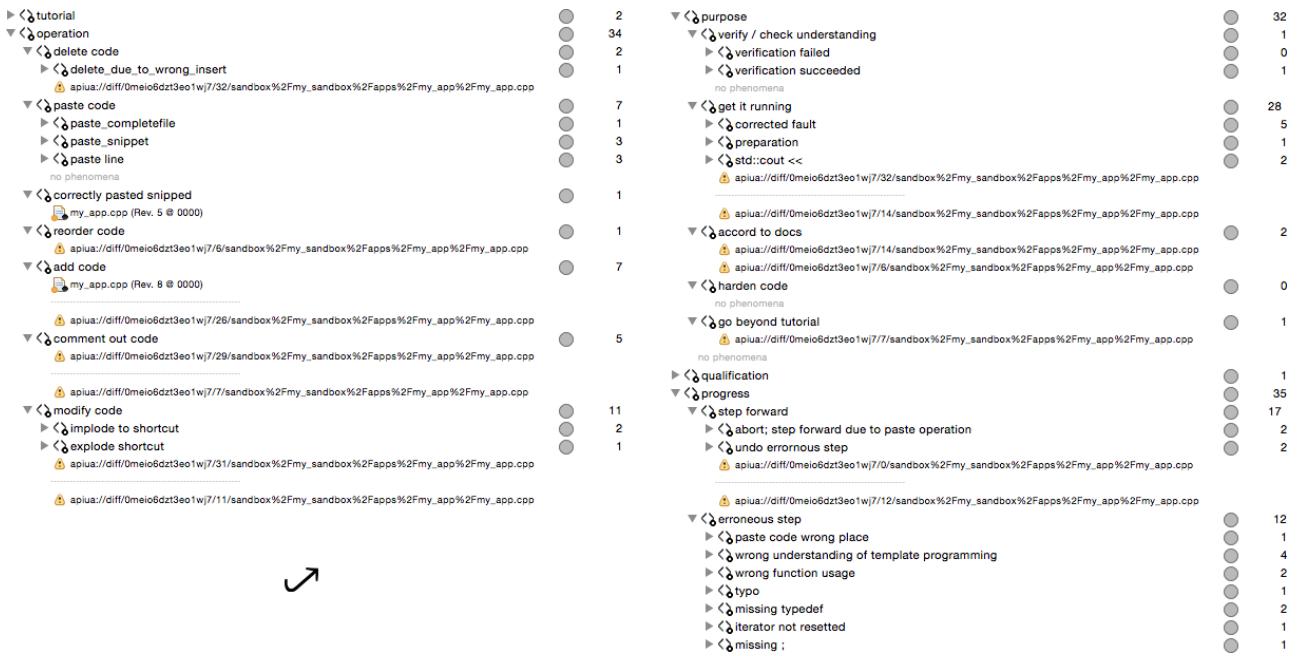


ABBILDUNG 3.40: Ergebnisse der ersten Analyseversuchs: Die Abbildung zeigt die hierarchische Ko-destruktur die bei dem ersten Versuch einer Analyse entstand. Aus technischen Gründen ist der Baum in zwei Spalten aufgetrennt. Innerhalb einer Spalte gibt es drei Spalten, von denen die erste den Namen des Kodes bzw. deren Phänomene/Verankerungen darstellt. Die grauen Linien geben an, dass aus Darstellungsgründen weite Phänomene ausgeblendet sind. Die tatsächliche Anzahl von Phänomenen steht in der dritten Spalte. Die zweite Spalte gibt die Farben der Kodes. Sie ist hier grau, weil diese Kodes nicht weiter verwendet wurde.

In diesem Beispiel interessierte mich die Datei `my_app.cpp` der *Revision 5*, die den fünften Kom-pilierversuch symbolisiert.

3. Die Diff-Ansicht stellt die vorangegangene und aktuelle Version der Datei `my_app.cpp` gegen-über. Offenbar wurden sehr viele Änderungen vorgenommen. Eine derartige, fehlerfrei wirkende Operation ist für einen SeqAn-Anfänger ungewöhnlich. Dafür musste es also eine Erklärung geben.

4. Der Bereich (4a) stellt listenartig alle Ereignisse auf der Online-Dokumentation dar. Der Be-reich (4b) stellt dieselben Informationen in der untersten Zeile der Zeitleiste dar. Der bläuliche Einfärbung auf der Zeitleiste markiert dabei den Zeitraum des Workshops. Die neongrüne Einfärbung hebt sämtliche Ereignisse hervor, die in die Zeitspanne der Arbeiten an der fünften Revision der Datei `my_app.cpp` fallen.

Bleibt der Forscher mit der Maus über einem Ereignis stehen, werden detaillierte Informationen in (5) dargestellt.

5. Der Detaildialog ist ein nicht-modales Fenster, wie man es aus der Eclipse-Entwicklungsumgebung, beispielsweise bei der Codevervollständigung kennt. In diesem Fall zeigt das Fenster an, was genau der Proband in diesem Moment auf der Online-Dokumentation gesehen hat. Der blasse Pfeil im Hintergrund des Screenshots bedeutet, dass der Anwen-der herunter gescrollt ist, um den dargestellten Bereich zu sehen. Der Screenshot zeigt ein SeqAn-Online-Tutorial. Durch den Vergleich des Tutorial-Inhalts mit dem, was der Anwender

programmiert hat, wird deutlich, dass er zwei Code-Fragmente aus dem Tutorial kopiert und in seiner C++-Datei eingefügt hat.

6. Die beiden Fragmente habe ich mit dem Kode  correctly pasted snipped kodiert. Da die Koddeansicht alle existierenden Kodes darstellt, bietet es sich an dieser Stelle an, die Kodierung mit den bereits kodierten Phänomenen im Sinne des *ständiges Vergleichens* gegenüberzustellen, um zu klären, ob es sich wirklich noch um die Semantik des Kodes handelt.
7. Erkenntnisse können in der Memoansicht festgehalten werden. In diesem Fall war das Memo bereits in Kode  paste\_snippet festgehalten, der mit  correctly pasted snipped zusammengefasst werden sollte.
8. Die Kodierung der fünften Revision der Datei `my_app.cpp` ist abgeschlossen. Die Programmierfortschritte-Ansicht kann nun verwendet werden, um die sechste Revision zu kodieren.

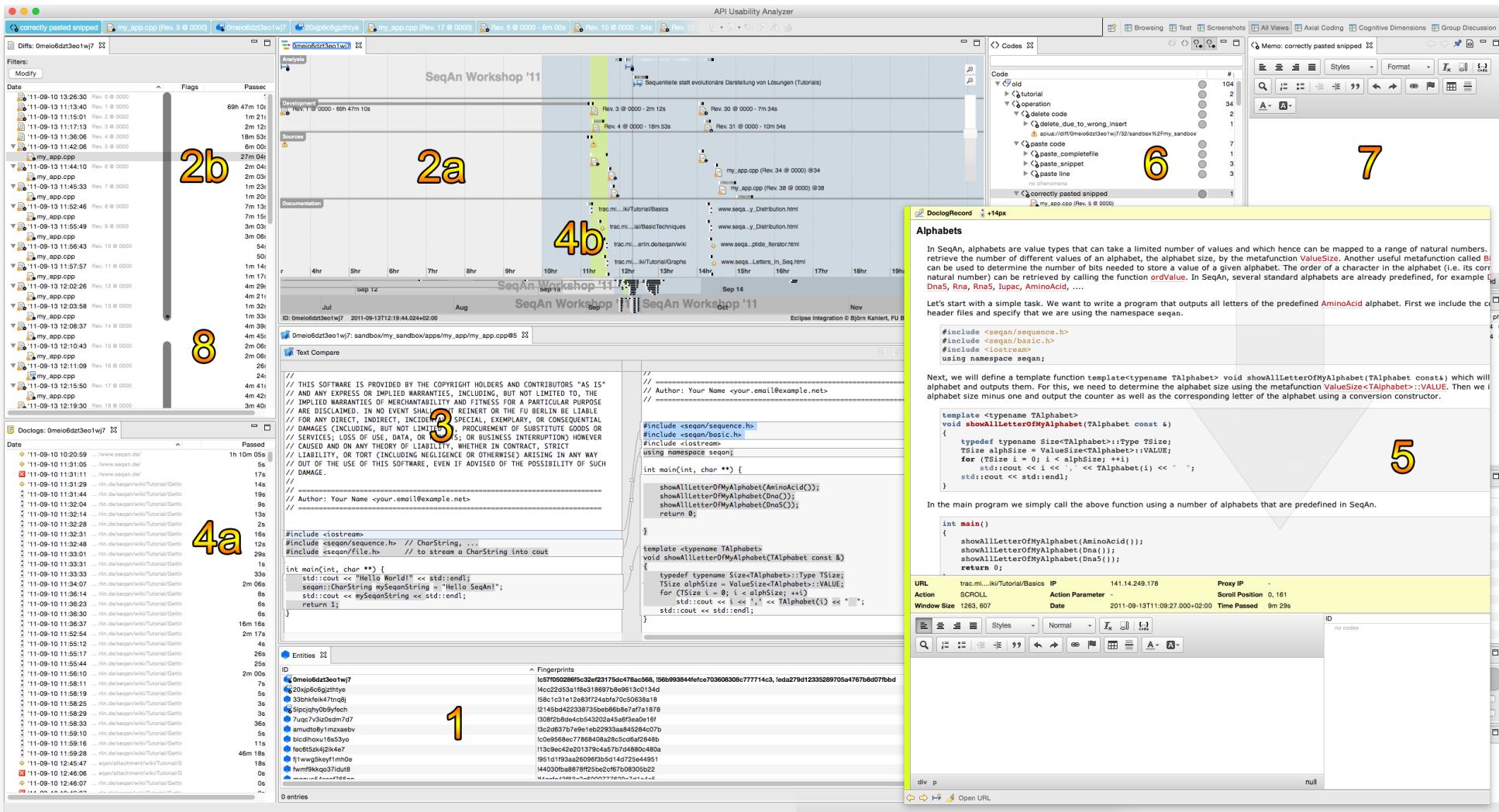


ABBILDUNG 3.41: Analyse von Programmierfortschritte-Daten mit Hilfe von APIUA: (1) Auswahl des Probanden, dessen Programmierfortschritte analysiert werden sollen. (2a) zeigt die Programmierfortschritte entlang einer Zeitleiste. (2b) beschränkt sich auf eine Listendarstellung. (3) stellt die vorangegangene und aktuelle Version der selektierten Datei gegenüber. (4a) stellt listenartig alle Ereignisse auf der Online-Dokumentation dar. (4b) Dasselbe macht unterste Zeile der Zeitleiste. (5) stellt detaillierte Informationen zu einem Datenpunkt dar. (6) zeigt die existierenden Kodes. In (7) werden Erkenntnisse zum aktuell selektierten Element festgehalten.

Wurde die betrachtete Datei kodiert, kann der Forscher in (8) fortfahren.

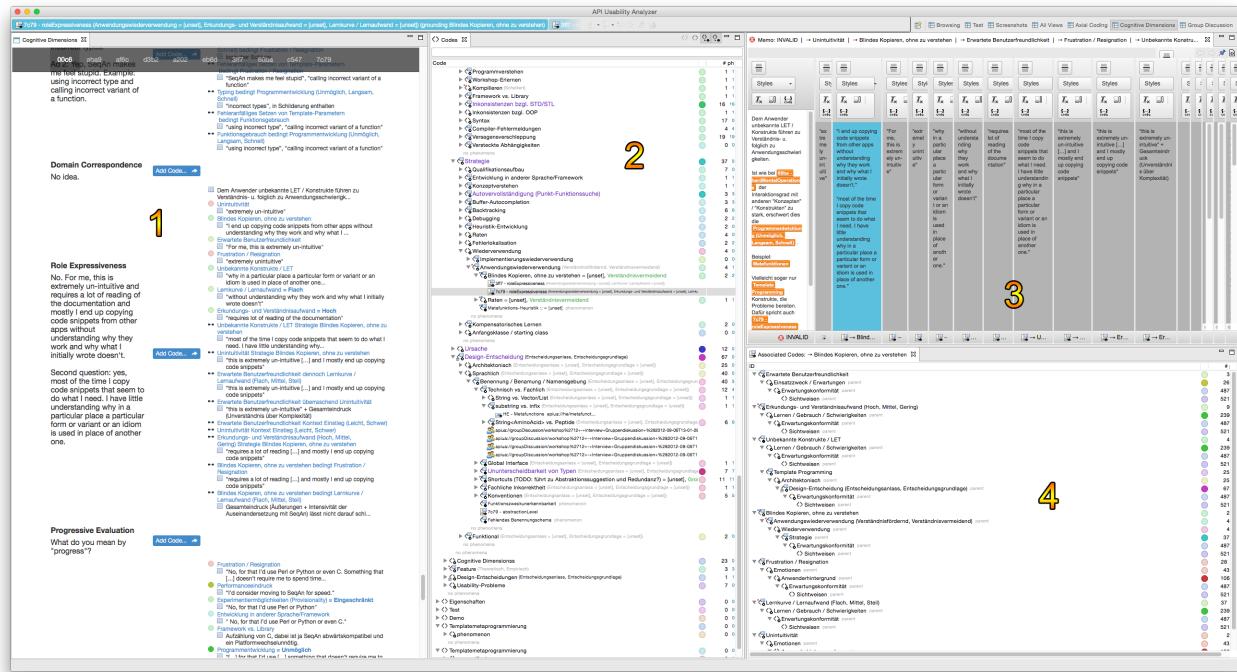


ABBILDUNG 3.42: Analyse der Cognitive-Dimensions-Fragebögen mit Hilfe von APIUA: (1) Dieser Bereich stellt alle Antworten der Fragebögen dar. (2) zeigt alle gefundenen Kodes. Neben einem Kode ist dessen Dimensionalisierung dargestellt. Unterhalb eines Kodes sind die Fundstellen/Phänomene aufgelistet. (3) stellt das Memo des aktuell selektierten Elements (blau hervorgehoben), das Memo des zur Selektion gehörenden Kodes (weiß) und die Memos der anderen dem Phänomen zugewiesenen Kodes (grau) dar. (4) listet die zugewiesenen Kodes und deren Elternkode-Hierarchie auf.

Auch wenn man auf diese Weise sehr gründliche Analysen durchführen kann, erfordert es viel Übung, ein gewisses Abstraktionsniveau zu erreichen, welches dem selbst gesteckten Ziel genügt. In meinem Fall war dies die Erforschung und insbesondere die Verbesserung der API-Usability von SeqAn.

Um meine Sensibilität zu erhöhen und eine effizientere, gezieltere Analyse der objektiven Programmierfortschritte-Daten zu erreichen, habe ich mich entschlossen, zunächst mit der Analysen der subjektiven Datenquellen *Cognitive-Dimensions-Fragebogen* und *Gruppendiskussion* fortzufahren. Immerhin konnten die Forscher der API-Usability-Studie von Grill et al. (2012) 55% der entdeckten Usability-Probleme in der direkten Interaktion mit Benutzern (insb. Interviews) finden.

### 3.5.2.2 Analyse der Cognitive-Dimensions-Fragebögen

Die im Abschnitt 3.3.4 vorgestellten Cognitive-Dimensions-Fragebögen kamen am Ende des Workshops'13 zum Einsatz. Abbildung 3.42 zeigt, wie die entsprechende Analyseansicht in APIUA aussieht. Der Kodierprozess ist dem im vorangegangenen Abschnitt ähnlich genug, um ihn an dieser Stelle kein zweites Mal zu erläutern.

Die Analyse der Fragebögen war weitaus ergiebiger, auch wenn sie mehrere Monate umfasste. In keiner Datenquelle fand ich so viele Informationen zur Kategorie ● **Funktionsbezogene Probleme** (u.a. ● **Fehlende Funktionskategorisierung** und ● **Funktionszweckunerkenntbarkeit**, siehe 268) wie in dieser.

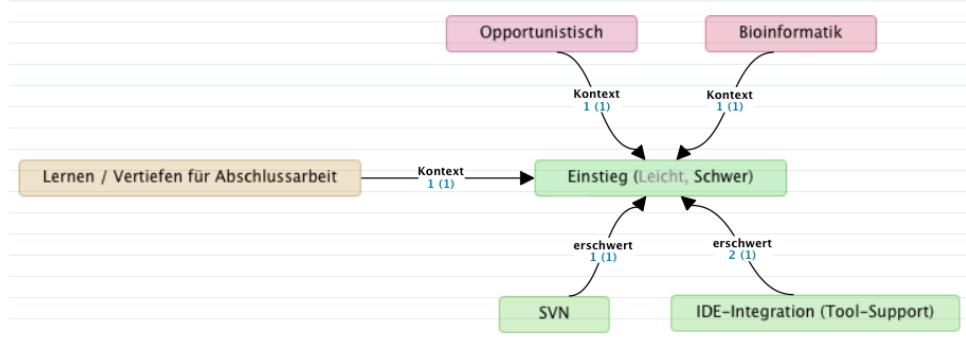


ABBILDUNG 3.43: Einfache kausale axiale Kodierung<sup>8</sup> eines Teilnehmers<sup>8</sup>

An das Beispiel der Wiederverwendung anknüpfend, konnte ich bei der Analyse der Fragebögen den Kode Blindes Kopieren entdecken. Ein solcher Bezeichner klingt zunächst nach einer mutigen Behauptung, die mit den Programmierfortschritte-Daten — ohne entsprechende Übung — nicht einfach zu belegen wäre. In dieser subjektiven Datenquelle fiel das deutlich leichter. So hat beispielsweise ein Proband zu der Frage zur CD *Rolenerkennbarkeit* gesagt: “[Writing code] is extremely un-intuitive [...] and mostly I end up copying code snippets from other apps without understanding why they work”<sup>8</sup>.

Manche Fragen wurden von Teilnehmer nicht so verstanden, wie ich das beabsichtigte (Details siehe Abschnitt 3.3.4). Das stellt zwar Forscher vor ein Problem, wenn sie den Fragebogen im Sinne des Cognitive Dimensions Frameworks (siehe Abschnitte 2.3.2 und 2.4.2.4) auswerten wollen. Für die GTM-Analyse ist dies jedoch unproblematisch, geben die Befragten ja dennoch relevante und wertvolle Informationen.

Bis eben habe ich mich lediglich auf das *offene Kodieren* beschränkt. Für das *axiale Kodieren* waren die Fragebögen weniger geeignet, denn ihnen fehlt weitgehend der Prozesscharakter, der in den Programmierfortschritte-Daten inherent vorhanden ist. Bei den Fragebögen haben die meisten Antworten einen resümierenden, momentbezogenen Charakter, was axiales Kodieren im Sinne des *paradigmatischen Modells* (siehe Abschnitt 1.4) erschwert. Viele meiner erarbeiteten *axialen Kodierungen* geben daher auch eher ein Résumé bzw. einen Kausalitätsgraphen wieder. Abbildung 3.43 zeigt ein entsprechendes einfaches Beispiel.

In Abbildung 3.42 (Bereich 2) kann man bereits erahnen, wie erkenntnisreich die zehn analysierten Fragebögen waren. Ich entschloss mich also, mit der Analyse der zweiten subjektiven Datenquelle *Gruppendiskussion* fortzufahren.

### 3.5.2.3 Analyse der Gruppendiskussion

In der im Abschnitt 3.3.3 vorgestellten und beim Workshop’12 durchgeführten Gruppendiskussion verwendete ich das Programmierparadigma *Templatemetaprogrammierung* als Haupttreiz. Weitere Reizargumente habe ich hauptsächlich auf der Grundlage von zuvor ausgegebenen Feedback-Zetteln erarbeitet (Details siehe Abschnitt 3.3.3).

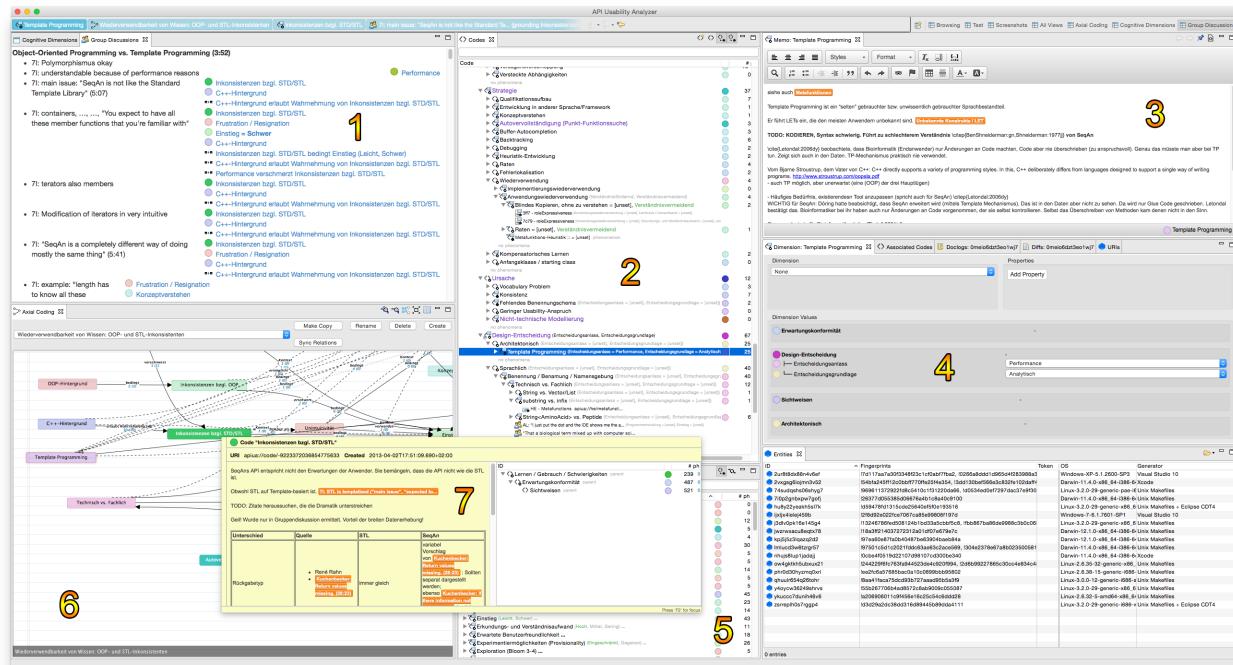


ABBILDUNG 3.44: Analyse der Gruppendiskussion mit Hilfe von APIUA: (1) zeigt das informelle Transkript der Gruppendiskussion. (2) zeigt die gefundenen Konzepte. (3) zeigt das Memo zur **Template metaprogrammierung**. (4) zeigt die von **Entwurfsentscheidung** gererbten Eigenschaften von **Template metaprogrammierung**: **Entwurfsentscheidungsmotivation** und **Entwurfsentscheidungsgrundlage**. (5) zeigt die gefundenen Relationen, wie z.B. **Inkonsistenzen bzgl. STL**  $\rightarrow$  **Einstieg (schwer)**. (6) zeigt ein relevantes axiales Kodiermodell<sup>63</sup>. (7) stellt in einem schwelbenden Dialog das Memo zum gerade selektierten Kode **Inkonsistenzen bzgl. STL** dar.

Neben den zu beobachtenden Wiederverwendungsstrategien gaben die Cognitive-Dimensions-Fragebögen umfangreich Aufschluss über unterschiedliche Facetten zur, in SeqAn verwendeten **Template metaprogrammierung**. Ein zuvor unentdeckter Aspekt sind **Inkonsistenzen bzgl. STL** (*STL* steht für die *C++ Standard Template Library*<sup>63</sup>), die einen Schwerpunkt der Diskussion selbst und meiner mehrmonatigen Analyse darstellte. Die gewonnenen Erkenntnisse sind ein zentraler Punkt meiner im nächsten Kapitel vorgestellten GT, weshalb ich an dieser Stelle nicht weiter darauf eingehe.

Abbildung 3.44 zeigt die APIUA-Perspektive zur Analyse von Gruppendiskussion und veranschaulicht meine Arbeitsweise mit dem APIUA-Werkzeug.

### 3.5.2.4 Erneute Analyse der Programmierfortschritte-Daten

Mit der neu gewonnenen Sensibilisierung für SeqAn-Usability-Probleme, wollte ich meine erhobenen Programmierfortschritte-Daten ein weiteres Mal analysieren. Wegen des enormen Zeitdrucks (siehe Abschnitt 3.1.4) habe ich nur eine Handvoll bereits beobachteter Usability-Probleme, wie die **Operatoreninkonsistenz**, trianguliert.

63 Diese Bezeichnung ist eigentlich nicht ganz korrekt, denn die Standard Template Library (ohne C++) ist eine von Alexander Stepanov entwickelte Softwarebibliothek für generische Programmierung (Lee u. Stepanov 1994), die 1998 in die C++ Standard Library aufgenommen wurde (ISO/IEC 14882:1998).

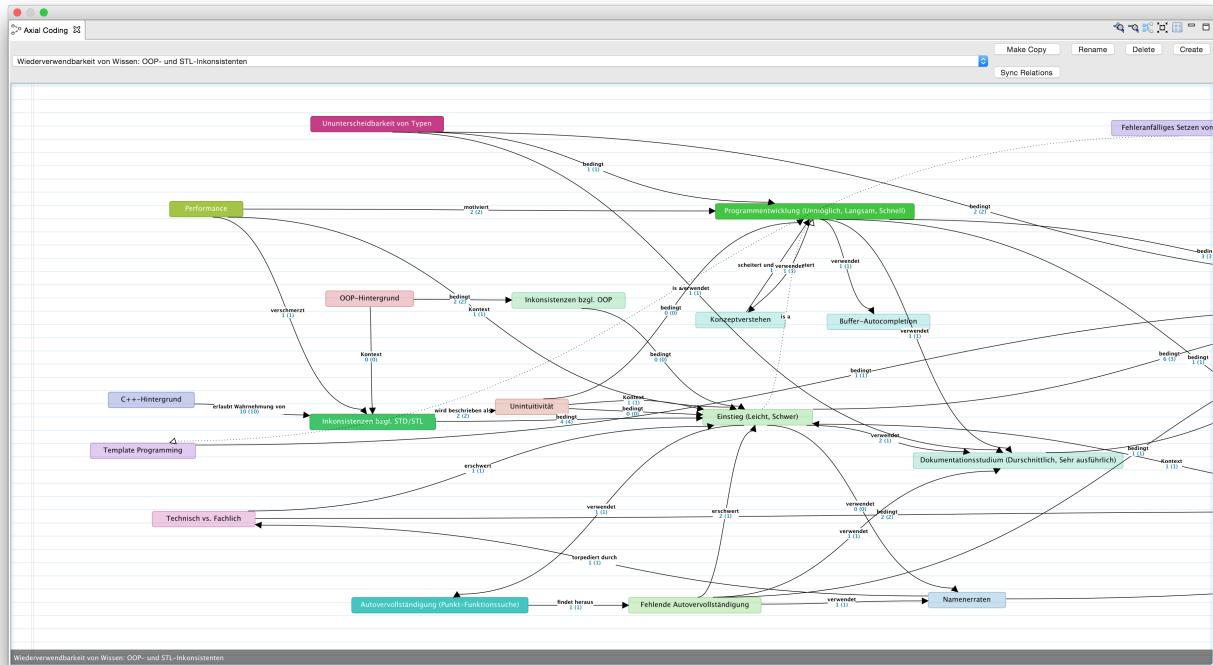


ABBILDUNG 3.45: Axiale Kodierung des Konzepts ● Inkonsistenzen bzgl. STL

Auf die Analyse der 1.200 Revisionen umfassenden Langzeitbeobachtung musste ich leider ganz verzichten.

Wie die im nächsten Kapitel vorgestellte GT zeigen wird, gab es allerdings auch keine zwingende Notwendigkeit einer zweiten Analyse der Programmierfortschritte. Dies soll nicht bedeuten, dass diese Analyse nicht wertvoll und erkenntnisreich gewesen wäre. Allerdings musste ich mich entschließen, diesen Analyseschritt für zukünftige Forschungsvorhaben aufzusparen.

### 3.5.2.5 Selektives Kodieren

Im Verlauf meiner Forschung analysierte ich fortwährend meine automatisch generierten und händisch angepassten axialen Kodiermodelle, um Erklärungs- und Kausalitätslücken an Hand meiner erfassten Daten zu füllen. Dabei platzte geradezu der Knoten, als ich versuchte, das Konzept ● Inkonsistenzen bzgl. STL axial zu kodieren (siehe Abbildung 3.45). Im Nachhinein würde ich diesen Moment als *abduktiven Blitz*<sup>64</sup> beschreiben.

<sup>64</sup> Strübing (2005) zitiert den Begründer des *abduktiven Schließens* Charles Sanders Peirce wie folgt: "Die abduktive Vermutung <suggestion> kommt uns wie ein Blitz. Sie ist ein Akt der *Einsicht*, obwohl extrem fehlbarer Natur. Zwar waren die verschiedenen Elemente der Hypothese schon vorher in unserem Verstande; aber erst die Idee, das zusammenzubringen, welches zusammenzubringen wir uns vorher nicht hätten träumen lassen, lässt die neu eingegebene Vermutung vor unserer Betrachtung aufblitzen"

Ich stellte mir die Frage, worin sich ● Inkonsistenzen bzgl. STL und ● Inkonsistenzen bzgl. OOP unterscheiden. Mir wurde plötzlich klar, dass die ● Paradigmatische Prägung<sup>65</sup> der Anwender eine elementare Rolle in der Bewertung der API-Usability spielt. In diesem Erklärungsansatz fühlte ich mich durch die Beobachtung bestätigt, dass sich der Gebrauch der Strategie ● Punkt-Funktionssuche<sup>66</sup> ebenfalls damit erklären ließ.

Abstrakt betrachtet, war allen Anwendern gemeinsam, dass sich die Usability von SeqAn signifikant aus den Vorerfahrungen und Vorkenntnissen seiner Anwender ergab, was ich als *Erwartungskonformität* bezeichnet habe, und die Perspektive für das selektive Kodieren darstellte.

Je mehr ich wichtige von weniger wichtigen Konzepten unterschied, desto deutlicher wurde, dass eine ganze Reihe von unempirischen ● Entwurfsentscheidung ursächlich für die von mir beobachteten Usability-Probleme war.

Beim selektiven Kodieren habe ich ein ganzheitliches axiales Kodiermodell entwickelt<sup>67</sup>, das durch einen sehr hohen Komplexitätsgrad gekennzeichnet war (siehe Abbildung 3.46a). Um dieses in seiner Komplexität, aber dennoch sinnerhaltend zu reduzieren, entwickelte ich eine APIUA-Funktion, die es mir erlaubte, Relationen zu abstrahieren, d.h. entlang der Konzept-Hierarchien der verbundenen Konzepte nach oben zu bewegen. Ob es sich dabei um eine legitime Operation handelt, konnte ich durch die APIUA-Funktion der hypothetischen Relationen (siehe Abschnitt 3.4.6.4) beantworten. Auf diese Weise konnte ich in dem Dickicht an gefundenen Relationen solche finden, die zusammengefasst und abstrahiert werden können.

Schienen hypothetische Relationen vollkommen unpassend, deutete dies auf Modellierungsschwächen hin. Beispielsweise waren die Folgen von ● Metafunktionen (● Komplizierte Konstruktion von Typen und ● Fehleranfälliges Setzen von Template-Parametern) fälschlicherweise Unterkonzepte von ● Metafunktionen selbst. Das Verschieben dieser Folgen in den anderen Teilbaum ● Funktionsgebrauch entsprach mehr der in den Daten manifesten Ordnung und reduzierte die Komplexität des zunehmend theoretischen Modells (siehe Abbildung 3.47c). Das Zwischenergebnis nach gut fünf Wochen aufwändigen selektiven Kodierens zeigt Abbildung 3.46b<sup>67</sup>.

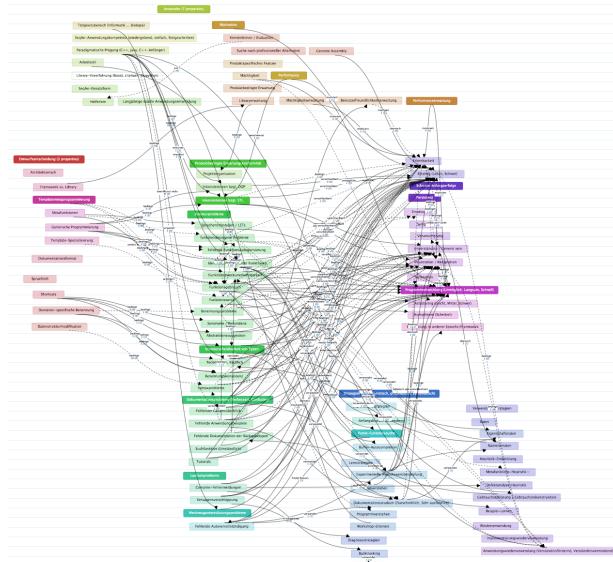
### 3.5.3 Probleme

Neben den bereits genannten Problemen traten auch solche auf, die den Gebrauch der GTM selbst betreffen. Drei Problemklassen stelle ich im Folgenden vor.

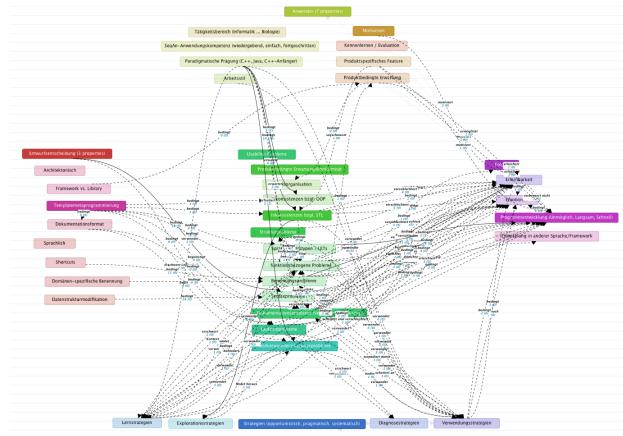
65 Darunter verstehe ich einfach ausgedrückt, welche Paradigmen (objektorientierte Programmierung, Templatemetaprogrammierung, etc.) von einem Anwender auf Grund seiner Ausbildung oder Tätigkeit besonders präferiert und damit besonders gut beherrscht werden.

66 Darunter verstehe ich die Auflistung zur Verfügung stehender Funktionen durch die Eingabe eines Punktes nach einer Variablen innerhalb der eigenen IDE.

67 Leider kam das selektive Kodieren auf derart ungünstige Weise ins Stocken, dass ich die Diagnose und die Behebung des Problems im Anhang A.2 ausführlicher beschreibe.

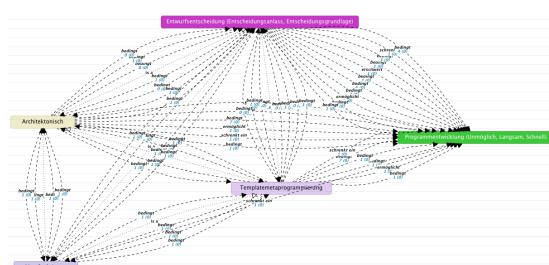


(A) Stand: 15.02.2015

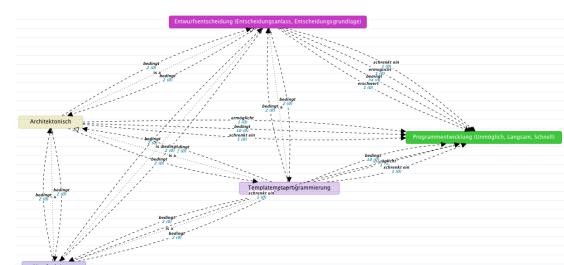


(B) Stand: 24.03.2015

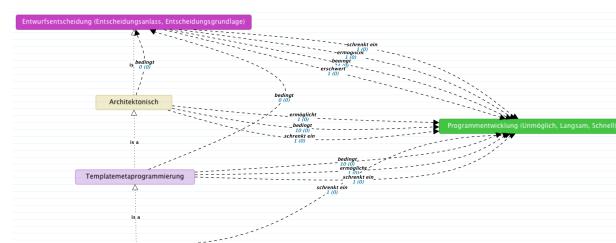
ABBILDUNG 3.46: Selektives Kodieren in APIUA



(A) ohne Zusammenfassung von hypothetischen  
Relationen



(B) mit Zusammenfassung von hypothetischen  
Relationen



(C) nach dem Verschieben von zwei Konzentren

ABBILDUNG 3.47: Selektives Kodieren in APIUA mit Hilfe zusammengefasster hypothetischer Relationen

### 3.5.3.1 Komplexität und Fokus

Empirische Forschungsmethode, zu denen die GTM gehört, werden von Farooq u. Zirkler (2009) nur zu Evaluation eines einzelnen API-Aspektes als geeignet betrachtet. Das bestätigen auch andere Studien (vgl. u.a. Beaton et al. 2008a, b; Ellis et al. 2007; Stylos u. Clarke 2007; Stylos u. Myers 2008). Ich habe versucht, SeqAn in seiner ganzen Breite zu betrachten, was eine Reihe von Aspekten — angefangen bei der ● Anwenderschaft bis hin zu ● Benennungsproblemen — umfasst. Meinen Anspruch, diese Breite in der gegebenen Zeit auch in der selben Tiefe zu betrachten, ist mir nicht vollständig gelungen, wie die Forschungsergebnisse zeigen werden. Darum hatte ich mich entschlossen, mich auf die ● Templatemetaprogrammierung und die Folgen der ● Paradigmatische Prägung paradigmatischen Prägung zu konzentrieren, was sich beispielsweise in der geringen Entdeckung von Eigenschaften anderer Konzepte äußert.

### 3.5.3.2 Modellierung der Ergebnisse

Die Modellierung meiner Theorie fiel mir alles andere als leicht. Durch die APIUA-Unterstützung hierarchischer Konzepte stellte sich häufig die Frage, welches Kriterium als Zerlegungskriterium für Unterkonzepte verwendet werden soll und welche Kriterien als Eigenschaften dienen. Für meine Daten haben sich gleich fünf mögliche Konzepte gebildet, die als Elternkonzepte bzw. Zerlegungskriterium für die möglichen Unterkonzepte dienen konnten:

- ● Erwartungskonformität  
Inwiefern kann existierendes Wissen durch den Anwender wiederverwendet werden?
- ○ Cognitive Dimensions  
Welche kognitiven Dimensionen sind betroffen?
- ● SeqAn-Feature  
Mit welcher Funktionalität/Merkmal von SeqAn hat meine Beobachtung zu tun?
- ● Entwurfsentscheidung  
Welche Art von Entwurfsentscheidung liegt vor?
- ● Usability-Probleme  
Was für eine Art Usability-Problem liegt vor?

Ich glaube, dass dieses Problem nicht nur mit meiner Unerfahrenheit im Gebrauch der GTM und der Möglichkeit, Konzepte hierarchisch anzugeordnen zu tun hat, sondern auch mit der geringen anwendungsbezogenen Strukturierung der GTM-Literatur (insb. Glaser 1978; Strauss 1987; Strauss u. Corbin 1990). In der Dissertation von Salinger (2013) beschreibt der Autor ähnliche Probleme (S. 107 ff.) und formuliert vier Praktiken, die ihm dabei halfen diese Probleme zu lösen (S. 108 ff.). Praktik 1 “Blickwinkel auf die Daten” hätte mein Modellierungsproblem sicherlich geschränkt, indem mir bewusster gewesen wäre, dass sich mein Blickwinkel auf die Daten ändert und welchen ich gerade habe.

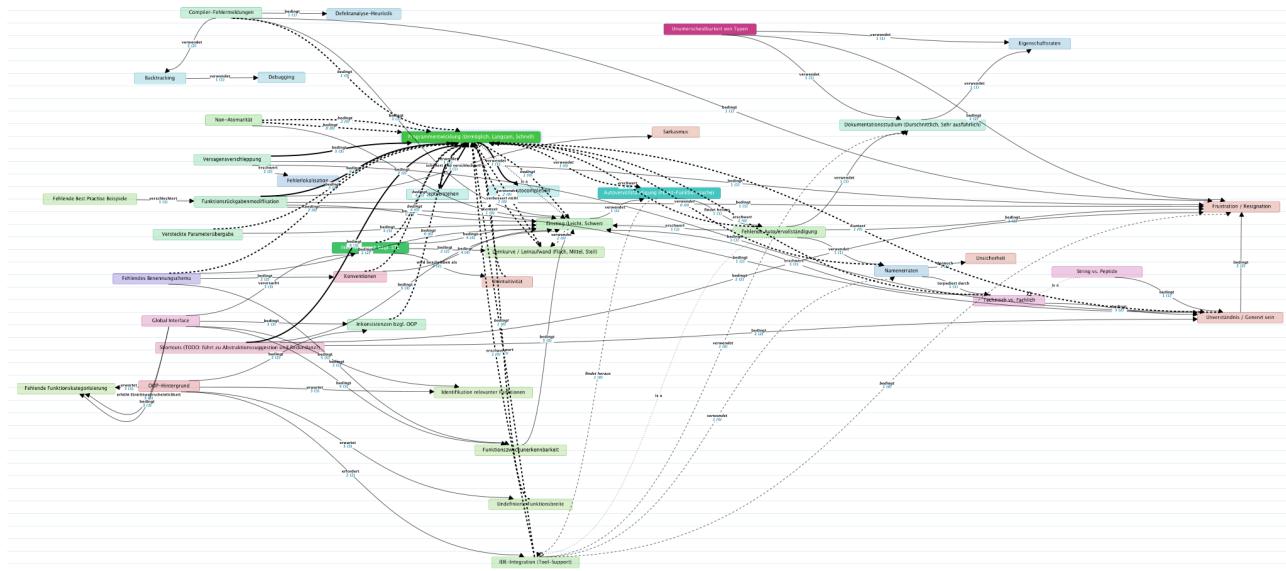


ABBILDUNG 3.48: Axiales Kodiermodell zur Gruppendiskussion<sup>3</sup>

Inhaltlich gesehen gibt es leider keine umfassenden API-Usability-Taxonomien (Daughtry et al. 2009a), die meine Modellierung vereinfacht haben könnten. Allerdings konnte ich für das oben bereits genannte Konzept ● Entwurfsentscheidung auf eine entsprechende Taxonomie von Robertson (2007) zurückgreifen. Taxonomien für Usability-Probleme werden von Keenan et al. (1999); Khajouei et al. (2011) vorgeschlagen. Beide Taxonomien wurden empirisch, allerdings ausschließlich durch die Auswertung grafischer Benutzeroberflächen mit textuellen Komponenten entwickelt. Einzig Grill et al. (2012) haben sich speziell mit API-Usability-Problemen befasst und für deren Unterteilung die vier Kategorien *Dokumentation*, *Laufzeit*, *Struktur* und *Benutzererlebnis* verwendet.

Ein weiteres Problem ist, dass bestimmte Werte für Eigenschaften gemeinsam mit dem Konzept wiederum ein Konzept bilden können. Beispiel: ● Entwurfsentscheidung haben die Eigenschaft ● Entwurfsentscheidungsgrundlage, welche die Ordinalskala (*implizit, explizit-intuitiv, explizit-argumentativ, explizit-empirisch*) verwendet. Implizite und explizite-intuitive Entwurfsentscheidungen bilden in meiner Theorie das Konzept ○ Bauchgefühlsentscheidung. Inhaltlich ist das nachvollziehbar und kanonisch. Jedoch ist die technische Realisierung schwierig und führt zu einer weiteren Abstraktionsebene innerhalb einer Grounded Theory (GT), weshalb ich diese Funktion nicht implementiert habe.

Axiale Kodiermodelle können sehr groß werden, wie die Abbildung 3.48 veranschaulicht. Solche Modellierungen auf das Wesentliche zu reduzieren, stellte mich vor große Probleme. Bei deren Reduktion halfen mir das APIUA-Werkzeug durch die Darstellung impliziter und hypothetischer Relationen, auf die ich im Abschnitt 3.4.6.4 auf Seite 222 eingegangen bin.

### 3.5.3.3 Validität

Während die Programmierfortschritte-Daten in situ sind, handelt es sich bei den Cognitive-Dimensions-Fragebögen um Ex-post-facto-Daten, die eine andere Perspektive für die axiale Kodierung

im Sinne des paradigmatischen Modells erfordern. Dariüber musste ich mir erst durch mehrere Analyse-anläufe bewusst werden. Das betrifft insbesondere das axiale Kodieren von mehr als einem Fragebogen: Lässt sich beispielsweise in einem Fragebogen die Relation A → B und in einem anderem Fragebogen die Relation B → C finden, kann daraus nicht ohne Weiteres auf einen Zusammenhang A → C geschlossen werden. Dazu würde ein gutes Verständnis vom Kontext vorliegen müssen, der den Fragebögen häufig aber kaum zu entnehmen ist.

Im Gegensatz zu den Programmierfortschritte-Daten, sind die subjektiven Datenquellen häufig ärmer an Kontextinformationen. Beide Datenquellen waren aber überraschend breit in ihrem Aussagegehalt, was dazu führte, dass ich viele Konzepte entdecken, aber nur wenige Verankerungen/Phänomene dazu finden konnte. Dies führte mich zu der Frage, welche Aussagekraft überhaupt die Anzahl der zu einem Konzept gefundenen Phänomene haben. In einer GTM der zweiten Generation (Charmaz 2006) können textuelle Daten explizit Wort-, Zeilen- und Absatz-weise kodiert werden, was massiven Einfluss auf die Anzahl gefundener Phänomene hat. Entsprechend eignet sich dieses quantitative Maß nur bedingt nur Diskussion der Validität.

### 3.5.4 Zusammenfassung

In dieser Phase 4 habe die eigentliche Forschung mit Hilfe der GTM und meinem Datenanalysewerkzeug APIUA besprochen. Dazu habe ich zunächst die Forschungsmethoden anderer Studien vorgestellt. Dabei bin ich auf die Eignung und die seltene Verwendung der GTM für explorative Studien eingegangen. Ich habe den unterschiedlichen Gebrauch der HE erläutert und bin auf die Schwächen anderer Forschungsmethoden eingegangen.

Anschließend habe ich die Analyse der Programmierfortschritte-Daten, der Cognitive-Dimensions-Fragebögen und der Gruppendiskussion beispielhaft skizziert und die konkrete Umsetzung innerhalb von APIUA dargestellt.

In meiner Beschreibung des selektiven Kodierens, habe ich erläutert, welchen Beitrag die Konzepte Inkonsistenzen bzgl. STL und Paradigmatische Prägung beim Durchbruch meiner Forschung hatten und welche APIUA-Funktionen mir bei der Reduktion des “globalen” axialen Kodiermodells auf ein theoretisches Kodiermodell behilflich waren.

Weiterhin bin ich auf Komplexitäts-, Modellierungs- und Validitätsprobleme beim Gebrauch der GTM eingegangen.



## ZUSAMMENFASSUNG DER FORSCHUNG

In diesem Kapitel habe ich die Erforschung der API-Usability von SeqAn vorgestellt. Dazu bin ich zunächst auf die Rahmenbedingungen, sowie auf die ursprünglich geplante und die tatsächlich durchgeführte Forschung eingegangen.

In Phase 1 habe ich die Notwendigkeit und Durchführung der Beseitigung grober SeqAn-Usability-Probleme beschrieben. Dazu habe ich verschiedene Datenerhebungsverfahren (Online-Umfrage, Interviews, Feedback) und eine vereinfachte Evaluation mit Hilfe der Heuristischen Evaluation durchgeführt.

In Phase 2 habe ich die Planung und Durchführung der für meine Forschung notwendigen Datenerhebung erläutert. Eingegangen bin ich dabei auf die Gruppendiskussion, auf den eigens für die Evaluation von APIs entwickelten Cognitive-Dimensions-Fragebogen und das neuartige Programmierfortschritte-Datenerhebungsverfahren.

Phase 3 beschäftigt sich mit dem qualitativen Datenanalysewerkzeug APIUA, dessen Entwicklung die spezielle Datenerhebung in Verbindung mit der GTM erforderte.

Schließlich stellte ich in Phase 4 die Analyse der erhobenen Daten mit Hilfe der GTM und des Datenanalysewerkzeugs vor.

In dem folgenden Kapitel präsentiere ich meine eigentlichen Forschungsergebnisse.



## KAPITEL

### 4

## ERGEBNISSE

In diesem Kapitel präsentiere ich meine GT<sup>G</sup> über die Entstehung und Auswirkungen von Entwurfsentscheidungen in SeqAn vor und formuliere allgemeinere sich daraus ergebende Erkenntnisse. Bestandteil dieser Theorie sind Usability-Probleme, die es zu beseitigen gilt. Die dafür notwendigen Maßnahmen und deren Umsetzung stelle ich im zweiten Teil dieses Kapitels vor. Abschließend bespreche ich die Güte meiner Forschung und die Validierung bzw. Verallgemeinerbarkeit meiner Forschungsergebnisse.

Das darauf folgende, letzte Kapitel fasst meine Arbeit in seiner Gesamtheit zusammen und stellt Vorschläge für ein weiteres Vorgehen vor.



## THEORIE ÜBER DIE ENTSTEHUNG UND AUSWIRKUNGEN VON ENTWURFSENTSCHEIDUNGEN IN SEQAN

In diesem Abschnitt präsentiere ich meine GT über das Zustandekommen von Entwurfsentscheidungen und deren Auswirkungen auf die API-Usability von SeqAn. Dabei beleuchte ich besonders die sich als außerordentlich wichtig herausgestellte ● Templatemetaprogrammierung<sup>1</sup>.

Abbildung 4.1 visualisiert meine Theorie. Gruppiert sind dabei die gefundenen bzw. entwickelten Konzepte in Form der fünf Hauptkategorien ● Entwurfsentscheidung, ● Anwender, ● Usability-Probleme, ● Strategien und ● Folgen, wobei ● Inkonsistenzen bzgl. STL die Kernkategorie bildet.

Ich präsentiere meine GT in Form einer *Story*, bei der ich auf alle oben genannten Kategorien eingehen werde. Nicht alle Erkenntnisse sind datengetrieben im strengen Sinne der GTM<sup>G</sup> zu Stande gekommen. Ein Teil meiner Theorie basiert auf einer technisch-deduktiven Argumentation. Das heißt, habe ich ein Konzept in den Daten erst einmal erkennen können, reichten in solchen Fällen bereits Deduktionsschritte basierend auf existierenden wissenschaftlichen Erkenntnissen, um eben jene Konzepte hinreichend valide zu untermauern. Aussagen mit einem hohen technisch-deduktiven Anteil werden mit <sup>TD</sup> ausgezeichnet und sind weniger stark in den aufgezeichneten Daten verankert. Aussagen, die beide Argumentationsstile kombinieren, werden durch <sup>DG&TD</sup> gekennzeichnet.

---

<sup>1</sup> Zur Erinnerung: Die Kode-/Konzept-/Kategorie-Bezeichner können angeklickt werden. Sie sind verlinkt mit meinen Forschungsdaten. In dieser Darstellung der Theory nenne ich nur einige Phänomene von Konzepten. **Alle gefundenen Phänomene können in den verlinkten Forschungsdaten nachgelesen werden.** Die Farben geben Aufschluss über die Zusammenghörigkeit von Konzepten und werden durch einen semi-transparenten Kreis (●) dargestellt. Ein vollfarbiger Kreis (●) markiert besonders wichtige Konzepte. Die Notation wird ausführlich im Abschnitt 1.4.5.2 ab Seite 54 beschrieben.

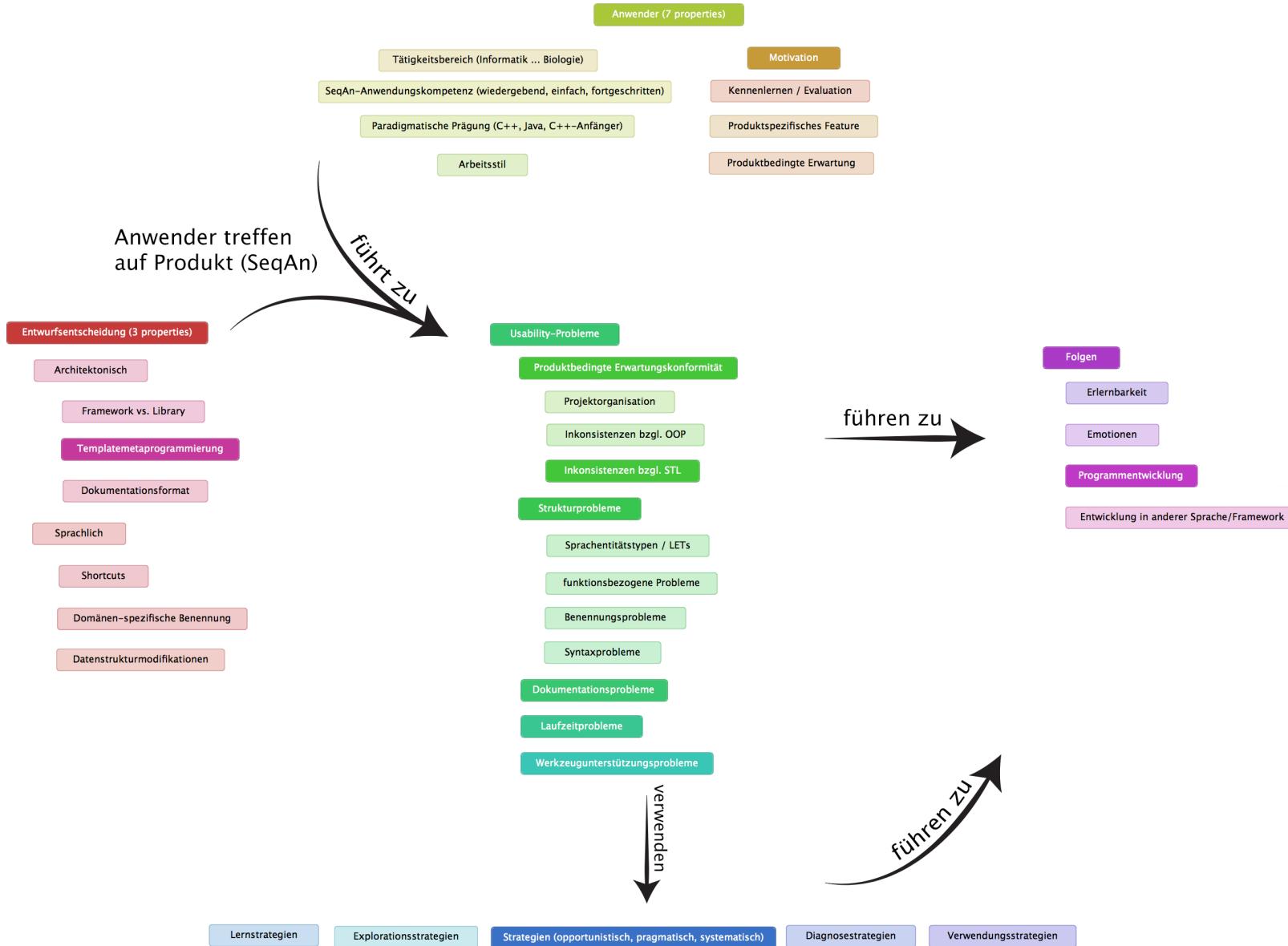


ABBILDUNG 4.1: Die Theorie über die Entstehung und Auswirkungen von Entwurfsentscheidungen in SeqAn besteht aus fünf Hauptkategorien, die in Beziehung stehen und dem paradigmatischen Modell ähneln: ● Entwurfsentscheidung beschreibt den Entwurf von SeqAn. Mit diesem sind SeqAns ● Anwender konfrontiert, was zu einer Reihe von ● Usability-Problemen führt. Zu Bewältigung kommen häufig ● Strategien zum Einsatz, deren Resultat mit den ● Folgen beschrieben werden.

### 4.1.1 ● Anwender

Diese Kategorie beschreibt den Anwender von SeqAn und umfasst alle relevanten Eigenschaften, die zu dessen Charakterisierung notwendig sind. Die API-Anwender zu verstehen ist existenziell notwendig, um die Usability einer API bewerten zu können. Denn Usability-Probleme entstehen erst durch die Verwendung eines Systems durch seine Anwender.

Basierend auf den Ergebnissen meiner GTM-Analyse und meiner im Abschnitt 3.2 vorgestellten Be seitigung grober Usability-Probleme, habe ich die folgenden relevanten Eigenschaften entdeckt<sup>2</sup>:

#### ● Tätigkeitsbereich

In welchem Gebiet ist der Anwender vornehmlich tätig? Die Anwenderschaft von SeqAn besteht — von wenigen Ausnahmen abgesehen — aus Informatikern, Bioinformatikern und in geringerem Umfang aus Biologen (siehe Abschnitt 3.2.3.1).<sup>3</sup>

Diese Eigenschaft ist wichtig für die weiteren Betrachtungen, da einzelne beobachtete Bioinformatiker<sup>3</sup> und Biologen vollständig zu der Gruppe der API-Endanwender gezählt werden. Diese Gruppe zeichnet sich durch einen opportunistischen Arbeitsstil aus.<sup>DG&TD<sup>3</sup></sup>

#### ● Arbeitsstil

Der Arbeitsstil beschreibt, auf welche Weise Anwender arbeiten. Basierend auf der Arbeit von Clarke (2007) (siehe Abschnitt 2.4.3) unterscheide ich den *opportunistischen*, *pragmatischen* und *systematischen* Arbeitsstil. Diese Eigenschaft ist für die Analyse von ● Problemlösungsstrategien relevant.<sup>TD</sup>

#### ● Paradigmatische Prägung

Von dieser Eigenschaft wurde in der Literatur nach meinem Kenntnisstand noch nicht berichtet. Sie beschreibt, durch welche Programmierparadigmen der Anwender geprägt sein kann.

Erstmalig wurde ich auf diese Eigenschaft aufmerksam, als ich bei der Auswertung der Workshop'12-Fragebögen auf die folgenden Aussagen stieß:

- “I’m noticing that of the constructs are different from what is common in the STL. This might be justified but it makes it \*much\* harder to learn.”<sup>3</sup>
- “Why don’t you use the naming convention (for iterators) used also by STL?”<sup>3</sup>

<sup>2</sup> Um Verwunderungen in Bezug auf das von mir verwendete Vokabular auszuräumen: In der Präsentation meiner GT verwende ich Verben wie “beobachten”, “suchen” und “entdecken”, weil dies der Philosophie der GTM entspricht (siehe Abschnitt 1.4 und Glaser u. Strauss 1967).

<sup>3</sup> Lediglich bei den Cognitive-Dimensions-Fragebögen habe ich nach dem Arbeitsstil gefragt. Dort gab jedoch keiner der Befragten an, einen opportunistischen Arbeitsstil zu haben. Ich gehe davon aus, dass dieser als laienhaft wahrgenommen wird und daher die Frage nicht immer wahrheitsgemäß beantwortet wurde. Tatsächlich kann man in den Programmierfortschritten opportunistisches Verhalten erkennen. In einem Fall<sup>3</sup> wird ein unerwartetes Verhalten einfach durch das manuelle Heraufsetzen einer Zählvariablen umgangen, ohne dass die Dokumentation konsultiert wurde. Ein (anderer) Diskussionsteilnehmer<sup>3</sup> deutet mit seiner Aussage “a metafunction and this double colon and a type or value behind”, die den Versuch vermissen lässt, Metafunktionen zu verstehen, ebenfalls auf ein opportunistisches Verhalten hin. Diese Beobachtungen, die im Abschnitt 2.4.3 beschriebene Literatur zu *Personas* (Clarke 2007) und die im Abschnitt 2.4.1 beschriebene Literatur zu *end-user software engineering*<sup>G</sup> (Ko et al. 2011) lassen den Schluss zu, dass der opportunistische Arbeitsstil auch unter der SeqAn-Anwenderschaft signifikant vertreten ist.

- “Not STL-like. Tries to reinvent STL with global function and it adds a lot of complexity that seems unnecessary.”<sup>3</sup>

Ich konnte grob die folgenden zwei Prägungen erkennen: *Java-Objektorientierung* und *C++/STL-Objektorientierung*. Beiden Prägungen ist gemein, dass sie sich mit der objektorientierten Programmierung befassen. Sie unterscheiden sich jedoch in der Strenge. Während Java streng objektorientiert und sozusagen “mono-paradigmatisch” ist, vermischen sich bei der C++-Objektorientierung verschiedene Paradigmen — insbesondere das der Templatemetaprogrammierung, welche durch die STL jedoch teilweise durch die Verwendung von `typedefs` verborgen wird. Beispielsweise ist die `string`-Klasse<sup>4</sup> nichts weiter als ein `typedef basic_string<char>`.

Diese Prägung habe ich in meiner Datenerhebung nicht explizit, jedoch indirekt über Fragestellungen zu Fertigkeiten mit bekannten Programmiersprachen erfragt. In Zusammenhang mit der Nennung verschiedener Usability-Probleme war es häufig nicht schwierig, auf die paradigmatische Prägung zu schließen. Gab ein Anwender<sup>3</sup> beispielsweise an, die besten Kenntnisse in Bezug auf Java zu haben und bezeichnete sich dieser auch noch als Java-Entwickler, wurde klar, dass er eine ● Java-objektorientierte Prägung besaß.

Diese Eigenschaft ist ursächlich für eine Reihe von ● Usability-Problemen.

### ● SeqAn-Einsatzform

Diese Eigenschaft beschreibt, in welcher Form SeqAn eingesetzt werden kann. Zu beobachten war, dass Anwender SeqAn entweder zur Implementierung von *Hilfsprogrammen*<sup>3</sup> oder zur Entwicklung einer ganzen *Pipeline*<sup>5</sup> nutzten. Dies wurde auch in persönlichen Gesprächen mit den Workshop-Teilnehmer bestätigt.

Die Relevanz dieser Eigenschaft wird weiter unten deutlich.

### ● SeqAn-Anwendungskompetenz

Diese Eigenschaft beschreibt, welche Erfahrung Anwender im Gebrauch von SeqAn besitzen können. Differenziert wird diese Eigenschaft in Anlehnung an die verschiedenen Tutorial-Übungsaufgaben-Schwierigkeitsgrade, die im Abschnitt 3.2.4.5 beschrieben und für meine Zwecke hinreichend validiert sind<sup>DG&TD</sup>. Sie lauten:

**Lernen** Der Anwender lernt SeqAn erst kennen und kann bestenfalls den SeqAn-Anwendungscode so weit verstehen, dass er einfache Anpassungen vornehmen kann.

**Anwendung** Der Anwender beherrscht SeqAn hinreichend, um existierenden Code neu komponieren und mehrzeilige Anwendungen selbstständig implementieren zu können.

**Beherrschung** Der Anwender verfügt über genügend Kenntnisse, dass er selbstständig SeqAn-Anwendungen für seinen eigenen Bedarf schreiben kann.

Meine Betrachtungen beschränken sich weitgehend auf Anwender, die SeqAn nicht beherrschen und dürften für Anwender mit guter SeqAn-Beherrschung nur geringe Gültigkeit haben.

<sup>4</sup> <http://www.cplusplus.com/reference/string/string/>

<sup>5</sup> Für die Sequenzanalyse werden häufig mehrere Phasen durchlaufen. Dazu gehört u.a. die Vorbereitung der Daten (*Preprocessing*) und die Visualisierung der Analyseergebnisse. Die technische Aneinanderreihung dieser Phasen wird häufig als *Pipeline* bezeichnet.

## ● Motivation

Die Eigenschaft befasst sich mit den Gründen, aus denen sich Anwender mit SeqAn befassen. Diese Eigenschaft verfügt über die folgenden Untereigenschaften:

### ● Kennenlernen / Evaluation

Es handelt sich hierbei um eine boolesche Eigenschaft. Sie gibt Auskunft darüber, ob potentielle Anwender SeqAn zunächst kennen lernen wollen. Dieses Kennenlernen kann zielgerichtet sein. In diesem Fall spreche ich von Evaluation. Die Anwender gaben wenig überraschend an, SeqAn für die Sequenzanalyse zu verwenden und SeqAns Eignung zu diesem Zweck überprüfen zu wollen. Ein Anwender<sup>3</sup> hatte besonders konkrete Vorstellungen. Er hatte bereits eine entsprechende Pipeline in der Programmiersprache C entwickelt und wollte prüfen, inwiefern er diese nach SeqAn portieren kann, um von SeqAns ● Performance-Eigenschaft profitieren zu können.

### ● Produktbedingte Erwartung

Diese Eigenschaft beschreibt Erwartungen, die beim Anwender durch das Produkt — also durch SeqAn — geweckt werden können. Wie noch weiter unten erläutert wird, weckt SeqAn die folgenden Erwartungen:

#### ● Libraryerwartung

Erwartung, dass es sich bei SeqAn um eine Softwarebibliothek handelt, die in das eigene Projekt eingebunden werden kann. Interessanterweise wurde ich auf diese Erwartung nur durch persönliche Gespräche — vielfach — aufmerksam. In meinen technisch erfasssten Daten ist diese Erwartung nicht ohne Weiteres zu finden. Verantwortlich mache ich dafür das Umfeld, in dem die Datenaufzeichnungen stattfanden. Diese waren nämlich stets moderiert (Workshop bzw. Projektseminar, siehe Abschnitt 3.1.1) und hatten nicht den Fokus auf die Integration von SeqAn in einem eigenen Projekt, sondern auf das Erlernen von SeqAn in einer vergleichsweise praxisfremden Umgebung.<sup>DG&TD</sup>

#### ● Performanceerwartung

Erwartung, dass die mit SeqAn entwickelten Anwendungen besonders schnell sind.<sup>3</sup>

#### ● Benutzerfreundlichkeitserwartung

Erwartung, dass SeqAn einfach in der Anwendung ist.<sup>38</sup>

### ● Produktspezifisches Feature

Die Eigenschaft unterscheidet sich von ● produktbedingen Erwartungen in dem Grad der Gewissheit. Sie beschreibt keine Erwartung, sondern das Wissen um eine Eigenschaft und betrifft hauptsächlich Anwender, die sich bereits von den Vorzügen der folgenden Eigenschaft überzeugen konnten und aus diesem Grund SeqAn einsetzen.

#### ● Performance

Wissen um die hohe Performance von mit SeqAn entwickelten Programmen.<sup>38</sup>

Die auffälligste Beobachtung ist, dass in keiner meiner Datenerhebungen ein Proband von der Benutzerfreundlichkeit von SeqAn berichtete, obwohl es die dazugehörige Erwartung gibt.

### 4.1.2 ● Entwurfsentscheidungen

SqAn wurde von Andreas Gogol-Döring im Rahmen seiner Dissertation “SeqAn — A Generic Software Library for Sequence Analysis” (Gogol-Döring 2009) entwickelt. Auf der Grundlage seiner Arbeit habe ich die Kategorie ● Entwurfsentscheidungen erarbeitet, welche alle Entscheidungen auf Seiten der API-Entwickler beschreiben können soll, die den Entwurf von SeqAn prägen. Unterschieden wird dabei grundsätzlich in ● architektonische und ● sprachliche Entwurfsentscheidungen. Diese Unterscheidung stammt von Robertson (2007) (siehe Abbildung 2.4 auf Seite 100).

Entwurfsentscheidungen können mit Hilfe der folgenden Eigenschaften charakterisiert werden:

#### ● Entwurfsentscheidungsziel

Diese Eigenschaft beschreibt das Ziel, mit der eine Entwurfsentscheidung getroffen wurde. Entdeckte Ziele sind *Performance* und *Usability*. Andere Ziele wie *Simplizität* habe ich unter *Usability* subsummiert.

#### ● Entwurfsentscheidungsmotivation

Diese Eigenschaft beschreibt die Motivation der Entwurfsentscheidung. Dabei kann es sich entweder um die Absicht handeln, das erklärte Ziel zu erreichen (*Zielverfolgungsabsicht*) oder um eine *notwendige Folgeentscheidung*, die sich aus einer vorangegangenen Entwurfsentscheidung ergibt. Ist Letzteres der Fall, so kann es ein Entwurfsentscheidungsziel geben — und zwar dann, wenn der API-Entwickler beispielsweise etwas kompensieren muss. Die weiter unten besprochene ● Folge-Entwurfsentscheidung ● Template-Spezialisierung diente dazu, die Usability zu steigern, indem Polymorphie ermöglicht wurde. Dieser Schritt war jedoch nur notwendig, weil durch den Einsatz von ● Templatemetaprogrammierung und den Verzicht auf objektorientierte Programmierung Polymorphie zunächst nicht mehr nutzbar gewesen ist.

#### ● Entwurfsentscheidungsgrundlage

Diese Eigenschaft beschreibt die Art, wie die Entwurfsentscheidung zustande kam bzw. umgesetzt wurde. Die entsprechende Dimension umfasst dabei das Spektrum von *implizit* bis *explizit-empirisch*.

**Implizite** Entwurfsentscheidungen sind solche, die unbewusst getroffen wurden. Diese Art ist häufig anzutreffen, wenn es sich um eine notwendige Folgeentscheidung handelt. In dieser Konstellation hat der API-Entwickler nicht erkannt, dass er gerade eine Entwurfsentscheidung trifft, was tendenziell ein weniger vorhersagbares Ergebnis bedingt.

**Explizite-intuitive** Entwurfsentscheidungen sind solche, die zwar erkannt wurden, jedoch lediglich intuitiv getroffen wurden. Implizite und explizit-intuitive Entscheidungen bezeichne ich auch als ● Bauchgefühlentscheidung.

**Explizite-argumentative** Entwurfsentscheidungen unterscheiden sich von explizit-intuitiven Entwurfsentscheidungen dadurch, dass sie argumentiert werden. Als Argumente dienen u.a. Erfahrungen des API-Entwicklers oder Entwurfsentscheidungen, die von Dritten in anderen Produkten getroffen und adaptiert wurden.

**Explizite-empirische** Entwurfsentscheidungen haben die höchste Wahrscheinlichkeit, das Entscheidungsziel zu erreichen. Die Wirksamkeit derartiger Entwurfsentscheidungen wird vom Entscheider empirisch belegt. Der empirische Wirkungsnachweis kann dabei selbst erbracht werden oder auf empirisches Untersuchen in verwandten Arbeiten fußen.

Bei der vorgestellten Unterteilung handelt es sich um eine Dimension mit zwei Extremen und erlaubt viele Schattierungen. So sind die Grenzen zwischen den expliziten Entscheidungsgrundlagen fließend — je nachdem, wie solide die Argumentation ist.

Implizite Entwurfsentscheidungen sind weniger differenziert, denn das Fehlen der Explikation der Entscheidungsgrundlage lässt keine zuverlässige Unterscheidung zu.

In Bezug auf ● Entwurfsentscheidungen kann SeqAn also wie folgt beschrieben werden:

SeqAn wurde als quelloffene Softwarebibliothek (*Library*) entworfen, welche auf C++ basiert und zum ultimativen ● Entwurfsentscheidungsziel die Geschwindigkeit (*Performance*) von mit SeqAn entwickelten Programmen hatte. Ein weiteres Entwurfsentscheidungsziel war die Benutzerfreundlichkeit (*Usability*) von SeqAn.

Diese Beschreibung, welche sich auch auf der SeqAn-Website<sup>6</sup> befindet, weckt auf ● Anwenderseite vier ● produktbedingte Erwartungen, von denen drei nicht erfüllt werden. Diese bezeichne ich informell gerne auch als *die drei Urlügen / -täuschungen*. Die im Folgenden vorgestellten, unerfüllten ● produktbedingten Erwartungen bzw. *Urlügen* basieren auf meinen Analysen der in Abschnitt 3.3 vorgestellten Datenquellen, sowie der im vorherigen Absatz genannten Dissertation (Gogol-Döring 2009) und einem längeren Telefonat mit Andreas Gogol-Döring (Gogol-Döring u. Kahlert 2013):

### 1. Framework statt Softwarebibliothek (● Libraryerwartung)

SeqAn sollte eine Library sein, wurde jedoch als Framework entwickelt.

Der Unterschied besteht in der Art, wie das “architektonische Skelett” der entwickelten Anwendungen durch die Library bzw. das Framework vorgegeben wird. Bei einer Library ist der API-Anwender weitgehend unbeeinflusst — er bindet die durch die Library bereitgestellte Funktionalität lediglich ein. Bei einem Framework wird die Struktur der entwickelten Anwendungen weitgehend von dem Framework vorgegeben. (Fairbanks et al. 2006)

Diese Form der Implementierung wird durch die ● architektonische Entwurfsentscheidung ● Framework vs. Library erfasst. Dabei handelt es sich nach meiner Einschätzung um eine *implizite* Entscheidung, denn Gogol-Döring lässt in seiner Dissertation kein Bewusstsein vermuten, dass die folgenden Beobachtungen de facto zur Implementierung eines Frameworks führen<sup>TD</sup>:

- SeqAn verwendet das plattformübergreifende Build-System *CMake*<sup>7</sup>. Das hat zur Folge, dass SeqAn-Anwender Programme nur innerhalb dieses Systems entwickeln können. Eine

---

6 <http://www.seqan.de>

7 <http://www.cmake.org>

Einbindung von SeqAn in das eigene C++-Projekt — durch Inkludierung einer entsprechenden Header-Datei — ist *nicht* vorgesehen.

- Die SeqAn-Entwickler nutzen Vorausdeklarationen (*forward declarations*), um API-interne Abhängigkeiten aufzulösen. Diese werden durch ein, in das Build-System integriertes Python-Skript bei jedem Kompilieren automatisch generiert, was die Komplexität von Vorausdeklarationen verringert. Ein SeqAn-Anwender müsste für den Gebrauch von SeqAn als Library dieses Skript selbst ausführen.
- SeqAn verwendet eine dreigliedrige Organisation. Der *Core* umfasst dabei alles, was zu SeqAn gehört. *Extras* beinhaltet funktionale Erweiterungen, die weitgehend auf die Ergebnisse von Abschlussarbeiten zurückgehen. Ist ein Extra von großer Relevanz und genügt es nicht näher spezifizierten Qualitätsanforderungen, wird es in den Core aufgenommen. Schließlich gibt es noch die *Sandboxes*. Jeder SeqAn-Anwender muss mit Hilfe eines Python-Skripts eine Sandbox erstellen. Innerhalb seiner Sandbox wiederum werden durch abermäßige Python-Skript-Aufrufe *Apps* erzeugt. Dabei handelt es sich um ein Skelett zur Entwicklung von SeqAn-Anwendungen. Die Verwendung des Python-Skripts ist obligatorisch, denn es erweitert das Build-System so, dass die selbst entwickelten SeqAn-Anwendungen / -Apps auch kompiliert werden können.

Aus diesen Gründen, ist SeqAn keine Library, sondern ein Framework. SeqAn kann nicht ohne weiteres in eigene Entwicklungsprojekte eingebunden werden, was in seiner Verwendung von CMake, dem Gebrauch von Vorausdeklarationen und der Vorgabe der Projektstruktur begründet ist. Damit wird massiv die oben beschriebene ● Libraryerwartung verletzt. Außerdem wird die ● SeqAn-Einsatzform eingeschränkt, wenn der Gebrauch als Library vorgezogen wird. Mehrere Workshop-Teilnehmer<sup>8</sup> gaben in mündlichen Gesprächen an, dass sie die Framework-Gestalt von SeqAn für inakzeptabel halten, da sie bereits ein eigenes Build-System betreiben und ein Wechsel nicht in Frage kommt. Dieses praxisrelevanten Äußerungen legen nahe, dass SeqAns Framework-Gestalt in der Praxis für viele potentielle Anwender ein Problem darstellt.<sup>DG&TD</sup>

## 2. Programmierparadigma (● Paradigmenerwartung)

SeqAn nutzt als Programmiersprache C++. Dies weckt Erwartungen, die im Konflikt zur anwenderspezifischen ● paradigmatischen Prägung stehen können und u.a. folgende Wortmeldungen zur Folge haben:

- Ein Anwender mit einer ● Java-Objektorientierungsprägung bezeichnete SeqAn als “Vergewaltigung der OO-Programmierung”<sup>8</sup>.
- Ein Anwender mit einer ● C++-/STL-Objektorientierungsprägung sagte: “If SeqAn [just] had the same way of doing things like the STL”<sup>8</sup>.
- Ein Anwender sagte allgemein im Zusammenhang mit der ● Templatemetaprogrammierung: “[It’s] not clear why you have to go through all this pain”<sup>8</sup>.

Offenbar “tickt” SeqAn also anders, als es seine Anwender erwarten. Gogol-Döring traf die architektonische Entwurfsentscheidung, für die Entwicklung von SeqAn keine klassische Objektorientierung, sondern das Programmierparadigma ● Templatemetaprogrammierung zu verwenden.

In seiner Dissertation stellt Gogol-Döring die Vorzüge der Templatemetaprogrammierung dar, ohne jedoch die Nachteile der Objektorientierung näher zu beleuchten. In dem Telefonat mit Gogol-Döring (Gogol-Döring u. Kahlert 2013) erfragte ich diese:

- Gogol-Döring teilte mir mit, dass er kein Freund von objektorientierter Programmierung (OOP) sei. Als Beispiel nannte er, dass die Implementierung symmetrischer Operationen, wie der Addition, in der OOP auf eine ihm “befremdliche” Art gelöst würden — nämlich durch Memberfunktionen und nicht durch “symmetrische”, globale Funktionen.
- Hauptargument für die Verwendung der Templatemetaprogrammierung war jedoch die dadurch hohe zu erreichende Performance. Zum Erreichen von Polymorphie werden in C++ virtuelle Funktionen benötigt. Welche Funktion tatsächlich zur Ausführung kommt, wird zur Laufzeit bestimmt — und das kostet Rechenzeit (*virtual lookup overhead*). Um die Bestimmung der aufzurufenden Funktion performance-steigernd zur Kompilierzeit zu berechnen (*static binding*) und gleichzeitig Polymorphie zu erlauben, traf der SeqAn-Autor die Entscheidung, ● Template-Spezialisierung (*Template Subclassing*) zu verwenden.
- Gogol-Döring experimentierte mit Möglichkeiten, eine hohe Performance mit Hilfe der OOP zu erreichen. Dies gelang ihm jedoch nicht.

### 3. Usability (● Benutzerfreundlichkeitserwartung)

Gogol-Döring beschreibt als weiteres Entwurfsziel die *Usability*. Um dies zu erreichen, entschloss sich der Autor zu der ebenfalls architektonischen Entwurfsentscheidung ● Generische Programmierung<sup>8</sup>. Generische Funktionen lösen Probleme allgemeingültig. Kann ein Problem für bestimmte Eingaben effizienter gelöst werden (z.B. bei der Sortierung), kann eine spezialisierte Funktion implementiert werden. Der Compiler wählt beim Kompilieren die spezialisierteste Implementierung aus und bindet diese Funktion statisch, was die Performance von SeqAn sicherstellt (Details siehe Gogol-Döring 2009).

Ungünstigerweise werden generische Funktionen nicht als Memberfunktionen, sondern als globale Funktionen implementiert. Damit entfällt jedoch technisch die immer noch existierende inhaltliche Zusammengehörigkeit von Funktionen, was in der OOP durch die Klassenzugehörigkeit implementiert wird. Üblicherweise erwartet eine generische Funktion ihr *Hauptobjekt* als erstes Argument. Schreibt man in OOP also `myString.length()`, lautet der gleiche Aufruf in SeqAn `length(myString)`. In SeqAn wird die inhaltliche Zusammengehörigkeit von Funktionen als *global function interface* bezeichnet und nur noch in der Dokumentation beschrieben, was insbesondere ● Explorationsstrategien behindert.

Durch die globale Implementierung von generischen Funktionen, firmieren unterschiedliche Funktionen unter dem gleichen Namen im globalen Namespace. Entsprechend variabel ist auch deren

<sup>8</sup> Tatsächlich verfolgte Gogol-Döring mit der ● Generischen Programmierung, neben der *Usability*, weitere Ziele wie *Simplizität*, *Generalisierbarkeit* und *Erweiterbarkeit*. Für den Zweck meiner Arbeit reicht die zusammenfassende Betrachtung *Usability*.

Rückgabetyp. Den korrekten Rückgabetyp zu ermitteln, ist mühsam und fehlerträchtig. Um u.a. dieses Problem zu lösen, wurde die Entwurfsentscheidung ● Metafunktionen getroffen. Dabei handelt es sich um Funktionen, die nicht zur Laufzeit, sondern zur Kompilierzeit ausgeführt werden. Sie werden dazu verwendet, den korrekten Typ für die Rückgabe einer generischen Funktion zu berechnen. Syntaktisch ähneln Metafunktionen allerdings Klassen, da Metafunktionen in SeqAn ebenfalls mit einem Großbuchstaben beginnen<sup>TD</sup>. Da die SeqAn-Anwender über eine unpassende ● Paradigmatische Prägung verfügen, führt dies zu ● Usability-Problemen, die weiter unten erläutert werden.

● Generische Programmierung und ● Metafunktionen sind im Falle von SeqAn *explizite-argumentative* ● Entwurfsentscheidungen mit dem ● Entwurfsentscheidungsziel *Usability*. Ein empirischer Beleg für die Wirksamkeit der Entscheidungen fehlt. Stattdessen wird die Wirksamkeit nur unzureichend argumentativ an drei Stellen seiner Dissertation (Gogol-Döring 2009) erbracht:

- In Unterkapitel 14.1 argumentiert der Autor, dass SeqAn lediglich eine geringe Anzahl von Techniken kombiniert, von denen jede nur eine begrenzte Komplexität (“limited complexity”) aufweist. Gleichzeitig räumt er aber ein, dass sein Ansatz als unüblich von den meisten Programmierern empfunden werden könnte und Fehlerausgaben der Compilers “ziemlich” schwer verständlich sind. In dieser Betrachtung unterschätzt Gogol-Döring im erheblichen Maß die Auswirkungen seines Entwurfs.
- In Unterkapitel 14.2 argumentiert der Autor, dass der erfolgreiche Einsatz von SeqAn im Rahmen von Arbeiten innerhalb der Bioinformatik-Arbeitsgruppe die Usability bestätigt. Allerdings ignoriert er dabei, dass die meisten Anwender die Entwicklung von SeqAn miterlebten/-gestalteten und sich bei Problemen, direkt an den Autoren wenden konnten.
- In Kapitel 15 demonstriert Gogol-Döring die Re-Implementierung der Basisfunktionalität eines Bioinformatik-Werkzeugs. Damit stellt er allerdings lediglich unter Beweis, dass SeqAn für ihn selbst benutzbar ist.

Die in Kapitel 15 angeklungene ● Bauchgefühl-Usability setzt sich in sämtlichen ● sprachlichen Entwurfsentscheidungen fort, d.h. all diese Entscheidungen sind *implizit* oder *explizit-intuitiv* — werden also gar nicht oder nur intuitiv begründet:

### ● Shortcuts

Hierbei handelt es sich um C++-**typedefs**, die eine einfachere Schreibweise von häufigen Templatespezialisierungen bereitstellen sollen. Beispiel: Das Shortcut **DnaString** kann anstelle von **String<Dna>** verwendet werden.

### ● Domänen-spezifische Benennung

In SeqAn mischt Gogol-Döring Termini aus den Domänen Informatik (z.B. **Alloc**), Bioinformatik (z.B. **Alphabet**, **Gaps**) und Biologie (z.B. **Peptide**), was u.a. zum weiter unten erläuterten ● Namennerraten führt.<sup>DG&TD</sup>

### ● Datenstrukturmodifikation

Datenstrukturen können auf unterschiedliche Weise verändert werden. In SeqAn kommen

verschiedene Ansätze zum Einsatz, die entlang zweier Dimensionen charakterisiert werden können<sup>TD</sup>:

**Direktheit** Werden Datenstrukturen direkt oder indirekt verändert?

**Explizitheit** Werden Datenstrukturen explizit oder implizit verändert?

Es gibt also vier Möglichkeiten Datenstrukturen zu verändern, die alle in SeqAn Anwendung finden. Im Folgenden veranschauliche ich diese Möglichkeiten mittels Pseudocode. Verändert wird immer der Inhalt der Variablen `x`.

1. direkt, explizit: `x = 1`
2. direkt, implizit: `fn(x, 1)`
3. indirekt, explizit: `fn(x) = 1`
4. indirekt, implizit: `fn(fm(x), 1)`

Die impliziten Datenstrukturmodifikationen sind möglich, weil SeqAn häufig — aber nicht immer — Referenzen zurückgibt und häufig — aber nicht immer — den Zuweisungsoperator überschreibt.

Die eben beschriebenen  sprachlichen Entwurfsentscheidungen können eine Reihe von schweren Usability-Problemen zur Folge haben, die unter  Benennungsprobleme und  Syntaxprobleme zusammengefasst sind.

#### 4.1.2.1 Zwischenzusammenfassung

SeqAn weckt bei seinen Anwendern die  produktbedingte Erwartung einer performanten, objektorientierten und benutzerfreundlichen Library. Die Performance konnte der SeqAn-Entwickler Gogol-Döring nachweisen. Jedoch ist SeqAn keine Library, sondern ein Framework, verwendet keine objektorientierte Programmierung, sondern  Templatemetaprogrammierung und hat, wie wir gleich sehen werden, schwere Defizite in Bezug auf seine Usability.

Abgesehen von der  Templatemetaprogrammierung, handelt es sich im Falle von SeqAn bei den  architektonischen Entwurfsentscheidungen um *explizite-argumentative* Entwurfsentscheidungen. Unter den  sprachlichen Entwurfsentscheidungen befindet sich eine *explizite-intuitive*; die beiden anderen sind lediglich *implizit*. Wenn überhaupt, wurden die potentiellen Anwender nur unzureichend in die Validierung der Usability einbezogen. Anders sind Anwender-Äußerungen wie “SeqAn makes me feel stupid”<sup>ø</sup> nicht zu erklären.

Bevor ich die beobachteten  Usability-Probleme genauer vorstelle, stelle ich zunächst die möglichen, schlussendlichen  Folgen des API-Entwurfs vor.

### 4.1.3 Folgen

In diesem Abschnitt stelle ich die möglichen Folgen eines schlechten SeqAn-Entwurfs für die Anwender vor. Diese Kenntnisse erleichtern das Verständnis der im nächsten Abschnitt präsentierten **Usability-Probleme** und **Strategien**.

Bei der Analyse meiner erhobenen Daten konnte ich vier Kategorien von **Folgen** entdecken:

#### **Erlernbarkeit**

Die Erlernbarkeit beschreibt, wie einfach sich SeqAn erlernen lässt. Ich habe den Fokus auf Einschränkungen der Erlernbarkeit gesetzt und daher keine weiteren Eigenschaften gesucht.

Charakteristisch für die Erlernbarkeit von SeqAn ist, dass sie von den Anwendern als mühsam bezeichnet wird. Immer wieder beschreiben die Anwender, dass das Erlernen von SeqAn viel Übung erfordert. Ein Workshop-Teilnehmer<sup>8</sup> verwendet sogar die Vokabel “Disziplin”. In dem Statement “once you get through it” eines anderen Anwenders<sup>8</sup> wird ebenfalls die Mühseligkeit der SeqAn-Erlernbarkeit deutlich.

Ein Aspekt der Erlernbarkeit sind **schnelle Anfangserfolge**, welche bei SeqAn die Wirksamkeit der Überarbeitung der Tutorials in der Behebungsphase grober Usability-Probleme belegen (siehe Abschnitt 3.2). Negativ an diesem Aspekt ist allerdings, dass manchen Anwendern ein Zugewinn an **Anwendungskompetenz** nicht gelingen könnte, was die folgenden Aussagen eines Teilnehmers<sup>8</sup> verdeutlichen:

- “However, while I found the tutorials easy to follow and to complete, I found it very hard to start my own work.”
- “I [had] a very good time writing simple apps. But when I tried something more complex, I immediately run into a wall.”
- “It is easy at the beginning, but becomes rapidly much harder when the problems complexity arises.”

Ein weiterer Aspekt ist die **Persistenz**. Selbst langjährige, SeqAn sporadisch einsetzende Anwender haben das Problem, sich immer wieder in SeqAn hineindenken zu müssen<sup>88</sup>. Der fremdaristige Entwurf von SeqAn erlaubt es Anwendern kaum, bestehendes Wissen auf SeqAn anwenden zu können. Die Folge: SeqAn erfordert ein hohes Maß an **produkt-spezifischem Wissen**.

#### **Programmentwicklung**

Unter diesem Konzept verbergen sich Auswirkungen von **Usability-Problemen** auf die Entwicklung von Programmen mit Hilfe von SeqAn. Viele der weiter unten vorgestellten Probleme können die Programmentwicklung *verlangsamen* (“Das fällt mir in der Tat noch recht schwer.”<sup>8</sup>, “Wenn viele Konzepte und Konstrukte ineinander greifen ist es teils die Verwendung teils schwer.”<sup>8</sup>). Andere Probleme wie das **Typing**<sup>9</sup> können sogar ursächlich für eine *quasi-unmögliche* Programmentwicklung sein (“two algorithms which I tried to implement (and failed)”<sup>8</sup>, “when I tried

---

<sup>9</sup> **Typing** bezeichnet das Problem, den korrekten Typ für eine Variable zu bestimmen.

something more complex, I immediately run into a wall<sup>3</sup>). Bemerkenswert daran ist, dass gerade die ● architektonische Entwurfsentscheidung ● Metafunktionen dieses Problem verhindern sollte, indem mit deren Hilfe Rückgabetypen einfach berechnet werden können sollten. Allerdings konnte ich auch eine Aussage finden, die der in SeqAn eingesetzten generischen Programmierung eine *vereinfachte* Programmierung zuschreibt<sup>3</sup>.

Besonders hervorzuheben sind zwei Aspekte der Programmierung: Häufig werden ● Refaktorisierungs-Operationen durch das eben genannte ● Typing massiv behindert<sup>3</sup>. Der zweite wichtige Aspekt ist das erfolgreiche ● Kompilieren eigener Programme, was ebenfalls am schwierigen ● Typing scheitern kann<sup>3</sup>.

### ● Entwicklung in anderer Sprache/Framework

Eine mögliche Folge der mangelhaften Usability ist die Verwendung einer Alternative<sup>DG&TD</sup>. In einem Fragebogen äußerte sich dies bzgl. eines Teilnehmers<sup>3</sup>, der bereits eigene Bioinformatik-Werkzeuge in der Programmiersprache C entwickelt hat<sup>3</sup>. Zwei Algorithmen versuchte er nach SeqAn zu portieren, was ihm jedoch nicht gelang<sup>3</sup>. Die ebenfalls von ihm geäußerte ● Frustration / Resignation<sup>333</sup> und die viele investierte Zeit<sup>3</sup> legen den Schluss nahe, dass der Anwender entweder bei seiner aktuellen Lösung bleibt oder sich nach einem anderen Produkt umschaut. Ich betrachte es für naheliegend, dass auch andere potentielle API-Anwender auf diese Weise handeln.

### ● Emotionen

Die letzte Klasse von Folgen sind Gefühle, die Anwender äußern, wenn sie auf ● Usability-Probleme treffen.

#### ● Sympathie

Um kein einseitiges, sondern hinreichend breites Bild von SeqAn zu erfassen, habe ich auch positive Emotionen kodiert, von denen ich allerdings nur wenige gefunden habe. Diese habe ich unter ● Sympathie zusammengefasst. Sie wird von einem Teilnehmer<sup>3</sup> in Anbe tracht der großen Anstrengungen, welche die SeqAn-Entwickler in die Library und seiner Dokumentation investieren, zum Ausdruck gebracht. Sympathie wurde beim Großteil der Workshop-Teilnehmer deutlich sichtbar, als Benchmarks von SeqAn-Programmen im Vergleich zu Konkurrenzprodukten vorgestellt wurden, was die enorme Geschwindigkeit der mit SeqAn entwickelten Programme veranschaulichte.

#### ● Verunsicherung

Gefühle der Verunsicherung konnte ich vornehmlich in Bezug auf die mangelhafte ● Benennungskonsistenz und dem ● Namennerraten feststellen. Beispielsweise sagte ein Gruppendiskussionsteilnehmer: "Even if you find the global function by name, you have always the unsureness if it's actually applicable."<sup>3</sup>

#### ● Unverständnis / Genervtheit

Das Gefühl, genervt zu sein und Unverständnis zu empfinden, wurde während der Gruppendiskussion am häufigsten zum Ausdruck gebracht. Die folgenden Äußerungen sprechen für sich:

- \* “If you look up shortcuts: ‘Oh, it’s just that!’”<sup>8</sup>
- \* “Why would you need to have shortcuts if you had to look them all up?!?”<sup>8</sup>
- \* “It’s just stupid! What’s the difference?! It’s just a string array!”<sup>8</sup>

Besonders bemerkenswert sind Aussagen eines Teilnehmers, der bezweifelt, dass die dominante ● Entwurfsentscheidung ● Templatemetaprogrammierung tatsächlich so alternativlos für die Erreichung von ● Performance sei, wie alle SeqAn-Entwickler immer sagen: “Where is the performance improvement? [...] There are [other] ways of saving that [virtual] lookup.”<sup>8</sup>

### ● Frustration / Resignation

Frustration konnte ich in allen subjektiven Datenquellen gleichermaßen finden. Frustration kann durch ein ganzes Spektrum von ● Usability-Problemen ausgelöst werden. Fasst man diese zusammen, kommt man zu dem Schluss, dass sie bei SeqAn von einer enttäuschten ● Benutzerfreundlichkeitserwartung herrühren.

Die folgenden Aussagen geben einen Eindruck von empfundenen Frustrationen:

- \* “99% error messages that I have seen until now were due to incorrect types”<sup>8</sup>
- \* “I have tried to implement two algorithms I have created and implemented before in C, but failed.”<sup>8</sup>
- \* “It’s this inconsistency that makes it hard.”<sup>8</sup>
- \* “[It’s] hard when the compiler tell you that the error is in some SeqAn file [...]”<sup>8</sup>

### ● Sarkasmus

Sarkastische Äußerungen waren selten und wurden bis auf eine Ausnahme in den Pausen der Workshops gemacht. Die eine Ausnahme stammt von einem Diskussionsteilnehmer: “To make things more funny there are places in SeqAn where the subscript operator returns values and not references.”<sup>8</sup>

#### 4.1.3.1 Zwischenzusammenfassung

Die schlussendlichen ● Folgen der ● Entwurfsentscheidungen sind vielgestaltig. ● SeqAn-Anwender haben Probleme mit der ● Erlernbarkeit und empfinden diese als mühselig. Die entsprechende Disziplin vorausgesetzt, können Anwender zwar zu ● schnellen Anfangserfolgen kommen, aber auch scheitern, wenn es darum geht, eigene Probleme selbstständig zu lösen. Selbst langjährige SeqAn-Anwender kämpfen ihres Wissens mit dem Problem der ● Persistenz, weil SeqAn durch seine ungewöhnliche Gestalt ein hohes Maß an ● produkt-spezifischem Wissen erfordert.

Ebenso kann die ● Programmentwicklung — also die eigentliche Arbeit mit SeqAn — behindert werden. Anwender beklagen, dass einfachste ● Refaktorisierungen immer wieder am erfolglosen ● Kompilieren scheitern.

All dies kann zu negativen, kopfschüttelnden ● Emotionen auf Seiten der SeqAn-Anwender führen, was in manchen Fällen zur Folge hat, dass ● Alternativen für die Lösung der eigenen Aufgabenstellungen in Betracht gezogen oder bereits genutzt werden.

Meine Beobachtungen sind Folgen, die auf die durch ● Bauchgefühl-Entwurfsentscheidungen verursachten ● Usability-Probleme zurückgehen und die Nichterfüllung der geweckten ● Benutzerfreundlichkeitserwartung belegen. Zwar konnten sich die bisherigen Dokumentationsverbesserungen (insbesondere die der Tutorials) positiv auf die Usability auswirken, jedoch nicht die negativen Folgen der ● Bauchgefühl-Entwurfsentscheidungen kompensieren.

Die bei SeqAn *explizite-empirische* ● Entwurfsentscheidung ● Templatemetaprogrammierung erfüllt zwar das ● Entwurfsentscheidungsziel ● Performance und damit auch die anwenderseitige ● Performanceerwartung. Jedoch kann die ● Templatemetaprogrammierung die anwenderseitige ● Benutzerfreundlichkeitserwartung kaum erfüllen. Auch andere Entwurfsentscheidungen wie ● Metafunktionen ändern daran nichts. Ganz im Gegenteil: ● Metafunktionen können weitere schwerwiegende ● Usability-Probleme bedingen, die ich im folgenden Abschnitt vorstellen werde.

#### 4.1.4 ● Usability-Probleme & ● Strategien

Bisher habe ich beschrieben, wer die ● Anwender von SeqAn eigentlich sind und welche expliziten und impliziten ● Entwurfsentscheidungen der Entwicklung von SeqAn zu Grunde lagen. Weiterhin habe ich die schlussendlichen ● Folgen der Entwurfsentscheidungen beschrieben.

In diesem Abschnitt stelle ich nun die Brückenglieder zwischen diesen drei Kategorien vor. Die ● Usability-Probleme, die durch die Konfrontation der Anwender mit den Entwurfsentscheidungen entstehen können und den ● Strategien, welche die Anwender bewusst oder unbewusst nutzen und die abhängig vom Erfolg zu unterschiedlichen Folgen führen können (siehe Abbildung 4.1).

Ich habe unterschiedliche ● Usability-Probleme gefunden und auf ähnliche Weise gegliedert, wie dies Grill et al. (2012) bereits in ihrer API-Evaluations-Studie taten: ● Strukturprobleme, ● Dokumentationsprobleme und ● Laufzeitprobleme. Die Kategorie *User Experience* gibt es bei mir nicht. Sie wird bereits durch die ● Emotionen (ausführlicher) beschrieben. Ergänzt habe ich die neu entdeckte Klasse der ● Werkzeugunterstützungsprobleme.

Wie man an der eben eingeführten Gliederung sehen kann, werden die Usability-Probleme an Hand ihrer betroffenen Kategorie gegliedert — nämlich der API selbst, seiner Dokumentation, den Problemen, die bei der Ausführung entstehen und dem Zusammenspiel mit anderen Werkzeugen. Einige in den Kategorien beschriebene Usability-Probleme haben jedoch einen besonderen Stellenwert in meiner Darstellung, weshalb ich diese in einer fünften Kategorie (● Produktbedingte Erwartungskonformitätsprobleme) zusammenfasse. Die ● Strategien stelle ich bei der Beschreibung des jeweiligen relevantesten Usability-Problems vor. Für eine schematische Einordnung verweise ich auf Abbildung 4.1.

##### ● Produktbedingte Erwartungskonformitätsprobleme

Diese Kategorie umfasste Usability-Probleme, die besonders stark durch die geweckten anwenderseitigen ● produktbedingten Erwartungen verursacht werden.

### ● Projektorganisation<sup>DG&TD</sup>

SqAn gibt vor, eine Library zu sein (● Framework vs. Library) und weckt damit die ● produktbedingte Libraryerwartung. Daraus entsteht das Problem der ● Projektorganisation. Darunter verstehe ich, dass der Anwender entgegen seiner Erwartung, es mit einer Library zu tun zu haben, tatsächlich aber mit einem Framework konfrontiert ist. Phänomene für dieses Problem sind schwere bis katastrophale Probleme bei der Einrichtung von SqAn. Zu nennen sind hier die Arbeit mit dem plattformunabhängigen CMake-Build-System und die Einrichtung von SqAn innerhalb der eigenen IDE (siehe Abschnitt 3.2). SqAn unterstützt offiziell fünf Entwicklungsumgebungen auf insgesamt drei Betriebssystemen. Möchte der Anwender eine andere Entwicklungsumgebung verwenden, ist er auf sich allein gestellt.

Verantwortlich dafür ist die Verwendung von Vorausdeklarationen, die von einem Python-Skript innerhalb des Build-Vorgangs berechnet werden und damit einen überdurchschnittlich individuellen Build-Prozess erfordern. In der Konsequenz kann die ● Programmentwicklung bei den betroffenen Anwendern massiv beeinträchtigt werden.

### ● Inkonsistenzen bzgl. STL<sup>DG&TD</sup>

SqAn basiert praktisch in Reinkultur auf dem Programmierparadigma ● Templatemetaprogrammierung mit allen seinen Folgeentwurfsentscheidungen wie den ● Metafunktionen und ignoriert dabei weitgehend die ● Paradigmatische Prägung seiner Anwender. Dies kann dazu führen dazu, dass Anwender mit einer *C++-/STL-Objektorientierungsprägung* auf zwei Weisen behindert werden. Einerseits können sie in ihrer Erwartung, objektorientierte Programmierung anzutreffen, vollkommen überrascht werden (“Vergewaltigung der OO-Programmierung”)<sup>8</sup>. Andererseits kann es passieren, dass kaum ein Anwender auf vorhandene Kenntnisse zurückgreifen kann (● Wiederverwendungsstrategien), die ihnen beim ● Erlernen von und bei der ● Programmentwicklung mit SqAn helfen könnten. Ein entsprechendes Phänomen ist die folgende Aussage: “SeqAn is a completely different way of doing mostly the same thing [as the STL].”<sup>8</sup>

Bei diesem Problem handelt es sich um ein besonders schweres Problem, denn die Auswirkungen erstrecken sich sowohl auf die Erlernbarkeit (“It wasn’t that straight forward as you would have it expected”<sup>8</sup>), wie auch die ● Programmentwicklung (“Mir fällt es sehr schwer zu experimentieren, weil das handling von Ein- und Ausgabe schwierig ist [...].”<sup>8</sup>), als auch die ● Emotionen (“[SeqAn] is kind of annoying.”<sup>8</sup>).

SqAn unterscheidet sich stark von der C++ Standard Template Library (STL), was weitere Usability-Probleme zur Folge haben kann<sup>DG&TD</sup>:

- \* SeqAn verwendet ● generische Programmierung, was dazu führt, dass alle Memberfunktionen technisch global implementiert sind. Die STL hingegen nutzt größtenteils Memberfunktionen. Bei der Vorstellung des Problems ● Identifikation relevanter Funktionen gehe ich auf mögliche Auswirkungen ein.

- \* Durch die globale Implementierung von Memberfunktionen können gleichnamige generische Funktionen unterschiedliche Rückgabetypen zurückgeben. Dies wird hauptsächlich durch das Problem ● Komplizierte Konstruktion von Typen erfasst.
- \* Die ● Templatemetaprogrammierung führt weitere sprachliche Mittel ein, die in der STL nicht existieren und welche ich unter ● Sprachentitätstypen erläutere.
- \* Die Verwendung von Iteratoren unterscheidet sich in SeqAn von der in der STL. Dies wird in ● Iteratorengebrauch beschrieben.

## ● Strukturprobleme

Strukturprobleme umfassen alle Probleme, welche die statische Struktur von SeqAn betreffen. Sie werden in vier Klassen wie folgt unterteilt:

### ● Syntaxprobleme

Die bereits auf Seite 259 beschriebene *explizite-intuitive* ● sprachliche Entwurfsentscheidung ● Datenstrukturmodifikation hat die zwei Eigenschaften *Direktheit* und *Explizitheit*, wodurch vier Varianten ermöglicht werden, Datenstrukturen zu verändern. In SeqAn kommen — soweit ich das beurteilen kann weitgehend unüberlegt — alle vier Möglichkeiten zum Einsatz, was zu den folgenden drei Usability-Problemen führen kann:

### ● Funktionsrückgabenmodifikation

Die *indirekte-explizite* ● Datenstrukturmodifikation wird negativ als unnötige Neuerfindung des Klammern-Operators bezeichnet<sup>10</sup>. Ein anderer Diskussionsteilnehmer drückt mit seiner Frage, ob dies der übliche Weg sei, Daten zu verändern, seine Überraschung über diese Variante aus<sup>11</sup>.

Ein konkretes Beispiel ist die Funktion `infix`<sup>10</sup>, die beispielsweise die Variable `myString` mit dem Wert Eingabe ATTACGG, unter dem Aufruf von `infix(myString, 1, 1) = "cgt"`, in ACGTTTACGG ändert.

### ● Versteckte Parameterübergabe

Die *indirekte-implizite* ● Datenstrukturmodifikation verändert “unsichtbar” Datenstrukturen, was die Anwender ebenfalls überrascht und ärgert<sup>12</sup>.

In dem folgenden Beispiel, das aus der SeqAn-Dokumentation stammt<sup>11</sup>, verändert die `hash`-Funktion die Variable `index` indirekt, sodass die Funktion `getOccurrences` damit arbeiten kann. Man kann also sagen, dass ein eigentlich expliziter Parameter versteckt an `getOccurrences` übergeben wird.

```
typedef Index<DnaString, IndexQGram<UngappedShape<3>>> TIndex;
TIndex index("CATGATTACATA");
hash(indexShape(index), "CAT");
for (unsigned i = 0; i < length(getOccurrences(index,
    indexShape(index))); ++i)
    std::cout << getOccurrences(index, indexShape(index))[i] << std::endl;
return 0;
```

10 [http://docs.seqan.de/seqan/develop/concept\\_SegmentableConcept.html#SegmentableConcept%23infix](http://docs.seqan.de/seqan/develop/concept_SegmentableConcept.html#SegmentableConcept%23infix)

11 <http://seqan.readthedocs.org/en/latest/Tutorial/IndexQGram.html#example>

### ● Operatoreninkonsistenz

Die *indirekte-explizite* Datenstrukturmodifikation wird als schwierig empfunden, denn selbst wenn sich die Anwender an diesen Stil gewöhnen, werden sie davon überrascht, dass dies nicht immer funktioniert<sup>8</sup>.

Die Ursache dafür ist, dass nicht jede Funktion in SeqAn eine Referenz zurückgibt und/oder der `=`-Operator nicht immer spezialisiert wurde.

Diese Inkonsistenzen sind bereits bei einfachen Beispielen zu finden. So kann ein SeqAn-String nicht ohne Weiteres einem C++-String mit Hilfe von `std::string myString = seqanString;` zugewiesen werden. Stattdessen muss die Zuweisung mittels `std::string myString; assign(myString, seqanString);` erfolgen. Dieses Beispiel ist besonders relevant, war doch eine Facette des Usability-Entwurfsziels die Kompatibilität (Gogol-Döring 2009, S. 25).

### Zwischenzusammenfassung

● Syntaxprobleme werden hauptsächlich durch die *explizite-intuitive* sprachliche Entwurfsentscheidung Datenstrukturmodifikation verursacht und verringern die Erlernbarkeit, behindern die Programmierung und lösen negative Emotionen aus.

### ● Benennungsprobleme

Für diese Problemklasse im Falle von SeqAn sind alle *impliziten* bis *expliziten-intuitiven* sprachlichen Entwurfsentscheidungen verantwortlich.

### ● Synonyme / Redundanz

Dieses Usability-Problem kann durch Shortcuts verursacht werden. Dabei werden Typdefinitionen mit Hilfe von `typedef` vorgenommen, um den Anwendern einen Komfort zu bieten. Beispielsweise kann man an Stelle von `String<Dna>` auch den Shortcut `DnaString` verwenden.

Ein Gruppendiskussionsteilnehmer bezeichnet dieses Feature als “stupid”<sup>9</sup>. Die sich zu Wort meldenden Gruppendiskussionsteilnehmer sprechen sich mehrheitlich dafür aus, es den Anwendern zu überlassen, selbst Shortcuts zu definieren<sup>10</sup>.

### ● Abstraktionssuggestion

● Shortcuts können Abstraktionen vortäuschen, die nicht vorhanden sind. Beispiel: `Peptide` ist ein Shortcut für `String<AminoAcid>`. Ein Anwender bringt den Kern des Problems auf den Punkt: “If you read Peptide it makes you think that it’s more than a shortcut.”<sup>11</sup>

Die Diskussionsteilnehmer haben zum größten Teil eine objektorientierte Paradigmatische Prägung, was ein starkes Indiz dafür ist, dass es zwischen dieser Prägung und dem Problem der Abstraktionssuggestion einen Zusammenhang gibt.

### ● Ununterscheidbarkeit von Typen

Dieses Benennungsproblem besteht darin, dass der Unterschied zwischen verschiedenen Klassen den Anwendern nicht klar ist. Das trifft beispielsweise auf die spezialisierte

Klasse `String<String>` und `StringSet` zu, was von Anwendern als verwirrend bezeichnet wird<sup>10</sup>. In diesem konkreten Beispiel ist es *nicht* so, dass mit dem `StringSet` etwa eine ungeordnete Menge gemeint wäre, bei der jedes Element maximal einmal enthalten sein darf. Stattdessen bietet das `StringSet` Funktionen an, über die `String<String>` nicht verfügt<sup>12</sup>.

### ● Technisch vs. Fachlich<sup>DG&TD</sup>

Die *implizite* sprachliche Entwurfsentscheidung Domänen-spezifische Benennung vermischt Termini aus verschiedenen Domänen — bei SeqAn aus der Informatik (z.B. `Alloc`), der Bioinformatik (z.B. `String`, `Alphabets`, `Gaps`) und der Biologie (z.B. `Peptide`). Dies kann zu Problemen führen, wenn die Anwender eine Funktion mit einer bestimmten Funktionalität suchen.

Ein Beispiel dafür ist die in der Informatik bekannte `substring`-Funktion, die eine Teilzeichenkette ihrer Eingabe zurückgibt. In SeqAn wurde jedoch diese Funktion, im Sinne der Bioinformatik, `infix` genannt. Allerdings suchen viele Anwender die entsprechende Funktion unter dem Namen `substring`<sup>1000</sup>. Sie fanden die Funktion nicht unter diesem Namen und waren dazu gezwungen, synonyme Bezeichner zu erraten (z.B. `segment`<sup>33</sup>), was ich als die Explorationsstrategie Namenerratte bezeichne und in SeqAn bei einigen Teilnehmer kläglich scheitert und massiven Unmut auslöst.

Dieses Problem ist in der Literatur (siehe Abschnitt 2.6, Furnas et al. 1987; Good et al. 1984) als Vocabulary Problem bekannt, findet aber in keiner mir bekannten Arbeit Beachtung. Es beschreibt die geringe Wahrscheinlichkeit, dass zwei Menschen für eine noch nicht benannte Sache den selben Namen vergeben. Dies lässt sich auf SeqAn übertragen. Dabei müssen der verantwortliche SeqAn-Entwickler und ein SeqAn-Anwender auf die gleiche Bezeichnung für eine Funktion kommen, damit dieses Problem nicht auftritt.

In diesem konkreten Fall wird das *Vocabulary Problem* durch die Herkunft der Bezeichner geprägt, weshalb ich von der Vermischung technischer und fachlicher Termini spreche.

### ● Benennungskonsistenz<sup>TD</sup>

Dieses Problem thematisiert inkonsistente Benennungen und hat verschiedene Ausprägungen. So gibt es beispielsweise unterschiedliche Bezeichnungen für Getter- und Setter-Funktionen. Für die SeqAn-Align-Klasse gibt es die Set-Funktion `assignSource`. Eine ähnliche Get-Funktion heißt jedoch `row` anstelle von `getRow` oder `retrieveRow`.

Ein weiteres Beispiel sind die bereits genannten Shortcuts. Die Anwender sprechen sich dafür aus, dass, wenn schon Shortcuts eingesetzt werden, diese wenigstens konsistent sein sollten. Sie schlagen vor, dass das Shortcut `Peptide` (= `String<AminoAcid>`) `AminoAcidString` heißen müsste<sup>9</sup>.

## Zwischenzusammenfassung

● Benennungsprobleme werden hauptsächlich durch die *impliziten* bis *expliziten-intuitiven*

12 <http://seqan.readthedocs.org/en/latest/Tutorial/StringSets.html#background>

Die sprachlichen Entwurfsentscheidungen verursachen die Erlernbarkeit, behindern die Programmierung und lösen negative Emotionen aus. Anwender werden dazu gedrungen, die Strategie Namennerraten zu verwenden, was nur einen begrenzten Erfolg hat.

### ● **Funktionsbezogene Probleme**

Diese Problemklasse umfasst alle Probleme, die sich auf Funktionen beziehen.

#### ● **Fehlende Funktionskategorisierung<sup>TD</sup>**

oder: *Wie werden Funktionen kategorisiert?*

Dieses Problem wird bei SeqAn durch die *explizite-argumentative* architektonische Entwurfsentscheidung Generische Programmierung verursacht und besteht darin, dass generische Funktionen nicht kategorisiert sind. Anwender haben entsprechend Probleme, zu einer Klasse gehörende Funktionen zu finden, was zu dem folgenden Problem führt.

#### ● **Identifikation relevanter Funktionen<sup>DG&TD</sup>**

oder: *Welche Funktionen stehen mir zur Verfügung?*

Durch die fehlende Funktionskategorisierung kann es dazu kommen, dass Anwender mit einer *objektorientierten* paradigmatischen Prägung nicht die Explorationsstrategie Anfangsklassenverwendung nutzen können, um die zur Verfügung stehenden Funktionen für ein bestimmtes Objekt zu identifizieren<sup>13</sup>. Bei dieser Strategie, die in der Arbeit von Stylos u. Myers (2008) nachgewiesen wurde (siehe Abschnitt 2.6.6.3), starten Anwender ihre Suche nach verfügbaren Funktionen auf der Grundlage einer so genannten *Anfangsklasse (starter class)*. Diese Klasse bildet meist der Typ des Objektes, das im Mittelpunkt der geplanten Operation steht. Beispiel: Möchte ein Anwender eine Teilzeichenkette für einen **String** berechnen wollen, wird er höchstwahrscheinlich die **String**-Klasse als die Anfangsklasse für die Suche nach der entsprechenden Funktion verwenden.

Bei der Anfangsklassenverwendung handelt es sich um eine Explorationsstrategie, weil diese Strategie auch genutzt wird, um sich einen Überblick über die überhaupt zur Verfügung stehenden Funktionen zu verschaffen<sup>13</sup>.

#### ● **Funktionszweckunerkenntbarkeit**

oder: *Macht die Funktion jetzt X oder Y?*

Dieses Problem hat seine Ursache ebenfalls in der fehlenden Funktionskategorisierung und damit in der Entwurfsentscheidung Generische Programmierung.

In SeqAn existieren unterschiedliche Funktionen mit gleichem generischem Namen global. Abhängig von den Eingabetypen, erledigen solche Funktionen unterschiedliche Dinge und geben auch unterschiedlich typisierte Rückgaben zurück, was zu der Forderung führt, unterschiedliche Namen für unterschiedliche Funktionen zu verwenden<sup>13</sup>.

Ein Beispiel sind die folgenden beiden Funktionen:

- `TInfix label(frag, stringSet, seqID)13`

<sup>13</sup> [http://docs.seqan.de/seqan/develop/class\\_Fragment.html#Fragment%23label](http://docs.seqan.de/seqan/develop/class_Fragment.html#Fragment%23label)

- TAlphabet label(it)<sup>14</sup>

### ● Funktionsgebrauch<sup>DG&TD</sup>

oder: *Wie verwende ich diese Funktion?*

Dieses Problem hat eine Reihe von Ausprägungen (● Uneindeutige Signaturbeschreibung, ● Iteratorengebrauch, ● Versteckte Abhängigkeiten), auf die ich aus Platz- und Relevanzgründen nicht weiter eingehe.

Am wichtigsten ist die ● Typing-Problem-Ausprägung, die bei SeqAn ihre Ursache in den *expliziten-argumentativen* ● architektonischen Entwurfsentscheidungen ● Metafunktionen und ● Generische Programmierung hat. Dabei werden Memberfunktionen von Klassen global implementiert, um dem Compiler zu erlauben, die jeweils passendste Implementierung eines generischen Algorithmus zur Kompilierzeit statt zur Laufzeit zu berechnen und damit besonders performante Programme zu ermöglichen. Dadurch können unterschiedliche Implementierungen für Funktionen mit gleichem generischem Namen existieren. In der OOP würde jede Klasse seine eigene Implementierung innerhalb seiner Klasse kapseln. Der Rückgabetyp hängt in beiden Fällen von den Argumenten der Funktion ab und ist in der OOP, durch die technische Klassenzugehörigkeit von Funktionen, einfach zu bestimmen. In SeqAn jedoch muss dieser Typ jedoch mit Hilfe von ● Metafunktionen berechnet werden (z.B. `Row<Align<String<Dna>, ArrayGaps>>::Type &row1 = row(align, 0)`, wobei `Row`<sup>15</sup> die Metafunktion ist).

Ebenso müssen die Eingaben korrekt typisiert sein, damit die richtige Funktion vom Compiler ausgewählt wird. Existieren zwei gleichwertige Funktionen (z.B. bei Alignments beispielsweise `NeedlemanWunsch` oder `MyersHirschberg`) müssen darüber hinaus so genannte *Tags* als weitere Funktionsparameter verwendet werden.

Nicht zuletzt müssen Ein- und Ausgabe-Typen zusammenspielen, was in seiner Gesamtheit die SeqAn-Anwender vor große Probleme stellt. In meinen Programmierfortschritte-Daten befinden sich Trial-and-Error-Phasen<sup>✉ u. ↗</sup>, die sich über mehr als 10 Minuten erstrecken und häufig durch die Verwendung der ● Anwendungswiederverwendungsstrategie ● Blindes Kopieren beendet werden. Die folgenden beiden Zitate verdeutlichen das Problem und die Strategie:

- “Metafunktionen kopiere ich oft nur ohne wirklich zu verstehen was sie tun.”<sup>✉</sup>
- “I end up copying code snippets from other apps without understanding why they work and why what I initially wrote doesn’t.”<sup>✉</sup>
- “most of the time I copy code snippets that seem to do what I need. I have little understanding why in a particular place a particular form or variant or an idiom is used in place of another one.”<sup>✉</sup>

Von der Strategie ● Blindes Kopieren machen nicht nur Anwender mit einem ● opportunistischen Arbeitsstil Gebrauch, was an den weiter unten erläuterten ● Dokumentationsproblemen liegt.

Während die eine Strategie gezwungener Maßen zum Einsatz kommt, wird eine andere Strategie gestört. In der Literatur gibt es die Idee der Hypothesenüberprüfung

14 [http://docs.seqan.de/seqan/develop/specialization\\_OutEdgeIterator.html#OutEdgeIterator%23label](http://docs.seqan.de/seqan/develop/specialization_OutEdgeIterator.html#OutEdgeIterator%23label)

15 [http://docs.seqan.de/seqan/develop/class\\_Align.html#Align%23Row](http://docs.seqan.de/seqan/develop/class_Align.html#Align%23Row)

zum Verstehen (siehe Abschnitt 2.2.2, Brooks 1983) bzw. zum Verwendung (siehe Abschnitt 2.3.2, Blackwell u. Green 2000; Green 1989; Green u. Petre 1995) von APIs. Diese Beobachtungen fasse ich unter der ● Lernstrategie ● Experimentelle Hypothesenüberprüfung zusammen und verstehe darunter das Herumexperimentieren mit Code, um auf diesem Weg die Funktionsweise einer API zu verstehen<sup>TD</sup>. Diese Strategie wird durch das ● Typing-Problem-verursachende Nicht-● Kompilieren gestört: “Meistens lassen sich Dinge nicht überprüfen, weil der Code nicht kompiliert.”<sup>8</sup>

### Zwischenzusammenfassung

● Funktionsbezogene Probleme werden in SeqAn durch *explizite-argumentative* ● architektonische Entwurfsentscheidungen verursacht. Sie verringern die ● Erlernbarkeit, behindern die ● Programmentwicklung und lösen negative ● Emotionen aus. Anwender werden dazu gedrungen, die Strategie ● Blindes Kopieren zu verwenden, was sich allem Anschein nach nicht positiv auf die Erlernbarkeit auswirkt. Darüber hinaus werden Anwender an der erfolgreichen Nutzung der Strategien ● Anfangsklassenverwendung und ● Experimentelle Hypothesenüberprüfung gehindert.

### ● Sprachentitätstypen<sup>TD</sup>

Betrachtet man eine Programmiersprache/API aus phänomenologischer Endanwender-Sicht, besteht sie aus verschiedenen Elementen wie Klassen, Funktionen mit bzw. ohne führendem Punkt (Memberfunktion bzw. globale Funktion), Variablen, etc.

Die von mir beobachteten Anwender bezeichnen diese Elemente als “Konstrukte”<sup>9</sup>, “Konzepte”<sup>10</sup> oder “Idiome”<sup>11</sup>. Konkrete Elemente werden auf merkwürdige Art attribuiert, was auf ein geringes Verständnis dieser Elemente hindeutet: “metafunction and this double colon”<sup>12</sup>.

Hinzu kommt, dass Klassen und Metafunktionen phänomenologisch sehr ähnlich aussehen: Beide fangen mit Großbuchstaben an und verwenden Camel-Case<sup>16</sup>.

Wie die obigen Probleme zeigen, ist ein tieferes Verständnis über die Bedeutung und Funktionsweise dieser Elemente notwendig, damit Anwender mit der API arbeiten können. Zeit also, diesen Elementen einen Namen zu geben. Ich verwende dazu die Bezeichnung ● Sprachentitätstypen.

Das fehlende Verständnis vieler Anwender äußert sich in der ● Verwendungsstrategie ● Metafunktions-Heuristik. Sie besteht darin, bei der Verwendung einer Klasse bzw. Metafunktion und einem gleichzeitigen fehlschlagenden Kompilierversuch, der Klasse bzw. Metafunktion Zeichenketten wie “::Type” oder “<>”<sup>13</sup> anzufügen oder zu entfernen, bis das eigene Programm wieder kompiliert.

Wegen der ● generischen Programmierung entfällt technisch die ● Funktionskategorisierung. Dennoch existiert diese Zugehörigkeit weiterhin inhaltlich, denn nicht jede generische Funktion funktioniert mit jedem Eingabetyp.<sup>17</sup>

<sup>16</sup> ... was auch als Pascal-Case bezeichnet wird.

<sup>17</sup> Einmal abgesehen von der `length`-Funktion, die bei unbekannten Eingaben immer 1 zurückgibt, was für sich genommen bereits ein Usability-Problem darstellt.

Diese Zugehörigkeit wird in SeqAn mit Hilfe des Sprachentitätstyps *Concept*<sup>18</sup> ausgedrückt. Concepts sind eng verwandt mit den aus Java bekannten *Interfaces* und formulieren Typ-Anforderungen. So gibt es beispielsweise das **StringConcept**<sup>19</sup>, das Anforderungen an Klassen stellt, die dieses Concept implementieren. Damit steht also fest, welche Funktionen beispielsweise Eingaben vom Typ **String** unterstützen.

Theoretisch erlauben Concepts C++-generischen Funktionen Anforderungen an ihre Eingabe so zu formulieren, dass sie vom Compiler überprüft werden können. Praktisch ist dies jedoch nicht der Fall, da Concepts nicht in den aktuellen C++11-Sprachstandard aufgenommen wurden. Dennoch werden Concepts in der SeqAn-Dokumentation dargestellt, was zu neuen Sprachentitätstypen führt. Obwohl beides technisch identisch ist, unterscheiden sich inhaltlich *globale Metafunktionen* und *Interfacemetafunktionen*, denn Letztere werden von einem Concept formuliert. Beide können technisch gesehen Argumente beliebigen Typs verarbeiten. Jedoch kommt es bei Interfacemetafunktionen höchstwahrscheinlich zu einem Laufzeitfehler, wenn ein Argument mit einem ungeeigneten Typ übergeben wird, da in diesem Fall die generische Funktion Operationen ausführt, die nicht vom Typ unterstützt werden.

Noch komplizierter wird es beim Sprachentitätstyp *Funktion*. In SeqAn wird dieser unterschieden in die Sprachentitätstyp-Varianten *globale Funktion*, *Memberfunktion* und *Interfacefunktion*. Technisch sind wiederum globale und Interfacefunktion identisch. Interfacefunktionen sind allerdings — analog zu den Interfacemetafunktionen — durch ein Concept formuliert und führen bei ungeeigneten Eingaben wiederum zu Laufzeitproblemen.

Meine technisch-deduktiven Ausführungen machen deutlich, wie komplex auf inhaltlicher Seite eine Differenzierung durch API-Entwickler vorgenommen wird, welche man technisch und syntaktisch nicht unterscheiden kann, ohne das dahinter stehende Konzept verstanden zu haben. Die damit einhergehenden Probleme kann ich nicht vollständig sicher benennen, denn die dafür notwendigen Daten waren in den Fragebögen und in der Gruppen-diskussion nur unzureichend enthalten. Lediglich die objektive Programmierfortschritte-Datenquelle enthielt entsprechende Daten, deren weitere Auswertung meine Zeitplanung (vgl. Abschnitt 3.1.4) nicht mehr zuließ. Es besteht aber der dringende Verdacht, dass das mangelnde Anwender-Verständnis von ● Sprachentitätstypen, die ● Metafunktions-Heuristik-Strategie und die diversen schweren Usability-Probleme in einem unmittelbaren Zusammenhang stehen.

## Zwischenzusammenfassung

- Syntaxprobleme werden sowohl durch die *impliziten* bis *expliziten-intuitiven* ● sprachlichen Entwurfsentscheidungen als auch durch die *expliziten-argumentativen* ● architektonischen Entwurfsentscheidungen verursacht. ● Syntaxprobleme hindern Anwender beim Gebrauch wichtiger ● Strategien wie der ● Anfangsklassenverwendung und der ● experimentellen Hypothesenüberprüfung. Auf andere Strategien wie dem ● Namenerratoren und dem ● blinden Kopieren weichen Anwender hingegen aus. Die Folgen erstrecken sich von einer verringerten

18 <http://en.cppreference.com/w/cpp/concept>

19 [http://docs.seqan.de/seqan/develop/concept\\_StringConcept.html](http://docs.seqan.de/seqan/develop/concept_StringConcept.html)

Erlernbarkeit über die gestörte Programmierung bis hin zu negativen Emotionen. In SeqAn neu eingeführte und für seine Anwender vollkommen unbekannte Sprachentitätstypen scheinen eine vielversprechender Grundlage für die Evaluation von APIs zu sein.

### ● Dokumentationsprobleme

Die konkreten Usability-Probleme dieser Klasse halte ich für selbsterklärend und erforscht genug (siehe Abschnitt 2.6), um sie im Folgenden etwas knapper vorzustellen.

### ● Fehlender Gesamtüberblick

Die eben beschriebenen Probleme im Zusammenhang mit den Sprachentitätstypen und eine Vielzahl von Beobachtungen weisen eindeutig darauf hin, dass der SeqAn-Dokumentation ein Gesamtüberblick fehlt, der den konzeptionellen Rahmen des SeqAn-Entwurfs gebündelt darstellt. Einige relevante Äußerungen lauten:

- “Um manche Dinge effizienter zu lösen wäre es gut eine Art Gesamtüberblick zu erlangen.”<sup>8</sup>
- “[...] once you know the concepts and tools [...]”<sup>8</sup>
- “[I need] a book: a properly written hard-copy book-manual-reference.”<sup>8</sup>

Das Fehlen dieser Gesamtübersicht kann dazu führen, dass die Lernstrategie Konzeptverstehen behindert wird<sup>8</sup>. Darunter verstehe ich das systematische Erlernen von SeqAn.

### ● Fehlende Anwendungsbeispiele

Viele Dokumentationseinträge verfügen über keine oder nur unzureichend verständliche Beispiele<sup>8</sup>.

Anwender mit einem pragmatischen Arbeitsstil nutzen Beispiele, um so SeqAn kostengünstig zu erlernen (Ye u. Fischer 2002). Ich bezeichne diese Strategie als Beispiel-Lernen<sup>TD</sup>. Dabei werden Beispiele gelesen und höchstens in Teilen kopiert — ein Teilnehmer teilte mir im Gespräch mit, sogar häufig den Dokumentationstext zu ignorieren und direkt zum Beispiel zu springen<sup>8</sup>. Diese Strategie scheitert jedoch im Falle fehlender Beispiele.

Eine weitere Verwendungsstrategie ist das Blinde Kopieren<sup>DG&TD</sup>, was zugleich eine Variante der Verständnisvermeidung (Lange u. Moher 1989) darstellt, mit der Floskel *Weg des geringsten Widerstands* umschrieben werden kann (Ko u. Myers 2005) und die häufig in der Gruppe der API-Endanwender zu beobachten ist (Ko et al. 2011). In dieser Gruppe wird häufig ein opportunistischer Arbeitsstil festgestellt (Ko et al. 2011).

Ich konnte Blinde Kopieren vielfach sowohl in meinen objektiven<sup>8</sup>, als auch subjektiven Daten<sup>8</sup> beobachten, bei denen Anwender keine Tutorials bearbeitet, sondern eigenen Code geschrieben hatten. Offensichtlich wird diese Strategie durch das Fehlen von Beispielen behindert. Das ist besonders dramatisch, als das die insgesamt geringe Usability von SeqAn allem Anschein nach auch pragmatische und systematische Anwender zu dieser Strategie zwingt<sup>TD</sup>.

### ● Fehlende Dokumentation der Rückgabetypen

Ein weiteres Problem kann die ● Fehlende Dokumentation der Rückgabetypen von Funktionen sein. In der SeqAn-Dokumentation wird der Rückgabetyp einer Funktion vollkommen unzureichend beschrieben.

Beispielsweise lautet die Signatur der `indexText`-Funktion<sup>20</sup> in der SeqAn-Dokumentation `TFibre indexText(index)`, wobei `TFibre` mit den Worten “A reference to the text index.” erklärt wird. Korrekter Weise müsste die Signatur `Fibre<TContainer, TSpec>::Type indexText(index)` lauten.

Viele Anwender<sup>21</sup> stehen also vor dem Problem, nicht zu wissen, welchen Typ die Variable haben muss, die das Ergebnis der Funktion speichert. Genauer gesagt wissen sie nicht, welche Metafunktion verwendet werden muss, um den korrekten Typ zur Kompilierzeit zu berechnen — nämlich die `Fibre`-Metafunktion<sup>21</sup>. Nähere Details zu den Auswirkungen dieses Problems habe ich bereits unter ● Typing beschrieben.

### ● Suchfunktion

Die Suchfunktion der SeqAn-Dokumentation ist unzureichend und wird u.a. als “umständlich”<sup>22</sup> beschrieben. Weiterhin wird die Art der Sortierung bemängelt<sup>23</sup>, bei der weder eine Gruppierung, nach ● Sprachentitätstyp noch eine gewichtete Sortierung anzutreffen ist.

### ○ Tutorials

Abgesehen von der Enttäuschung, die auf die ● schnellen Anfangserfolge folgt, konnte ich keine nennenswerten Usability-Probleme bzgl. der SeqAn-Tutorials feststellen. Dies ist ein gewichtiges Indiz für die im Abschnitt 3.2.4.5 beschriebenen Tutorialverbesserungen.

## Zwischenzusammenfassung

- Dokumentationsprobleme hindern Anwender beim Gebrauch wichtiger ● Strategien abhängig von ihrem ● Arbeitsstil: Systematiker können kein ● Konzeptverstehen betreiben und Pragmatiker kein ● Beispiel-Lernen. Stattdessen werden sie durch die bestehenden Probleme dazu gedrängt, das bei Opportunisten häufig beobachtete ● Blinde Kopieren zu nutzen. Infolgedessen wird die ● Erlernbarkeit verringert, die ● Programmierung gestört und es werden negative ● Emotionen ausgelöst.

### ● Laufzeitprobleme

Diese Art von Problemen treten erst zu Tage, wenn der Anwender mit SeqAn entwickelte Programme ausführt. Zwei Probleme konnte ich dabei entdecken:

### ● Compiler-Fehlermeldungen

Schlägt ein Kompilierversuch fehl, meldet der Compiler schwer zu lesende Fehlermeldungen, was bereits Gogol-Döring (2009) bei seiner ● Entwurfsentscheidung ● Templatemetaprogrammierung einräumte und von den Anwendern wie folgt bestätigt wird:

- “Got some compiler errors. [I] tried to guess what it wants [but] couldn’t understand them.”<sup>24</sup>

20 [http://docs.seqan.de/seqan/develop/class\\_Index.html#Index%23indexText](http://docs.seqan.de/seqan/develop/class_Index.html#Index%23indexText)

21 [http://docs.seqan.de/seqan/develop/global\\_metafunction\\_Fibre.html](http://docs.seqan.de/seqan/develop/global_metafunction_Fibre.html)

- “Errors concerning missing semicolons, includes or brackets can be read easily — but these are the easy mistakes.”<sup>8</sup>

### Versagensverschleppung

Dieses Usability-Problem ist in der Literatur bekannt und wird aus einer lösungsorientierten Perspektive als *fail fast* (Bloch 2006a) bezeichnet.

In SeqAn wird zur Erfüllung des  Entwurfsentscheidungsziels *Performance* auf Prüfroutinen verzichtet.

Zwei Beispiele sollen dieses Verhalten verdeutlichen:

- Die generische Funktion `length`<sup>22</sup> gibt für alle Eingaben, für die `length` nicht spezialisiert wurde, 1 zurück.
- Die Funktionen zum Lesen von Sequenzdaten akzeptieren auch ungültige Daten. Wird beispielsweise eine RNA gelesen, die nur aus den gültigen Nukleotiden A, C, G und U bestehen sollte, reagieren die entsprechenden Lesefunktion mit einer automatischen Umwandlung aller ungültigen Zeichen zu A, ohne einen Hinweis auf derartige Lesefehler zu melden.

Die Anwender reagierten auf dieses Verhalten mit  Verunsicherung und  Unverständnis / Genervtheit:

- “You wanna know if something is wrong.”<sup>9</sup>
- “How do you know something is not okay?”<sup>10</sup>
- “[This] posed a lot of problems just because of a small mistake.”<sup>11</sup>

Die Anwender waren einhellig der Meinung, dass es zumindest die Möglichkeit geben müsse, eine Validitätsprüfung durchzuführen: “Yes, it takes time, but the time you need to find the bug in further phases is much higher.”<sup>12</sup>

### Zwischenzusammenfassung

 Laufzeitprobleme werden durch die  Entwurfsentscheidung  Templatemetaprogrammierung selbst und dessen  Entwurfsentscheidungsziel *Performance* verursacht. Die schwer verständlichen  Compiler-Fehlermeldungen sind insbesondere für API-Endanwender problematisch. Die  Versagensverschleppung wiederum führt zu  Verunsicherung und  Unverständnis / Genervtheit und kann die Ursache für eine lange Defektsuche bedeuten, was die  Programmierung ausbremsst.

### Werkzeugunterstützungsprobleme<sup>DG&TD</sup>

Durch die  Entwurfsentscheidung  Templatemetaprogrammierung und der Verwendung des eigens entwickelten  Dokumentationsformats *DDDoc*, erfährt SeqAn nur eine geringe Unterstützung durch Entwicklungsumgebungen. Die Aussage “not comparable to the Java API [IDE support]”<sup>13</sup> verdeutlicht diese Problemklasse.

### Fehlende Integration der Dokumentation<sup>TD</sup>

Ein Problem kann darin bestehen, dass die Dokumentation stets manuell aufgerufen werden

---

<sup>22</sup> [http://docs.seqan.de/seqan/develop/concept\\_ConsoleConcept.html#ContainerConcept%23length](http://docs.seqan.de/seqan/develop/concept_ConsoleConcept.html#ContainerConcept%23length)

muss, weil keine Entwicklungsumgebung das speziell für SeqAn entwickelte, exotische *DD-Doc*-Format lesen kann. Dieses Problem habe ich jedoch nicht weiter untersucht, weil es mir in der GTM-Analyse nicht weiter auffiel. Lediglich bei meiner in Abschnitt 3.2 vorgestellten Heuristischen Evaluation wurde ich darauf aufmerksam.

### ● Fehlende Auto vervollständigung<sup>DG&TD</sup>

Ein anderes Problem kann die fehlende bzw. rudimentäre Auto vervollständigung sein. Da SeqAn praktisch sämtliche Funktionen global implementiert, gibt es für SeqAn-Anwender keine Möglichkeit, die auf ein Objekt anwendbaren Funktionen aufzulisten.

Bei Anwendern mit einer ● objektorientierten paradigmatischen Prägung, zu denen fast alle der von mir beobachteten Anwender gehören, kommt dabei die ● Explorationsstrategie ● Punkt-Funktionssuche zum Einsatz. Sie besteht darin, hinter dem Objekt der *Anfangsklasse* (vgl. ● Anfangsklassenverwendung bzw. Stylos u. Myers 2008) einen Punkt zu setzen und die zur Verfügung stehenden Funktionen einblenden zu lassen. Oder, um es mit den Worten eines Anwenders zu beschreiben,: “I just put a dot and the IDE shows me the available functions”<sup>23</sup>. Allerdings kann diese Strategie im Falle von SeqAn aus den oben genannten Gründen nicht eingesetzt werden, was zu einer negativen Beeinträchtigung der ● Erlernbarkeit führt.

Ein Teilnehmer<sup>23</sup> hat wegen dieser Einschränkung die Strategie ● Buffer-Auto vervollständigung entwickelt. Sie besteht in der Verwendung der Buffer-Auto vervollständigung, wie sie von bekannten Editoren wie *Vim*<sup>23</sup> unterstützt werden.

Omar et al. (2012) haben beobachtet, dass die Code vervollständigungsfunktion in IDEs immer häufiger von API-Anwendern genutzt werden. Kann diese nicht genutzt werden, müssen Anwender auf die Dokumentation zurückgreifen, was die Autoren als “mental overhead” bewerten.

---

23 [http://vim.wikia.com/wiki/Any\\_word\\_completion](http://vim.wikia.com/wiki/Any_word_completion)



## ZUSAMMENFASSUNG

### 4.2.1 Theorie über die Entstehung und Auswirkungen von Entwurfsentscheidungen in SeqAn

Die quelloffene C++-basierte Softwarebibliothek SeqAn dient der Sequenzanalyse und wurde ursprünglich von Andreas Gogol-Döring im Rahmen seiner gleichnamigen Dissertation (Gogol-Döring 2009) entwickelt. Dabei verfolgte der Autor das ultimative Entwurfsentscheidungsziel *Performance*, das er durch die architektonische Entwurfsentscheidung Templatemetaprogrammierung nachweislich erreichte.

Die übrigen Entwurfsentscheidungen dienten anderen Zielen, die sich zu *Usability* zusammenfassen lassen. Im Unterschied zur Templatemetaprogrammierung wurden diese Entscheidungen auf einer weniger soliden Entwurfsentscheidungsgrundlage getroffen. Während Metafunktionen, Generische Programmierung und Template-Spezialisierung noch *argumentiert* wurden, existiert für die Entwurfsentscheidungen Shortcuts und Datenstrukturmodifikation nur eine *intuitive* Begründung. Die Frameworkgestalt von und die Domänen-spezifische Benennung in SeqAn werden von dem Autor nur *implizit* thematisiert.

Die Schwächen dieser intuitiven und argumentativen Entwurfsentscheidungen liegen einerseits in den teilweise persönlichen Präferenzen des Autors. Andererseits sind die Schwächen der unzureichenden Berücksichtigung der Anforderungen zukünftiger Anwender geschuldet, denn der SeqAn-Entwurf weckt bei seinen Anwendern die produktbedingte Erwartung einer performanten, objektorientierten und benutzerfreundlichen Library. Jedoch ist SeqAn keine Library, sondern ein Framework, verwendet keine objektorientierte Programmierung, sondern Templatemetaprogrammierung und hat schwere Defizite in Bezug auf seine Usability.

Fasst man alle Ursachen zu einer zusammen, kommt man zu dem Schluss, dass es zwar gelungen ist, SeqAn-basierte Programme extrem schnell zu machen, es jedoch nicht gelungen ist, die übrigen Erwartungen der Anwender — insbesondere bezüglich der Wiederverwendbarkeit bestehender paradigmatisch geprägter Vorkenntnisse — zu erfüllen, was sich in der Aussage “SeqAn makes me feel stupid”<sup>8</sup> widerspiegelt.

In Bezug auf die Anwenderschaft ergeben sich durch die getroffenen Entwurfsentscheidungen eine Reihe von Usability-Problemen. Unterteilt werden können die Usability-Probleme in querschnittige produktbedingte Erwartungskonformitätsprobleme und in längsschnittige Struktur-, Dokumentations-, Laufzeit- und Werkzeugunterstützungsprobleme.

Treffen Anwender auf Usability-Probleme, werden Strategien, die auf den anwenderseitigen Arbeitsstilen und paradigmatischen Prägungen basieren, häufig behindert. Dabei sind besonders

die ● experimentelle Hypothesenüberprüfung, das systematische ● Konzeptverstehen, das pragmatische ● Beispiel-Lernen, die ● Anfangsklassenverwendung und die ● Punkt-Funktionssuche zu nennen.

Hingegen müssen Anwender auf andere Strategien ausweichen. Zu diesen Strategien gehören das ● Namenerraten und das opportunistische ● blinde Kopieren.

Die Folgen der teilweise ungelösten Usability-Probleme, die in den Entwurfsentscheidungen ihre Ursache haben, erstrecken sich auf eine deutlich verringerte ● Erlernbarkeit, eine teils ausgebremste, teils gestörte ● Programmentwicklung und die Empfindung auffallend negativer ● Emotionen.

In SeqAn neu eingeführte und für seine Anwender vollkommen unbekannte ● Sprachentitätstypen scheinen eine vielversprechender Grundlage für die Evaluation von APIs zu sein.

### 4.2.2 Einsichten

Die GTM-Analyse bestätigt das Ergebnis der zuvor durchgeführten Datenauswertung (siehe Abschnitt 3.2), bei der ich bereits herausfand, dass viele Probleme auf die Templatemetaprogrammierung zurückzuführen sind und die Erlernbarkeit negativ beeinflusst wird. Allerdings brachte die GTM weitaus tiefere Einblicke in die Zusammenhänge, sodass ich Fokus-bedingt nicht auf alle Konzepte im gleichen Maß eingehen konnte (z.B. ● Sichtbarmachung der Implementierung und ● Dokumentationsformat).

Betrachtet man die Usability-Probleme unter dem Aspekt der Zugehörigkeit, kann man die Beobachtung machen, dass einige Probleme relevant für jede Art von APIs sind, während andere Probleme erst im Umfeld von Templatemetaprogrammierung-basierenden APIs auftreten können.

#### Allgemeine Probleme

- Strukturprobleme
    - Technisch vs. Fachlich, ● Ununterscheidbarkeit von Typen, ● Abstraktionssuggestion,
    - Benennungsinkonsistenz
  - Dokumentationsprobleme
    - Fehlende Anwendungsbeispiele
  - Laufzeitprobleme
    - Versagensverschleppung
  - Werkzeugunterstützungsprobleme
-

## Probleme der Templatemetaprogrammierung

### Inhärente Probleme

- Strukturprobleme
  - Inkonsistenzen bzgl. OOP, ● Inkonsistenzen bzgl. STL, ● Sprachentitätstypen,
  - Funktionszweckunerkennbarkeit, ● Funktionsgebrauch
- Dokumentationsprobleme
  - Fehlende Funktionskategorisierung, ● Identifikation relevanter Funktionen
- Laufzeitprobleme
  - Compiler-Fehlermeldungen
- Werkzeugunterstützungsprobleme
  - Fehlende Autovervollständigung

### Mögliche Probleme (insbesondere in Hinblick auf generische Programmierung)

- Strukturprobleme
  - Funktionsrückgabemodifikation, ● Operatoreninkonsistenz, ● Versteckte Parameterübergabe
- Dokumentationsprobleme
  - Fehlender Gesamtüberblick, ● Fehlende Dokumentation der Rückgabetypen,
  - Suchfunktion, didaktische Lernressourcen wie ○ Tutorials
- Laufzeitprobleme  
—
- Werkzeugunterstützungsprobleme  
—

Tatsächlich rein individuelle Probleme in Bezug auf SeqAn sind lediglich ● Projektorganisation und ● Synonyme / Redundanz

Abgesehen von der schlechten Lesbarkeit von Compiler-Fehlermeldungen, hat Gogol-Döring (2009) bei seiner ● Entwurfsentscheidung ● Templatemetaprogrammierung die Spannbreite der Folgen dieser Entscheidung weit unterschätzt. Diese Gliederung zeigt jedoch, dass Gogol-Döring diese Folgen kaum hätte abschätzen können, denn wie ich bereits in meiner Literaturübersicht (siehe Abschnitt 2) gezeigt habe, gibt es keinerlei Erkenntnisse in Bezug auf die Usability von Templatemetaprogrammierung-basierenden APIs, für die man hätte sensibilisiert sein können. Umso wichtiger scheint bei derartigen Entwurfsentscheidungen eine explizite-empirische Entwurfsentscheidungsgrundlage notwendig zu sein.

### 4.2.3 Ur-Ursache

Abschließend möchte ich auf subtilere Ursachen für den SeqAn-Entwurf eingehen.

Ich konnte feststellen, dass die Performance eindeutig das Superziel für SeqAn gewesen ist. Kompromisse wurden dabei nur einseitig für die Usability getroffen. Dies zeigt sich auch daran, dass nur die Performance empirisch belegt wurde, während die Usability — etwas spitz ausgedrückt — in Gogol-Dörings Dissertation (Gogol-Döring 2009) lediglich herbei argumentiert wurde.

SeqAn ist unter dem unausgesprochenen Kontext einer Bioinformatik-Arbeitsgruppe entwickelt worden. Viele wissenschaftliche Abschlussarbeiten und mehr als zwei Dutzend Veröffentlichungen<sup>24</sup> kamen unter dem Einsatz von SeqAn zu Stande. Dabei spielte die Performance — neben der Funktionalität — der entwickelten Algorithmen die dominant wichtige Rolle. Usability hatte hingegen eine weit geringere Wichtigkeit, denn die Bereitschaft, Usability-Probleme zu ertragen, ist in einer wissenschaftlichen Arbeitsgruppenumgebung weit höher als im kommerziellen Umfeld.

Genau diese kommerzielle Erweiterung vollzog SeqAn jedoch im Rahmen des Programms *VIP – Validierung des Innovationspotenzials wissenschaftlicher Forschung* (siehe Abschnitt 3.1.1), was eine weit stärkere Betonung der Usability notwendig macht, damit sich SeqAn am Markt behaupten kann.

Allerdings gehen die Vorstellungen der Anwenderschaft innerhalb der verantwortlichen (teils ehemaligen) SeqAn-Entwickler weit auseinander. Während die Einen auch API-Endanwender durch eine sehr niedrigschwellige Usability einbeziehen wollen, fordern die Anderen hohe Vorkenntnisse auf Seiten der API-Anwender. Dies zeigt sich an der eingeschränkten Sensibilität für die Bedürfnisse der SeqAn-Anwender, welche häufig *technisch wegargumentiert* (Sarodnick u. Brau 2006) werden, obwohl offiziell der Usability ein hoher Stellenwert zukommt. Damit verwandt ist eine gewisse Selbstüberschätzung, was ich mit dem Begriff der *Bauchgefühlusability* bezeichne.

Die Verbreiterung des Einsatzgebietes von SeqAn auf den kommerziellen Markt durch ein Consultance-Vermarktungsmodell — dabei bleibt SeqAn kostenlos und quelloffen, lediglich in Anspruch genommene Dienstleistungen werden bezahlt — zwingen die SeqAn-Entwickler dazu, Benutzerfreundlichkeit aktiv zu betreiben, da sie sonst ein Zufall bleibt (Stylos 2009) und den kommerziellen Erfolg gefährdet (Sunshine et al. 2014). Die Vermutung liegt nahe, dass eine schlechte Usability das neue Geschäftsmodell eigentlich befeuern müsste. Jedoch gelangen SeqAn-Anwender häufig gar nicht an den Punkt, wo sie sich von der ausgezeichneten SeqAn-Performance überzeugen können<sup>3</sup>. Für Studenten, Freelancer und finanzienschwache Projekte würde dies eine Abkehr von SeqAn bedeuten, wenn diese Kundengruppe nicht die finanziellen Mittel aufbringen kann. Dadurch sinkt auch die Chance, dass SeqAn in zukünftigen Projekten zum Einsatz kommt. Nicht ohne Grund stellen große Unternehmen ihre Software für Studenten und nicht kommerzielle Unternehmungen kostenlos zur Verfügung. Entscheider innerhalb größerer Projekte hingegen werden sich wahrscheinlich nur schwer von einer Investition überzeugen lassen, wenn die eigenen Entwickler nicht einmal im Stande sind, zwei Dutzend Zeilen Code fehlerfrei selbst zu schreiben.

Die SeqAn-Entwickler müssen akzeptieren, dass die Existenzberechtigung von SeqAn nicht mehr nur darin besteht, wissenschaftliche Arbeiten zu ermöglichen, sondern von kommerziellen Kunden genutzt werden können muss, von denen viele das Design einer API nicht systematisch studieren (Hou et al. 2005) und häufig den Weg des geringsten Widerstandes gehen (Ko u. Myers 2005).

---

24 <http://www.seqan.de/publications/>

In diesem Unterkapitel habe ich meine Theorie über die Entstehung und Auswirkungen von Entwurfsentscheidungen in SeqAn vorgestellt. Auf der Grundlage der gefundenen Usability-Probleme und Strategien lassen sich Verbesserungsvorschläge erarbeiten, die ich im nächsten Unterkapitel vorstelle.



## VORSCHLÄGE ZUR VERBESSERUNG DER API-USABILITY VON SEQAN

In dem vorangegangenen Unterkapitel habe ich meine Theorie über die Entstehung und Auswirkungen von Entwurfsentscheidungen in SeqAn vorgestellt. Basierend auf den gefundenen Usability-Problemen und Strategien lassen sich Verbesserungsvorschläge erarbeiten, die ich in diesem Unterkapitel vorstelle.

Dazu entwickle ich *Maßnahmen*, die unterschiedliche Usability-Probleme teilweise oder praktisch vollständig lösen. Bei der Formulierung nehme ich keine Rücksicht auf die Abwärtskompatibilität der SeqAn-API. Dieses Vorgehen ist durch den VIP-Förderantrag (siehe Abschnitt 3.1.1) abgedeckt und dank der eng angebundenen SeqAn-Anwender (siehe Abschnitt 1.3.4) auch mit weniger Rücksichtnahme möglich.

### 4.3.1 Bewertung von Fatalität und Aufwand

Für die Priorisierung einer Maßnahme sind die Schweregrade (*Fatalitäten*) der adressierten Usability-Probleme und der Maßnahmendurchführungsaufwand zu ermitteln. Je höher die Gesamtfatalität und je geringer der Aufwand sind, desto höher ist die Priorität einer Maßnahme. Leider kann die Gesamtfatalität einer Reihe von Usability-Problemen nicht einfach durch Addition ermittelt werden (Kahlert 2011).

Bereits die valide Ermittlung der Fatalität eines jeden einzelnen Usability-Problems ist nicht einfach, denn dazu müssten praktisch alle Probleme in einer längeren Beobachtung überprüft worden sein. Schließlich definiert sich die Fatalität u.a. über die Eigenschaft *Persistenz* (Nielsen u. Mack 1994). Ob ein Usability-Problem dauerhaft, oder nur bei den ersten Erscheinungen Schaden anrichtet, kann aber nur so bestimmt werden.

Angesichts dieser Schwierigkeiten und meiner Beobachtung, dass sich praktisch alle Usability-Probleme auf ● Erlernbarkeit, ● Programmentwicklung und ● Emotionen auswirken, habe ich mich dazu entschlossen, die Gesamtfatalität und den Aufwand für jede Maßnahme nur grob zu schätzen.

Für die Schätzung der Gesamtfatalität habe ich mein, im Rahmen der wissenschaftlichen Literaturstudie erworbenes Wissen und die Phänomene meiner gefundenen Usability-Probleme genutzt. Ich verwende dabei die zur Gesamtfatalität inverse Metrik *Kosten* mit der Ordinalskala: *gering*, *mittel* und *hoch*.

Für die Schätzung der Kosten verwende ich dieselbe Ordinalskala wie folgt:

- gering: < 1 Personenmonat
- mittel:  $\geq 1$  Personenmonat und  $< 4$  Personenmonate

- hoch:  $\geq 4$  Personenmonate

### 4.3.2 Maßnahmenkatalog

Der im Folgenden dargestellte Maßnahmenkatalog listet alle gefundenen Usability-Probleme auf. Hinter jedem Usability-Problem sind die Maßnahmen genannt, welche die jeweiligen Usability-Probleme (teilweise) lösen sollen. Im Anschluss an diesen Katalog beschreibe ich die einzelnen Maßnahmen.

#### ● Produktbedingte Erwartungskonformitätsprobleme

- Projektorganisation: *Frameworkumbau*
- Inkonsistenzen bzgl. STL: *STL-Angleichung, Dokumentation*

#### ● Strukturprobleme

##### ● Syntaxprobleme

- Funktionsrückgabenmodifikation: *Inkonsistenzbeseitigung*
- Versteckte Parameterübergabe: *Intransparenzbeseitigung*
- Operatoreninkonsistenz: *Inkonsistenzbeseitigung*

##### ● Benennungsprobleme

- Synonyme / Redundanz: *Shortcuts*
- Abstraktionssuggestion: *Shortcuts, Inkonsistenzbeseitigung*
- Ununterscheidbarkeit von Typen: *Dokumentation*
- Technisch vs. Fachlich: *Inkonsistenzbeseitigung*
- Benennungskonsistenz: *Shortcuts, Inkonsistenzbeseitigung*

##### ● Funktionsbezogene Probleme

- Fehlende Funktionskategorisierung: *Dokumentation*
- Identifikation relevanter Funktionen: *Dokumentation*
- Funktionszweckunerkenntbarkeit: *keine Lösung*
- Funktionsgebrauch: *Dokumentation*

##### ● Sprachentitätstypen: *Dokumentation*

#### ● Dokumentationsprobleme

- Fehlender Gesamtüberblick: *Dokumentation*
- Fehlende Anwendungsbeispiele: *Dokumentation*
- Fehlende Dokumentation der Rückgabetypen: *Dokumentation*
- Suchfunktion: *Dokumentation*
- Tutorials: *Dokumentation*

#### ● Laufzeitprobleme

- Compiler-Fehlermeldungen: *keine Lösung*
- Versagensverschleppung: *Fail-Fast*

### ● Werkzeugunterstützungsprobleme

- Fehlende Integration der Dokumentation: *keine Lösung*
- Fehlende Autovervollständigung: *keine Lösung*

## 4.3.3 Maßnahme: Frameworkumbau

**Nutzen:** hoch; **Kosten:** mittel

SqAn ist aktuell keine Library, sondern ein Framework (siehe Abschnitt 4.1.2). Die Ursache besteht hauptsächlich in der Verwendung von Vorwärtsdeklarationen. Diese müssen beseitigt werden. Auf diese Weise sind Anwender nicht mehr dazu gezwungen, das CMake-Build-System zu verwenden und ihre eigene ● Projektorganisation umzustellen.

## 4.3.4 Maßnahme: STL-Angleichung

**Nutzen:** hoch; **Kosten:** hoch

Ich kam während meiner ersten Analyse-Versuche zu dem Schluss, dass SeqAns größtes Usability-Problem die fehlende objektorientierte Programmierung (OOP), wie man sie aus Java kennt, ist. Allerdings musste ich diesen Glauben nach Analyse der Gruppendiskussion korrigieren. Tatsächlich gibt es zwei wichtige ● paradigmatische Prägungen auf Seiten der SeqAn-Anwender.

Die Eine betrifft tatsächlich die aus Java bekannte OOP. Die zweite Ausprägung jedoch betrifft die OOP, wie sie in C++/STL vorkommt. Beiden gemeinsam ist die Möglichkeit der ● generischen Programmierung, die exzessiv in SeqAn genutzt wird. Jedoch werden generische Funktionen in Java als Memberfunktionen und in C++ häufig als globale Funktionen implementiert<sup>25</sup>.

SqAn unterscheidet sich insbesondere in den folgenden zwei Punkten zu C++/STL, und wird von Anwendern als “extremely messy”<sup>3</sup> beschrieben:

1. Funktionen wie `length` und `resize` sind in C++ Memberfunktionen und in SeqAn globale Funktionen.
2. In C++ sind der Iteratoren-Typ und die entsprechenden Iteratoren-Funktionen als Member der Klasse definiert, über deren Instanzen iteriert werden soll: `std::string::iterator it = s.begin();`

---

<sup>25</sup> Genauer gesagt, inhaltlich generische Funktionen wie die Funktionen der Algorithmen-Library; siehe <http://en.cppreference.com/w/cpp/algorithms>.

In SeqAn hingegen wird der Typ des Iterators über eine Metafunktion berechnet und die Iteratoren-Funktionen global implementiert: `Iterator<String<Dna> >::Type it = begin(genome);`.

Ich empfehle daher, die globalen Funktionen, die in C++ Memberfunktionen sind, in SeqAn zu Memberfunktionen umzuwandeln. Außerdem sollte die Typ-Definition eines Iterators und die entsprechenden Funktionen als Member der entsprechenden Klasse bereitgestellt werden (`CLASS::iterator` bzw. `CLASS.begin`).

Das Problem an dieser Umstrukturierung ist, dass damit die statische Bindung zur Kompilierzeit gefährdet ist, sobald Polymorphismus eingesetzt wird. Dadurch kann es dazu kommen, dass verschiedene Implementierungen — beispielsweise für `resize` — existieren, die aufzurufende Funktion über einen Virtual-Lookup erst zur Laufzeit berechnet wird und kein *Inlining* mehr stattfinden kann. Dies würde die Performance von SeqAn stellenweise zunichte machen. Dieses Problem kann jedoch mit Hilfe des CRTP vermieden werden.

#### 4.3.4.1 Curiously Recurring Template Pattern — CRTP

Während meiner Recherchen traf ich auf das *curiously recurring template pattern* (CRTP), das auch als *simulated dynamic binding* bezeichnet wird und eine spezielle Form des *F-bounded Polymorphismus* darstellt (Canning et al. 1989).

Auch ein besonders qualifizierter SeqAn-Anwender und Gruppendiskussionsteilnehmer bezeichnete das CRTP als “polymorphistische Alternative”<sup>26</sup> zum aktuellen SeqAn-Entwurf.

Das CRTP kommt bereits in einigen *Boost*-Bibliotheken<sup>26</sup> und umfassend in Microsofts *Active Template Library*<sup>27</sup> zum Einsatz.

CRTP basiert auf dem Prinzip, der ererbten Klasse den Typ der erbenden Klasse zu übergeben, um damit eine Bindung zur Kompilierzeit (*static binding*) zu ermöglichen, was das folgende Beispiel veranschaulichen soll:

```
#include <iostream>
using namespace std;

template <typename Child>
struct Base
{
    void interface()
{
```

---

<sup>26</sup> <http://www.boost.org>

<sup>27</sup> <https://msdn.microsoft.com/en-us/library/3ax346b7.aspx>

```

        static_cast<Child*>(this)->implementation();
    }
};

struct Derived : Base<Derived>
{
    void implementation()
    {
        cerr << "Derived";
    }
};

int main()
{
    Derived d;
    d.interface();
}

```

LISTUNG 11: Demonstration des CRTP: Das Programm gibt “Derived” aus, wobei die Memberfunktion `interface` statisch gebunden wurde.

Mir ist es erfolgreich gelungen, für ein kleines Beispiel das CRTP auf SeqAn anzuwenden. Die dazu notwendigen Schritte beschreibe ich im Folgenden. Es ist jedoch weniger meine Absicht, dass jeder Leser die folgenden drei Listungen versteht, sondern vielmehr, meine Ergebnisse für die SeqAn-Entwickler-Nachwelt festzuhalten.

- Einführung einer neuen Basis-Klasse `OOPContainerConcept` (in diesem Beispiel umfasst sie nur die `length`-Memberfunktion):

```

template <typename TThis>
class OOPContainerConcept {
public:
    SEQAN_HOST_DEVICE inline typename Size<TThis>::Type
    length() {
        return seqan::length(*static_cast<TThis*>(this));
    }
};

```

- Einführung einer Vererbung für zwei `String`-Spezialisierungen

```
template <typename TValue, typename TSpec>
class String<TValue, Alloc<TSpec> > : public OOPContainerConcept<String<TValue,
→ Alloc<TSpec> > >
```

```
template <typename TValue, typename THostspec>
class String<TValue, Packed<THostspec> > : public
→ OOPContainerConcept<String<TValue, Packed<THostspec> > >
```

- Schreiben von Client-Code

```
#include <iostream>
#include <vector>
#include <string>

#include <seqan/sequence.h>
#include <seqan/file.h>
```

```
using namespace std;
```

```
template<typename T>
void printLength(T s)
{
    std::cout << "printLength: " << length(s) << std::endl;
}
```

```
int main(int, char const **)
{
    // The STL way:
    vector<string> stlString;
    stlString.push_back("The number is 10");
    stlString.push_back("The number is 20");
    stlString.push_back("The number is 30");
    std::cout << "stlString.size(): " << stlString.size() << std::endl;
```

```
// SeqAn OOP Test
seqan::CharString charString = "Hello SeqAn!";
int charStringGlobalLength = length(charString);
int charStringMemberLength = charString.length();
std::cout << "CharString global length: " << charStringGlobalLength <<
→ std::endl;
```

```

    std::cout << "CharString member length: " << charStringMemberLength <<
    ↵ std::endl;
    std::cout << "CharString.length() works? " << (charStringGlobalLength ==
    ↵ charStringMemberLength ? "yes" : "no") << std::endl;

    seqan::DnaString dnaString = "GATTACA";
    int dnaStringGlobalLength = length(dnaString);
    int dnaStringMemberLength = dnaString.length();
    std::cout << "DnaString global length: " << dnaStringGlobalLength <<
    ↵ std::endl;
    std::cout << "DnaString member length: " << dnaStringMemberLength <<
    ↵ std::endl;
    std::cout << "DnaString.length() works? " << (dnaStringGlobalLength ==
    ↵ dnaStringMemberLength ? "yes" : "no") << std::endl;

    seqan::String<seqan::Dna, seqan::Packed<> > packedDnaString = "GGATTACAG";
    int packedDnaStringGlobalLength = length(packedDnaString);
    int packedDnaStringMemberLength = packedDnaString.length();
    std::cout << "PackedDnaString global length: " <<
    ↵ packedDnaStringGlobalLength << std::endl;
    std::cout << "PackedDnaString member length: " <<
    ↵ packedDnaStringMemberLength << std::endl;
    std::cout << "PackedDnaString.length() works? " <<
    ↵ (packedDnaStringGlobalLength == packedDnaStringMemberLength ? "yes" :
    ↵ "no") << std::endl;

    return 1;
}

```

LISTUNG 12: Demonstration des CRTP in SeqAn

Dieser kleine Demonstrator zeigt, dass das CRTP auch für den mittels ermöglichten Polymorphismus funktioniert. Das Programm hat folgende Ausgabe:

```

stlString.size(): 3
CharString global length: 12
CharString member length: 12
CharString.length() works? yes
DnaString global length: 7

```

```
DnaString member length: 7
DnaString.length() works? yes
PackedDnaString global length: 9
PackedDnaString member length: 9
PackedDnaString.length() works? yes
```

In einem Telefonat mit *dem* SeqAn-Entwickler Gogol-Döring (Gogol-Döring u. Kahlert 2013), drückte dieser seine Begeisterung für meinen Erfolg aus<sup>28</sup>. Dabei nannte er die folgenden Vorteile meines Ansatzes:

- SeqAn würde durch diese Umarbeitung einen besseren Ruf bekommen, was die Anwendergruppe verbreitet und was sich wiederum im Falle von Finanzierungsanträgen positiv auswirken könnte.
- Das CRTP würde ein wichtiges ● Werkzeugunterstützungsproblem abschmälern — nämlich das der ● fehlenden Autovervollständigung und damit die Strategie der ● Punkt-Funktionssuche unterstützen.
- Aktuell kann eine IDE praktisch nur alle globalen SeqAn-Funktionen vorschlagen, was in Anbe tracht der globalen Funktionsvielzahl kaum einen Vorteil bringt: “A class with 10 functions looks tractable to most people, but a class with 100 functions is enough to make many programmers run and hide [and] end up discouraging people from learning how to use it.” (Meyers 1992))

#### 4.3.4.2 Metafunktionen

In C++ werden kaum ● Metafunktionen eingesetzt. In SeqAn jedoch schon.

Selbst wenn das CRTP in SeqAn eingesetzt wird, bleibt es bei dem Bedarf globaler generischer Funktionen, wie sie auch in C++ als *Algorithmen* existieren. In C++ können die Typen der Rückgaben sehr leicht bestimmt werden, weil alle mir bekannten Algorithmen immer nur auf einem bestimmten Container-Typ arbeiten (z.B. `vector<int>`) und sich der Rückgabetyp einfach ergibt (z.B. `int` bzw. `int *`).

In SeqAn hingegen ist die Rückgabentypbestimmung komplizierter, weshalb auf Metafunktionen zurückgegriffen wird, die den notwendigen Typ berechnen. Dies führt allerdings teilweise zu dem ● Typing-Problem: “Das erstellen der richtigen Typen mit Hilge [sic] von Metafunktionen kann sehr kompliziert sein.”<sup>29</sup>

Für diesen Anwendungsfall sind Metafunktion für API-Anwender durch den neuen C++-Sprachstandard obsolet geworden, was ich im nächsten Unterkapitel erläuterte.

---

<sup>28</sup> Gegenstand des Telefonats war auch die Frage, ob Gogol-Döring beim Entwurf von SeqAn das CRTP selbst evaluierte, denn diese Frage wird in keiner seiner SeqAn-Arbeiten (Gogol-Döring 2009; Gogol-Döring u. Reinert 2009) beantwortet. Die Antwort fiel nicht eindeutig aus. Nach längerem Hin und Her, glaubte er sich daran zu erinnern, eine Evaluation durchgeführt zu haben, die jedoch an Visual Studio gescheitert sei. Darüber hinaus sei Objektorientierung ohnehin nicht seine favorisierte Lösung gewesen.

#### 4.3.4.3 Vorschlag: CRTP

Ich schlage vor, das CRTP<sup>29</sup> auf folgende Weise einzusetzen:

1. Für jede Basis-Klasse wird ein *OOP-Interface* eingeführt, in dem alle generischen Memberfunktionen aufgelistet und die jeweiligen technisch globalen Funktionen gekapselt werden, die auch in C++/STL Memberfunktionen wären.

Beispiel: Für die Klasse `String` muss es das OOP-Interface `OOPInterfaceString` geben.

Konkrete Implementierung mit der Funktion `length`:

```
template <typename TThis>
class OOPInterfaceString {
public:
    SEQAN_HOST_DEVICE inline typename Size<TThis>::Type
    length() {
        // call the corresponding global function
        return seqan::length(*static_cast<TThis*>(this));
    }
};
```

2. Besitzen Template-Spezialisierungen weitere Funktionen, muss ein neues OOP-Interface implementiert werden, das bisherige erben und die entsprechenden Funktionen hinzufügen.

Beispiel: Für die Klasse `StringSpecial` muss es das OOP-Interface `OOPInterfaceStringSpecial` geben. `OOPInterfaceStringSpecial` erbt von `OOPInterfaceString`. `StringSpecial` erbt nun nicht mehr von `OOPInterfaceString`, sondern von `OOPInterfaceStringSpecial`.

3. Der standardmäßige Iterator für eine Container-Klasse wird mit Hilfe von `typedef` in die Container-Klasse als statisches Feld `::iterator` aufgeführt.
4. Das OOP-Interface eines Container wird um die Iteratoren-relevanten Funktionen wie `begin` und `end` ergänzt.

Auf diese Weise stehen sämtliche, bisher globale Funktionen als Memberfunktionen bereit und können durch die IDE-Autovervollständigung aufgelistet werden. Außerdem kann ein Anwender fortan über Container iterieren, wie er es aus C++ gewohnt ist.

---

<sup>29</sup> Die mit C++14 eingeführten erweiterten konstanten Ausdrücke (`constexpr`) habe ich nicht weiter evaluiert, da sie nicht von allen Compilern unterstützt werden, mit denen SeqAn kompatibel sein soll.

Beispiel: Visual Studio — <http://blogs.msdn.com/b/vcblog/archive/2015/04/29/c-11-14-17-features-in-vs-2015-rc.aspx> (Stand: 29.04.2015).

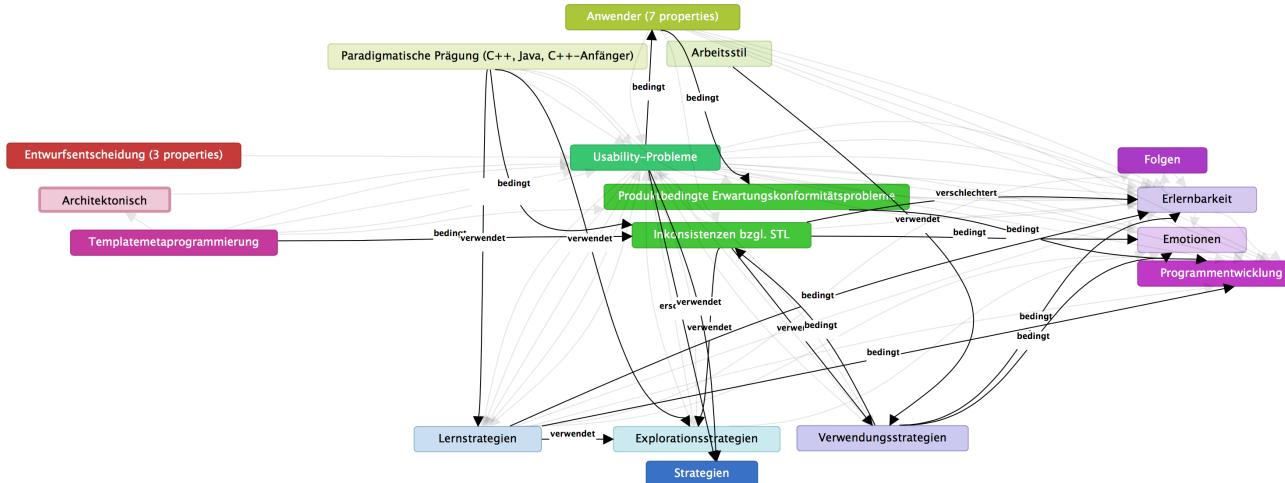


ABBILDUNG 4.2: Ursachen und Folgen des Problems ● Inkonsistenzen bzgl. STL

Das Resultat ist eine dramatische Annäherung an die Anwendungsweise der STL von C++, was das schwere Problem ● Inkonsistenzen bzgl. STL zu großen Teilen beheben sollte. Die Auswirkungen dieses Problems werden in dem in Abbildung 4.2 dargestellten Teilgraph meiner in 4.1.4 vorgestellten GT visualisiert. Anwender haben weiterhin die Möglichkeit, statt `str.length`, `length(str)` aufzurufen.

#### 4.3.4.4 Weiterer Vorschlag: Wrapper-API

Während ich die Umsetzung des CRTP-Vorschlags mit großer Überzeugung vertreten kann, fehlen mir für den folgenden Vorschlag hinreichend empirisch belegte Erkenntnisse — siehe ● Inkonsistenzen bzgl. OOP bzw. *Java-objektorientierte* ● paradigmatische Prägung.

In der API-Usability-Evaluationsstudie von Stylos et al. (2008) haben die Autoren festgestellt, dass ein Teil der Anwender API-Endanwender sind. Da diese Gruppe andere Anforderungen an eine API stellt, haben sich die Autoren dazu entschlossen, die existierende API nicht zu überarbeiten und damit womöglich die API-Usability für die technischen API-Anwender zu verschlechtern. Stattdessen haben sie eine auf der existierenden API aufbauende Wrapper-API geschaffen, die sich dadurch auszeichnet, dass sie API-Endanwender-zentrisch entwickelt wurde. Sie verfügt über ein weitaus höheres Abstraktionsniveau und verringert die Konfrontation mit technischen Belangen wie Thread-Sicherheit. Auf diese Weise verfügt die verbesserte Library nun über zwei aufeinander aufbauende APIs — eine für API-Anwender und eine für API-Endanwender.

Dasselbe tat der Workshop'14-Teilnehmer John Reid im Zuge einer Arbeit (Reid 2014), bei der SeqAn eingesetzt wurde. Auf Grund Reids großer Unzufriedenheit mit SeqAn<sup>30</sup>, implementierte er einen Python-basierten SeqAn-API-Wrapper mit Hilfe der *Boost.Python*-Library<sup>30</sup>. Dieser aufwändige Schritt unterstreicht zugleich die Fatalität der dazugehörigen Usability-Probleme. Der Wrapper sieht in seiner Anwendung wie folgt aus:

30 <http://www.boost.org/libs/python/>

```

# Create a set of strings
seqs = seqan.StringDNASet('ACGT', 'AAAA', 'GGGG', 'AC')

# Create an enhanced suffix array
index = seqan.IndexStringDNASetESA(seqs)

# Traverse the array (DFS)
it = index.topdownhistory()
it.goBegin()
while not it.atEnd:
    print it.numOccurrences, it.representative
    it.goNext()

```

Bereits bei dem ersten SeqAn-Workshop im Jahre 2011 äußerten einzelne Workshop-Teilnehmer im Rahmen der Feedback-Runde den Wunsch, SeqAn in ihre Python-Projekte verwenden zu können. Dieser Wunsch wurde jedoch schnell von Teilen der anwesenden SeqAn-Entwickler wegdiskutiert und nicht weiter verfolgt.

Die Arbeit von Stylos et al. (2008) zeigt, dass es durchaus Sinn machen kann, verschiedene APIs für verschiedene Anwendergruppen zu implementieren. Reid (2014) zeigte wiederum, dass eine zweite API möglich ist und wie sie objektorientiert aussehen kann, ohne Performanceverluste zu erleiden.

Erst spät wurde mir klar, dass der VIP-Förderantrag selbst eine solche Wrapper-API vorsah — ohne sie als solches zu benennen. Die Rede ist von der Workflow-Engine KNIME, auf die ich bereits in den Abschnitten 3.1.1 und 3.2.4.2 eingegangen bin. KNIME erlaubt es, als visuelle Knoten gekapselte Programme grafisch zu verknüpfen und auf diese Weise Workflows zu erstellen (siehe Abbildung 4.3). Eines der Ziele des ebenfalls im Abschnitt 3.1.1 vorgestellten BioStore-Projekts bestand in der Bereitstellung von SeqAn-Anwendungen innerhalb von KNIME. Es handelt sich dabei also um ein vom BioStore-Projekt erklärt Ziel, das ich nun auf der Grundlage meiner Forschungsergebnisse unterstütze, weil es sich dabei um eine grafische Wrapper-API handelt, die sich besonders gut für API-Endanwender eignet (vgl. Ko et al. 2011; Ko u. Myers 2004).

### 4.3.5 Maßnahme: Intransparenzbeseitigung

**Nutzen:** gering<sup>31</sup>; **Kosten:** mittel

Der Problem der  versteckten Parameterübergabe muss durch eine explizite Parameterübergabe gelöst werden, was ich an dem folgenden Beispiel verdeutlichen möchte:

---

<sup>31</sup> Nur ein Anwender schilderte dieses Problem — allerdings mit Vehemenz. Ich kann daher den genauen Nutzen nicht beurteilen.

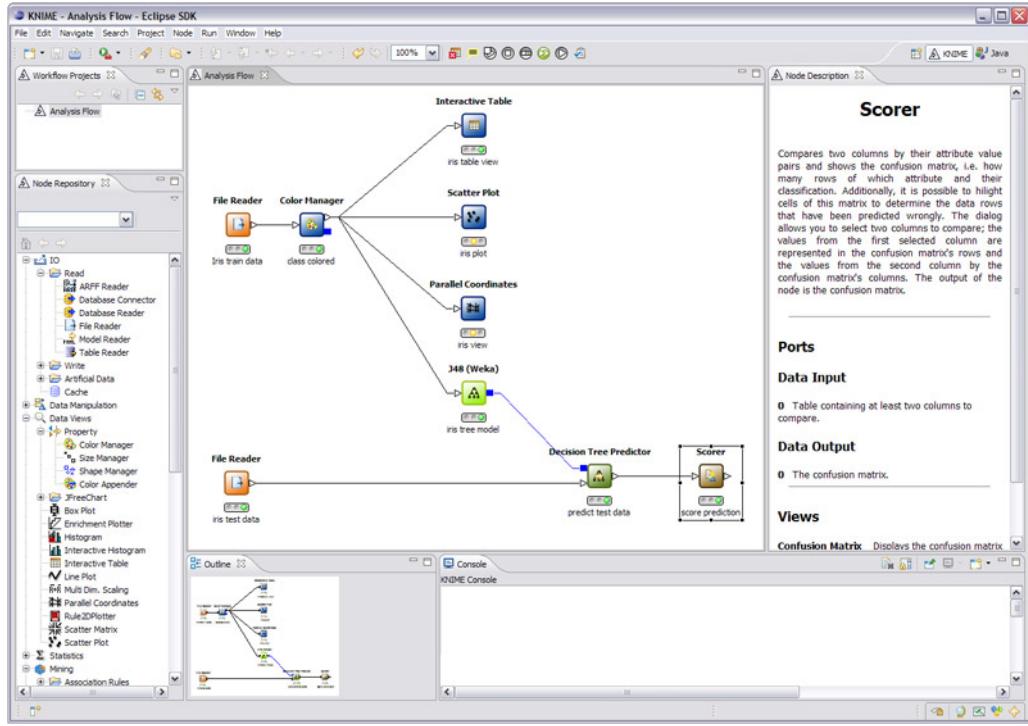


ABBILDUNG 4.3: Workflow-Engine KNIME

```

typedef Index<DnaString, IndexQGram<UngappedShape<3> > > TIndex;
TIndex index("CATGATTACATA");
hash(indexShape(index), "CAT");
for (unsigned i = 0; i < length(getOccurrences(index, indexShape(index))); ++i)
    std::cout << getOccurrences(index, indexShape(index))[i] << std::endl;
return 0;

```

Dieses Beispiel könnte verbessert lauten:

```

typedef Index<DnaString, IndexQGram<UngappedShape<3> > > TIndex;
TIndex index("CATGATTACATA");
TShape shape = indexShape(index);
THash shapeHash = hash(shape, "CAT");
for (unsigned i = 0; i < length(getOccurrences(index, shape, shapeHash)); ++i)
    std::cout << getOccurrences(index, indexShape(index))[i] << std::endl;
return 0;

```

In diesem Beispiel habe ich die Rückgaben der Funktionen `indexShape` und `hash` expliziert.

Da ich nur durch einen Proband auf dieses Problem aufmerksam wurde, dieses Thema in der API-Usability-Forschung nach meiner Kenntnis unerforscht ist und ich mich mit zeitlichen Problemen konfrontiert sah, habe ich diese Maßnahme nicht weiter verfolgt. Das mit dieser Maßnahme adressierte Problem ● versteckte Parameterübergabe ist jedoch Teil meines Ausblicks.

### 4.3.6 Maßnahme: Inkonsistenzbeseitigung

**Nutzen:** hoch; **Kosten:** hoch

- *Indirekte* Datenstrukturmodifikationen wie `fn(x) = 1` und `fn(fm(x), 1)` müssen durch *direkte-explizite* Datenstrukturmodifikationen wie `x = 1` oder `x = fn(y)` ersetzt werden. *Direkte-implizite* Datenstrukturmodifikationen wie `fn(x, 1)` sind tolerabel, wenn der Funktionsname auf eine Datenstrukturveränderung hinweist (z.B. `assignValue(x, 1)`). Auf diese Weise kann das Problem *Funktionsrückgabenmodifikation* gelöst werden.
- Zusätzlich müssen Funktionen — wenn kein dringender Grund dagegen spricht — Referenzen zurückgeben und der Zuweisungsoperator entsprechend überschrieben werden, um das Problem der *Operatoreninkonsistenz* zu beheben.
- In SeqAn werden Termini aus verschiedenen Domänen verwendet. Rein fachliche Termini wie *Peptide* dürfen nur dann verwendet werden, wenn tatsächlich eine domänenentsprechende KapSELung mit Mehrwert zur benannten Datenstruktur zu Grunde liegt. Damit wird die Fatalität der Probleme *Abstraktionssuggestion* und *Technisch vs. Fachlich verringert*.

### 4.3.7 Maßnahme: Shortcuts

**Nutzen:** mittel; **Kosten:** gering

Shortcuts sind ein Quell der Unmut und haben einen zweifelhaften Nutzen.

- Der Name eines Shortcuts sollte sich aus den Bestandteilen ergeben, die es synonymisiert. Damit wird die Wahrscheinlichkeit verringert, dass Shortcuts eine *Abstraktion* suggerieren.
- Offensichtliche Shortcuts sollten entfernt werden oder zumindest nicht in für API-Anwender sichtbaren Code — insbesondere Code-Beispielen — verwendet werden. Dadurch kann das Problem *Synonyme / Redundanz* gelöst werden.

### 4.3.8 Maßnahme: Fail-Fast

**Nutzen:** mittel; **Kosten:** gering

Die *Versagensverschleppung* hat zwei Ursachen und muss daher auch auf zweierlei Weise gelöst werden:

1. Globale (Meta-)Funktionen, die tatsächlich Interface-(Meta-)Funktionen sind, müssen auch als solche implementiert werden. Standardimplementierungen, die auch für ungeeignete Eingaben Werte zurückgeben, müssen beseitigt werden (z.B. `length`).

2. Leseoperationen müssen die Möglichkeit zur Verfügung stellen, Lesefehler in Erfahrung zu bringen. Da Performance ein wichtiges Entwurfsziel von SeqAn ist, schlage ich vor, standardmäßig eine Verifikation der eingelesenen Daten vorzunehmen. Anwendern, die nicht bereit sind, diesen Performanceverlust hinzunehmen, muss die Möglichkeit gegeben werden, diese Verifikation zu deaktivieren (z.B. mit Hilfe von `Tags`).

Ich schlage vor, eine Ausnahme im Falle eines Lesefehlers zu werfen. Der SeqAn-Core-Entwickler Manuel Holtgrewe begrüßte diesen Vorschlag bereits.

### 4.3.9 Maßnahme: Dokumentation

**Nutzen:** hoch; **Kosten:** hoch

● Dokumentationsprobleme haben neben den ● Inkonsistenzen bzgl. STL die höchste Gesamtfatalität.

Die Dokumentation muss sowohl inhaltlich, als auch technisch überarbeitet werden.

#### 4.3.9.1 Inhaltliche Überarbeitung

**Gesamtüberblick** Die SeqAn-Dokumentation leidet unter ihrem ● fehlenden Gesamtüberblick. Ein solcher muss daher bereitgestellt werden. Er muss folgendes umfassen:

- Funktionale Abgrenzung: Was bietet SeqAn an Funktionen an und was nicht?
- Entwurf: Wie “tickt” SeqAn? Dieser Punkt könnte durch eine Gegenüberstellung eines typischen SeqAn-Programms mit anderen Sprachen erreicht werden. Für die Gegenüberstellung empfehle ich zwei hypothetische, semantisch identische Programme in Java-OOP und “klassischem” objektorientierten C++. Beispielsweise könnte die SeqAn-Programmzeile `resize(score, length(text) - length(pattern) + 1);` in Java `score.resize(text.length() - pattern.length() + 1);` lauten. Sobald die Maßnahme *STL-Angleichung* umgesetzt ist, könnte der Vergleich zu C++/STL hinfällig sein.
- Entwurfsmotivation: Woher kommen die Unterschiede der Gegenüberstellung? Dieser Abschnitt muss die Motivation des Entwurfs — beispielsweise durch einen Benchmark — erläutern, um die Bereitschaft der Anwender zu erhöhen, sich weiterhin mit SeqAn zu befassen und eine höhere Toleranz für SeqAns Entwurf aufzubringen.

Eine ausführliche, aber nicht ausschweifende Erläuterung der grundlegenden Entwurfsentscheidungen unterstützt die ● Lernstrategie ● konzeptionelles Verstehen.

**Beispiele** ● Fehlende Anwendungsbeispiele behindern eine ganze Reihe von ● Explorations-, ● Lern-, und ● Verwendungsstrategien, was neben meiner Forschung auch die Arbeiten von Rosson u. Carroll (1996); Stylos et al. (2006) für API-Anwender und von Rosson et al. (2005); Wiedenbeck (2005) für API-Endanwender zeigen konnten.

Daher müssen alle nicht-trivialen Dokumentationseinträge über didaktische Code-Beispiele verfügen. Die Code-Beispiele müssen vollständig und voll funktionstüchtig sein, um es Anwendern zu erlauben, die Code-Beispiele ohne Anpassungen ausführen und besser verstehen zu können (Robillard u. DeLine 2010; Rosson u. Carroll 1996).

**Sprachentitätstypen** sind im Kontext von SeqAn ein wichtiges Konzept (siehe Abschnitt 4.1.4, Seite 270), da SeqAn wegen seiner ● Templatemetaprogrammierung eine Fülle neuer ● Sprachentitätstypen einführt, die für SeqAns Anwender auf Grund derer ● paradigmatischer Prägungen unbekannt sind.

Daher schlage ich vor, Sprachentitätstypen explizit in der Dokumentation zu benennen und sämtliche Dokumentationseinträge mit ihrem jeweiligen Sprachentitätstyp zu annotieren. Eine separate Seite soll darüber hinaus alle in SeqAn verwendeten Sprachentitätstypen erläutern.

**Punktuelle Verbesserungen** Zur Lösung des Problems ● Ununterscheidbarkeit von Typen müssen die Unterschiede und Gemeinsamkeiten zwischen ähnlich benannten Typen (z.B. `String<String>` und `StringSet`) deutlich erklärt werden.

Eine weiteres Problem ist die ● fehlende Dokumentation von Rückgabetypen. Die Lösung liegt auf der Hand: Jede Erläuterung einer Funktionsrückgabe muss den zurückgegebenen Typ oder die zur Berechnung des Typs notwendige Metafunktion nennen. Damit wird auch gleichzeitig das Problem ● Komplizierte Konstruktion von Typen abgeschwächt.

#### 4.3.9.2 Technische Überarbeitung

**Auflistung sämtlicher Funktionen** Die Dokumentation muss auf Grund der Usability-Probleme ● Identifikation relevanter Funktionen und ● Fehlende Funktionskategorisierung und zur Unterstützung der ● Explorationsstrategien ● Anfangsklassenverwendung unter jeder Klasse sämtliche zur Verfügung stehende Member-Funktionen, Interface-Funktionen und Member-Metafunktionen aufführen und klar voneinander unterscheiden. Um den Wartungsaufwand für die Dokumentation zu begrenzen, sollten sich die strukturellen Informationen aus den im Code manifesten Template-Spezialisierungen herleiten.

**Suche** Die ● Suchfunktion innerhalb der Dokumentation muss so angepasst werden, dass sie die Suchergebnisse gewichtet sortiert und auf der Grundlage von Sprachentitätstypen gruppiert.

Das durch das *Vocabulary Problem* bedingte Usability-Problem ● Technisch vs. Fachlich soll durch die Einführung von Aliassen gelöst werden. So muss beispielsweise der Suchbegriff `substring` den Dokumentationseintrag `infix` zu Tage fördern.

Wichtig an dieser Stelle ist die Unterscheidung zwischen den von mir vorgeschlagenen Aliassen und den als problematisch diagnostizierten Shortcuts. Während Shortcuts als eigenständige Entität dokumentiert sind und auch während der Programmierung alternativ genutzt werden können, dienen Aliasse lediglich dazu, unterschiedliche Suchbegriffe für einen bestimmten Dokumentationseintrag zu ermöglichen. Aliasse sind also weder eigenständig dokumentiert, noch können diese in Programmcode verwendet werden. Vielmehr erhöhen sie die Wahrscheinlichkeit eines erfolgreichen Suchtreffers und vermitteln dabei zugleich die in SeqAn verwendete Terminologie.

**Fehlerfreiheit** Die in der Dokumentation und in den ○ Tutorials verwendeten Code-Beispiele müssen defektfrei sein und kompilieren. Dazu müssen die Beispiele einer Quelle entspringen, die im Sinne der kontinuierlichen Integration (engl. *continuous integration*) getestet wird.

#### 4.3.10 Maßnahme: Werkzeugunterstützung

**Nutzen:** mittel; **Kosten:** mittel

Schaut man sich die in Abschnitt 2.7 vorgestellten API-Werkzeuge an, muss man leider feststellen, dass keines der Arbeiten den Prototypen-Status je verlassen hat oder zumindest nicht als schlüsselfertiges Produkt vorliegt.

Eine Klasse von Werkzeugen unterstützt API-(End-)Anwender bei der Suche von Beispielen. Leider sind alle mir bekannten Lösungen, mit einer Emacs-Ausnahme, als Eclipse-Plugins implementiert. Des Weiteren haben sich die meisten Werkzeuge auf die Programmiersprache Java konzentriert.

Fasst man die Unreife der Werkzeuge, deren IDE- und Sprachabhängigkeit zusammen, muss man leider zu dem Schluss kommen, dass die Kosten für deren Anpassung, ihren Nutzen vermutlich übersteigen.

Die einzige Ausnahme bildet ein Algorithmus von Buse u. Weimer (2012), der das Problem ● Fehlende Anwendungsbeispiele stark abmildern könnte. Dieser Algorithmus generiert automatisch Code-Beispiele für APIs, die auf typisierten Programmiersprachen basieren. Die Qualität der generierten Code-Beispiele wurde empirisch validiert. Ich halte es für sinnvoll, sich mit diesem Algorithmus näher zu beschäftigen und zu prüfen, mit welchen Kosten eine Anpassung an die Templatemetaprogrammierung-basierte generische Programmierung in C++ möglich ist.

#### 4.3.11 Maßnahme: Kollaborationsplattform

**Nutzen:** mittel; **Kosten:** mittel

Kollaborationsplattformen wie Webforen haben sich als nützliche Möglichkeit herausgestellt, eine andauernde Kommunikation zwischen API-Entwicklern und -Anwendern zu ermöglichen. Sie stellen eine exzellente Ressource für die Themen Debugging, Bugs und Entwurfsfragen dar. (Hou et al. 2005)

Darüber hinaus bergen Webforen das Potential, API-Endanwendern das Wissen und die Hilfe von erfahrenen API-Anwendern zugänglich zu machen. (Ko u. Myers 2005)

Damit handelt es sich um eine relativ global wirkende Maßnahme, die sich dann positiv auf die Erlernbarkeit von SeqAn auswirken sollte, wenn (1) dem Anwender das Problem bekannt ist und (2) der Anwender die Konsultation von Foren nicht scheut.

Aus diesen Gründen schlage ich die Einrichtung einer Kollaborationsplattform vor.

### 4.3.12 Probleme ohne Lösung

Compiler-Fehlermeldungen können nach meiner Kenntnis leider nicht verbessert werden, sondern höchstens durch die Maßnahme *Fail-Fast* verringert werden. Für API-Endanwender käme theoretisch das Debugging-Interface *Whyline* (Ko u. Myers 2004) in Frage. Praktisch scheitert dieser Vorschlag jedoch an der Unreife dieser Lösung und der extrem hohen Kosten für die Anpassung an SeqAn (vgl. Abschnitt 2.7). Nur die Aufnahme von *Concepts* in einen der kommenden C++-Sprachstandards könnte nach meiner Einschätzung Abhilfe schaffen.

Funktionszweckunerkenntbarkeit besteht darin, dass mindestens zwei Funktionen, die den gleichen Namen tragen, unterschiedliche Dinge tun. Nach Bloch (2006a) sollten solche Funktionen unterschiedliche Namen tragen. Dieser Empfehlung stimmen auch SeqAn-Anwender zu. Ungünstigerweise ist die Verwendung generischer — und damit häufig gleich lautender Namen — ja gerade das Grundprinzip der generischen Programmierung. Würde man die Funktion `calc` auftrennen in `calcA` und `calcB` würde man die Erweiterbarkeit einer API massiv beeinträchtigen. Dieses Problem kann also lediglich durch Aufklärung in der Dokumentation teilweise gelöst werden.

Das von mir während meiner Heuristischen Evaluation gefundene Problem der *fehlenden IDE-Integration der Dokumentation* ist der Tatsache geschuldet, dass das aktuelle *DDDoc*-Dokumentationsformat von SeqAn nicht mit dem Dokumentationssystem *Doxygen*<sup>32</sup> kompatibel ist. SeqAn hat besondere Anforderungen an das Dokumentationsformat, da es die in SeqAn verwendeten *Concepts* abbilden muss, die noch nicht in den aktuellen C++-Sprachstandard aufgenommen wurden. Jedoch schlage ich unter der Maßnahme Dokumentation eine Änderung des aktuellen Dokumentationsformats vor, die Doxygen ähnelt. Sobald Concepts in den C++-Sprachstandard aufgenommen wurden, sind sehr wahrscheinlich nur noch kleine Anpassungen notwendig, um die Dokumentation in die IDE des Anwenders integrieren zu können.

Die fehlende Auto vervollständigung ist der Werkzeugunterstützungsprobleme von generischer Programmierung durch Entwicklungsumgebungen geschuldet. Es ist davon auszugehen, dass die

<sup>32</sup> Es handelt sich dabei um das C++-Pendant zu JavaDoc. Siehe [www.doxygen.org](http://www.doxygen.org)

Aufnahme von Concepts in den C++-Sprachstandard zu einer besseren Auto vervollständigung führen werden. Diese Entwicklung würde auch den Strategien ● Punkt-Funktionssuche und ● Anfangsklassenverwendung entgegen kommen.

### 4.3.13 Weitere Maßnahme: Usability-Priorisierung

**Nutzen:** hoch; **Kosten:** mittel

Wie ich bereits im Abschnitt 4.2.3 angesprochen habe, befindet sich SeqAn gerade dabei, sein Einsatzgebiet auf den kommerziellen Bereich auszuweiten. Diese Entwicklung gibt der zuvor vernachlässigten Usability einen neuen Stellenwert.

SeqAn-Entwickler müssen bei zukünftigen Entwicklungen, wie der Unterstützung von GPUs<sup>33</sup> und der Parallelisierung von Algorithmen, nun nicht mehr nur Performance und Funktionalität, sondern auch Usability *gleichwertig* miteinander vereinbaren und ihre wissenschaftliche Arbeitsweise ebenfalls auf die Usability übertragen (Stichwort: Usability-Engineering).

Dafür gibt es zwei Gründe:

1. Durch die Verwendung eines Consultance-Vermarktungsmodells stehen SeqAn-Entwickler im direkten Kontakt zu ihren Kunden. *Technisches Wegargumentieren* wäre für die Kommunikation mit den Kunden schädlich (Sarodnick u. Brau 2006).
2. SeqAn ist zwar eine der schnellsten, aber nicht die einzige Bioinformatik-Softwarebibliothek auf dem Markt. Eine hohe Performance kann eine schlechte Usability nur in Maßen aufwiegen. Tut sie das nicht, wechseln die Kunden zu einem anderen Produkt (Sunshine et al. 2014). Dies belegen auch meine erhobenen Daten<sup>34</sup>.

### 4.3.14 Zusammenfassung

In diesem Unterkapitel habe ich Vorschläge zur Verbesserung der API-Usability von SeqAn auf der Grundlage meiner zuvor erhobenen Usability-Probleme und Strategien unterbreitet. Diese Vorschläge sind in Bezug auf die Usability-Probleme weitgehend disjunkt. Die meisten Maßnahmen lassen sich daher hinreichend isoliert bearbeiten.

Die Entwurfsentscheidungen ● Templatemetaprogrammierung und ● generische Programmierung machen Gebrauch von *Concepts* — also Interface-ähnlichen Typ-Anforderungsbeschreibungen — notwendig, von denen SeqAn bereits Gebrauch macht. Allerdings ist in diesem Punkt SeqAn seiner Zeit voraus, denn Concepts sind bis heute nicht in den C++-Sprachstandard aufgenommen worden. Erst C++17 wird voraussichtlich Concepts in die Sprache aufnehmen (Schmidt 2014).

---

<sup>33</sup> Grafikprozessor — *Graphics Processing Unit*

Nach Aufnahme von Concepts in C++ ist kurzfristig damit zu rechnen, dass Werkzeugunterstützungsprobleme wie die fehlende Autovervollständigung und die IDE-Integration der Dokumentation entfallen werden. Ich rechne außerdem damit, dass die Lesbarkeit von Compiler-Fehlermeldungen zunehmen wird, da durch die Unterstützung von Concepts eine Reihe von bisher nur in SeqAn verwendeten Sprachentitätstypen auch im Compiler als solche benannt werden.

Mittelfristig rechne ich damit, dass sich die durch Concepts eingeführten neuen Sprachentitätstypen, wie *Interface-Funktionen*, innerhalb der C++ geprägten Anwenderschaft etablieren werden. Zwar profitieren C++ unerfahrene, aber Java-erfahrene Anwender weniger davon, diese werden jedoch bereits durch die Beseitigung der Werkzeugunterstützungsprobleme unterstützt.

Die Aufnahme von Concepts in den C+-Sprachstandard bilden also nur einen Baustein bei der Verbesserung der API-Usability von SeqAn.

Die vier wichtigsten Usability-Verbesserungsmaßnahmen sind

- SeqAns Umstellung von einem Framework auf eine Library,
- SeqAns Angleichung an die STL,
- die Implementierung einer speziell für API-Endanwender entwickelten Wrapper-API und
- die Verbesserung der Dokumentation.

Ergänzt werden diese Maßnahmen durch punktuellere Maßnahmen wie der Überarbeitung von Shortcuts sowie der Beseitigung von Intransparenzen, Inkonsistenzen und Versagensverschleppungen.

Abschließend habe ich vorgeschlagen, den Dokumentationsaufwand zu begrenzen. Dazu empfiehlt es sich, bereits im Code vorhandene strukturelle Informationen für die Generierung der Dokumentation zu nutzen, das Dokumentationsformat DDDoc syntaktisch zu vereinfachen und den Code-Beispiel-Generationsalgorithmus von Buse u. Weimer (2012) zu evaluieren.

Das nächste Unterkapitel befasst sich mit den bereits erreichten API-Usability-Verbesserungen von SeqAn.



## VERBESSERUNG DER API-USABILITY VON SEQAN

Dieses Unterkapitel behandelt die erreichten API-Usability-Verbesserungen an Hand der im vorherigen Unterkapitel vorgeschlagenen Maßnahmen und wiederholt in sehr knapper Form die bereits in Phase 1 erläuterten Verbesserungen im Zuge der ersten Beseitigung grober Usability-Probleme.

Auf Grund der im gleichnamigen Abschnitt erläuterten Schwierigkeiten (S. 148) konnte ich nicht für die Umsetzung aller Maßnahmen sorgen. Ganz besonders sind dabei, neben organisatorischen Gründen wie dem Wegfall von Kollegen und meiner auf drei Jahre begrenzten Stelle, wissenschaftliche Gründe, wie den Anwendungsschwierigkeiten der GTM und dem unerwartet aufwändigen Analysewerkzeugbau, zu nennen. Aus den damit einhergehenden zeitlichen Problemen musste ich auch die im nächsten Unterkapitel vorgestellte Validierung stark kürzen.

### 4.4.1 Prozessverbesserungen

Im Zuge der ersten Beseitigung grober Usability-Probleme habe ich in Zusammenarbeit mit der Bioinformatik-Arbeitsgruppe die folgenden Prozessverbesserungen vorgenommen, die sich wie folgt positiv auf die Usability auswirken:

- Die Einführung standardisierter Commit-Nachrichten führte zu verständlicheren und ausführlicheren Beschreibungen der Entwicklungen an SeqAn. Dies half mir als API-Evaluator die Arbeiten an SeqAn besser nachvollziehen zu können. Augenscheinlich mag dieser Schritt unwichtig wirken. Durch die hohe Dynamik in meinem Arbeitsumfeld (insb. die vielen Akteure und die stetige Fortentwicklung von SeqAn; siehe Abschnitt 3.1.1) war diese Verbesserung jedoch von großer Wichtigkeit für mich und wirkte sich positiv auf mein Arbeitsergebnis aus.
- Die Umstellung von Subversion auf Git verbesserte den Entwicklungsprozess für die SeqAn-Entwickler, da sie nun lokale Revisionen erstellen konnten, ohne das zentrale Repository zu kompromittieren. Dieser Zugewinn an Freiheitsgraden erhöht die *sequenzielle Vollständigkeit* der Entwicklungs-*Arbeitsaufgabe*<sup>34</sup> und damit die Wahrscheinlichkeit einer höheren Softwarequalität. Ich gehe nicht davon aus, dass ein Zugewinn an Softwarequalität zu einer Verschlechterung der API-Usability führt — eher im Gegenteil.
- Die Einführung von Code-Reviews als Instrument der Qualitätssicherung verbessert ebenfalls die Softwarequalität.

Weitere Details können im Abschnitt 3.2 nachgelesen werden.

---

<sup>34</sup> Hierbei handelt es sich um Begriffe aus der Arbeitspsychologie. „Eine Handlung ist sequenziell vollständig, wenn in ihr der gesamte Handlungszyklus abgedeckt ist, d.h. es kommt sowohl zu Zielbildungs-, Planungs- und Ausführungsprozessen als auch zu Kontrollprozessen.“ (Bamberg et al. 2011)

## 4.4.2 Frameworkumbau

SqAn wurde erfolgreich von einem Framework zu einer Library umgebaut. Die SqAn-Entwickler haben sämtliche Vorwärtsdeklarationen beseitigt. Fortan ist es möglich, SqAn sowohl als Framework als auch als Library zu verwenden. Für Letzteres gibt es eine eigene Anleitung, die in allen Installationsanleitungen verlinkt ist<sup>35</sup>.

In Folge gibt es nun auch keine Abhängigkeiten mehr zum CMake-Build-System, der Anwender dazu zwingen könnte, den eigenen Build-Prozess umzustellen.

## 4.4.3 STL-Angleichung

### 4.4.3.1 Curiously Recurring Template Pattern — CRTP

Ich habe das CRTP einigen SqAn-Entwicklern zu einer Zeit präsentiert, in der ich noch in dem Glauben war, SeqAns Anwender würden eine streng objektorientierte Library erwarten. Dies entsprach zum Einen nicht den Tatsachen, denn ein großer Teil erwartete lediglich eine STL-konforme Softwarebibliothek. Zum Anderen stieß ich mit diesem Vorschlag auf Ablehnung. Ich hatte den Eindruck, dass die Idee, die SqAn-API zu einer reinen OOP-API umzustrukturieren, für die SqAn-Entwickler ein zu radikaler Schritt wäre.

Aus zeitlichen Gründen kam ich nicht mehr dazu, dem SqAn-Team meinen neuen Erkenntnisstand mitzuteilen.

Die in Abschnitt 4.1.4 beschriebenen Beobachtungen lassen erwarten, dass die Umwandlung von in SqAn globalen Funktionen, die in C++ als Memberfunktionen implementiert wären, eine äußerst positive Auswirkung auf die Usability haben wird.

### 4.4.3.2 Metafunktionen

Metafunktionen werden in der SqAn-API vornehmlich zur Berechnung von Rückgabetypen verwendet, was — obwohl Gogol-Döring damit eine Verbesserung der Usability beabsichtigte — tatsächlich eine Verschlechterung selbiger (siehe ● Typing-Problem) verursachte.

Dieser Anwendungsfall ist durch das `auto`-Schlüsselwort im neuen C++-Sprachstandard obsolet geworden, was auch die Dynamik meines Forschungsvorhabens unterstreicht (vgl. Abschnitt 3.1.4). Mittels `auto` wird der Compiler angewiesen, den Datentyp selbstständig herzuleiten. Auf diese Weise würde man statt `Row<Align<String<Dna>, ArrayGaps>>::Type &row1 = row(align, 0);` nur noch `auto &row1 = row(align, 0);` schreiben müssen.

---

<sup>35</sup> <http://seqan.readthedocs.org/en/latest/BuildManual/IntegrationWithYourOwnBuildSystem.html>

#### 4.4.4 KNIME als API-Endanwender-Werkzeug

Während die STL-Angleichung die bestehende SeqAn-API an die Erwartung von C++-geprägten API-Anwendern annähern soll, geht es bei der Wrapper-API darum, eine zweite, speziell auf die Bedürfnisse von API-Endanwendern angepasste API zu entwickeln. Die Bereitstellung von SeqAn-Programmen innerhalb der Workflow-Engine KNIME war eines der Ziele des BioStore-Projekts (siehe Abschnitte 3.1.1). Dieses Ziel haben die Bioinformatik-Arbeitsgruppe und ich wie folgt erreicht:

##### 1. KNIME-Anpassungen

1. Das von der Universität Tübingen entwickelte *Common Tool Description (CTD)* Format wurde weiterentwickelt und erlaubt nun eine mächtige Beschreibung der Schnittstelle von Kommandozeilenprogrammen. Beschreibungsmöglichkeiten umfassen u.a. typisierte Argumente und Wertebereiche (vgl. Abschnitt 3.2.4.2).
2. Das ebenfalls ursprünglich an der Universität Tübingen entwickelte *Generic KNIME Nodes*, wurde unter dem Namen *Generic Workflow Nodes*<sup>36</sup> weiterentwickelt. Es kann, unter Verwendung von CTD-Dateien, die entsprechenden Kommandozeilenprogramme als KNIME-Knoten kapseln. Diese können dann über das Internet für alle KNIME-Anwender bereitgestellt werden.

##### 2. SeqAn-Anpassungen

1. Es wurde ein neuer Parser für Argumente implementiert (siehe Abschnitt 3.2.4.2).
  - Der Argument-Parser ist Bestandteil der SeqAn-API und bietet mächtige Funktionen zur Beschreibung von Kommandozeilenprogrammschnittstellen.
  - Ein Argument-Parser verwendendes SeqAn-Programm kann seine eigene Schnittstellenbeschreibung als CTD-Datei exportieren.
  - Sämtliche SeqAn-Anwendungen (siehe Abschnitt 1.3.3.5) wurden an den Argument-Parser angepasst und verwenden diesen nun.
2. Die kontinuierliche Integration (engl. *continuous integration* — *CI*) wurde angepasst.
  - Der SeqAn-CI-Server generiert nun für alle SeqAn-Anwendungen CTD-Dateien.
  - Die SeqAn-Anwendungen werden mit Hilfe der CTD-Dateien und der Generic Workflow Nodes Anwendung zu KNIME-Knoten gepackt.
  - Die SeqAn-KNIME-Knoten<sup>37</sup> werden auf den Community-Server von KNIME hochgeladen und stehen damit allen KNIME-Anwendern zu Verfügung<sup>38</sup>.
3. Analog zu SeqAn-Code-Beispielen, wurden beispielhafte Workflows entwickelt<sup>39</sup>, die in KNIME importiert werden können.

<sup>36</sup> <https://github.com/genericworkflownodes>

<sup>37</sup> <https://tech.knime.org/seqan-nodes-for-knime>

<sup>38</sup> <http://seqan.readthedocs.org/en/master/HowTo/UseSeqAnNodesInKnime.html>

<sup>39</sup> [https://github.com/seqan/knime\\_seqan\\_workflows](https://github.com/seqan/knime_seqan_workflows)

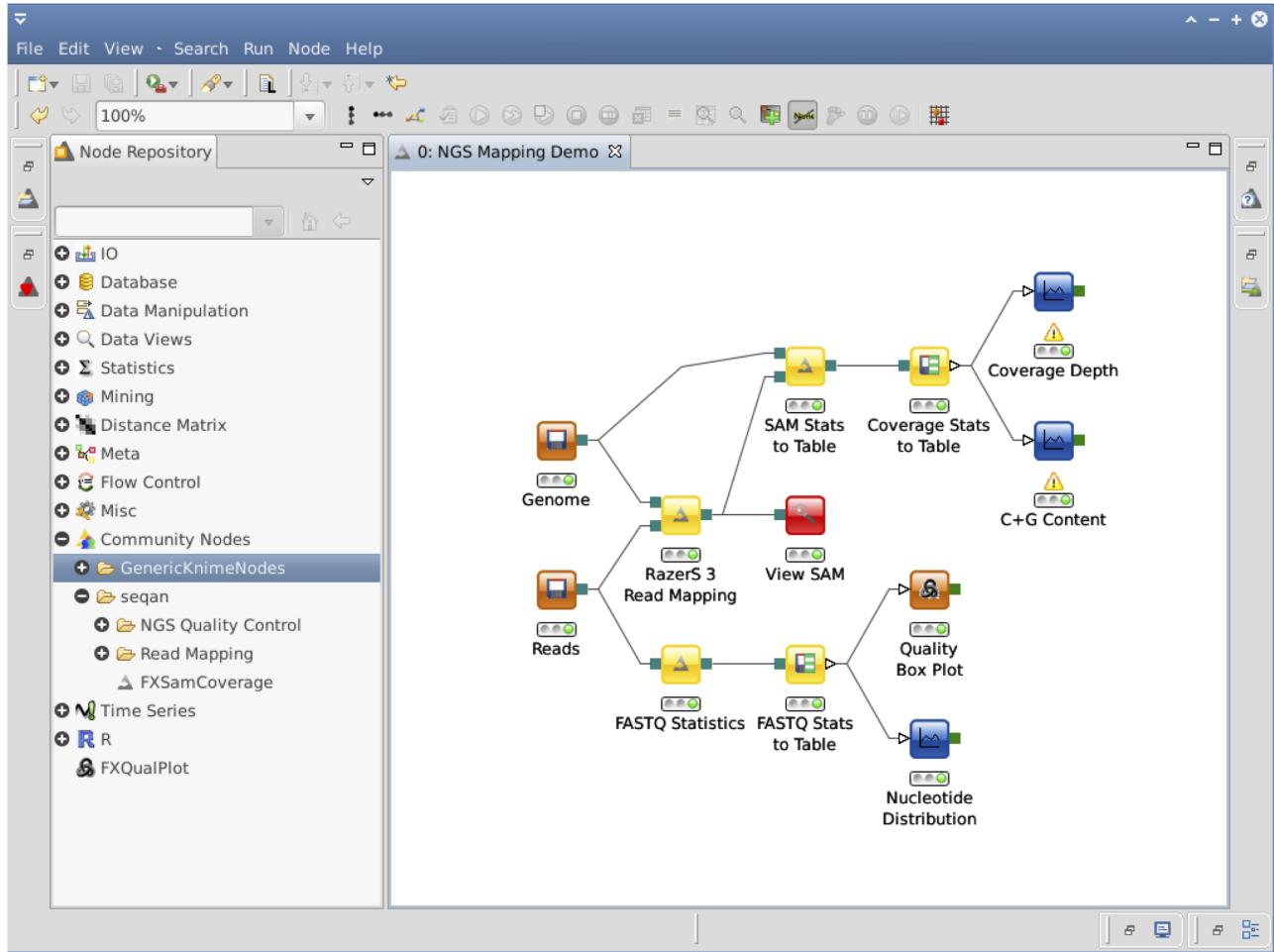


ABBILDUNG 4.4: Beispiel-SeqAn-Workflow in KNIME

Die SeqAn-Anpassungen erlauben es darüber hinaus SeqAn-Anwendern KNIME-Knoten aus ihren selbst entwickelten SeqAn-Anwendungen zu generieren.<sup>40</sup>

Das Ergebnis besteht darin, dass KNIME-Anwender nun grafisch mit SeqAn arbeiten können. Dazu können SeqAn-Anwendungen in Form von Knoten auf einer KNIME-Arbeitsfläche platziert, verbunden und konfiguriert werden (siehe Abbildungen 4.4 und 4.5). Auf diese Weise können API-Endanwender SeqAn-Workflows entwickeln, ohne eine einzige Zeile Code schreiben zu müssen.

#### 4.4.5 Inkonsistenzbeseitigung

Die Probleme Funktionsrückgabenmodifikation, Operatoreninkonsistenz und Abstraktionssuggestion wurden mit den SeqAn-Core-Entwicklern besprochen. Jedoch kam es nicht zu einer konkreten Planung zur Umsetzung dieser Maßnahme, was zwei Gründe hatte:

- Zum Zeitpunkt der Vorstellung meiner Ergebnisse waren meine Lösungsvorschläge nicht konkret genug.

<sup>40</sup> <http://seqan.readthedocs.org/en/latest/HowTo/GenerateSeqAnKnimeNodes.html>

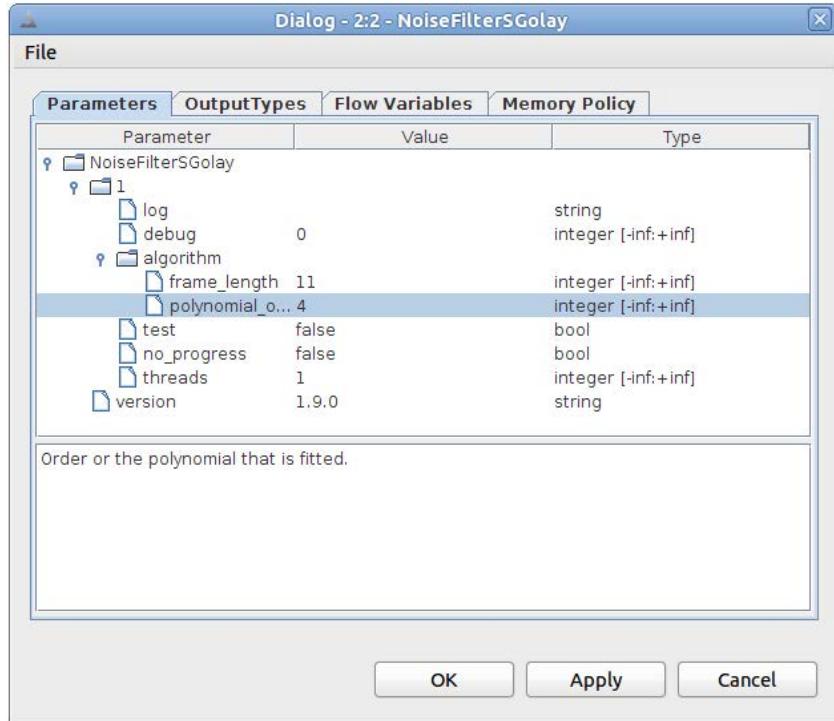


ABBILDUNG 4.5: KNIME-Konfigurationsdialog für einen CTD-basierten Knoten

- Meine Kollegen verfolgten während der BioStore-Projektzeit genauso eigene Ziele, wie ich das mit der API-Usability-Erforschung tat. Nach meiner Einschätzung des Standpunktes der SeqAn-Entwickler stand der Nutzen dieser Maßnahme in keinem guten Verhältnis zu den Kosten.

#### 4.4.6 Shortcuts

Während des BioStore-Projekts, das vor einem knappen Jahr endete, konnte ich die Sensibilität für die mit den Shortcuts einhergehenden Probleme auf Seiten der SeqAn-Entwickler erhöhen. Jedoch waren meine damaligen Lösungsansätze nicht hinreichend fundiert, um die SeqAn-Entwickler zu einer Lösung der Probleme ● Abstraktionssuggestion und ● Synonyme / Redundanz zu bewegen.

#### 4.4.7 Fail-Fast

Diese Maßnahme bestand aus zwei Schritten:

1. Dass es Funktionen wie `length` gibt, die bei bestimmten Eingaben ungültige Rückgaben liefern, war einigen SeqAn-Entwicklern unabhängig von meinen Forschungsergebnissen bekannt. Der Entwickler Manuel Holtgrewe erachtete deren Korrektur ebenfalls für sinnvoll. Aktuell ist dieser Schritt noch nicht umgesetzt.
2. Die Einschätzung, dass Leseoperationen Ausnahmen werfen müssen, wenn es zu einem Lesefehler kommt, teilten anfangs nur sehr wenige SeqAn-Entwickler aus Angst, SeqAn würde dadurch

langsamer arbeiten. Die Notwendigkeit dieser Maßnahme und dessen Vereinbarkeit mit dem SeqAn-Entwurfsziel *Performance* mittels *Tags* fand zunehmend Zustimmung. Allerdings ging diese Maßnahme in den letzten Projektmonaten — insbesondere aus Zeitgründen bei allen Beteiligten — unter. Die Beseitigung steht also noch aus.

## 4.4.8 Dokumentation

Die SeqAn-Dokumentation besteht im weiteren Sinne aus den Installationsanleitungen, didaktischen Lernressourcen (Tutorials) und der Dokumentation selbst, die als Nachschlagewerk für SeqAn-Anwender dient.

Im Zuge der ersten Beseitigung grober Usability-Probleme habe ich gemeinsam mit meinen Kollegen bereits die Installationsanleitungen und die Tutorials umfassend überarbeitet. Die Wirksamkeit dieser Überarbeitungen habe ich bereits im Abschnitt 3.2.5 gezeigt. An dieser Stelle fasse ich diese Änderungen nur sehr knapp zusammen:

### 4.4.8.1 Installationsanleitungen

Die Installationsanleitungen waren fehlerhaft, uneinheitlich, unstrukturiert und verfügten über zu wenig Beispiele, um den Anleitungen folgen zu können. Ich habe eine inhaltliche und grafische Vorlage für alle plattformabhängigen Installationsanleitungen erstellt und dabei die folgenden Installationsanleitungsbestandteile etabliert:

1. Prerequisites — Was bereits installiert sein muss + Verweise
2. Install — Die eigentliche SeqAn-Installation
3. A First Build — Überprüfung, ob Installation korrekt verlief
4. Hello World! — Skelett für erste eigene SeqAn-Anwendung
5. Further Steps — Verweise auf Dokumentation und Tutorials

Sämtliche Installationsanleitungen wurden korrigiert und vereinheitlicht. Abbildung 4.6 zeigt einen Ausschnitt aus der Installationsanleitung für Windows.

### 4.4.8.2 Tutorials

Basierend auf dem etablierten Lernphasenmodell (Gagné 1985) und den Analyseergebnissen einer Vielzahl von Daten (Details siehe Abschnitt 3.2) habe ich die Struktur der Tutorials überarbeitet und Qualitätskriterien formuliert. Diese Ergebnisse habe ich in Form eines Tutorials über das Schreiben von Tutorials zusammengefasst und in die Entwickler-Dokumentation integriert<sup>41</sup>. Das Dokument richtet sich an die Autoren von SeqAn-Tutorials, wurde von diesen sehr positiv aufgenommen, gilt seitdem als

---

<sup>41</sup> <http://seqan.readthedocs.org/en/master/HowTo/WriteTutorials.html>

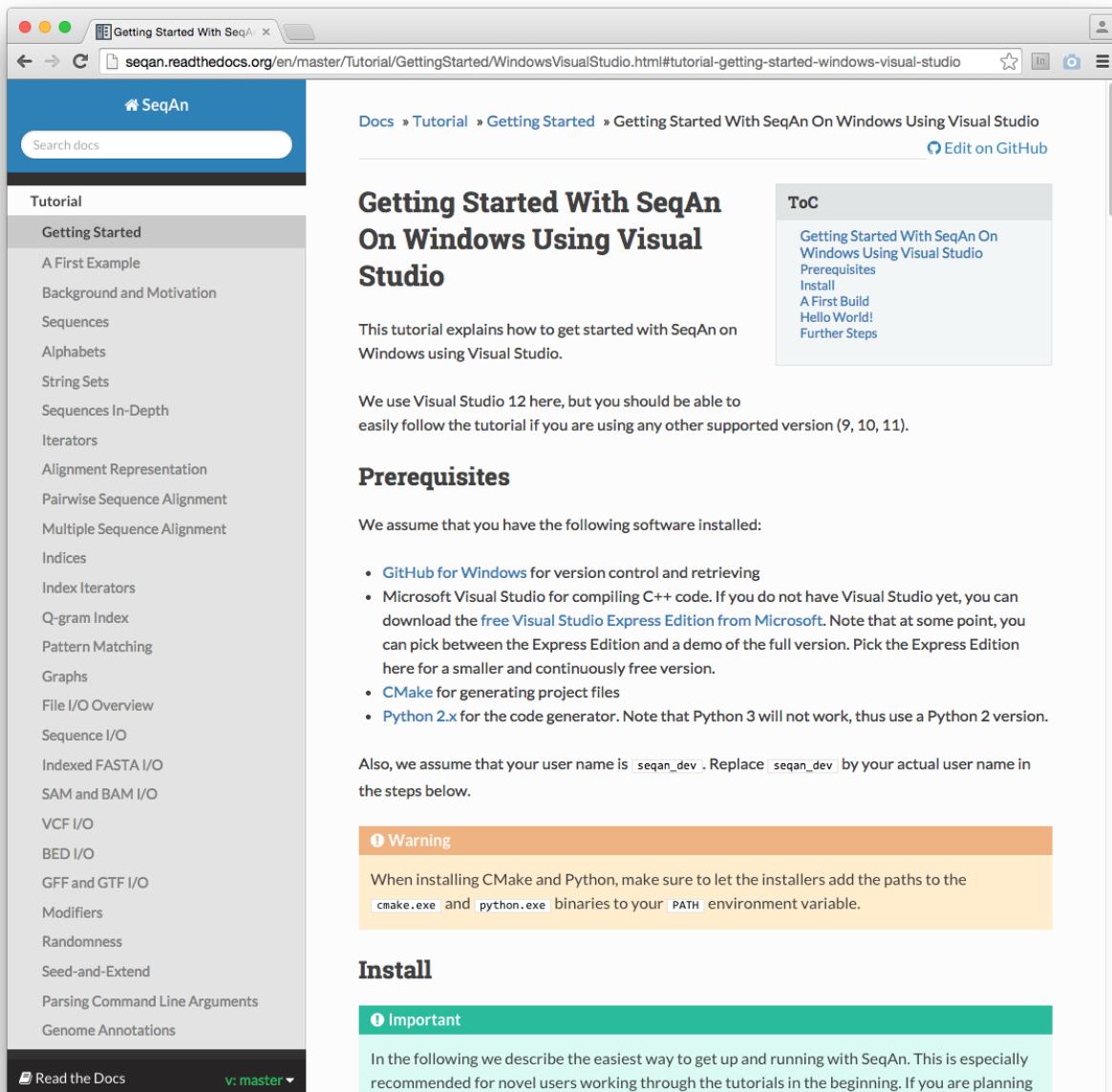


ABBILDUNG 4.6: Ausschnitt aus der verbesserten SeqAn-Installationsanleitung für Windows

verbindliche Vorlage für neue Tutorials und stellt damit einen langfristigen Beitrag für die verbesserte API-Usability von SeqAn dar.

Das Dokument besteht aus den folgenden Abschnitten:

1. Konventionen, die beim Verfassen von Tutorials zu beachten sind
2. Struktur — Aufbau eines Tutorials, explizite Metaangaben
3. Didaktik — Anwenderzentrierung, Übungsaufgaben
4. Integration — Vernetzung des Tutorials für bessere Auffindbarkeit
5. Vorlage — für die Erstellung eines neuen Tutorials

Es wurden mehrere neue Tutorials durch das Auf trennen existierender Tutorials geschaffen. Besonders erwähnenswert ist dabei das neue Anfänger-Tutorial “A First Example”<sup>42</sup>, das eine besonders geringe Einstiegshürde für Anwender mit den in der GTM-Analyse gefundenen ● paradigmatischen Prägungen aufweist. Es lohnt sich für den Leser dieser Dissertation das Anfänger-Tutorial einmal selbst zu öffnen.

Die Tutorials wurden sukzessive über eine längere Diskussionsphase hinweg gemeinsam von mir und den jeweiligen Autoren verbessert (siehe Abbildung 4.7) und dienen fortan als Aufgaben-bezogener SeqAn-Überblick (vgl. Fairbanks et al. 2006; Ko u. Riche 2011; Pugh 2006; Robillard 2009).

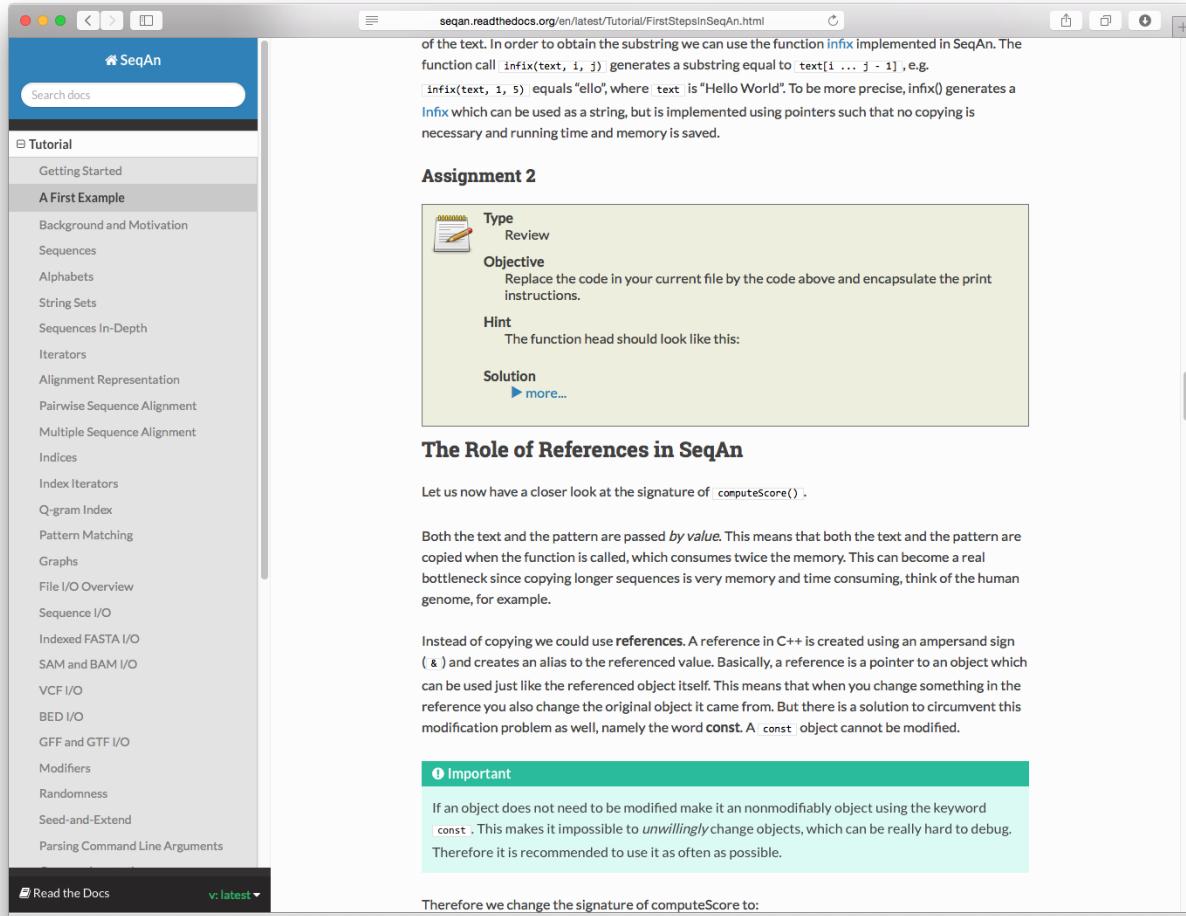


ABBILDUNG 4.7: Beispiel für ein überarbeitetes Tutorial

#### 4.4.8.3 Dokumentation

Dokumentationen bilden ein wichtiges Bindeglied zwischen dem, was der Anwender möchte und dem, was die Library anbietet (Kintsch 1988; Pennington 1987; Robillard 2009). Aus diesem Grund und meinen GTM-Analyseergebnissen habe ich die eigentliche SeqAn-Dokumentation technisch von Grund auf neu entwickelt. Die inhaltliche Überarbeitung wurde von den SeqAn-Entwicklern durchgeführt. Die

42 <http://seqan.readthedocs.org/en/master/Tutorial/FirstStepsInSeqAn.html>

Abbildung 4.8 zeigt die alte und die neue Dokumentation im Vergleich. Alternativ bietet es sich an, während des Lesens dieses Abschnitts die Dokumentation unter [docs.seqan.de/seqan/develop/](https://docs.seqan.de/seqan/develop/) selbst zu öffnen.

Im Folgenden erläutere ich die Änderungen gegenüber der alten Dokumentation.

(A) Startseite in Version 2.0.0

(B) Startseite in Version 3.0.0

(C) Geöffnete String-Klasse in Version 2.0.0

(D) Geöffnete String-Klasse in Version 3.0.0

ABBILDUNG 4.8: Die Abbildungen zeigen die alte und neue Dokumentation.

**Dokumentationssystem** In Vorbereitung auf die technische Neuentwicklung der Dokumentation haben meine Kollegen und ich zunächst das in SeqAn verwendete Dokumentationssystem überarbeitet.

Der SeqAn-Entwickler Gogol Döring entwarf das ● Dokumentationssystem *DDDoc* im Zuge einer *expliziten-argumentativen* ● Entwurfsentscheidung. Notwendig war (und ist) ein selbst entwickeltes Dokumentationssystem, da SeqAn, die noch nicht in den C++-Sprachstandard aufgenommene Sprachentität *Concept* verwendet.<sup>43</sup>

Wie das folgende kleine Beispiel zeigt, unterscheidet sich *DDDoc* von bekannten Dokumentationssystemen wie JavaDoc oder Doxygen, indem es ungewöhnlich zu lesen und zu schreiben ist.

```
/**  
 .Spec.SimpleScore  
 ..general:Class.Score  
 ..summary:Basic scoring scheme.  
 ..description:  
 This class allows to do alignments with simple match/mismatch-based scores.  
 ..remarks:This class also supports different gap open and extension costs.  
 ..include:seqan/score.h  
 */
```

LISTUNG 13: *DDDoc*-Beispiel für die Template-Spezialisierung `SimpleScore`

Mit meinem Vorschlag, das Format zu überarbeiten, rannte ich bei den SeqAn-Entwicklern offene Türen ein. Eine Überarbeitung würde außerdem die Möglichkeit bieten, die in *DDDoc* fehlende Unterscheidung zwischen den Beziehungen “wird von Funktion verwendet” und “ist Teil der Schnittstelle eines Typs” zu beheben. Aus diesen Gründen wurde ein Doxygen-Derivat namens *Dox* entwickelt, das auch Concepts und damit auch globale Funktionsinterfaces und Templatevererbung unterstützt. Implementiert wurde ebenso ein C++-Parser.

Das folgende Beispiel zeigt denselben Dokumentationseintrag im neuen *Dox*-Format:

```
/*!  
 * @class SimpleScore  
 * @extends Score  
 * @summary Basic scoring scheme.  
 *
```

<sup>43</sup> Gogol-Dörings Einschätzung, dass Concepts unbedingt Bestandteil der Dokumentation sein müssen, konnte ich empirisch bestätigen. Wie die Probleme ● Fehlende Funktionskategorisierung und ● Identifikation relevanter Funktionen zeigen, können ohne Concepts die inhaltliche Zusammengehörigkeit technisch global implementierter (Meta-)Funktionen vom Anwender nicht erkannt werden. Tatsächlich handelt es sich nämlich bei den meisten globalen Funktionen inhaltlich um Interface-(Meta-)Funktionen.

```

* @signature template <typename TValue>
*           class Score<Tvalue, Simple>;
*
* This class allows to do alignments with simple match/mismatch-based
* scores.
*/
template <typename TValue>
class Score<TValue, Simple> {...};

```

LISTUNG 14: Dox-Beispiel für die Template-Spezialisierung `SimpleScore`

Wie man erkennen kann, ist das neue Doxygen-Format durch seine Ähnlichkeit zu JavaDoc und Doxygen leichter zu lesen und zu schreiben.

Das nächste und letzte Beispiel zeigt den Dokumentationseintrag für die technisch globale, aber inhaltlich zum `Score` gehörige Interface-Funktion `scoreGapOpen`:

```

1  /**
2   * @fn Score#scoreGapOpen
3   *
4   * @signature TValue scoreGapOpen(sc);
5   *
6   * @return TValue The gap open score value for <tt>sc</tt>.
7   *           Tvalue is the value type of <tt>sc</tt>.
8   * @param sc The @link Score @endlink object to query.
9   */
10 template <typename TValue, typename TSpec, typename TSeqHValue, typename TSeqVValue>
11 inline TValue
12 scoreGapOpen(...)

```

LISTUNG 15: Dox-Beispiel für die zur Klasse `Score` gehörige Interface-Funktion `scoreGapOpen`

Über mehrere Monate haben die SeqAn-Entwickler sämtliche DDDoc-Einträge zu Dox-Einträgen umgewandelt. Beim Debuggen dieser Umwandlungen half u.a. ein in der neuen Dokumentation implementierter Entwicklermodus.

**Entwicklermodus** Für die neue Dokumentation habe ich einen Entwicklermodus entwickelt, der sich an die SeqAn-Entwickler richtet. Der Entwicklermodus kann bei geöffneter Dokumentation durch die Tastenkombination Shift+Ctrl<sup>44</sup> aktiviert werden und wurde von mir so implementiert, dass er um beliebige Funktionen erweitert werden kann. Aktuell unterstützt er SeqAn-Entwickler bei den folgenden Tätigkeiten:

- SeqAn-Entwickler können für jeden Dokumentationseintrag den dazugehörigen Dox-Eintrag im Quellcode einblenden lassen (siehe Abbildung 4.9).
- Insbesondere für das Schreiben von Tutorials benötigen SeqAn-Entwickler den Link zu einem bestimmten Eintrag. Dieser kann nicht ohne Weiteres aus der Adresszeile entnommen werden. Einerseits verwendet die Dokumentation *Iframes*<sup>45</sup>. Andererseits gibt es den Bedarf, auch einen ganz bestimmten Untereintrag innerhalb eines Dokumentationseintrages zu verlinken. Im Entwicklermodus werden alle verlinkbaren Elemente hervorgehoben. Ein Klick auf eines dieser Elemente bringt einen modalen Dialog hervor, der alle Verlinkungsmöglichkeiten anzeigt und auf Wunsch in die Zwischenablage kopiert (siehe Abbildung 4.10).

Der Entwicklermodus wurde von den SeqAn-Entwicklern sehr gut angenommen, was man leicht an den schnellen Beschwerden, im Falle von entdeckten Bugs, sehen konnte.

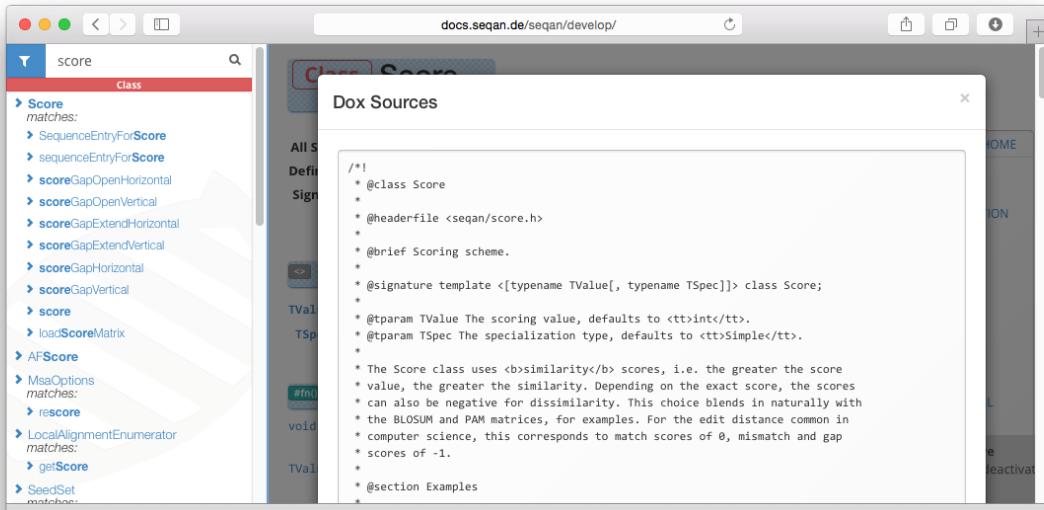


ABBILDUNG 4.9: Entwicklermodus — Dox-Quelle für den aktuellen Dokumentationseintrag

**Gesamtüberblick** Die Notwendigkeit eines Gesamtübersichts konnte ich empirisch zeigen und lässt sich auch direkt und indirekt aus der Literatur entnehmen. So verhindert ein fehlender Gesamtübersicht Top-Down-Lernen (vgl. Brooks 1983) und das Erkennen von Zusammengehörigkeit verteilter

44 Zu beachten ist, dass der rechte Bereich, also nicht der Suchbereich, über den Fokus verfügen muss.

45 <https://www.w3.org/wiki/HTML/Elements/iframe>

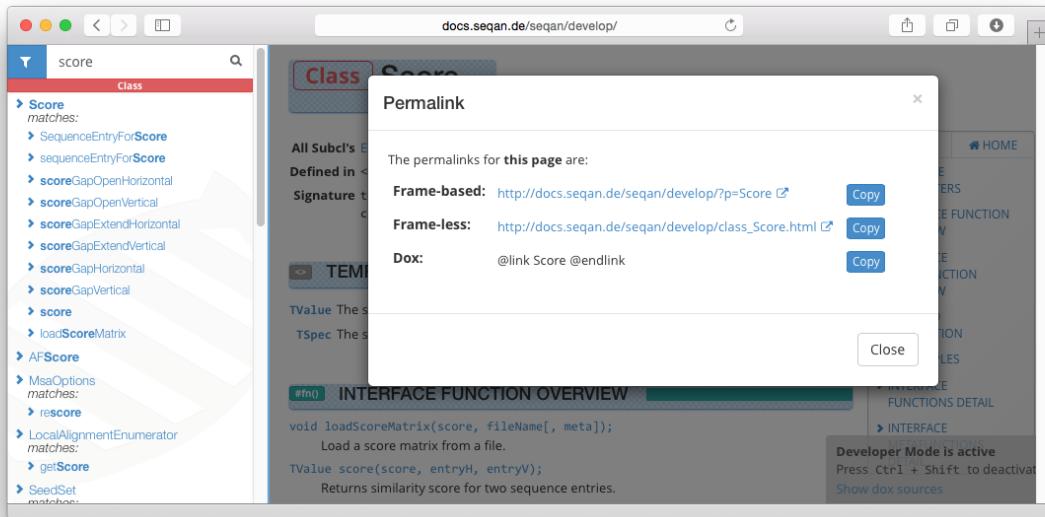


ABBILDUNG 4.10: Entwicklermodus: Anzeige aller Links für das aktuell selektierte Element. Im Hintergrund des modalen Dialogs kann man drei bläulich hinterlegte Elemente erkennen, die verlinkbar sind.

Informationen (vgl. Schneiderman u. Mayer 1979) — insbesondere für Anfänger (Piccioni et al.; Stylos u. Myers 2008).

Werden an keiner Stelle Entwurfsentscheidungen gebündelt dargestellt, werden Anwender beim Anpassungen von Beispiel-Code eingeschränkt (Bruch et al. 2006). Dies führt dazu, dass Anwender häufiger Fehler bei der Anpassung machen, und mit geringerer Wahrscheinlichkeit die korrekte Intention des Codes wiedergeben (Fairbanks et al. 2006).

Außerdem muss die Startseite der Dokumentation über eine Aufgaben-bezogene, logische Gruppierung der Library-Elemente verfügen, da es klassenübergreifende Aufgaben gibt, deren Lösung sich für API-Anfänger sonst nicht ohne Weiteres erschließen würde. (Clarke 2005b; Hou et al. 2005)

Die überarbeitete Startseite der SeqAn-Dokumentation verfügt nun über folgende Elemente:

- Knappe Beschreibung von SeqAn
- *Getting Started*
  - Verweis auf die Installationsanleitungen
  - Verweise auf die Tutorials und ähnliche Inhalte (Profiling, CMake, etc.)
  - Verweis auf die Sprachentitätstypen-Übersichtseite
- Typische SeqAn-Aufgaben mit Verweisungen auf die dazugehörigen Tutorials, Klassen und Funktionen. Die vier typischen Aufgaben lauten:
  - Read-Mapping
  - Ein- u. Ausgabe

- Sequence-Alignment
- Graphen

Des Weiteren wurde das kaum auffindbare und veraltete Background-and-Motivation-Tutorial<sup>46</sup> umfassend überarbeitet. Es erklärt nun die Motivation der SeqAn zu Grunde liegenden Entwurfsentscheidungen und veranschaulicht den SeqAn-Performance-Gewinn an Hand eines ohne SeqAn geschriebenen Beispiels.

**Sprachentitätstypen** Das Konzept der Sprachentitätstypen (engl. *language entity types*) ist in SeqAn von besonders großer Relevanz, da SeqAn durch den Gebrauch von Concepts Sprachentitätstypen verwendet, die dem Gros der SeqAn-Anwender unbekannt sind (siehe Abschnitt 4.1.4). Dabei ist es von elementarer Wichtigkeit, bedeutende Konzepte in einer Dokumentation zu erklären (Jeong et al. 2009; Ko u. Riche 2011; Monperrus et al. 2011).

Mein Ziel war es, Anwender unaufdringlich auf die Existenz neuer Sprachentitätstypen hinzuweisen und es ihnen zu erlauben, sich auf leichtgewichtige Art darüber zu informieren. Dieser Ansatz wird als *knowledge pushing* bezeichnet und wurde bereits in ähnlicher Weise von Dekel u. Herbsleb (2009); Sunshine et al. (2014) erfolgreich angewendet.

Die Umsetzung erfolgte wie folgt:

- Alle in der Dokumentation genannten Sprachentitäten, wie `length` oder `String`, werden durch ein kleines Label annotiert.
- Jedes Label kennzeichnet einen Sprachentitätstyp.
- Die Labels werden durch zwei Merkmale unterschieden:
  1. Farbe, welche die Zusammengehörigkeit verschiedener Sprachentitätstypen symbolisiert.
  2. Ideogramm, also ein textuelles Piktogramm, das den Sprachentitätstyp phänotypisch ausdrückt.
- Die existierenden Sprachentitätstypen lauten:

<code>typedef</code>	Typdefinition mittels <code>typedef</code>	<code>int x</code>	Variable <sup>47</sup>
<code>Concept</code>	Concept	<code>Class</code>	Klasse
<code>Spec</code>	Spezialisierung	<code>enum</code>	Enum
<code>Fn&lt;&gt;</code>	Globale Metafunktion	<code>#Fn&lt;&gt;</code>	Interface-Metafunktion
<code>Tag</code>			Tag
<code>fn()</code>	Globale Funktion	<code>#fn()</code>	Interface-Funktion
<code>.fn()</code>			Member-Funktion
<code>foreign:</code>	Adaption	<code>#define</code>	Makro

<sup>46</sup> <http://seqan.readthedocs.org/en/master/Tutorial/BackgroundAndMotivation.html>

<sup>47</sup> Beim Schreiben dieser Dissertation wurde mir klar, dass `var` ein geeigneteres Ideogramm darstellt. Ein entsprechender Issue-Tracker-Eintrag existiert: <https://github.com/seqan/seqan/issues/1034>

- Bewegt der Anwender die Maus über ein Label, wird eine minimale Beschreibung des dazugehörigen Sprachentitätstyps eingeblendet (siehe Abbildung 4.11).
- Klickt der Anwender bei eingeblendetem minimaler Beschreibung auf das Label, wird die ausführlichere Beschreibung des Sprachentitätstyps geöffnet (siehe Abbildung 4.12).
- Sowohl Suchergebnisse als auch Funktionsauflistungen innerhalb von Dokumentationseinträgen sind nach den Sprachentitätstypen gruppiert (siehe Abbildung 4.8d).
- Die Suchfunktion der Dokumentation erlaubt die Filterung der Ergebnisse nach Sprachentitätstypen. Tatsächlich ging es mir aber nicht um die Funktion selbst, sondern darum, dass neugierige Anwender so auf eine vollständige Auflistung der in SeqAn präsenten Sprachentitätstypen stoßen, ohne sich mit diesen gleich aktiv auseinandersetzen zu müssen.

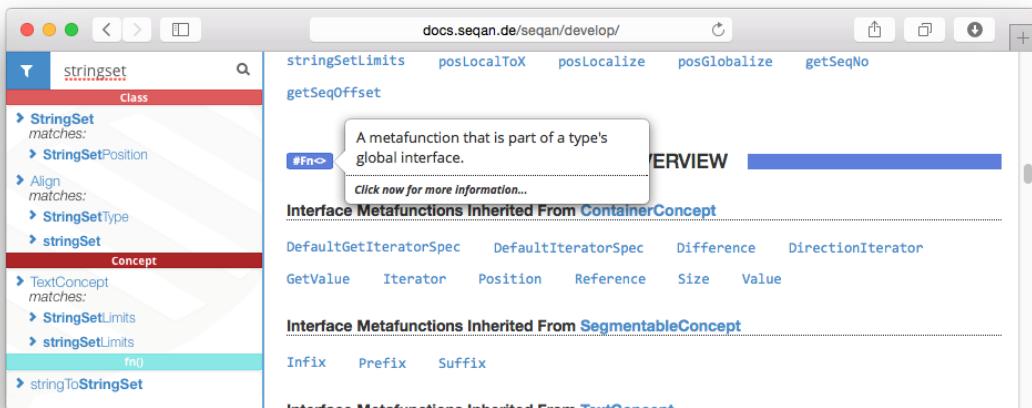


ABBILDUNG 4.11: Kurzbeschreibung des Sprachentitätstyps *Interface-Metafunktion*

**Seitenaufbau** Der Aufbau von Dokumentationsseiten wurde von mir und einem SeqAn-Entwickler überarbeitet. Abstrakt sind Seiten wie folgt gegliedert:

1. Zusammenfassung zum Nachschlagen
2. Detaillierte Beschreibung

Im Grunde gibt es zwei Seitentypen:

**Containerseiten** kommen zur Dokumentation von Concepts, Klassen und Spezialisierungen zum Einsatz<sup>48</sup>, da diese Sprachentitätstypen über Untereinträge verfügen. Hierbei umfasst die Zusammenfassung einen Bereich für Metainformationen und eine Auflistung aller verfügbaren Funktionen. Die Metainformationen zählen bei Klassen beispielsweise auf, welche Concepts direkt und indirekt implementiert werden, welche Spezialisierungen existieren, in welcher Datei die Klasse

48 Beispiel String-Klasse: <http://docs.seqan.de/seqan/develop/?p=String>

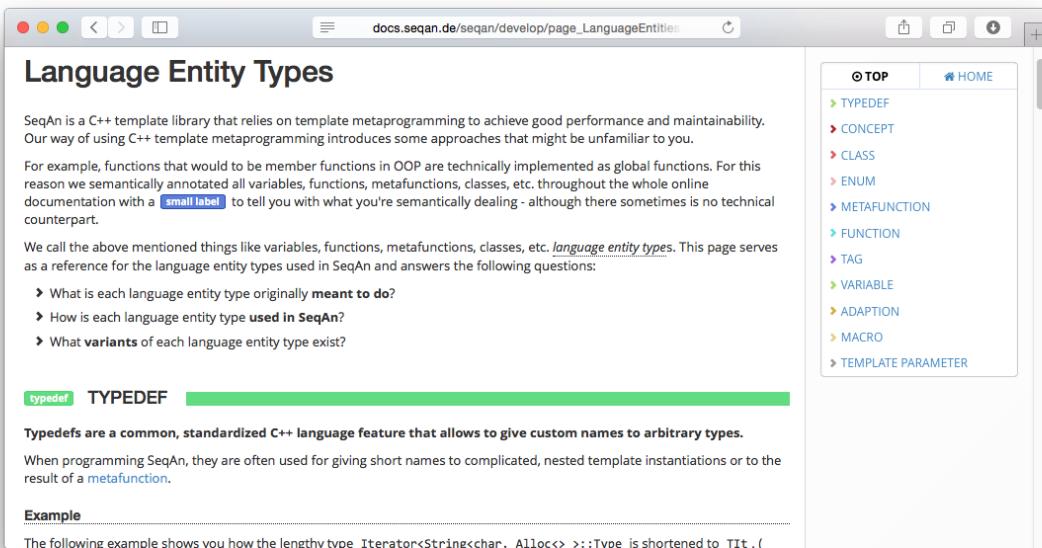


ABBILDUNG 4.12: Ausführliche Beschreibung der in SeqAn verwendeten Sprachentitätstypen

definiert ist und welche Signatur die Klasse besitzt. Die Auflistung der Funktionen ist gegliedert nach Member-Funktionen, Interface-Funktionen und Interface-Metafunktionen. Der Detailbereich erläutert im Falle einer Klasse den Gebrauch der Klasse samt Beispielen und erklärt detailliert sämtliche oben genannten Funktionen im Detail.

**Elementseiten** sind atomar und kommen für alle anderen Sprachentitätstypen zum Einsatz<sup>49</sup>, da diese über keine Untereinträge verfügen. Im Falle von (semantisch) globalen Funktionen umfasst die Zusammenfassung eine Auflistung aller Parameter und die Beschreibung der Rückgaben. Der Detailbereich erläutert dann wiederum ausführlich Sinn und Zweck der Funktion und beschreibt dies an Hand von Beispielen.

In SeqAn werden häufig Referenzen technisch als Eingaben übergeben, inhaltlich aber als Rückgaben behandelt. Ob ein Parameter als Ein- und/oder Ausgabe behandelt wird, wird durch die folgenden Icons ausgezeichnet: ↗, ↘ und ↗↘.

**Beispiele** Beispiele sind neben der Gesamtübersicht der wichtigste Bestandteil einer benutzerfreundlichen Dokumentation (Robillard u. DeLine 2010), was ich an Hand der verschiedenen, durch die Abwesenheit (guter) Beispiele eingeschränkten Beispiel-bezogenen Strategien empirisch belegen konnte (siehe Abschnitt 4.3.9).

Im Zuge der inhaltlichen Überarbeitung der Dokumentation wurden für die wichtigsten Dokumentationseinträge existierende Beispiele verbessert und fehlende Beispiele ergänzt. Dabei wurden alle Beispiele um die Programmausgabe ergänzt, was besonders für API-Endanwender wichtig ist (Gross u. Kelleher 2009).

49 Beispiel globalAlignment-Funktion: <http://docs.seqan.de/seqan/develop/?p=globalAlignment>

**Suchfunktion** Die vollständig neu entwickelte Suchfunktion verfügt über die folgenden Funktionen:

- Sprachentitäten können nun über Aliasse/Synonyme verfügen. Auf diese Weise wurden die im *Vocabulary Problem* (Furnas et al. 1987) veranlagten Usability-Probleme gemildert. Der Suchbegriff `substring` beispielweise führt nun auch zu dem Treffer `infix`. Wurde ein Eintrag mit Hilfe eines Synonyms gefunden, wird dem Anwender dies mitgeteilt. Auf diese Weise lernt der Anwender bedarfsgerecht die korrekte Bezeichnung in SeqAn. Anwender, die ohnehin den korrekten Begriff verwendet haben, erfahren nicht von den Synonymen. Auf diese Weise wird verhindert, dass unterschiedlich benannte, aber inhaltlich identische Sprachentitäten innerhalb von SeqAn gepflegt werden müssen.
- Die Suchergebnisse sind nun gewichtet, so dass beispielsweise der Suchbegriff `string` auch tatsächlich die `String`-Klasse als ersten Suchtreffer hervorbringt.
- Die Suchergebnisse sind nach Sprachentitätstypen gruppiert und auf maximal fünf Einträge je Typ begrenzt. Auslassungspunkte am Ende einer Treffergruppe deuten darauf hin. Ein Klick darauf zeigt alle Suchtreffer der Gruppe.
- Suchergebnisse können auch Dokumentationsteileinträge sein. So ist der erste Suchtreffer für `infix`, die `Infix`-Interface-Metafunktion innerhalb des Concepts `SegmentableConcept`. Der Anwender kann selbst wählen, ob der den Dokumentationseintrag oder -untereintrag öffnen möchte.
- Die Suche kann ohne Maus und ausschließlich mit der Tastatur bedient werden. Dazu erhält das Suchfeld automatisch den Fokus, die Sucheingabe kann mit `Esc` gelöscht und innerhalb der Suchergebnisse mit den Pfeiltasten navigiert werden.

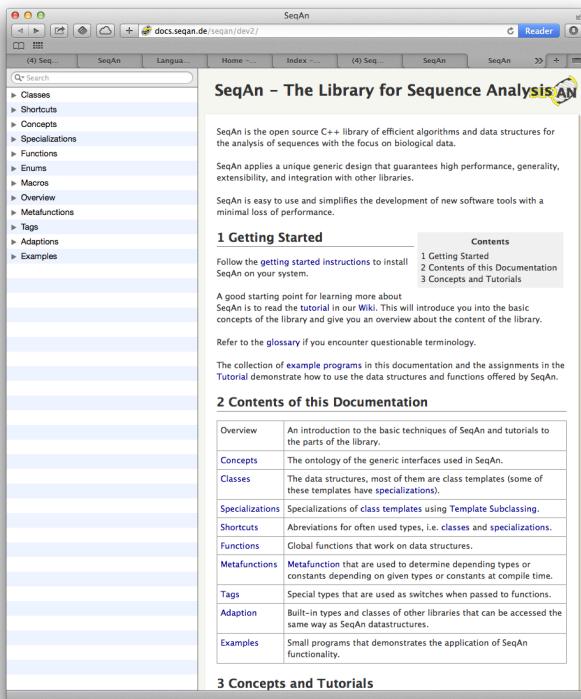
**Darstellung** Die Online-Dokumentation ist *responsive*<sup>50</sup>, passt sich also einer großen Bandbreite von Fenstergrößen an. Dies ist nicht nur schick, sondern auch hilfreich. Bei SeqAn-Anfängern konnte ich einen häufigen Wechsel zwischen Entwicklungsumgebung und Online-Dokumentation beobachten. Da die neue SeqAn-Dokumentation selbst bei einem 600 Pixel schmalen Fenster (bei 72 DPI) ohne Einschränkung nutzbar ist, können SeqAn-Anwender die Dokumentation auch neben der Entwicklungsumgebung platzieren. Auch ist eine Nutzung der Dokumentation auf mobilen Geräten möglich, wie es einige Anwender forderten.

Des Weiteren habe ich großen Wert auf eine hohe Ästhetik gelegt. Ich denke, dieses u.a. durch den Einsatz von subtilen Farbverläufen und dezenten Animationen erreicht zu haben. Spaß an der Bedienung ist ein wesentlicher Aspekt des Anwendererlebnisses und spielt damit auch für SeqAns kommerziellen Erfolg eine Rolle.

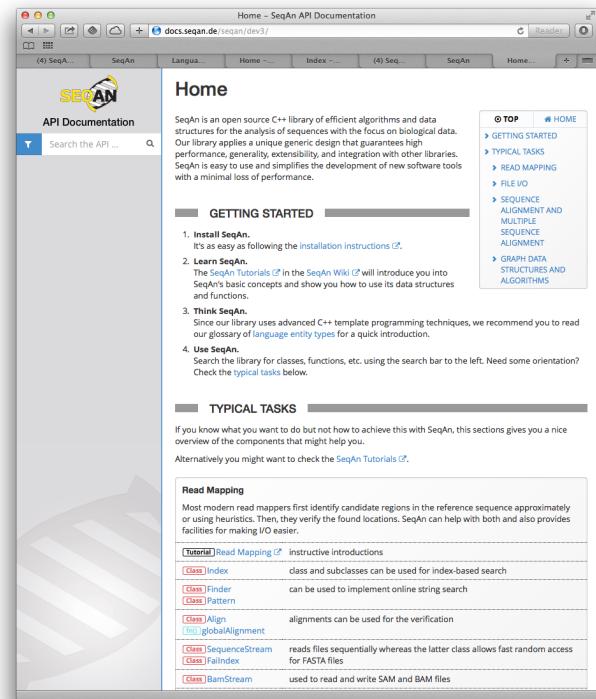
Abbildung 4.13 stellt die gleichen Inhalte wie Abbildung 4.8 auf Seite 312 dar — nur mit dem Unterschied eines schmäleren Browser-Fensters.

---

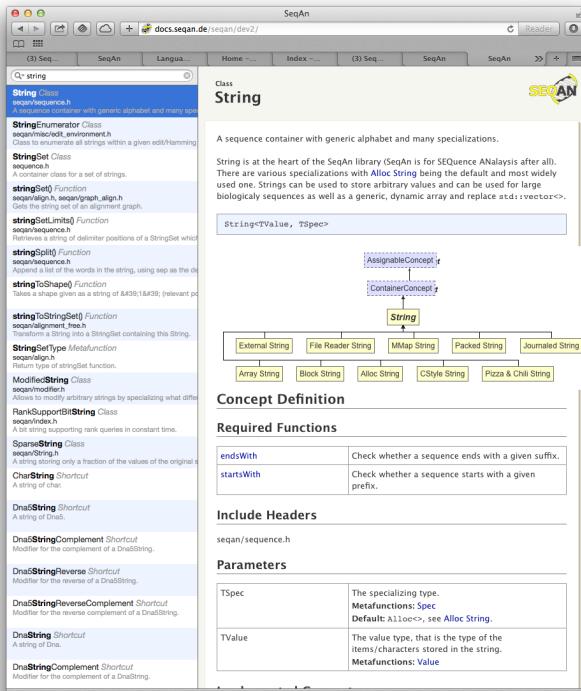
<sup>50</sup> <https://developers.google.com/web/fundamentals/layouts/rwd-fundamentals/>



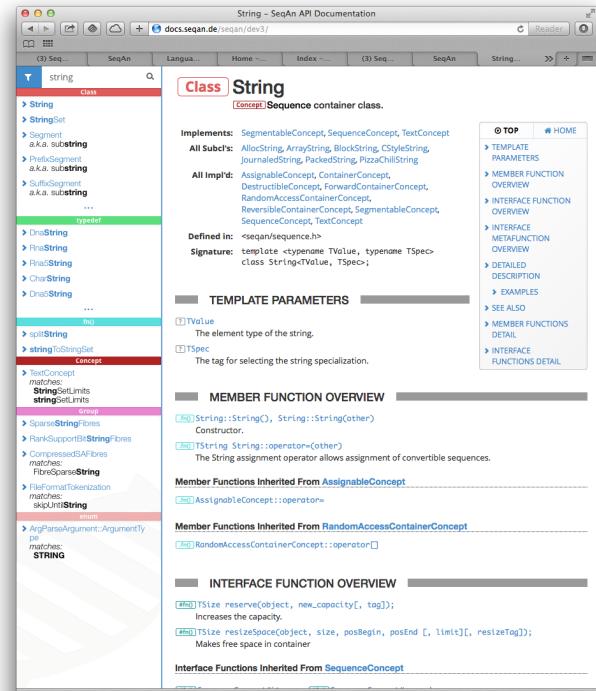
(A) Startseite in Version 2.0.0



(B) Startseite in Version 3.0.0



(c) Geöffnete String-Klasse in Version 2.0.0



(d) Geöffnete String-Klasse in Version 3.0.0

ABBILDUNG 4.13: Die Abbildungen zeigen die alte und neue Dokumentation in schmaler Breite.

**Integration** Die neue Dokumentation ist besser in anderen Lernressourcen, wie den Tutorials, integriert. Dies wurde nicht zuletzt durch den weiter oben beschrieben Entwicklermodus gefördert, der SeqAn-Entwicklern alle möglichen Links für einen Dokumentations-(teil-)eintrag bereitstellt.

Außerdem habe ich bei der Neuentwicklung auf die Indexierbarkeit durch Suchmaschinen geachtet. Dazu habe ich das Programm, welches die Dokumentation generiert, so angepasst, dass Inhalte sowohl statisch (für Suchmaschinen), als auch dynamisch (für Anwender) vorliegen.

Des Weiteren erfordert die Dokumentation — trotz seines Funktionszuwachses — keine Server-Komponente. Die mit SeqAn heruntergeladene Dokumentation verfügt online wie offline über den gleichen Funktionsumfang.

#### 4.4.9 Kollaborationsplattform

Im Zuge der Umstellung des Versionsverwaltungssystems von Subversion auf Git wurde von einer selbst gehosteten Lösung auf *GitHub*<sup>51</sup> umgestellt. Diese Plattform bietet einen *Issue-Tracker*<sup>52</sup>, der von SeqAn-Anwendern sehr gut angenommen und von SeqAn-Entwicklern aktiv verwaltet wird. Auf dieser Plattform findet ein fruchtbarer Austausch zwischen Anwendern und Entwicklern statt<sup>53</sup>.

#### 4.4.10 Usability-Priorisierung

Die gesamtheitliche Erforschung der Entwicklung der SeqAn-Library brachte erst spät die trivial erscheinende Erkenntnis, dass es Gründe für die besondere Betonung der Performance gab und erst die kommerzielle Verbreiterung des SeqAn-Einsatzzweckes zu einer notwendigen Neugewichtung der Usability führt bzw. führen muss.

Mit dieser Arbeit liefere ich belastbare Argumente für meine Theorie, die dem SeqAn-Entwicklungsteam dabei helfen sollen, diesen Perspektivwechsel wahrzunehmen und eine verstärkte Betonung der Usability in Form von expliziten-empirischen Entwurfsentscheidungen zu fördern. Diese Einsicht auf Entwicklerseite würde mittelfristig zu einer ganzheitlichen Verbesserung der API-Usability von SeqAn führen.

#### 4.4.11 Zusammenfassung

Für die Verbesserung der API-Usability von SeqAn haben die Bioinformatik-Arbeitsgruppe und ich umfangreiche Maßnahmen umgesetzt.

Neben den aus der ersten Verbesserungsiteration stammenden Prozessverbesserungen, konnten meine SeqAn-Kollegen und ich drei der vier wichtigsten Maßnahmen vollständig bearbeiten:

---

<sup>51</sup> <https://github.com>

<sup>52</sup> <https://github.com/seqan/seqan/issues>

<sup>53</sup> Beispiel: <https://github.com/seqan/seqan/issues/919>

1. SeqAn kann nun auch als Library verwendet werden, indem insbesondere Vorwärtsdeklarationen beseitigt wurden. So simpel dieser Punkt klingen mag, so fundamental war er auch für potentielle SeqAn-Anwender, die ihren bestehenden Build-Prozess nicht anpassen wollten oder konnten.
2. Die Dokumentation wurde technisch neu entwickelt und inhaltlich überarbeitet. Dazu wurde das neue Dokumentationsformat Dox entwickelt, ein Gesamtüberblick erarbeitet sowie Seitenaufbau, Beispiele, Suchfunktion, Darstellung und die Integration verbessert. Außerdem wurde das Konzept der Sprachentitätstypen gesamtheitlich innerhalb der Dokumentation implementiert. Ergänzt wird die Verbesserung der Dokumentation durch die bereits in der ersten Verbesserungsiteration generalüberholten Installationsanleitungen und Tutorials.
3. SeqAn kann nun von API-Endanwendern genutzt werden. Dazu wurde eine Wrapper-API in Form einer Integration in die Workflow-Engine KNIME implementiert.

Außerdem abgeschlossen wurde die Einrichtung einer Kollaborationsplattform — in Form eines GitHub Issue-Trackers — für den Austausch zwischen SeqAn-Entwicklern und -Anwendern.

Aus diversen, insbesondere Zeitgründen konnte ich leider nicht darauf hinwirken, dass die wichtige STL-Angleichung in Angriff genommen wurde. Zwar sind Metafunktionen zur Berechnung von Rückgabetypen, wegen des in C++11 eingeführten `auto`-Schlüsselworts theoretisch nicht mehr nötig, was die mit dieser Maßnahme in Verbindung stehenden Usability-Probleme teilweise entschärft. Dies muss jedoch noch praktisch gezeigt werden. Im Erfolgsfall müssen außerdem sämtliche Lernressourcen entsprechend angepasst werden. Diese Entwicklung hat jedoch keinen Einfluss auf meine dringende Empfehlung, das CRTP einzusetzen, um die globalen Funktionen zu Memberfunktionen zu refaktorieren, die auch in C++ als Memberfunktionen implementiert sind.

Ebenfalls aus Zeitgründen konnten die Maßnahmen Inkonsistenzbeseitigung, Shortcuts und die Evaluation des Code-Beispiel-Erzeugungs-Algorithmus nicht umgesetzt werden.

Das dem der Maßnahme Intransparenzbeseitigung zu Grunde liegende Usability-Problem ● versteckte Parameterübergabe ist in Bezug auf seine Fatalität nicht hinreichend gut verstanden, weshalb ich die Umsetzung der Maßnahme angesichts der hohen Kosten momentan nicht empfehlen kann.

Die Umsetzung der abstrakten Maßnahme Usability-Priorisierung ist längst nicht abgeschlossen und soll durch diese Arbeit motiviert werden.



## GÜTE, VALIDIERUNG UND VERALLGEMEINERBARKEIT

In den vorangegangenen Unterkapiteln habe ich meine GT vorgestellt. Für die darin beschriebenen Usability-Probleme habe ich Lösungsmaßnahmen vorgeschlagen und deren Bearbeitung durch die SeqAn-Entwickler beschrieben.

In diesem Unterkapitel möchte ich die Fragen beantworten, inwiefern meine Forschung überhaupt den Gütekriterien qualitativer Forschung entspricht und ob die Umsetzung meiner vorgeschlagenen Maßnahmen tatsächlich zu der erhofften Usability-Verbesserung führten. Des Weiteren möchte ich die Frage nach der Verallgemeinerbarkeit meiner Forschungsmethode und -ergebnisse klären.

### 4.5.1 Güte

In dieser Arbeit habe ich einen qualitativen Forschungsansatz gewählt, bei dem die GTM eine große Rolle spielt. Für die Bewertung der Güte meiner Arbeit gelten also die am Anfang dieser Arbeit, im Abschnitt 1.4, vorgestellten Gütekriterien qualitativer Forschung. Entlang dieser Gütekriterien werde ich meine Arbeit bewerten.

#### Verfahrensdokumentation

Ich habe meine durch die wissenschaftliche Literaturstudie erworbenen Vorkenntnisse, die Forschungsplanung, die Datenerhebung, den Bau des Datenanalysewerkzeugs, die Herleitung der GT, das Zustandekommen der Verbesserungsvorschläge und deren Umsetzung ausführlich beschrieben. Dabei habe ich die erhobenen Daten und die Grundlagen der von mir entwickelten Konzepte detailliert mittels entsprechender URIs zugänglich gemacht. Durch den Einsatz einer Versionsverwaltung kann sogar das Zustandekommen meiner als XML-Datei gesicherten Theorie über die Zeit nachvollzogen werden.

#### Argumentative Interpretationsabsicherung

Aus Platz-, Zeit- und Plausibilitätsgründen habe ich der Darstellung von Interpretationsalternativen nur wenig Raum in dieser Arbeit gegeben. Wenn es zu Konzepten relevante Literatur gab, habe ich diese genannt und zu meinen Beobachtungen in Beziehung gesetzt. Hatte ich Unsicherheiten in der Interpretation von Konzepten, habe ich diese bei für wichtig geglaubten Konzepten ausgeräumt. Weniger wichtige Konzepte habe ich in dieser Arbeit nicht aufgeführt. Die schlecht verstandenen Konzepte (z.B.  versteckte Parameterübergabe) habe ich als solche deklariert.

#### Regelgeleitetheit

Meine Forschungsergebnisse entstanden innerhalb eines Prozesses, der stark durch die GTM geprägt war. Durch die Implementierung eines dazugehörigen Analysewerkzeugs habe ich insbesondere die Phasen des offenen und axialen Kodierens und damit einen umfangreichen Prozessteil

kodifiziert. Bei meiner Forschung kam es zu Abweichungen, die ich nachvollziehbar erläutert habe. Beispielsweise habe ich begründet, weshalb eine erste Beseitigung grober Usability-Probleme notwendig war und auf welche Weise ich dazu eine vereinfachte Form der Heuristischen Evaluation (HE) durchgeführt habe. Weiterhin habe ich erläutert, dass ich eine besonders gründliche und breit aufgestellte Datenerhebung durchgeführt habe, um trotz der widrigen Umstände theoretischen Sampling durchführen zu können.

### Nähe zum Gegenstand

Dank der Workshops, zu denen auch langjährige SeqAn-Anwender gehörten, konnte ich reale SeqAn-Anwender und die, die es möglicherweise werden wollten, beobachten. Meine Art der Datenaufzeichnung hat es mir erlaubt, auf ein Laborsetting zu verzichten, bei dem Anwender an “unnatürlichen”, speziell für die Datenaufzeichnung vorbereiteten Arbeitsplätzen arbeiten müssten. Ich habe sowohl rein objektive Daten, als auch subjektive Daten erhoben. Für die Erhebung Letzterer habe ich zu den SeqAn-Anwendern ein gegenseitiges und offenes Verhältnis gepflegt und mein Arbeitsziel klar und ohne “Täuschung” formuliert. Auf diese Weise wurden den SeqAn-Anwendern deutlich, dass sie und ich ein gemeinsames Ziel verfolgen — nämlich ein benutzerfreundliches SeqAn. Schwächen hatte meine Forschung bzgl. dieses Kriteriums, weil ich Anwender nicht an ihrem natürlichen Arbeitsplatz, sondern in Praktika und Workshops beobachtet habe. Meine Langzeitaufzeichnungen könnten diesen Punkt entkräften, jedoch habe ich für deren Analyse keine Zeit aufbringen können.

### Kommunikative Validierung

Die Gültigkeit vieler Ergebnisse — insbesondere der ● Inkonsistenzen bzgl. STL — konnte ich auf zwei Kommunikationswegen mit den SeqAn-Anwendern validieren. Einerseits zeigten die Datenerhebungen nach Überarbeitungen von SeqAn (z.B. Dokumentation) viel positives Feedback. Andererseits wurden Analyseergebnisse direkt mit SeqAn-Anwendern besprochen, wie dies beispielsweise bei der Gruppendiskussion der Fall war. Manche Ergebnisse konnte ich jedoch nicht validieren, weil mir entweder die Zeit fehlte oder ich die betroffene Person wegen meiner streng-pseudonymisierten Datenerhebung nicht identifizieren konnte (z.B. ● versteckte Parameterübergabe). Die Triangulation vieler meiner Ergebnisse mit existierender Literatur verbessert die Verallgemeinerbarkeit dieser Ergebnisse.

### Triangulation

Für die Datenerhebung und -analyse kamen sowohl subjektive (Umfrage, Interview, Feedback, Gruppendiskussion und CDF-Fragebogen) als auch objektive Datenquellen (Programmierfortschritte) zum Einsatz (Datentriangulation). Für die Forschung selbst verwendete ich die HE<sup>G</sup> und die GTM (Methodentriangulation). Eine Reihe von Erkenntnissen konnte auf diese Weise trianguliert werden. Dabei traten in unterschiedlichen Datenquellen Konzepte mit unterschiedlichen Schwerpunkten auf (z.B. CDF-Fragebogen: ● Fehlende Funktionskategorisierung; Gruppendiskussion: ● Inkonsistenzen bzgl. STL; Programmierfortschritte: ● Wiederverwendung).

Aus Zeitgründen konnte ich leider den Großteil der sich aus den subjektiven Daten ergebenen Ergebnisse nicht mit Hilfe meiner Programmierfortschritte-Datenquelle validieren. Dennoch bin

ich der festen Überzeugung, dass dies für viele Konzepte möglich ist. Besonders gut dafür geeignet sind die Daten meines Langzeitprobanden, da bei ihm nicht davon auszugehen ist, dass Beobachtungen auf fachlich schlechte Kenntnisse zurückzuführen sind. Die Auswertung dieser Daten bleibt nachfolgenden Arbeiten überlassen.

Im Abschnitt 1.4 habe ich neben den Gütekriterien nach Mayring (2002) auch weitere Gütekriterien nach Steinke (1999) vorgestellt, die sich zwar überschneiden, ich an dieser Stelle aber dennoch für die Bewertung meiner Arbeit heranziehen möchte:

### **Intersubjektive Nachvollziehbarkeit**

Meine Argumentation in Bezug auf die *Verfahrensdokumentation* trifft auch auf dieses Gütekriterium zu.

### **Reflektierte Subjektivität**

Während meiner Arbeit habe ich meine eigene Rolle als Bestandteil des Forschungsprozesses reflektiert, weshalb ich für meine Ausarbeitung auch die Ich-Form gewählt habe. Ich war mir bewusst, dass ich ein GTM-Anfänger und zu Forschungsbeginn in keiner Weise mit der Bioinformatik vertraut war. Während ich Ersteres strukturiert, und, so gut es geht, ausgeräumt habe<sup>54</sup>, hätte ich mich mit Letzterem intensiver — z.B. durch den Besuch von Bioinformatik-Lehrveranstaltungen — auseinandersetzen können. Meine Beschäftigung mit der Bioinformatik bestand hauptsächlich in der Teilnahme an Bioinformatik-Vorträgen und vielen Fragen an meine Bioinformatik-Kollegen. Probleme mit meinem Bioinformatik-Kenntnisstand konnte ich bei der Analyse der Programmierfortschritte-Daten feststellen, was die ohnehin zeitraubende Analyse weiter verlangsamte. Bei der Analyse subjektiver Daten fiel mir hingegen keinerlei Mangel auf. Auch wenn ich es bezweifle, kann ich abschließend nicht sagen, ob sich andere Theorieschwerpunkte ergeben hätten, hätte ich über bessere Bioinformatik-Kenntnisse verfügt.

Meine Sensibilität für Usability ist interessenbedingt hoch. In Bezug auf API-Usability habe ich meine Sensibilität insbesondere durch die Durchführung der in dieser Arbeit vorgestellten, umfassenden Literaturforschung erhöht.

Meine Nähe zu den SeqAn-Anwendern habe ich bereits unter dem Gütekriterium *Nähe zum Gegenstand* erläutert.

### **Limitation**

Dieses Kriterium befasst sich mit den Grenzen der Verallgemeinerbarkeit der entwickelten Theorie, worauf ich im nächsten Abschnitt eingehen.

Auch wenn es nach meiner Kenntnis kein Gütekriterium *Intrasubjektive Nachvollziehbarkeit* in der Literatur gibt, habe ich ein solches dennoch feststellen können. Mir fiel auf, dass meine Bewertungen einer Vielzahl von Phänomenen und Konzepten immer wieder mit den vor Monaten bereits in Memos

---

<sup>54</sup> Dazu habe ich auf die Erfahrungen meiner GTM-geübten Kollegen zurückgegriffen, viel GTM-Literatur gelesen und eine einwöchige GTM-Weiterbildung absolviert. Durch die Entwicklung eines qualitativen Analysewerkzeugs, habe ich mich besonders intensiv mit der GTM auseinander gesetzt.

festgehaltenen Bewertungen weitgehend übereinstimmten. Dies deutet auf eine stabile Interpretation hin.

## 4.5.2 Validierung

In der ursprünglichen Planung (siehe Abschnitt 3.1.3) war vorgesehen, dass ich eine weitere Datenerhebung durchföhre, die erhobenen Daten analysiere und prüfe, ob die vermeintlich behobenen Usability-Probleme tatsächlich zu einer API-Usability-Verbesserung beitrugen.

Dies war mir aus diversen Gründen (siehe Abschnitt 3.1.4) nicht möglich. Daher entschloss ich mich, eine vereinfachte Validierung mit Hilfe des Cognitive Dimension Frameworks durchzuführen.

### 4.5.2.1 Validierung mit Hilfe des Cognitive Dimension Frameworks

Das Cognitive Dimensions Framework habe ich ausführlich im Abschnitt 2.3.2 vorgestellt. Um zumindest eine vereinfachte Validierung vorzunehmen, wollte ich den von mir entwickelten Cognitive-Dimension-Fragebogen (siehe Abschnitt 3.3.4) einmal vor und einmal nach der Umsetzung der von mir vorgeschlagenen Verbesserungsmaßnahmen einsetzen. Dafür hätte ich zwei Idealprofile — eins für die SeqAn-Anwender und eins für die SeqAn-Endanwender — erarbeiten müssen. Abweichungen der SeqAn-Profile vor und nach der Umsetzung der Maßnahmen würden dann mit den Anwenderbestimmten Idealprofilen verglichen werden müssen, um Aussagen über die Verbesserung treffen zu können.

Bedauerlicherweise scheiterte mein Versuch der Validierung mit Hilfe des CDF. Zum einen ist die konkrete Anwendung des CDF schlecht in Green (1996) erklärt (siehe Abschnitt 2.3.2.4). Des Weiteren wird kaum erläutert, wie das Idealprofil — oder in meinem Fall die zwei Idealprofile — zu ermitteln sind. Die Autoren empfehlen den Einsatz eines Fragebogens, der allerdings seine eigenen Probleme in sich birgt (siehe Abschnitt 3.3.4). Beispielsweise werden nicht alle Fragen hinreichend ausführlich beantwortet, weil der Fragebogen schlicht zu umfassend ist. Oder aber, Fragen werden von den Befragten falsch verstanden, wodurch man die entsprechende dimensionale Ausprägung nicht mehr belastbar bewerten kann, denn die dazu notwendigen Informationen liegen ja nicht vor. Diese Probleme führen dazu, dass weit mehr als zehn ausgefüllte Fragebögen notwendig sind, um jede kognitive Dimension valide einzuschätzen zu können.

Ein weiteres Problem ist, dass es Überlappungen und Abhängigkeiten zwischen den kognitiven Dimensionen gibt (Blackwell u. Green 2003). Auch fehlt es in der Literatur an Studien, die das CDF tatsächlich vollständig eingesetzt hätten (vgl. Blackwell u. Green 2003; Green u. Blackwell 1998; Roast et al. 2000). Selbst der umfangreichste, von mir gefundene Anwendungsfall behandelt nur 6 der 14 Dimensionen (Green u. Blackwell 1998). Da es sich bei der Bewertung der kognitiven Dimensionen jeweils um einen Einzelfall handelt, reicht diese Demonstration jedoch nicht aus, um die anderen kognitiven Dimensionen bewerten zu können.

Außerdem ist der Vergleich von zwei Profilen in Bezug auf eine kognitive Dimension schwierig, denn in der CDF-Literatur gibt es keinerlei Angaben dazu, wie Messungen quantitativ verglichen werden könnten.

#### 4.5.2.2 Argumentative Validierung

Da ich weder eine empirische (siehe Abschnitt 3.1.4), noch eine verkürzte Validierung mittels CDF vornehmen konnte, bin ich dazu gezwungen, eine argumentative Validierung durchzuführen.

Was die Auswahl der Probanden betrifft, kann ich feststellen, dass die beobachteten SeqAn-Anwender die SeqAn-Anwenderschaft für meine Forschung hinreichend repräsentieren<sup>55</sup>, was wichtig für valide Ergebnisse ist (Clarke 2004; Henning 2007). Lediglich, wenn einzelne Usability-Probleme besser hätten verstanden werden müssen, wäre eine zielgerichteter Auswahl der Probanden und eine langfristigere Beobachtung notwendig gewesen. Die wenigen Usability-Probleme, die ich nicht ausreichend verstanden habe, habe ich auch als solche benannt.

Im Falle einer GTM-basierten Forschung müsste normalerweise über das ständige Vergleichen argumentiert werden und dabei das unermüdliche Infragestellen und Überprüfen von Konzepten und Relationen gezeigt werden (siehe Abschnitt 1.4). Dieses Vorgehen ist mir aber nicht möglich, weil es für einige meiner Konzepte und Relationen nur wenige Exemplare gibt. Außerdem sind die meisten meiner Exemplare detailarm, da sie größtenteils Ex-post-facto-Datenquellen wie der Gruppendiskussion und den Cognitive-Dimensions-Fragebögen entspringen (siehe Abschnitt 3.5.3.3).

Meine Argumentation basiert daher auf einer Art indirekten Beweis, der nicht das ständige Vergleichen, sondern meinen auf im Abschnitt 3.5.2.5 vorgestellten abduktiven Blitz in den Mittelpunkt stellt. Dieser besteht darin, dass die wichtigsten Usability-Probleme auf die anwenderseitigen ●paradigmatischen Prägungen und den SeqAn-gestaltenden ●Entwurfsentscheidungen zurückzuführen sind.

Es stellt sich also die Frage, was schief gegangen sein müsste, wenn meine GT nicht stimmen sollte. Um diese Frage zu beantworten, beleuchte ich die folgenden drei möglichen Arten von Fehlern, die mir unterlaufen sein könnten:

##### 1. Wurde ein völlig unzutreffendes Konzept bzw. Relation erkannt?

Vollkommen unzutreffende, in dieser Arbeit präsentierte Konzepte und Relationen wären den Lesern bereits in der Vorstellung der GT aufgefallen. Für die Überprüfung stehen sowohl die angegebenen, mit einem ⚡-Symbol versehenen Exemplare, die zu den Konzepten gehörigen Memos und die insbesondere bei den vornehmlich technisch-deduktiven Konzepten angegebene Literatur zur Verfügung.

<sup>55</sup> In der ersten Analysephase habe ich die Anwenderschaft von SeqAn charakterisiert (siehe Abschnitt 3.2.3.1). Zu dieser gehören berufstätige, nationale und internationale Wissenschaftler aus den Bereichen Informatik, Bioinformatik und Physik. Aus diesem Grund halte ich diese Personen geeignet als Probanden für meine Forschung. Der Studienanteil stellt keine Probleme dar, da er einen beachtlichen Teil zur bioinformatischen Arbeit beiträgt (Letondal 2006) und damit mindestens zur zukünftigen Anwendergruppe gerechnet werden kann.

### Beispiel 1

Nehmen wir als Beispiel das folgende Exemplar, welches die Antwort eines Probanden<sup>55</sup> auf die Frage nach der kognitiven Dimension *Hard Mental Operations* (siehe Anhang B.2) war.

**Frage:** “Do some things in SeqAn seem especially complex or difficult to work out in your head (e.g. when combining several things)? What are they?”

**Antwort:** “Yes, remembering all the templates and variants of types and templates and trying to figure out what is the difference between say Dna5 and Dna5String.”<sup>56</sup>

Bei naiver Lesart könnte man zu dem Schluss kommen, dass die Kernaussage des Probanden darin besteht, dass die Vielzahl an Templates und Typvarianten deren Einprägen verhindert. Ein mögliches Konzept hieße also ● Template-/Typ-Variantenübermaß oder etwas deskriptiver ● Nicht einprägsames Template-/Typ-Variantenübermaß.

Wendet man hingegen die *wortgenaue Analyse* nach Charmaz (2006) an, ergibt sich folgende stichpunktartige Paraphrase:

- remembering complicated
  - many templates
  - many variants of types
  - many variants of templates
- difference unclear
  - e.g. Dna5 and Dna5String

Schaut man sich darüber hinaus die übrigens Antworten des Teilnehmers an (“typing / templates seem to be extremely messy”<sup>57</sup>, “you need to take care of types, type casting, [...] etc. “99% of the cases [...] will not even compile”<sup>58</sup>, “[requires] me to spend time on getting type casting right”<sup>59</sup>, ...) erkennt man, dass es gar nicht um das Einprägen von Templates oder Typvarianten geht. Vielmehr wird deutlich, dass der Proband jede gestellte Frage nutzt, um seinen Verdruss über die Art zum Ausdruck bringt, wie schwer der Umgang mit Typen in SeqAn ist.

Das in meinen Augen korrekte Konzept lautet also ● Typing (siehe Seite 269). Dieses Konzept befasst sich mit den Problemen, die Anwender bei der Deklaration von Typen, bei Type-Castings, beim Setzen von Parametern von Klassen- und Funktionstemplate und bei der Berechnung von Rückgabetypen mittels Metafunktionen haben.

Der zweite Teil der Aussage des Probanden bezieht sich auf die unklare Unterscheidbarkeit von der `String`-Klasse und ihren Spezialisierungen, was ich als ○ Fachliche Inkorrektheit bezeichne<sup>56</sup>, aber wegen der ohnehin schon vielen wichtigen Erkenntnisse nicht weiter verfolgt habe.

### Beispiel 2

Auf die Frage zur CD *Viskosität*<sup>57</sup> (siehe Anhang B.2) antworte ein anderer Proband<sup>58</sup> wie folgt:

<sup>56</sup> In SeqAn dienen die `Dna`-Varianten als Spezialisierung der `String`-Klasse, was ich fachlich für inkorrekt halte. Tatsächlich müsste es `Nucleotide`-Varianten geben, die `String` spezialisieren. Folglich müsste `Dna` als `typedef String<Nucleotide>` definiert sein.

<sup>57</sup> Der Fragebogen wurde sowohl auf Deutsch wie auch auf Englisch angeboten. Daher gebe ich die Übersetzung an, die der Proband mit großer Wahrscheinlichkeit auch gelesen hat.

**Frage:** “Wenn du Änderungen am Code durchführst, welche fallen dir am schwersten/aufwendigsten? Warum?”

**Antwort:** “Wenn es wenig Beispiele gibt bzw. die Funktionalität mancher Konstrukte zu ergründen ist”<sup>8</sup>

Bei dieser ziemlich knappen Antwort könnte man glauben, dass der Proband hinterfragt, was manche “Konstrukte” genau tun. Das Konzept könnte also ⚪ Unklare Konstrukt-Funktionalität heißen.

Aber auch hier wäre die naive Lesart unpassend und die Betrachtung der übrigen Antworten desselben Probanden anzuraten:

- Antwort auf die Frage zur CD *Schwierige mentale Operationen*: “Wenn viele Konzepte und Konstrukte ineinander greifen ist es teils die Verwendung teils schwer. Klar ;)”<sup>8</sup>
- Antwort auf die Frage zur CD *Fortschreitende Evaluation*: “Wenn man lange nach bestimmten Funktionen sucht bzw. lange braucht deren Verwendung zu verstehen ist der Fortschritt manchmal schwer zu ueberblicken. Im Prinzip aber ja.”<sup>8</sup>

Bei Betrachtung dieser beiden weiteren Antworten wird deutlich, dass es nicht um die “Funktionalität” von “Konstrukten”, sondern um deren “Verwendung” geht. In Verbindung mit dem gleichzeitigen Auftreten der Begriffe “Konzepte”, “Konstrukte” und “Funktionen” schließe ich, dass es neben den ⚩ fehlenden Anwendungsbeispielen im Kern um das Usability-Problem ⚩ Funktionsgebrauch (siehe Seite 269) geht. Dieses Konzept befasst sich mit der anwenderseitigen Frage, wie Funktionen verwendet werden. Es handelt sich dabei um ein Elternkonzept des schon genannten ⚩ Typing-Konzepts (vgl. Abschnitt 3.5.3.2: “Modellierung der Ergebnisse”).

### Beispiel 3

Wenn man sich noch einmal Beispiel 2 ansieht und sich fragt, welche Relationen in der Antwort “Wenn es wenig Beispiele gibt bzw. die Funktionalität mancher Konstrukte zu ergründen ist”<sup>8</sup> auf die Viskositätsfrage “Wenn du Änderungen am Code durchführst, welche fallen dir am schwersten/aufwendigsten? Warum?” stecken, sieht man die folgende Relation leicht ein: ⚩ Fehlende Anwendungsbeispiele  $\xrightarrow{\text{verlangsamen}}$  ⚪ Programmierung.

Wäre man dem Irrtum erlegen, der Proband spräche von der “Funktionalität mancher Konstrukte”, hieße eine weitere, inkorrekte Relation ⚪ Unklare Konstrukt-Funktionalität  $\xrightarrow{\text{verlangsamt}}$  ⚪ Programmierung. Tatsächlich verbirgt sich aber die folgende Relation in der Antwort: ⚩ Funktionsgebrauch  $\xrightarrow{\text{verlangsamt}}$  ⚪ Programmierung.

Nach mehreren Monaten der Datenanalyse habe ich eine gewisse theoretische Sensibilität erlangt, die mir auch dabei half, Begriffe wie “Konstrukte”, “Konzepte” und “Idiome” zu deuten. Aus dem breiten Kontext wird deutlich, dass diese Begrifflichkeiten auf den für die beobachteten ⚩ paradigmatischen Prägungen untypischen Entwurf von SeqAn abzielen. Dieser ist geprägt, durch Elemente wie Metafunktionen, Tags und Interface-Funktionen, welche ich zusammengefasst als ⚩ Sprachentitätstypen bezeichne (siehe Seite 270). Daraus ergibt sich die folgende weitere Relation: ⚩ Sprachentitätstypen  $\xrightarrow{\text{erschweren}}$  ⚩ Funktionsgebrauch. Ohne diese theoretische Sensibilität und ohne Betrachtung anderer Datenpunkte des gleichen Probanden, wäre man Gefahr gelaufen, den Begriff “Konstrukt” als Schleifenkonstrukt,

Funktionsbezeichner oder ähnliches fehlzudeuten. Schleifenkonstrukte sind in der Tat eine mögliche, alternative Deutung, denn in SeqAn werden in Schleifen Iteratoren verwendet. Da Iteratoren wiederum mit Hilfe von Metafunktionen konstruiert werden, werden auch die Schleifenkonstrukte schwerer verständlich. Derartige Fehlinterpretationen können jedoch weitgehend ausgeschlossen werden, denn häufig benennen Probanden Iteratoren auch als solche<sup>88</sup>. Im konkreten Beispiel kommt hinzu, dass der Proband bereits seit vier Jahren SeqAn nutzt<sup>89</sup> und Iteratoren zu den Pflicht-Tutorials eines jeden SeqAn-Anfängers gehören<sup>90</sup>.

## 2. Wurde ein Konzept bzw. Relation weit unterschätzt?

Die Wahrscheinlichkeit einer Unterschätzung wurde durch die weiter oben beschriebene Triangulation verringert. Es ist anzunehmen, dass wichtige Konzepte bzw. Relationen in meiner breit angelegten Datenerhebung (siehe Abschnitt 3.3) häufiger oder intensiver vertreten und damit weniger leicht zu unterschätzen sind.

Außerdem habe ich die Daten sehr genau (siehe obige Beispiele) analysiert. Dies verringerte ebenfalls die Wahrscheinlichkeit, dass ich wichtige Konzepte bzw. Relationen übersehen habe.

Des Weiteren erhebe ich keinen Anspruch auf Vollständigkeit. Es liegt in der Natur der Sache, dass ein anderer Forscher Problemarten gefunden hätte, für die ich blind war oder denen ich weniger Aufmerksamkeit zukommen ließ. Zum Beispiel habe ich dem Usability-Problem ○ Fehleranfälliger Gebrauch von `const` relativ früh wenig Aufmerksamkeit beigemessen, weil die ersten Phänomene mir klar zeigten, dass es sich um ein reines C++-Anfängerproblem handelte. Möglicherweise wäre ich eines besseren belehrt worden, hätte ich dieses Konzept intensiver beforscht.

Ähnlich verhält es sich mit der Relation ○ Tutorials → erleichtern Einstieg, die eine Folge der im Abschnitt 3.2 beschriebenen Usability-Verbesserung im Rahmen der ersten Beseitigung grober Usability-Probleme darstellt und darüber hinaus bereits im Abschnitt 3.2.5 validiert wurde.

## 3. Wurde ein Konzept bzw. Relation weit überschätzt?

Diese Frage kann mit der Tatsache beantwortet werden, dass ich die SeqAn-Entwickler zur Umsetzung vieler der auf meinen Ergebnissen basierenden Usability-Verbesserungsmaßnahmen bewegen konnte.

In Abschnitt 4.3 habe ich Verbesserungsmöglichkeiten vorgeschlagen. Den erwarteten Einfluss auf den Grad der Verbesserung habe ich wiederum auf Grundlage meiner Analyseergebnisse geschätzt. Von den vier wertvollsten Maßnahmen, wurden drei Maßnahmen umgesetzt — nämlich (1) die Verwendung von KNIME als API-Endanwender-Werkzeug, (2) den Umbau von SeqAn von einem Framework zu einer Library und (3) die Überarbeitung der SeqAn-Dokumentation (siehe Abschnitt 4.4). Abgesehen von der technischen Neuentwicklung der Dokumentation, wurde ein beachtlicher Teil der Arbeit durch die SeqAn-Entwickler gestemmt. Dies zeigt, dass die SeqAn-Entwickler meine Einschätzung der wichtigen gefundenen Konzepte und Relationen teilen.

Interessanter ist die vierte wichtige Maßnahme, die in der Angleichung an die STL besteht: Ich habe das für die Umsetzung der Maßnahme notwendige CRTP einigen SeqAn-Entwicklern zu einer Zeit präsentiert, in der ich noch in dem Glauben war, SeqAns Anwender würden eine streng

objektorientierte Library erwarten. Dies entsprach zum Einen nicht den Tatsachen, denn ein großer Teil erwartete lediglich eine STL-konforme Softwarebibliothek (siehe ● Paradigmatische Prägung, Seite 251). Zum Anderen stieß ich mit diesem Vorschlag auf Ablehnung, denn mit Ausnahme des übrigen SeqAn-Entwicklers Andreas Gogol-Döring, der begeistert von meinem Vorschlag war (Gogol-Döring u. Kahlert 2013, siehe Abschnitt 4.3.4.1), scheut die damaligen SeqAn-Entwickler verständlicherweise die damit verbundenen Kosten, welche sie zusätzlich zu tragen hätten. Aus zeitlichen Gründen kam ich nicht mehr dazu, dem SeqAn-Team meinen neuen Erkenntnisstand mitzuteilen. An dem Nutzen dieser Maßnahme besteht indes kein Zweifel, denn die teils hitzigen Wortmeldungen in den unterschiedlichen Datenquellen (“Vergewaltigung der OO-Programmierung”<sup>18</sup>, “[It’s] not clear why you have to go through all this pain”<sup>19</sup>) sprechen eine eindeutige Sprache (siehe ● Inkonsistenzen bzgl. STL, Seite 264).

Nicht alle Usability-Probleme wurden gelöst, da nicht alle vorgeschlagenen Maßnahmen umgesetzt wurden:

- Konzepte, die ich nicht hinreichend gut verstanden habe (z.B. ● Funktionsrückgabenmodifikation, ● Operatoreninkonsistenz und ● Abstraktionssuggestion, siehe Abschnitt 4.4.5), wurden auch als solche benannt und werden im Ausblick thematisiert. Hier besteht also keine Gefahr einer Überschätzung.
- Die übrigen Konzepte und Relationen wurden entweder nur durch einen Teil der SeqAn-Entwickler oder von keinem der SeqAn-Entwickler ähnlich eingeschätzt. Zu Ersteren gehört die ● Versagensverschleppung. Zu Letzteren gehört die ● Abstraktionssuggestion. An dieser Stelle der Argumentation kann ich lediglich auf die Plausibilität meiner ab Seite 249 dargestellten GT und auf meine oben angesprochene theoretische Sensibilität verweisen.

Da nun die Frage nach der Validität meiner in dieser Arbeit präsentierten GT geklärt ist, gilt es noch zu untersuchen, ob meine vorgeschlagenen Maßnahmen die entsprechenden Usability-Probleme korrekt adressieren und ob die umgesetzten Maßnahmen auch tatsächlich zu einer Usability-Verbesserung beigetragen haben.

Die von mir formulierten Maßnahmen betreffen weitgehend eine disjunkte Menge von Usability-Problemen. Bei einigen Usability-Problemen (Maßnahmen *Frameworkumbau*, *Fail-Fast*, *Kollaborationsplattform* und *Werkzeugunterstützung*) lag die Lösung auf der Hand. War dies nicht der Fall, habe ich argumentiert, an welcher Stelle die Maßnahme ansetzt und in welchem Umfang welche Usability-Probleme damit gelöst werden (z.B. *Maßnahme STL-Angleichung*).

Ob die praktische Ausgestaltung einer Maßnahme tatsächlich den gewünschten Effekt hatte, diskutiere ich im Folgenden:

- Die *STL-Angleichung* wurde leider noch nicht umgesetzt. Insbesondere die Analyseergebnisse der Gruppendiskussion sind bzgl. der Lösung derart eindeutig, dass kein Zweifel über die inhaltliche Eignung des CRTP besteht. Darüber hinaus wurde das CRTP bereits in einigen Boost-Libraries

und der Microsoft Active Template Library eingesetzt, was technische Bedenken bzgl. der Umsetzbarkeit weitgehend ausräumt.

- Das gleiche gilt für die, durch die ● Entwurfsentscheidung ● Datenstrukturmodifikation verursachten ● Syntaxprobleme, welche durch die Maßnahme *Inkonsistenzbeseitigung* behoben werden sollen.
- Ebenso eindeutig sind die Gruppendiskussionsergebnisse in Bezug auf die Maßnahme *Shortcuts*, welche die, durch die gleichnamige ● Entwurfsentscheidung verursachten Probleme ● Abstraktionssuggestion und ● Synonyme / Redundanz beheben soll.
- Die Entwicklung einer *Wrapper-API* durch die Integration von SeqAn-Knoten in die Workflow-Engine KNIME muss auf verschiedene Weise argumentiert werden.
  1. Die Eignung von grafischen Entwicklungsumgebungen für die Verbesserung der API-Usability für API-Endanwender ist vielfach gezeigt worden (Ko et al. 2011).
  2. Da die SeqAn-Knoten direkt in den KNIME-Mechanismus zum Beziehen von Dritthersteller-Knoten integriert wurden, kommt es an dieser Stelle zu keiner Kompromittierung der Usability.
  3. Allerdings kommt es durch die Einbindung von Kommandozeilenprogrammen in KNIME zu einem Paradigmenwechsel. KNIME transferiert zwischen den datenverarbeitenden Knoten Daten in Form von Tabellen. SeqAn-Anwendungen verwenden für die Ein- und Ausgabe hingegen Dateien. Um eine ideale Integration zwischen gewöhnlichen KNIME- und SeqAn-Knoten zu ermöglichen, wurden Konverterknoten implementiert, welche Tabellen zu Dateien und anders herum konvertieren können. Ob und welche Anwendungsschwierigkeiten sich daraus ergeben, habe ich nicht evaluiert.
  4. Nicht zuletzt hängt die Usability der Wrapper-API von der Usability von KNIME selbst ab, das ebenfalls nicht Gegenstand einer Evaluation war. Die weite Verbreitung von KNIME lässt zumindest darauf schließen, dass es sich hierbei nicht um ein K.O.-Kriterium<sup>58</sup> handelt.
- Die Validierung der Maßnahme *Intransparenzbeseitigung* kann wegen unzureichender empirischer Erkenntnisse nicht geleistet werden. Dazu muss das Usability-Problem ● versteckte Parameterübergabe besser erforscht werden.
- Die Dokumentation wurde umfangreich verbessert (siehe Abschnitt 4.4.8):
  1. Die Umstellung des Dokumentationssystems wurde von den SeqAn-Entwicklern als wichtig und sinnvoll erachtet, was sich auch an ihrer mehrwöchigen, inhaltlichen Mithilfe zeigte.
  2. Auch der Entwicklermodus wurde positiv von den SeqAn-Entwicklern angenommen. Dieser wird regelmäßig genutzt, um Dokumentationseinträge stärker zu verlinken.

---

<sup>58</sup> Wissenschaftlich korrekt ausgedrückt, handelt es sich hierbei um die vierte boolesche Eigenschaft *Markteinfluss*, die von Sarodnick u. Brau (2006) zur Bewertung der Fatalität eines Usability-Problems herangezogen wird. Diese Eigenschaft ergänzt die ursprünglichen drei von Nielsen u. Mack (1994) formulierten Eigenschaften.

3. Die positiven Auswirkungen eines Gesamtüberblicks und von Beispielen sind bereits vielfach in der Literatur beschrieben worden (siehe Abschnitt 2.6). Es ist daher naheliegend, dass der neu geschaffene Gesamtüberblick und die ergänzten bzw. korrigierten Beispiele (siehe jeweils Abschnitt 4.4.8) zu einer signifikante Verbesserung der API-Usability beigetragen haben.
4. Die Verbesserung der Suchfunktion ist eine direkte Konsequenz aus den von den Anwendern geäußerten Kritikpunkten. Die Unterstützung von Aliassen und einer gewichteten Suche (siehe Abschnitt 4.4.8) lassen den Schluss auf eine Usability-Verbesserung zu.
5. ● Sprachentitätstypen werden in keiner mir bekannten C++-Library, die auf Templatemetaprogrammierung basiert, explizit zu Dokumentationszwecken eingesetzt. Dafür gibt es zwei mögliche Gründe.
  - Entweder richten sich derartige Libraries an fortgeschrittene Entwickler<sup>59</sup>, was den Erklärungsbedarf anspruchsvoller Sprachentitätstypen deutlich verringert.
  - Oder aber entsprechende Libraries verbergen für Anfänger ungeeignete Sprachentitätstypen beispielsweise durch den Einsatz von CRTP<sup>60</sup>.

SqAn wendet sich sowohl an API-Anwendern als auch an API-Endanwender und damit auch an weniger erfahrene Entwickler. Meine Forschungsergebnisse haben gezeigt, dass das fehlende Verständnis neuartiger Sprachentitätstypen wie Interface-Metafunktionen zu Usability-Problemen führt. Daher besteht für mich kein Zweifel daran, dass die explizite Einführung dieser Sprachentitätstypen zu einer Usability-Verbesserung beiträgt.

- Den positiven Effekt von Verbesserungen aus der ersten Beseitigung grober Usability-Probleme habe ich — insbesondere für die überarbeiteten Installationsanleitungen und Tutorials — bereits im Abschnitt 3.2.5 argumentativ und empirisch gezeigt.

Es ist davon auszugehen, dass der Umfang der Überarbeitung der SqAn-API eine erneute Validierung sinnvoll macht. Es ist nicht auszuschließen, dass das Zusammenspiel der Maßnahmen zu neuen emergenten Usability-Problemen geführt hat. Diese Vermutung wird durch die Aussage von Stylos (2009) gestützt, dass bereits kleine API-Anpassungen große Auswirkungen auf die Usability haben können.

### 4.5.3 Verallgemeinerbarkeit

Es lassen sich sowohl methodische, softwaretechnische und Usability-Problem-bezogene Erkenntnisse verallgemeinern.

---

59 Typische Vertreter sind die Boost-Libraries wie BCCL ([http://www.boost.org/doc/libs/1\\_58\\_0/libs/concept\\_check/concept\\_check.htm](http://www.boost.org/doc/libs/1_58_0/libs/concept_check/concept_check.htm)) oder CRC([http://www.boost.org/doc/libs/1\\_58\\_0/libs/crc/](http://www.boost.org/doc/libs/1_58_0/libs/crc/)).

60 Zu den bekanntesten Vertretern gehört die Microsoft Active Template Library (<https://msdn.microsoft.com/en-us/library/4e07a759.aspx>).

#### 4.5.3.1 Methodische Erkenntnisse

Meine Erkenntnisse und die Generizität meiner Methode belegen, dass sich meine Forschungsmethode sehr gut für die Erforschung *untypischer APIs* eignet. Die Generizität ergibt sich einerseits aus der breit angelegten Datenerhebung, und andererseits aus dem extrem offenen Forschungsansatz der GTM.

Insbesondere die Programmierschritte-Datenerhebung eignet sich auch unter Rahmenbedingungen, bei denen es nur wenige Möglichkeiten zur Datenerhebung gibt, weil besonders viele Daten automatisiert erhoben werden und praktisch keine Beeinflussung der Probanden stattfindet (siehe Abschnitt 3.3.6). Aus Zeitgründen konnte ich leider nur wenige Erkenntnissen aus dieser Datenquelle ziehen, weshalb ich letztere Aussage kaum empirisch belegen kann.

Die sich aus meiner Forschungsmethode ergebende Erkenntnisfülle hat aber auch eben diesen Preis: den hohen Zeitaufwand. Durch die Erhebung sowohl subjektiver, als auch objektiver Daten, verbunden mit der leichtgewichtigen HE<sup>G</sup> und der schwergewichtigen GTM, kann der Forscher nicht mit schnellen Ergebnissen rechnen. Außerdem benötigt er für die Programmierfortschritte-Daten ein geeignetes Datenanalysewerkzeug, wie der im Abschnitt 3.4 vorgestellte APIUA<sup>G</sup>.

APIUA ist aktuell zwar explizit auf meine Forschung zugeschnitten, erlaubt durch seine komponentenbasierte Architektur aber eine einfache Erweiterung um die Unterstützung neuer Datenformate. Jedoch kann ich nicht verschweigen, dass der praktischen Nutzung noch die relativ geringe Stabilität im Wege steht. Der für die Datenerhebung notwendige Webclient kann ohne Anpassungen für andere Forschungsvorhaben eingesetzt werden. Der lokal arbeitende Client hingegen muss in den lokalen Build-Prozess integriert werden, was im Falle des von SeqAn eingesetzten CMake problemlos möglich war, anderenfalls aber aufwändiger sein könnte.

Die Eignung meiner Forschungsmethode für meine durchgeführte Forschung beweisen meine Ergebnisse. Die Eignung für andere Forschungsvorhaben habe ich soeben argumentiert. Ob dies tatsächlich zutrifft, müsste durch die Durchführung einer Vielzahl unterschiedlicher Forschungsvorhaben mit meiner Methode belegt werden. Zumindest kann ich zu meiner Verteidigung ins Feld führen, dass der empirische Beleg von keiner der in der Literatur beschriebenen komplexen API-Usability-Evaluationsmethoden erbracht wurde (siehe Abschnitt 3.3).

Die folgenden zwei Unterabschnitte sind gegliedert in methodische und problembezogene Erkenntnisse. Methodische Erkenntnisse umfassen Einsichten, die sich auf den Entwicklungsprozess von APIs beziehen. Problembezogene Erkenntnisse hingegen behandeln die Frage, mit welchen Usability-Problemen API-Entwickler bei der Entwicklung rechnen müssen.

#### 4.5.3.2 Softwaretechnische Erkenntnisse

1. Die Entwicklung von SeqAn hat gezeigt, dass nicht-empirische Entwurfsentscheidungen nicht zwingend zum Erreichen der erdachten Ziele führen. Ist es *tatsächlich* so, dass ein Entwurfsziel primär und die anderen sekundär sind, deuten meine Forschungsergebnisse darauf hin, dass

die sekundären Entwurfsentscheidungen nicht gründlich genug umgesetzt werden, im Zweifel unterliegen und zu schweren bis katastrophalen Problemen führen können.

Im Falle von SeqAn stellt Performance das primäre Ziel und Usability de facto das sekundäre Ziel dar (siehe Abschnitt 4.1).

2. Ein Prioritätenwechsel, wie dies bei SeqAn die neue Betonung der Usability war (siehe Abschnitt 4.2.3), zeigt, dass für tolerierbar gehaltene oder gar nicht wahrgenommene Usability-Probleme plötzlich eine große Wichtigkeit bekommen können. Die nachträgliche Diagnose und Behebung solcher Usability-Probleme kann sehr aufwändig sein. Meine Arbeitszeit erstreckte sich über knapp vier Jahre. Hinzu kamen die unzähligen Personenmonate, die meine SeqAn-Kollegen aufbrachten.
3. Bei der Entwicklung eines Produkts ist ein gründliches und gemeinsames Verständnis über die (zukünftigen) Anwender existenziell wichtig. Dies ist bereits in der Literatur bekannt (u.a. Clarke 2004; Henning 2007).

Im SeqAn-Team lag nur ein unzureichendes Verständnis von den Anwendern vor, worin eine der Hauptursachen für die entdeckten Probleme bestand. Des Weiteren gab es Uneinigkeit in der Frage, ob Entwickler mit wenig Programmiererfahrung zur Anwenderschaft gehören. Dies führte dazu, dass bekannte Anwendungsschwierigkeiten nicht als zu behebende bzw. zu vermeidende Usability-Probleme eingestuft wurden. Meine Beobachtungen bestätigen also die Notwendigkeit, eine fundierte und gemeine Vorstellung von den Zielgruppen zu haben, wenn Usability-Probleme vermieden werden sollen.

#### 4.5.3.3 Usability-Problem-bezogene Erkenntnisse

Bis auf die beiden weitgehend SeqAn- bzw. Gogol-Döring-spezifischen Probleme ● Projektorganisation und ● Synonyme / Redundanz, lassen sich die übrigen von mir entdeckten Probleme unterschiedlich stark verallgemeinern. Die Gründe der Verallgemeinerung können den Usability-Problem-Beschreibungen im Abschnitt 4.1 entnommen werden.

Da ich mich mit einer C++-basierten Library beschäftigt habe, gelten die im Folgenden genannten Usability-Probleme möglicherweise nicht bei Programmiersprachen, die nicht über bestimmte Sprachfeatures, wie die Überladbarkeit von Operatoren, verfügen. Andere Probleme lassen sich hingegen leicht auf die eigene Programmiersprache übertragen. So würde das Problem ● Inkonsistenzen bzgl. STL im Falle von Java beispielsweise *Inkonsistenzen bzgl. JDK* lauten, denn der Kern dieses Problems ist nicht die STL, sondern die Enttäuschung der sich aus der ● paradigmatischen Prägung ergebenden Erwartungen.

##### Allgemeine Probleme

Probleme, die in jeder API auftreten können.

- Strukturprobleme

- Technisch vs. Fachlich, ● Ununterscheidbarkeit von Typen, ● Abstraktionssuggestion,
- Benennungskonsistenz
- Dokumentationsprobleme
  - Fehlende Anwendungsbeispiele
- Laufzeitprobleme
  - Versagensverschleppung

### Probleme der Templatemetaprogrammierung

Probleme, die in APIs auftreten können, die Templatemetaprogrammierung einsetzen.

#### Inhärente Probleme

Probleme, mit deren Auftreten bei der Verwendung von Templatemetaprogrammierung gerechnet werden muss und die proaktiv ausgeschlossen werden müssen.

- Strukturprobleme
  - Inkonsistenzen bzgl. OOP, ● Inkonsistenzen bzgl. STL, ● Sprachentitätstypen,
  - Funktionszweckunerkennbarkeit, ● Funktionsgebrauch
- Dokumentationsprobleme
  - Fehlende Funktionskategorisierung, ● Identifikation relevanter Funktionen
- Laufzeitprobleme
  - Compiler-Fehlermeldungen
- Werkzeugunterstützungsprobleme
  - Fehlende Autovervollständigung

#### Mögliche Probleme

Probleme, die mit einer gewissen Wahrscheinlichkeit — insbesondere im Hinblick auf generische Programmierung — auftreten.

- Strukturprobleme
  - Funktionsrückgabenmodifikation, ● Operatoreninkonsistenz, ● Versteckte Parameterübergabe
- Dokumentationsprobleme
  - Fehlender Gesamtüberblick, ● Fehlende Dokumentation der Rückgabetypen,
  - Suchfunktion, didaktische Lernressourcen wie ○ Tutorials

Für den Verallgemeinerungsgrad der oben genannten Usability-Probleme habe ich auch die zu erwartenden Fatalitäten der Probleme betrachtet. So stellt ein ● Fehlender Gesamtüberblick wahrscheinlich bei jeder API ein Usability-Problem dar. Jedoch ist es bei generischer Programmierung besonders schwerwiegend, weshalb ich es entsprechend eingeordnet habe.

#### 4.5.4 Zusammenfassung

In diesem Unterkapitel habe ich die Güte, Validität und Verallgemeinerbarkeit meiner Forschung bzw. Forschungsergebnisse besprochen.

Die **Güte** meiner Forschung habe ich belegen können. Durch die Verwendung von URIs habe ich meine Forschung nicht nur inhaltlich, sondern auch technisch nachvollziehbar dokumentiert. Die Regelgeleitetheit habe ich an Hand der GTM und der Dokumentation des Forschungsprozesses dargestellt. Ebenso habe ich die Nähe zum Forschungsgegenstand über die Nähe zum Anwender und den Verzicht auf ein Laborsetting gezeigt. Wenn dies der zeitliche Verlauf zuließ, habe ich Teilergebnisse mit den Anwendern, sowohl direkt, als auch indirekt besprochen. Für die Triangulation habe ich verschiedene Analysemethoden, Datenerhebungen und existierende Literatur genutzt.

Aus Platz- Zeit und Plausibilitätsgründen<sup>61</sup> habe ich alternative Interpretationen nur geringfügig in dieser Arbeit dargestellt. Meine intensive Auseinandersetzung mit der GTM konnte ich insbesondere durch den Bau eines GTM-unterstützenden Datenanalysewerkzeugs unter Beweis stellen. Meine fachliche Eignung habe ich wahrheitsgemäß dargelegt.

Die **Validität** meiner Ergebnisse basiert zu großen Teilen auf meiner gründlichen Anwendung der GTM, was sich auch an dem Bau des dafür geeigneten Datenanalysewerkzeugs zeigt. Die Zusammensetzung der beobachteten Probanden lässt die Annahme zu, dass diese hinreichend repräsentativ für meine qualitative Forschung waren. Für meine vorgeschlagenen Maßnahmen habe ich gezeigt, dass sie sich entweder direkt aus den Analyseergebnissen ergaben, oder ich habe genau beschrieben, weshalb davon auszugehen ist, dass die entsprechenden Usability-Probleme durch meine Maßnahmen gelöst werden. Dies trifft insbesondere auf die noch nicht vollzogene STL-Angleichung zu. Die Wirksamkeit umgesetzter Maßnahmen habe ich bei den Installationsanleitungen und Tutorials empirisch belegen können und für die neue Wrapper-API sowie der neuen Dokumentation ausführlich argumentiert. Wegen der umfangreichen Arbeiten an der SeqAn-API empfiehlt sich eine erneute Evaluation, um insbesondere auch emergente, und für andere Forscher für wichtiger erachtete Usability-Probleme zu entdecken. Dabei wäre es wünschenswert, erfahrene SeqAn-Anwender stärker in den Fokus zu rücken.

Die **Verallgemeinerbarkeit** meiner Forschungsmethode habe ich argumentativ beschrieben. Sie sollte insbesondere dann angewendet werden, wenn eine vergleichsweise untypische API eingesetzt wird, da sich der große Aufwand in solch einem Fall auch rechtfertigen lässt.

Ich habe gezeigt, dass nicht-empirische bzw. sekundäre Entwurfsentscheidungen zu schweren bis katastrophalen Ergebnissen führen können. Bei einem Prioritätenwechsel können zuvor wenig beachtete Usability-Probleme eine Relevanz erhalten, die zu einer extrem kostenintensiven Beseitigung führen kann. Vermieden werden kann dies, wenn Usability von Anfang an einen hohen Stellenwert bekommt und untypische Entwurfsentscheidungen empirisch gesichert sind. Dazu gehört auch ein gründliches Studium der (zukünftigen) Anwenderschaft des eigenen Produkts.

Neben einigen allgemein gültigen Usability-Problemen konnte ich solche finden, die speziell bei auf Templatemetaprogrammierung und generischer Programmierung basierenden APIs auftreten. Besonders relevant sind die Probleme ● Inkonsistenzen bzgl. STL, ● Sprachentitätstypen, ● Dokumentationsprobleme und ● Werkzeugunterstützungsprobleme.

---

61 Die Plausibilität ergibt sich über die Integrität meiner GT und der soeben beschriebenen Triangulation.



## KAPITEL

# 5

## FAZIT

### 5.1 Ausgangslage

Der Ausgangspunkt dieser Arbeit war die für notwendig erachtete Usability-Verbesserung der API<sup>G</sup> der bioinformatischen Softwarebibliothek SeqAn. Der spezielle Entwurf von SeqAn, durch den auf SeqAn basierte Anwendungen extrem performant sind, besteht in der Verwendung der C++-Templatemetaprogrammierung. Dieses Verfahren blieb jedoch in der seit etwa dem Jahr 2000 erforschten API-Usability vollkommen unbeleuchtet, wodurch die geplante API-Usability-Verbesserung nicht ohne Weiteres vorgenommen werden konnte.

### 5.2 Zielsetzung

Das Ziel meines Vorhabens bestand also darin, die API-Usability von SeqAn zu erforschen, um mit diesem Wissen SeqAns Usability zu verbessern und verallgemeinerbare Erkenntnisse für andere auf Templatemetaprogrammierung basierenden APIs zu gewinnen. Dabei galt es darauf zu achten, dass die Verbesserungen nicht nur den SeqAn-Anwendern — also Informatikern und Bioinformatikern —

nutzen, sondern eine hohe Usability auch für SeqAn-Endanwender — also Biologen, Medizinern und anderen Lebenswissenschaftlern — hergestellt wird.

### 5.3 Methode

Für die Erforschung von SeqAns API-Usability habe ich mich für eine explorative, empirische Fallstudie unter Verwendung der GTM<sup>G</sup> nach Strauss u. Corbin (1990) entschlossen (siehe Abschnitte 1.4 und 3.5). Die GTM hat den Vorteil, dass sie kaum inhaltliche Annahmen verlangt.

Innerhalb einiger Wochen wurde deutlich, dass es um SeqAns Usability schlechter bestellt war als angenommen (siehe Abschnitt 3.2). Dies war besonders verwunderlich, wurde man doch in SeqAn-Präsentationen nicht müde, dessen Benutzerfreundlichkeit zu bewerben. Um nicht Gefahr zu laufen, tiefer liegende Usability-Probleme zu übersehen, mussten die offensichtlicheren, groben Probleme zunächst beseitigt werden. Dazu nutzte ich eine Online-Umfrage, Interviews, einen Feedback-Zettel und eine vereinfachte HE<sup>G</sup>. Beseitigt habe ich in Zusammenarbeit mit der Bioinformatik-Arbeitsgruppe in diesem Zuge eine Reihe von OOBE<sup>G</sup> relevanten Problemen, insbesondere durch die Verbesserung der SeqAn-Installationsanleitungen und -Tutorials.

Die speziellen organisatorischen Rahmenbedingungen (siehe Abschnitt 3.1.1) erforderten in Verbindung mit der GTM, eine Datenerhebung, bei der die SeqAn-Anwender im Zentrum der Betrachtung stehen, so wenig wie möglich beeinflusst werden und sich besonders reichhaltige Daten ergeben. Dazu entwickelte ich ein kombiniertes Datenerhebungsverfahren, bei dem sowohl subjektive, als auch objektive Daten erhoben werden (siehe Abschnitt 3.3). Für die subjektiven Daten kamen eine Gruppendiskussion, in der bis dato erarbeitete Ergebnisse thematisiert wurden, und ein eigens entwickelter Fragebogen, der auf dem Cognitive Dimensions Framework basiert, zum Einsatz. Für die Erhebung der objektiven Daten entwickelte ich ein leistungsfähiges, automatisiertes, plattformübergreifendes und weitgehend transparent arbeitendes Verfahren, das seinen angestrebten Zweck sehr gut erfüllte.

Für die integrierte, GTM-basierte Analyse dieser unterschiedlichen und teils hochstrukturierten Daten entwickelte ich das qualitative Datenanalysewerkzeug *APIUA*<sup>G</sup>. Der im Abschnitt 3.4 beschriebene APIUA war ursprünglich als reine Datenvisualisierungslösung geplant, entwickelte sich aber schließlich zu einer Anwendung, die eine Reihe von Funktionen bietet, die selbst beim “Platzhirsch” *ATLAS.ti* nicht zu finden sind, aber für meine Forschung unerlässlich waren.

Meine Forschung war durch einen hochdynamischen Prozess gekennzeichnet, bei dem sich die Datenerhebung und -analyse, der Werkzeugbau und das Erlernen der GTM gegenseitig beeinflussten. Erschwert wurde mein Vorhaben durch die stetige Weiterentwicklung von SeqAn und der Weiterentwicklung der Programmiersprache C++. Nach meiner Kenntnis ist das Zustandekommen der Ergebnisse mittels selektiven Kodierens und die Präsentation der im anschließenden Abschnitt zusammengefassten Ergebnisse in Form einer GT<sup>G</sup> einmalig in der API-Usability-Forschung. Zwar konnte ich meine Ergebnisse nur teilweise empirisch validieren, jedoch habe ich sehr großen Wert auf die Verfahrensdokumentation gelegt. Durch die eindeutige Identifikation sämtlicher von mir präsentierter Konzepte

und Zitate mit Hilfe von APIUA-URIs kann meine Argumentation nicht nur inhaltlich, sondern auch technisch lückenlos nachvollzogen werden. Dieser Grad der Transparenz erfüllt die Anforderungen an die Veröffentlichung und Zugänglichkeit von Primärdaten gemäß *Open Science*.

## 5.4 Ergebnisse

Die Ergebnisse dieser Arbeit umfassen

- eine Theorie über die Entstehung und Auswirkungen von Entwurfsentscheidungen in SeqAn,
- Verbesserungen der API-Usability von SeqAn,
- das Datenanalysewerkzeug API Usability Analyzer,
- eine wissenschaftliche Literaturstudie zu API-Usability und
- verallgemeinerbare wissenschaftliche Erkenntnisse

und werden im Folgenden erläutert.

### 5.4.1 Theorie über die Entstehung und Auswirkungen von Entwurfsentscheidungen in SeqAn

Bei meinem im Abschnitt 4.1 vorgestellten Ergebnis handelt es sich um eine GT. Sie beginnt mit der Entwicklung von SeqAn durch Andreas Gogol-Döring (Gogol-Döring 2009), der das ultimative ● Entwurfsentscheidungsziel *Performance* verfolgte, welches er durch die *explizite-empirische* ● architektonische Entwurfsentscheidung ● Templatemetaprogrammierung nachweislich erreichte.

Seine übrigen Entwurfsentscheidungen dienten, zusammenfassend betrachtet, der *Usability*. Im Unterschied zur Templatemetaprogrammierung basierten ● Generische Programmierung und ● Template-Spezialisierung auf einer *expliziten-argumentativen*, und ● Shortcuts sowie ● Datenstrukturmodifikation nur auf einer *expliziten-intuitiven* ● Entwurfsentscheidungsgrundlage. Die ● Frameworkgestalt von und die ● Domänen-spezifische Benennung in SeqAn, wurden von dem Autor nur *implizit* thematisiert.

Geprägt waren diese intuitiven und argumentativen Entwurfsentscheidungen einerseits durch die persönlichen Präferenzen des Autors und andererseits durch die unzureichende Berücksichtigung der Anforderungen zukünftiger ● SeqAn-Anwender. Folglich erweckte der SeqAn-Entwurf bei seinen Anwendern die ● produktbedingte Erwartung einer performanten, objektorientierten und benutzerfreundlichen Library. Jedoch war SeqAn keine Library, sondern ein Framework, verwendete keine objektorientierte Programmierung, sondern Templatemetaprogrammierung und hatte schwere Defizite in Bezug auf seine Usability.

Fasst man alle Ursachen zu einer zusammen, kommt man zu dem Schluss, dass es zwar gelungen ist, SeqAn-basierte Programme extrem schnell zu machen, es jedoch nicht gelungen ist, die übrigen Erwartungen der Anwender — insbesondere bezüglich der ● Wiederverwendbarkeit bestehender

● paradigmatisch geprägter Vorkenntnisse — zu erfüllen, was sich in der Aussage “SeqAn makes me feel stupid”<sup>38</sup> widerspiegelt.

In Bezug auf die Anwenderschaft ergaben sich durch die getroffenen Entwurfsentscheidungen eine Reihe von ● Usability-Problemen, die sich in ● produktbedingte Erwartungskonformitätsprobleme und in ● Struktur-, ● Dokumentations-, ● Laufzeit-, sowie ● Werkzeugunterstützungsprobleme unterteilen lassen.

Trafen Anwender auf Usability-Probleme, so wurden ● Strategien, die auf den anwenderseitigen ● Arbeitsstilen und ● paradigmatischen Prägungen beruhen, häufig behindert. Dabei sind besonders die ● experimentelle Hypothesenüberprüfung, das systematische ● Konzeptverstehen, das pragmatische ● Beispiel-Lernen, die ● Anfangsklassenverwendung und die ● Punkt-Funktionssuche zu nennen.

Andererseits zwangen die bestehenden Usability-Probleme die Anwender dazu, auf andere Strategien zurückzugreifen. Zu diesen Strategien gehören das ● Namennerraten und das opportunistische ● blinde Kopieren.

Die Folgen erstreckten sich auf eine deutlich verringerte ● Erlernbarkeit SeqAns, eine teils ausgebremste, teils gestörte ● Programmentwicklung und die Empfindung auffallend negativer ● Emotionen.

Meine Ergebnisse haben die von allen Stakeholdern geteilte Hypothese, dass die Entwurfsentscheidung Templatemetaprogrammierung eine der Hauptursachen für SeqAns schlechte Usability ist, empirisch bestätigt — ohne für alternative Erklärungen verschlossen gewesen zu sein.

Tritt man einen Schritt zurück, stellt man fest, dass Performance kein zur Usability gleichgewichtiges Ziel, sondern das Superziel für SeqAn gewesen ist. Im Unterschied zur empirisch belegten Performance, wurde die Usability lediglich “herbei argumentiert” oder basiert auf Bauchgefühlentscheidungen. Wesentliche Ursache dafür war SeqAns hauptsächlicher Einsatzzweck im unausgesprochenen Kontext der Bioinformatik-Arbeitsgruppe, bei dem es um das Verfassen wissenschaftlicher Abschlussarbeiten und Veröffentlichungen ging. In diesem Zusammenhang spielte die Usability eine untergeordnete Rolle, weil Experten, die den damaligen Anwendern helfen konnten, jederzeit greifbar waren. Erst die Ausweitung des Einsatzzwecks auf kommerzielle Anwendungen im Rahmen des durch das BMBF geförderte VIP-Projekt (siehe Abschnitt 3.1.1) bescherte der Usability ein neues Gewicht — und dem SeqAn-Team eine aufwändige Nachkorrekturphase.

Ein weiterer Grund für SeqAns Entwurf lag in den unterschiedlichen Vorstellungen der SeqAn-Entwickler von SeqAns Anwenderschaft begründet, was sich besonders in der von den SeqAn-Entwicklern unterschiedlich beantworteten Frage widerspiegelt, ob API-Endanwender zu dieser Gruppe gehören, oder nicht. Ungeachtet dessen, konnte ich feststellen, dass das Bewusstsein für die Wichtigkeit einer hohen Usability nicht bei jedem Beteiligten gleich ausgeprägt ist. Dies zeigt sich speziell am Phänomen des *technischen Wegargumentierens*.

### 5.4.2 Verbesserungen der API-Usability von SeqAn

Für die Verbesserung der API-Usability von SeqAn haben die Bioinformatik-Arbeitsgruppe und ich umfangreiche Maßnahmen umgesetzt.

Um meine Forschungsergebnisse nicht durch grobe Usability-Probleme zu gefährden, wurden in einer ersten Verbesserungsphase (siehe Abschnitt 3.2) besonders der SeqAn-Entwicklungsprozess überarbeitet, sowie die SeqAn-Installationsanleitungen und -Tutorials umfassend verbessert.

Auf der Grundlage meiner GTM-Forschungsergebnisse habe ich zehn priorisierte Maßnahmen vorgeschlagen, die die Verbesserung der Usability zum Ziel hatten (siehe Abschnitt 4.4). Unter diesen, befinden sich vier Maßnahmen mit besonders hoher Priorität, von denen drei vollständig realisiert wurden (siehe Abschnitt 4.3):

1. Die Dokumentation<sup>1</sup> wurde von mir technisch vollkommen neu entwickelt und gemeinsam mit meinen Kollegen inhaltlich überarbeitet. Dazu wurde das neue Dokumentationsformat Dox entwickelt, ein Gesamtüberblick erarbeitet sowie Seitenaufbau, Beispiele, Suchfunktion und Suchmaschinen-Indexierbarkeit, Darstellung und die Integration in andere Lernressourcen verbessert. Außerdem wurde das Konzept der ● Sprachentitätstypen gesamtheitlich innerhalb der Dokumentation implementiert. Dies soll die ● Erlernbarkeit von SeqAn fördern, indem Templatemetaprogrammierung-bedingte, neue Sprachentitätstypen, wie Interface-Metafunktionen, auch als solche benannt und erklärt werden. Ergänzt wird die Verbesserung der Dokumentation durch die bereits erwähnten generalüberholten Installationsanleitungen und Tutorials.
2. SeqAn war zunächst ein Framework und kann nun auch als Library verwendet werden, indem insbesondere Vorwärtsdeklarationen beseitigt wurden. So simpel dieser Punkt klingen mag, so fundamental war er auch für potentielle SeqAn-Anwender, die ihren bestehenden Build-Prozess nicht anpassen wollten oder konnten.
3. SeqAn wurde so erweitert, dass es mit Hilfe weiterer geschaffener technischer Lösungen in die Workflow-Engine KNIME integriert werden konnte. Damit wurde eine Möglichkeit geschaffen, API-Endanwendern mit minimalen Programmierfähigkeiten die Nutzung von SeqAn zu erlauben.

Außerdem abgeschlossen wurde die Einrichtung einer Kollaborationsplattform für den engen Austausch zwischen SeqAn-Entwicklern und -Anwendern.

Für die vierte wichtige Maßnahme, die in der Angleichung von SeqAn an die STL mittels CRTP besteht, habe ich die Machbarkeit gezeigt. Jedoch konnte diese Maßnahme aus Zeitgründen leider nicht mehr umgesetzt werden. Durch anstehende Erweiterungen des C++-Sprachstandards werden die mit dieser Maßnahme in Verbindung stehenden Usability-Probleme — insbesondere ● Werkzeugunterstützungsprobleme — aber teilweise entschärft. Die übrigen, weniger wichtigen Usability-Verbesserungen konnten, ebenfalls aus Zeitgründen, nicht umgesetzt werden und sind Bestandteil des Ausblicks.

---

<sup>1</sup> <http://docs.seqan.de/seqan/master/>

### 5.4.3 Datenanalysewerkzeug API Usability Analyzer

Die Fülle meiner erhobenen Daten (siehe Abschnitt 3.3) erforderte die Verwendung einer qualitativen Datenanalysesoftware. Jedoch sprach gegen die Verwendung eines der kommerziell verfügbaren Produkte die teilweise hochstrukturierte Gestalt meiner Daten, für die es keine Unterstützung gab. Darüber hinaus haben alle großen qualitativen Datenanalyseprodukte nicht zu vernachlässigende Schwächen in Bezug auf ihren Einsatz als GTM-Software, wodurch ich die Tiefe und Intensität meiner Analyse gefährdet sah (siehe Abschnitt 3.4). Um die durch die GTM ohnehin schon stark geforderte Sorgfalt und Disziplin des Forschers nicht noch mehr durch ein unzureichend geeignetes Datenanalysewerkzeug unnötig zu strapazieren, habe ich ein eigenes Datenanalysewerkzeug namens API Usability Analyzer (APIUA) entwickelt.

APIUA basiert auf der RCP<sup>G</sup>, die wiederum die Grundlage für die bekannte Entwicklungsumgebung Eclipse<sup>G</sup> darstellt. Dadurch kann das Werkzeug allgemein, aber besonders mit Hinblick auf die Unterstützung weiterer Datenformate, beliebig erweitert werden.

Zu den weiteren, im Vergleich zur weitverbreiteten Datenanalysesoftware ATLAS.ti exklusiven Funktionen gehören:

- URI<sup>G</sup>-referenzierbare Rohdaten und Forschungsergebnisse<sup>2</sup>
- hierarchische Kodes
- filterbare Kodes
- automatische Farbkodierung von Kodes
- verallgemeinerbare Relationen
- zusammenfassbare Relationen
- in den Daten verankerbare Relationen
- axiales und selektives Kodieren
- Speicherung und Wiederherstellung von Forschungssitzungen

APIUA ist bei weitem nicht perfekt. Jedoch war dieses Werkzeug für die in dieser Arbeit präsentierte Forschung unverzichtbar. Der Quellcode und Hinweise zur Installation von APIUA befinden sich unter <https://github.com/bkahlert/api-usability-analyzer>.

### 5.4.4 Wissenschaftliche Literaturstudie zu API-Usability

Im Rahmen der Erforschung der API-Usability von SeqAn stellte ich fest, dass es zwar bereits drei Literaturstudien zum Thema API-Usability gibt, sich diese aber ausschließlich auf Teilaspekte der API-Usability oder auf Teilaspekte API-Usability-Evaluation beschränken (siehe Kapitel 2). Eine wirklich holistische Literaturstudie, die Erkenntnisse aus der “klassischen” Usability und andere wichtige Erkenntnisse, wie Programmverständnismodelle, das Cognitive Dimensions Framework oder Personas,

<sup>2</sup> Diese Funktion ermöglichte die Verlinkung von Rohdaten und Forschungsergebnissen in dieser Arbeit (siehe Abschnitt 1.4.5.2).

nicht ignoriert, fehlte bis dato. Ergänzt wird meine Literaturstudie durch die erste mir bekannte Beschreibung und systematische Einordnung einer Vielzahl API relevanter Werkzeuge.

Ich rechne damit, dass meine Arbeit zukünftigen Forschern die Einarbeitung in das Gebiet der API-Usability-Forschung erleichtert.

#### 5.4.5 Verallgemeinerbare wissenschaftliche Erkenntnisse

**Methodische Erkenntnisse** Meine Erkenntnisse und die Generizität meiner Methode legen nahe, dass sich die in dieser Arbeit vorgestellte Forschungsmethode (siehe Abschnitte 3.3 und 3.5) sehr gut für die Erforschung *untypischer APIs* eignet. Die Generizität ergibt sich einerseits aus der breit angelegten Datenerhebung, und andererseits aus dem extrem offenen Forschungsansatz der GTM (siehe Abschnitt 4.5).

Meine Forschung bietet Grund zur Annahme, dass eine gut vorbereitete Gruppendiskussion nicht nur der Orientierung, sondern auch der Vertiefung bisher erworbener Erkenntnisse dienlich ist.

Hingegen kann ich Forschern nur raten, den Einsatz des Cognitive Dimension Framework wohl zu überlegen und in jedem Fall vorher für den eigenen Einsatzzweck zu testen. Den Einsatz des CDF-Fragebogens zu Explorationszwecken kann ich hingegen empfehlen. Der von mir entwickelte Fragebogen eignet sich für die Exploration von APIs.

Programmierfortschritte-Daten scheinen eine extrem wertvolle Datenquelle zu sein. Deren Analyse erfordert allerdings einen hohen Grad an theoretischer Sensibilität und ein geeignetes Datenanalysewerkzeug, wie APIUA.

Ein erfolgsversprechender Ansatzpunkt für die Evaluation von APIs ist die Berücksichtigung der anwendерseitigen ● paradigmatischen Prägung und der API-seitigen ● Sprachentitätstypen.

#### Softwaretechnische Erkenntnisse

1. Die Entwicklung von SeqAn hat gezeigt, dass nicht-empirische Entwurfsentscheidungen nicht zwingend in das Erreichen erdachter Ziele resultiert. Ist es *tatsächlich* so, dass ein Entwurfsziel primär und die anderen sekundär sind, deuten meine Forschungsergebnisse darauf hin, dass die sekundären Entwurfsentscheidungen nicht gründlich genug umgesetzt werden, im Zweifel unterliegen und zu schweren bis katastrophalen Problemen führen können.

Im Falle von SeqAn stellte Performance das primäre Ziel und Usability de facto das sekundäre Ziel dar (siehe Abschnitt 4.1).

2. Ein Prioritätenwechsel, wie bei SeqAn die neue Betonung der Usability (siehe Abschnitt 4.2.3), kann zur Folge haben, dass für tolerierbar gehaltene oder gar nicht wahrgenommene Usability-Probleme plötzlich eine große Wichtigkeit bekommen. Die nachträgliche Diagnose und Behebung solcher Usability-Probleme kann sehr aufwändig sein. Meine Arbeitszeit erstreckte sich über knapp vier Jahre. Hinzu kamen die unzähligen Personenmonate, die meine SeqAn-Kollegen aufbrachten.

3. Bei der Entwicklung eines Produkts ist ein gründliches und gemeinsames Verständnis über die (zukünftige) Anwenderschaft existenziell wichtig. Dies ist bereits in der Literatur bekannt (u.a. Clarke 2004; Henning 2007).

Im SeqAn-Team lag nur ein unzureichendes Verständnis von den Anwendern vor, worin eine der Hauptursachen für die entdeckten Probleme bestand. Des Weiteren gab es Uneinigkeit in der Frage, ob Entwickler mit wenig Programmiererfahrung zur Anwenderschaft gehören. Dies führte dazu, dass bekannte Anwendungsschwierigkeiten nicht als zu behebende bzw. zu vermeidende Usability-Probleme eingestuft wurden. Meine Beobachtungen bestätigen also die Notwendigkeit, eine fundierte und gemeine Vorstellung von den Zielgruppen zu haben, wenn Usability-Probleme vermieden werden sollen.

**Usability-Problem-bezogene Erkenntnisse** Meine Forschungsergebnisse zeigen, dass Template-metaprogrammierung und generische Programmierung verwendende APIs prädestiniert für das Auftreten bestimmter Usability-Probleme sind. Diese Probleme bestehen darin, dass derartige APIs Dinge anders lösen, als es die API-Anwender erwarten. Werden dabei Sprachentitätstypen benutzt, welche die Anwender nicht kennen, wird das Problem verschärft. Eine weitere Eskalation tritt dann auf, wenn bestimmte API-Konzepte nicht Bestandteil der verwendeten Programmiersprache sind und simuliert werden müssen. Dann liegt wahrscheinlich auch keine Unterstützung dieser Konzepte durch das Entwicklungswerkzeug vor, was eine weitere Problemquelle darstellt. Außerdem erhöht generische Programmierung die Wahrscheinlichkeit, dass syntaktische Konstrukte verwendet werden, die dem Anwender unbekannt sind.

All diese Probleme treten mit hoher Wahrscheinlichkeit auf, wenn sie nicht proaktiv von den API-Entwicklern adressiert werden.

Der Großteil der von mir entdeckten Probleme lässt sich unterschiedlich stark verallgemeinern:

### Allgemeine Probleme

Probleme, die in jeder API auftreten können.

- Strukturprobleme
  - Technisch vs. Fachlich, ● Ununterscheidbarkeit von Typen,
  - Abstraktionssuggestion, ● Benennungsinkonsistenz
- Dokumentationsprobleme
  - Fehlende Anwendungsbeispiele
- Laufzeitprobleme
  - Versagensverschleppung

### Probleme der Templatemetaprogrammierung

Probleme, die in APIs auftreten können, welche Templatemetaprogrammierung einsetzen.

### Inhärente Probleme

Probleme, mit deren Auftreten bei der Verwendung von Templatemetaprogrammierung gerechnet werden muss und die proaktiv ausgeschlossen werden müssen.

- Strukturprobleme<sup>3</sup>
  - Inkonsistenzen bzgl. OOP, ● Inkonsistenzen bzgl. STL, ● Sprachentitätstypen,
  - Funktionszweckunkernbarkeit, ● Funktionsgebrauch
- Dokumentationsprobleme
  - Fehlende Funktionskategorisierung, ● Identifikation relevanter Funktionen
- Laufzeitprobleme
  - Compiler-Fehlermeldungen
- Werkzeugunterstützungsprobleme
  - Fehlende Auto vervollständigung

### Mögliche Probleme

Probleme, die mit einer gewissen Wahrscheinlichkeit — insbesondere im Hinblick auf generische Programmierung — auftreten.

- Strukturprobleme
  - Funktionsrückgabenmodifikation, ● Operatoreninkonsistenz, ● Versteckte Parameterübergabe
- Dokumentationsprobleme
  - Fehlender Gesamtüberblick, ● Fehlende Dokumentation der Rückgabetypen,
  - Suchfunktion, didaktische Lernressourcen wie ○ Tutorials

## 5.5 Konklusion

Ausgehend von dem Zustand, den SeqAn in Bezug auf seine Usability hatte, bestand das Ziel dieser Arbeit in der Verbesserung dieser Usability. Dieses Ziel wurde erreicht, indem ich — mittels einer umfassenden Datenerhebung und einem eigens für die GTM-basierte Datenanalyse entwickelten Analysewerkzeug — eine GT über die Entstehung und Auswirkungen von Entwurfsentscheidungen in SeqAn entwickelte. Diese bildete wiederum die Grundlage für die Formulierung von Usability-Verbesserungs-Vorschlägen, die ich erfolgreich gemeinsam mit meinen SeqAn-Kollegen zu einem großen Teil umgesetzt habe. Im Zuge dieser Arbeit entstand außerdem eine breite und umfassende wissenschaftliche Literaturstudie. Zu guter Letzt konnte ich einen Beitrag zur wenig erforschten API-Usability leisten, die für die Entwickler anderer APIs, insbesondere auf Templatemetaprogrammierung-basierender APIs von Wert sind. Die Anzahl an Arbeiten mit vergleichbar konkreten Ergebnissen für den Entwurf von APIs ist überschaubar<sup>4</sup>.

---

3 Da ich mich mit einer C++-basierten Library beschäftigt habe, gelten einige Usability-Probleme möglicherweise nicht bei Programmiersprachen, die nicht über bestimmte Sprachfeatures, wie die Überladbarkeit von Operatoren, verfügen. Andere Probleme lassen sich hingegen leicht auf die eigene Programmiersprache übertragen. So würde das Problem ● Inkonsistenzen bzgl. STL im Falle von Java beispielsweise *Inkonsistenzen bzgl. JDK* lauten, denn der Kern dieses Problems ist nicht die STL, sondern die Enttäuschung der sich aus der ● paradigmatischen Prägung ergebenden Erwartungen.

4 Im Abschnitt 2.6 nenne ich einige. Ganz besonders hervorzuheben sind die Arbeiten von Jeffrey Stylos (Ellis et al. 2007; Stylos 2009; Stylos u. Clarke 2007; Stylos u. Myers 2008).

Dennoch gibt es einiges zu tun. Das nächste und zugleich letzte Kapitel dieser Arbeit befasst sich mit Empfehlungen, welche die API-Usability-Forschung weiter voranbringen und die softwaregestützte qualitative Forschung mittels GTM inspirieren soll. Zugleich ist die Usability-Verbesserung von SeqAn noch nicht abgeschlossen. Abgesehen von den noch nicht umgesetzten Usability-Verbesserungs-Maßnahmen, gilt es noch eine Einsicht im SeqAn-Team zu etablieren: Die Ausweitung des Einsatzgebietes von SeqAn vom akademischen auf den kommerziellen Bereich erfordert zukünftig eine zur Performance gleichwertige Priorisierung der Usability, um erneute hohe Diagnose- und Behebungskosten zu vermeiden und damit den Erfolg von SeqAn zu sichern. Zu dieser Einsicht soll diese Dissertation beitragen.

## KAPITEL

# 6

## AUSBLICK

Dieses abschließende Kapitel greift die im vorangegangen Kapitel zusammengefassten Ergebnisse meiner Forschung auf und stellt Vorschläge für mögliche weitere Forschungsvorhaben vor.

### 6.1 API-Usability

In diesem Abschnitt unterbreite ich Forschungsfragen, die sich auf das Thema API-Usability beziehen. Genauer: Fragen in Bezug auf API-Usability-Probleme und die API-Usability-Evaluation.

#### 6.1.1 API-Usability-Probleme

Das Usability-Problem ● Versteckte Parameterübergabe befasst sich mit der Übergabe von Parametern zwischen Funktionsaufrufen, ohne dass der Anwender diese Parameterübergabe ohne Weiteres sehen kann. Meine Forschung ergab, dass es sich wahrscheinlich um ein relevantes Problem handelt, ohne dies empirisch belastbar — abgesehen von den Schilderungen eines betroffenen Anwenders — belegen zu können. Dieses Problem zu erforschen erscheint mir außerordentlich sinnvoll, da es in jeder auf imperativer Programmierung basierenden API auftreten kann.

In Bezug auf die API-Usability ist das Konzept der ● Persistenz interessant. Es beschreibt, inwiefern Probleme, auch bei wiederholtem Auftreten, Schwierigkeiten beim Anwender verursachen. Persistente Probleme haben also eine höhere Fatalität als weniger persistente Probleme. Zur Erforschung der Persistenz bieten sich Langzeitbeobachtungen an. Die Persistenz der folgenden Probleme erachte ich für besonders interessant:

- Operatoreninkonsistenz
- Versteckte Parameterübergabe
- Benennungsprobleme
- Typing

Für eine Langzeitbeobachtung, welche ich bereits technisch realisiert habe, muss sichergestellt werden, dass die betrachteten Usability-Probleme nicht durch grobe Usability-Probleme überschattet werden. In Bezug auf SeqAn empfiehlt es sich daher, vorher die Maßnahme STL-Angleichung umzusetzen.

Als besonders spannend haben sich ● Sprachentitätstypen und damit verbundene Strategien, wie die ● Metafunktions-Heuristik, herausgestellt. Nach meiner Überzeugung, stellen sie ein Schlüsselkonzept für die Usability exotischer APIs dar. Für auf Templatemetaprogrammierung und generischer Programmierung basierender APIs scheint dies zu gelten. Es wäre aufschlussreich zu erfahren, ob diese sich auch zur Erklärung der Usability von APIs eignen, die auf anderen Entwurfsentscheidungen basieren.

Clarke (2007); Stylos u. Clarke (2007) haben drei Personas auf der Grundlage der Eigenschaft ● Arbeitsstil vorgestellt. Ich konnte damit im Zusammenhang stehende Strategien beobachten. Dazu gehören insbesondere das systematische ● Konzeptverstehen, das pragmatische ● Beispiel-Lernen und das opportunistische ● Blinde Kopieren. Diese weiter empirisch zu erforschen und Empfehlungen für den Entwurf von APIs und den dazugehörigen Dokumentationen vorzuschlagen, würde sowohl die ● Erlernbarkeit von und die ● Programmierung mit APIs verbessern.

### 6.1.2 API-Usability-Evaluation

**Methodenvergleich** Im Abschnitt 2.4 habe ich einen Überblick über klassische, also aus der HCI stammende Usability-Evaluationsmethoden, die sich potentiell zur Evaluation von APIs nutzen lassen, gegeben. Weitere, explizit für die API-Evaluation geeignete Methoden sind die Concept-Maps-Methode von (Gerken et al. 2011), die Methode von Grill et al. (2012) und schließlich das in dieser Arbeit eingesetzte Verfahren.

Leider existieren bis heute nur oberflächliche Vergleiche (Barth 2011; Beaton et al. 2008b) einer Teilmenge, der von mir zusammengestellten Methoden. Eine umfassende Gegenüberstellung, welche die verschiedenen Vor- und Nachteile im Detail überprüft und vorstellt, wäre wünschenswert. Insbesondere zum Kosten-Nutzen-Verhältnis und der Art, der durch die jeweiligen Methoden erhobenen API-Usability-Probleme, gibt es nur fragmentiertes Wissen.

**Heuristische Evaluation** Die unmittelbare Verwendung der Heuristischen Evaluation nach Nielsen u. Molich (1990) zur Evaluation von API-Usability wird von Beaton et al. (2008b) vorgeschlagen. Grill et al. (2012) haben auf der Basis von Zibran (2008) 16 Heuristiken zur API-Usability-Evaluation hergeleitet und erfolgreich angewendet — allerdings nennen die Autoren die Einschränkung, dass sich auf diese Weise keine Laufzeitprobleme finden lassen.

Welches Spektrum an API-Usability-Problemen tatsächlich gefunden werden kann, ist nicht geklärt. An der Vollständigkeit der Heuristiken habe ich Zweifel, da die Herleitungsgrundlage (Zibran 2008) den Anspruch der Vollständigkeit nie verfolgt hat. Etwas überspitzt kann man sagen, dass die Heuristiken von Grill et al. (2012) eher pragmatisch als systematisch entwickelt wurden.

Es fehlt also eine umfassende, systematische Herleitung und empirische Validierung von API-Heuristiken. Da die Anwendung solcher Heuristiken eine kostengünstige Verbesserung von API-Usability erlaubt, halte ich eine Forschung in diese Richtung für sinnvoll und lohnenswert.

Eine Grundlage für dieses Vorhaben kann die Arbeit von Watson (2012) bilden. In dieser entwickelt und validiert der Autor drei Heuristiken zur Bewertung von API-Dokumentationen. Auf den Konsistenz-Aspekt hat sich die erstaunlich wenig beachtete Studie von Correia et al. (2009) konzentriert. Die darin präsentierten Ergebnisse eignen sich ebenfalls, um Heuristiken für benutzerfreundliche API-Dokumentationen zu entwickeln und damit eine Teilmenge möglicher API-Heuristiken zu bilden.

**Cognitive Dimensions Framework** Für die Diskussion der Usability von API<sup>G</sup>s eignen sich die kognitiven Dimensionen (CD) nach Clarke u. Becker (2003). In den Abschnitten 3.3.4.2 und 4.5.2.1 habe ich gezeigt, dass das CDF<sup>G</sup> im Gebrauch als API-Evaluationsmethode seine Schwierigkeiten hat. Nicht ohne Grund konnte ich keine einzige vollständige Anwendung in der Literatur finden. Weiterhin habe ich im Abschnitt 2.4.2.4 auf Mängel in der Arbeit von Clarke u. Becker (2003) hingewiesen.

Um das CDF als API-Evaluationsmethode einsetzen zu können, müssen noch einige Fortschritte gemacht und Einsichten etabliert werden:

- Das CDF ist ein Diskussionswerkzeug für die Usability von Notationen. Für die Evaluation von API<sup>G</sup>s wird eine klare Gebrauchsanweisung benötigt. Insbesondere Metriken zur Messung der Notationsausprägungen entlang der verschiedenen kognitiven Dimensionen müssen entwickelt werden.
- Zur Auswertung der ausgefüllten generischen CDF-Fragebögen ist ein gutes Verständnis der durch die Notationsanwender gebildeten Anwendergruppen notwendig (Blackwell u. Green 2000). Im Abschnitt 2.3.2.4 und in der Arbeit von Nykaza et al. (2002) finden sich Belege, die zeigen, dass dieser Ansatz zu kurz greift. Vielmehr muss das Verständnis von Notation und Anwendergruppen genutzt werden, um den generischen Fragebogen entsprechend dem Kontext anzupassen. Eine Instanziierung des Fragebogens durch jeden einzelnen Befragten, wie dies Blackwell u. Green (2000) vorschlagen, entbehrt jeder Vernunft.
- Zur Diskussion und Evaluationen von APIs im Speziellen müssen Clarkes API-CDs weiterentwickelt werden. Die CDs von Green (1989) wurden nicht fehlerfrei in die spezielle Notationen-Klasse “API” überführt. So fehlen beispielsweise die CDs *Fehleranfälligkeit* und *Gegenüberstellbarkeit*. Existierende Anwendungsbeschreibungen der Clark'schen CDs (Clarke 2003, 2005b) sind nicht detailliert bzw. transparent genug beschrieben, um sie auf die eigene Forschung übertragen zu können. Klar spezifizierte und erprobte Anwendungsinstruktionen müssen entwickelt werden. Stylos u. Clarke (2007) verwenden sogar an einer Stelle die ursprüngliche CD *Diffuseness* und

die daraus abgeleitete CD *Work-Step Unit* von Clarke zur gleichen Zeit, was ebenfalls an der aktuellen Reife des CDF als API-Evaluationsmethode zweifeln lässt.

**Taxonomien** Für die Ordnung von ● Entwurfsentscheidung habe ich auf eine zweidimensionale Taxonomie von Robertson (2007) zurückgegriffen. Für die Ordnung der ● Usability-Probleme verwendete ich die sehr grobe Taxonomie von Grill et al. (2012), welche aus nur vier Kategorien besteht.

Während ich bei der ersten keine Anwendungsschwierigkeiten hatte, musste ich bei der zweiten feststellen, dass diese weder meine entdeckten ● Werkzeugunterstützungsprobleme, noch die für meine GT besonders wichtigen ● produktbedingten Erwartungskonformitätsprobleme berücksichtigte.

Es wäre wünschenswert, über eine vollständigere Taxonomie für API-Usability-Probleme zu verfügen, um API-Evaluationen zu vereinfachen. Ein weiterer Nutzen bestünde darin, explorativ forschende Wissenschaftler im Sinne der GTM — das notwendige Wissen über die bewusste Verwendung von Literatur vorausgesetzt — theoretisch zu sensibilisieren. Ein vielversprechender Ausgangspunkt bildet die Taxonomie von Khajouei et al. (2011), die augenscheinlich zwar mit grafischen Benutzeroberflächen im Hinterkopf entwickelt wurde, jedoch vergleichsweise generisch ist.

## 6.2 SeqAn

### 6.2.1 Anwender

Meine bisherige Forschung beschränkte sich auf eine vornehmlich qualitative Betrachtung der Anwenderschaft von SeqAn. Für zukünftiges Usability-Engineering kann es hilfreich sein, ein besseres quantitatives Verständnis von den Anwendern zu erarbeiten, um vorhandene Ressourcen zielgerichteter für die Verbesserung von SeqAn API-Usability einsetzen zu können.

### 6.2.2 Abschluss von Usability-Verbesserungsmaßnahmen

Die Maßnahme *Fail-Fast* wurde nicht vollständig umgesetzt. Es fehlt die Bereitstellung der Möglichkeit, Fehler beim Lesen von Dateien in Erfahrung bringen zu können.

Außerdem wurde die Angleichung SeqAns an die STL nicht verwirklicht. Bei dieser Maßnahme handelt es sich um die einzige sehr wichtige Maßnahme, die ich im Rahmen meiner Arbeit nicht realisieren konnte. Schlüsseltechniken sind dabei das CRTP und der in C++11 aufgenommene `auto`-Spezifizierer.

Weitere Maßnahmen, deren Umsetzung ich empfehle, sind die *Inkonsistenzbeseitigung* und die *Shortcuts*-Maßnahme.

Ein weiteres Problem ist, dass nicht jedes in den Tutorials verwendete Code-Beispiel getestet wurde. Eigentlich werden nur Code-Beispiele eingebunden, die automatisch durch den SeqAn-CI-Server getestet werden. Dieser Automatismus hilft jedoch nicht, wenn Code-Beispiele händisch eingepflegt werden.

Nicht kompilierbarer Code beeinflusst die Erlernbarkeit negativ. Ein konkretes Beispiel befindet sich im Sequences-Tutorial, in dem der nicht-kompilierbare Datentyp `String<AminAcid>`<sup>1</sup> verwendet wird.

Seit jeher ist die Dokumentation von SeqAn stark über mehrere Webseiten fragmentiert, was die Wahrscheinlichkeit erhöht, dass Anwender nicht auf Anhieb das finden, wonach sie suchen. Dies kann dazu führen, dass Anwender die Dokumentation nur noch vermindert nutzen (vgl. Jeong et al. 2009). Auch wenn die Verknüpfungen zwischen diesen Fragmenten verbessert wurden, bleibt das Problem bestehen. Daher empfehle ich, seqan.de zu einer zentralen Anlaufstelle für SeqAn-Anwender weiterzuentwickeln. Dabei müssen Dokumentation, Tutorials, Installationsanleitungen, etc. deutlich sichtbar und nicht mehr nur mit Hilfe von 40px kleinen Icons in der Ecke der Webseite verlinkt werden.

Da die alte Dokumentation aus Revisionsgründen weiterhin online ist, konkurrieren alte und neue Dokumentation in den Suchmaschinen bei der Suche nach SeqAn-Sprachentitäten wie Funktionen. Alle nicht aktuellen Revisionen sollten daher so geändert werden, dass sie bei der Indexierung durch einen Suchmaschinen-Bot, den HTTP-Status-Code 301 Moved Permanently mit der aktuellen Adresse zurückgeben.

### 6.2.3 Evaluation von Usability-Verbesserungsmaßnahmen

SeqAn wurde in die Workflow-Engine KNIME integriert. Die Usability dieser Lösung wurde noch nicht evaluiert. Soll dies geschehen, müssen insbesondere die Usability von KNIME selbst und der Paradigmenwechsel von tabellen- zu dateibasierter Datenverarbeitung evaluiert werden.

Die weiter oben angeregte quantitative Betrachtung der Anwender könnte die Portierung von SeqAn nach anderen Programmiersprachen rechtfertigen. Bereits beim ersten SeqAn-Workshop gab es vereinzelte, technisch leider wegargumentierte Anregungen, SeqAn nach Python zu portieren. Dabei scheint es sich um keinen unwichtigen Wunsch zu handeln, was die teilweise Portierung SeqAns durch einen seiner Anwender zeigt (Reid 2014).

Seit geraumer Zeit, verfolgt das SeqAn-Team das Ziel, SeqAn zu parallelisieren und dabei auch GPUs zu nutzen. Dabei darf die Usability kein zweites Mal im Schatten des primären Ziels stehen. Ich überschaue die Auswirkungen auf die API nicht. Sollte es jedoch welche geben, empfehle ich deren Evaluation in Bezug auf die Usability.

Sobald der in C++11 eingeführte `auto`-Spezifizierer in SeqAn verwendet wird, um die Berechnung des Typs von Funktionsrückgaben mit Hilfe von Metafunktionen entsprechend zu ersetzen, muss diese Änderung ebenfalls evaluiert werden. Sollte es beispielsweise Situationen geben, in denen auf den Gebrauch einer Metafunktion nicht verzichtet werden kann, ist es denkbar, dass das Konzept der Metafunktion noch schlechter von den Anwendern verstanden wird, als zuvor — schließlich kommt es ja dann deutlich seltener zum Einsatz.

---

<sup>1</sup> <http://seqan.readthedocs.org/en/latest/Tutorial/Sequences.html?highlight=aminacid>

Code-Beispiele sind ein leidiges Thema, denn sie können Defekte enthalten, müssen in großer Stückzahl vorhanden sein, eine gleichbleibende Qualität aufweisen und immer aktuell sein. Der eben genannte `auto`-Spezifizierer ist so ein Beispiel, wie schnell alle Code-Beispiele mit einem Mal überarbeitet werden müssen. Der von Buse u. Weimer (2012) vorgestellte Algorithmus zur automatischen Erzeugung von Code-Beispielen könnte Abhilfe schaffen. Ich halte es für sinnvoll, sich mit diesem Algorithmus näher zu beschäftigen und zu prüfen, mit welchem Aufwand eine Anpassung an die Templatemetaprogrammierung-basierte generische Programmierung in C++ möglich ist.

Suchergebnisse in SeqAns Dokumentation werden entsprechend ihrer Relevanz sortiert. In Anlehnung an das *degree-of-interest model* (Stylos u. Myers 2008), wäre zu überprüfen, ob in SeqAn die Relevanz von Suchergebnissen, oder überhaupt von Sprachentitäten, nicht auf eine andere visuelle Weise ausgedrückt werden könnte. Stylos et al. (2009a) haben eine Implementierung präsentiert, bei der u.a. die Schriftgröße einer Sprachentität, dessen Suchhäufigkeit bei Google anzeigt. Ein Vorteil an diesem Ansatz besteht darin, dass Sprachentitäten immer an der gleichen Stelle zu finden sind, denn nicht mehr die Position innerhalb einer Sortierung, sondern die Schriftgröße gibt Aufschluss über die Relevanz.

Ich habe die Wirksamkeit der umgesetzten Maßnahmen lediglich argumentiert. Durch die Vielzahl der Maßnahmen wäre eine empirische Validierung wünschenswert. Der Nutzen dieses Schrittes bestünde sowohl für SeqAn, als auch für die API-Usability-Forschung.

Es ist davon auszugehen, dass die voraussichtliche Aufnahme von *Concepts* in den C++17-Sprachstandard (Schmidt 2014) eine Reihe von durch ● Templatemetaprogrammierung und ● generischer Programmierung verursachten Usability-Problemen lösen wird. Grund dafür ist, dass SeqAn bereits heute Concepts einsetzt und damit seiner Zeit voraus ist. Beseitigt werden dürfen ● Werkzeugunterstützungsprobleme, wie die *fehlende IDE-Integration der Dokumentation*, die ● fehlende Autovervollständigung und die schlechte Lesbarkeit von ● Compiler-Fehlermeldungen. Sobald C++17 eine breite Unterstützung erfahren haben wird, müssen eventuell Anpassung am Dox-Format vorgenommen und die tatsächliche Behebung der vorgenannten Usability-Probleme überprüft werden.

### 6.3 Qualitative Datenanalysewerkzeuge mit Unterstützung der GTM

Im Abschnitt 3.4.6 bin ich unter anderem auf mögliche Verbesserungen von qualitativen Datenanalysewerkzeugen eingegangen, die sich für die Forschung mit Hilfe der GTM eignen. Der Vorschlag besteht darin, ein derartiges Werkzeug auf der Grundlage einer Meta-Ontologie zu entwickeln. Diese Meta-Ontologie bildet dabei das Schema für Ontologien, die Anwender im Rahmen ihrer Forschung mittels GTM entwickeln. Dabei kann das Werkzeug möglicherweise dabei helfen, mittels Integritätsregeln, Widersprüche in Ontologien zu entdecken, implizites Wissen unter Anwendung von vom Forscher definierten Interferenzregeln explizit zu machen, Einsichten zu fördern, die Sorgfalt der Anwendung der GTM und so schließlich die Qualität der resultierenden GT zu verbessern. Von der Software bereitgestellte Umstrukturierungsoperationen, wie das Zusammenfassen/Auftrennen von Kodes oder

die Verallgemeinerung/Spezialisierung von Relationen, können dabei helfen, das Theoriemodell reibungsloser, entsprechend der eigenen Analyseergebnisse zu verändern. Ich halte es für außerordentlich lohnenswert, mögliche Umstrukturierungsoperation und die Ontologie-Hypothese im Zuge weiterer Forschung in Hinblick auf ihre Effektivität und ihre Vereinbarkeit mit der GTM zu überprüfen.

Die ontologische Modellierung einer GT hätte auch das Potential, seine maschinelle Weiterverarbeitung zu ermöglichen. Dieser Frage sollte unter Berücksichtigung der Kenntnisse zu den Themen Wissensmanagement und Wissensrepräsentation nachgegangen werden. Ein Anwendungsfall im Kleinen ist diese Arbeit, in der ich alle Elemente meiner Theorie direkt mit meinem Datenanalysewerkzeug mittels URIs verknüpft habe.

Qualitative Datenanalysewerkzeuge können durch weitergehende Analysefunktionen verbessert werden. Nach meiner Kenntnis verfügt keines der existierenden Lösungen über derartige Funktionen. Eine könnte darin bestehen, theoretisches Sampling zu verbessern, indem wichtige Kodes in Hinblick auf ihre Verankerungen in den verschiedenen Datenquellen analysiert werden. Beispiel: ● Inkonsistenzen bzgl. STL habe ich erstmalig und am häufigsten in der Datenquelle Gruppendiskussion beobachtet.

APIUA hat sich als extrem nützlich für meine Forschung erwiesen. Für die Verwendung in anderen Forschungsprojekten müssen jedoch noch weitere Datenformate unterstützt werden (insb. Videos) und die Robustheit erhöht werden. Außerdem empfiehlt sich eine Portierung von RCP<sup>2</sup> in der Version 3 zur aktuellen Version 4. Als essentielle, noch zu implementierende Funktionen haben sich die Möglichkeit der Umformung von Kodes zu Eigenschaften und die Dimensionalisierbarkeit von Relationen herausgestellt.

## 6.4 UI-Entwicklung mittels Websprachen

Die folgenden Vorschläge bewegen sich nicht auf Forschungsebene, sondern auf rein technischer Ebene, da sie ein reines Mittel zum Zweck darstellen. Dennoch erachte ich diese Vorschläge als relevante Fragestellungen, deren Erforschung sich lohnt.

Viele Teile der UI<sup>G</sup> von APIUA sind programmatisch anspruchsvoll. Um den Entwicklungsaufwand gering zu halten, traf ich die Entscheidung, die umfangreiche Funktionalität moderner Webbrowser wiederzuverwenden und UI-Elemente basierend auf den Websprachen HTML, CSS und JavaScript zu entwickeln. Der *Nebula*-Browser ist dafür zuständig, die so entwickelten UI-Elemente darzustellen (siehe Abschnitt 3.4). Wie in Anhang A.2 beschrieben, hat der *Nebula*-Browser allerdings auch das Potential für beliebige, auf dem SWT<sup>G</sup> basierende Anwendungen, als Grundlage zu dienen. Mehrere Abschlussarbeiten des Saros<sup>G</sup>-Projekt<sup>3</sup> befassen sich bereits mit der Frage, ob und wie die vorhandenen UI-Elemente portiert werden können, um eine IDE<sup>G</sup>-unabhängige Benutzeroberfläche zu entwickeln. Auf diese Weise würden nicht mehr für die verschiedenen unterstützten IDE<sup>G</sup>'s separate Code-Teile gepflegt werden müssen. Im Rahmen des Seminars *Beiträge zum Software Engineering* am 15.01.2015

2 [http://wiki.eclipse.org/index.php/Rich\\_Client\\_Platform](http://wiki.eclipse.org/index.php/Rich_Client_Platform)

3 <http://www.saros-project.org>

wurde verabredet, dass der *Nebula*-Browser in einem eigenständigen Git<sup>G</sup>-Projekt weiterentwickelt werden soll.

An dieser Stelle bietet es sich an, weiteren Projekten, die SWT einsetzen, den Nebula-Browser als Option vorzuschlagen. Eine breite Anwendung würde diese Komponente verlässlicher und noch besser für den Zweck der plattformunabhängigen UI-Entwicklung machen.

Es gibt bereits Anwendungen, die zu diesem Zweck ebenfalls einen Browser kapseln. Dazu gehört Vaadin<sup>4</sup> — ein Java-Framework, das auf dem Google Web Toolkit<sup>5</sup> basiert. Es verwendet den Browser für einen grafischen Editor von auf Vaadin basierenden Benutzeroberflächen.

In dieser Arbeit haben sich parallel zwei Entwicklungsansätze für die Browser-basierte UI-Entwicklung herausgebildet. Die erste ist Java-zentrisch, d.h. es wird lediglich eine leere HTML-Seite geladen, die dann durch diverse JavaScript-Aufrufe zu einer UI-Komponente geformt wird. Der zweite Ansatz geht von einer anwendungsspezifischen Schnittstelle zwischen Java und JavaScript aus, d.h. die gesamte Funktionalität wird in Form einer HTML-CSS-JavaScript-Seite — also einer Webanwendung — realisiert und lediglich in dem *Nebula*-Browser eingebunden. Die Stärken und Schwächen beider Ansätze müssen besser verstanden werden.

Die in diesem letzten Kapitel vorgestellten Vorschläge, werfen eine Fülle neuer Fragen auf, die nur teilweise SeqAn-spezifisch sind. Der größere Teil hingegen umfasst sich als äußerst relevant herausgestellte, methodische, inhaltliche und technische Fragestellungen der API-Usability-Forschung im Speziellen und der softwaregestützten Forschung mit Hilfe der GTM im Allgemeinen.

---

<sup>4</sup> <https://vaadin.com>

<sup>5</sup> <http://www.gwtproject.org>

## ANHANG

# A

## API USABILITY ANALYZER

Dieser Teil vertieft die Erläuterungen zu Entwurfsentscheidungen und Komponenten des APIUA<sup>G</sup>.

### A.1 Services

#### LocatorService

Dieser Dienst ist dafür zuständig, URIs aufzulösen, zu laden und als Objekt bereitzustellen. Dazu wird — neben vielen Varianten — die Methode `<T extends ILocatable> Future<T> resolve(final URI uri, final Class<T> clazz, final IProgressMonitor monitor)` bereitgestellt. Die Erzeugung mancher Objekte kann viel Zeit in Anspruch nehmen. Daher wird diese Operation stets in einem separaten Thread ausgeführt und der Fortschritt innerhalb von APIUA durch eine Fortschrittsleiste dargestellt.

Sobald das zu suchende Objekt konstruiert wurde, kann darauf mittels `Future<T>.get()` zugegriffen werden. Diese Methode gibt `null` zurück, wenn das Objekt nicht existiert oder nicht zuweisungskompatibel zur übergebenen Klasse ist.

Der LocatorService verwendet intern einen Cache, der die Objekte verdrängt, auf die am seltensten zugegriffen wurde.

## DataService

Dieser Dienst kapselt die Funktionalität, die zur Verwaltung von Datenverzeichnissen notwendig ist. Ein Datenverzeichnis enthält dabei alle, zu einem Ereignis aufgezeichneten Daten. Solche Daten können Programmierfortschritte, Gruppendiskussion, etc. sein.

Ein Datenverzeichnis wird durch eine Instanz des `IBaseDataContainer`-Interfaces dargestellt. Unterverzeichnisse implementieren das Interface `IDataContainer` und enthalten die eigentlichen `IData`-implementierenden Daten.

Die Daten werden vollständig von ihrer Speicherform abstrahiert. Aktuell gibt es eine Implementierung für Dateisysteme (`FileBaseDataContainer`, `FileDataContainer`, `FileData`). Es wären aber auch Implementierungen für den Zugriff auf Datenbanken, etc. möglich.

Eine weitere Aufgabe des DataService besteht im Laden von Datenverzeichnissen. Der Lademechanismus ist deklarativ erweiterbar. Gibt es beispielweise ein Plugin, das Daten vom Typ A laden kann und auf geladene Daten vom Typ B eines anderen Plugins angewiesen ist, fordert DataService erst den La- deprozess für B und dann für A an. Datenformate, die untereinander keine Abhängigkeiten aufweisen, werden parallel geladen. Mehr Informationen zu Datentyp-Plugins finden sich im Abschnitt 3.4.5.3.

## CodeService

Der CodeService wird durch das GTM<sup>G</sup>-Plugin (siehe Abbildung 3.22) bereitgestellt. Seine Aufgabe besteht darin, das Erstellen, Lesen, Editieren und Löschen von Kodes, Kodeinstanzen, Relationen, Relationsinstanzen, axiale Kodiermodelle<sup>G</sup>, axiale Kodierungen<sup>G</sup>, Memos, etc. zu ermöglichen.

Die Implementierung basiert im Wesentlichen auf zwei Entwurfsentscheidungen, die weiter unten ausführlicher beschrieben sind:

1. Die Datenhaltung selbst ist in die Klasse `CodeStore` ausgelagert. CodeService kapselt diese Klasse und ist für die Invariantenprüfung zuständig.
2. Die Verwendung spezieller Caches vermeidet unnötige mehrfache Berechnungen.

**Datenhaltung** CodeService verwendet zur Speicherung seiner Daten die `CodeStorage`-Klasse. Sie speichert und lädt Daten ohne eine Invariante zu haben oder gar zu prüfen. So kann beispielsweise eine Kodeinstanz auf einen nicht mehr existierenden Kode zeigen. Diese Prüfung wird vom CodeService selbst übernommen.

Der `CodeStore` speichert mit zwei Ausnahmen alle Daten in einer XML-Datei. Ein beispielhafter `CodeStore.xml` könnte wie folgt aussehen:



```

7555      <calendar>
7556          <time>1343386987000</time>
7557          <timezone>Europe/Berlin</timezone>
7558      </calendar>
7559  </creation>
7560 </instance>

    ...

15311 <relations>
15312     <uri>apiua://relation/293gbmficnukqqg68ihv7atasdtc0rf7</uri>
15313     <from reference=" [...]nebula.dataTreeNode[18]/data/uri" />
15314     <to reference=" [...]nebula.dataTreeNode[2]/data/uri" />
15315     <name>bedingt</name>
15316     <timeZoneDate>
15317         <calendar>
15318             <time>1418791813845</time>
15319             <timezone>Europe/Berlin</timezone>
15320         </calendar>
15321     </timeZoneDate>
15322 </relations>

    ...

23431 </codeStore>
```

LISTUNG 16: Auszug aus der CodeStore.xml, auf der diese Arbeit basiert.

Memos und axiale Kodiermodelle werden in separaten Dateien gespeichert. Jede einzelne Memo wird in einer HTML-Datei gespeichert, deren Name der URI<sup>G</sup> des Objekts entspricht, zu dem diese Memo gehört. Ein axiales Kodiermodell wird in einer JSON-formatierten Datei gespeichert, die wie folgt aussehen kann:

```

1  {
2      "cells": [
3          {
4              "type": "html.Element",
5              "position": {
6                  "x": -885,
7                  "y": 1198
8              }
9          }
10     ]
11 }
```

```
8     },
9     "size": {
10        "width": 128,
11        "height": 28
12    },
13    "angle": 0,
14    "id": "apiua://code/-9223372036854775552",
15    "title": "Performance<div class='details'>[...]</div>" ,
16    "content": "",
17    "z": 0,
18    "customClasses": [
19
20    ],
21    "color": "rgb(255, 255, 255)" ,
22    "background-color": "rgb(163, 198, 57)" ,
23    "border-color": "rgb(138, 168, 49)" ,
24    "attrs": {
25
26    }
27 },
```

...

```
679 {
680   "type": "link",
681   "source": {
682     "id": "apiua://code/-9223372036854775623"
683   },
684   "target": {
685     "id": "apiua://code/-9223372036854775440"
686   },
687   "labels": [
688     {
689       "position": 0.5,
690       "attrs": {
691         "text": {
692           "text": "bedingt\\n3 (2)"
693         }
694     }
695   }
696 }
```

```

695         }
696     ],
697     "id": "apiua://relation/hjg2ikddubh1nrj9uc646kjr1uf6agha",
698     "smooth": true,
699     "z": 28,
700     "customClasses": [
701
702     ],
703     "vertices": [
704     {
705         "x": 370.25,
706         "y": 1741
707     }
708     ],
709     "attrs": {
710         ".marker-target": {
711             "d": "M 10 0 L 0 5 L 10 10 z"
712         }
713     }
714 }

    ...
2811 {
2812     "title": "Wiederverwendbarkeit von Wissen: OOP- und STL-Inkonsistenten",
2813     "zoom": 0.8000000000000005,
2814     "pan": {
2815         "x": -66.08609376562507,
2816         "y": -1103.5950979062497
2817     }
2818 }

    ...

```

LISTUNG 17: Auszug aus der JSON-formatierten ACM-Datei  
[acm.apiua%3A%2F%2Faxialcodingmodel%2FQpPaMj4mceVnuSYI](http://acm.apiua%3A%2F%2Faxialcodingmodel%2FQpPaMj4mceVnuSYI)

**Cache** Im Abschnitt 3.4.6 wird ausführlich beschrieben, dass APIUA über fest implementierte Interferenzregeln verfügt. Der CodeService ist der Ort, an dem die dazu notwendigen Daten (z.B. hypothetische Relationen) berechnet werden. Diese Berechnung muss sehr effizient vonstatten gehen, da jede Theoriemodelländerung, Teile dieser Berechnungen ungültig macht.

Dazu wurde eine *DataCache*-Klasse implementiert, die Berechnungen auf der Grundlage ihrer Eingabedaten abstellt und zwischenspeichert. Die Berechnung findet nur dann, wenn die Ergebnisse überhaupt erfragt werden und sich die zugrunde liegenden Daten verändert haben.

Im Laufe meiner Forschung mit APIUA musste ich feststellen, dass die Anwendung extrem langsam geworden ist. Grund dafür war die Einführung der Interferenzregeln, deren Anwendung auf das vom mir generierte Theoriemodell viel Zeit kostete. Jede Anfrage löste eine erneute Berechnung aus.

Der Einsatz der DataCache-Klasse löste das Problem. Nun gibt es vier DataCache-Erben (*CodeCache*, *CodeInstanceCache*, *RelationCache*, *RelationInstanceCache*), die auf Teilen des Theoriemodells arbeiten (*CodeCache* nutzt den Kodebaum als Eingabe) und durch Abhängigkeiten untereinander eine Baumstruktur bilden (*CodeInstanceCache* nutzt als Eingabe die Liste der KodeInstanzen und den *CodeCache*). Dieser Ansatz führt dazu, dass nur Teile der Interferenzberechnungen im Falle einer Änderung am Theoriemodell neu berechnet werden müssen. Die nun zwischengespeicherten Ergebnisse erfordern keine ständige Neuberechnung mehr.

Ursprünglich erkannte die *DataCache*-Klasse Änderungen an seinen Eingaben mit Hilfe der `hash`-Funktion. Diese Implementierung wurde allerdings auf eine explizite Methode (`DataView.setLastModification(long nanoseconds)`) umgestellt, da die Änderung des hash-Werts keine Aussage darüber zulässt, ob sich das Objekt im Auge des Entwicklers hinreichend verändert hat und damit einer Neuberechnung bedarf.

## LabelProviderService

Dieser Dienst erlaubt es, die Bezeichner, Icons, etc. von Objekten in Erfahrung zu bringen, die durch andere Plugins beigesteuert werden. Möchte zum Beispiel eine Ansicht alle Verankerungen einer bestimmten Relation darstellen, können diese mittels `Set<IRelationInstance> ICodeService.getExplicitRelationInstances(IRelation relation)` erfragt werden. Die Phänomene werden mit Hilfe von URI `IRelationInstance.getPhenomenon()` ermittelt. Da die Phänomene möglicherweise selbst nicht geladen sind, werden sie lediglich durch eine URI identifiziert.

Um nun Daten unbekannter Quelle, darstellen zu können, kann auf die Methoden `StyledString getStyledText(URI uri)`, `String getText(URI uri)` und `getImage(URI uri)` zugegriffen werden. Der LabelProviderService delegiert die Anfragen dann an den zuständigen LabelProvider und gibt den Bezeichner bzw. das Icon zurück.

Im Abschnitt 3.4.5.3 wurde bereits beschrieben, dass jedes Datentyp-Plugin einen LabelProvider bereitstellen muss, der die Anzeigeeinformationen für die Daten berechnen kann, für die das Plugin zuständig ist.

## A.2 Browser

Das SWT<sup>G</sup> stellt einen Wrapper und damit eine standardisierte Schnittstelle für den plattformabhängigen Standard-Webbrowser bereit. Seine Kernfunktionalität besteht im Laden von Webseiten und in der Möglichkeit, von Java<sup>G</sup> aus JavaScript-Code innerhalb des Browsers auszuführen. Der SWT-Browser sollte ursprünglich als Grundlage für die UI<sup>G</sup>-Komponenten von APIUA dienen. Leider war das Funktionsspektrum zu eingeschränkt, um es problemlos für die auf Websprachen basierte UI<sup>G</sup>-Entwicklung zu verwenden.

Ich habe das von mir entwickelte *Nebula*-Plugin<sup>G1</sup> um eine Browser-Komponente erweitert, um die Unzulänglichkeiten des SWT-Browsers zu beheben.

Einen Browser für die Implementierung von Benutzeroberflächen zu verwenden, hat den Vorteil, die extrem hohe Performanz aktueller Browser nutzen zu können. Darüber existiert eine Unmenge an wiederverwendbaren auf Websprachen basierten Lösungen für verschiedenste Probleme. Abbildung A.1 zeigt eine auf einen Browser basierende UI<sup>G</sup>-Komponente, die einen Rich-Text-Editor bereitstellt. Abbildung A.2 veranschaulicht, wie das Funktionsspektrum der *Nebula*-Utility-Klasse *ColorUtils* mit Hilfe des *Nebula*-Browser dargestellt wird.

Weiterer Bestandteil des *Nebula*-Plugins ist die *Widget Gallery*. Sie stellt alle vom *Nebula*-Plugin bereitgestellten UI<sup>G</sup>-Komponenten in verschiedenen Konfigurationen dar. Sie dient damit einerseits als Überblick und andererseits als Dokumentation der Verwendungsweise der vorhandenen UI<sup>G</sup>-Komponenten.

Die Abbildungen A.1, A.2, A.3, A.4 und A.5 zeigen, wie die verschiedenen Anwendungen der Browser-Komponenten in der Widget Gallery dargestellt werden.

Der *Nebula*-Browser kapselt den SWT-Browser und unterscheidet sich von diesem wie folgt:

### Erweiterungen

Eine in dem Browser geladene Seite kann erweitert werden. Erweiterungen bestehen aus einer beliebigen Anzahl von CSS- und JavaScript-Dateien. Erweiterungen können Abhängigkeiten zu anderen Erweiterungen definieren, die im Falle ihrer Abwesenheit automatisch geladen werden. Außerdem wird sichergestellt, dass eine Erweiterung maximal einmal geladen wird. Durch Erweiterungen können beispielsweise Bibliotheken wie jQuery<sup>2</sup> nachgeladen werden.

---

1 <https://github.com/bkahlert/com.bkahlert.nebula>

2 <http://jquery.com>

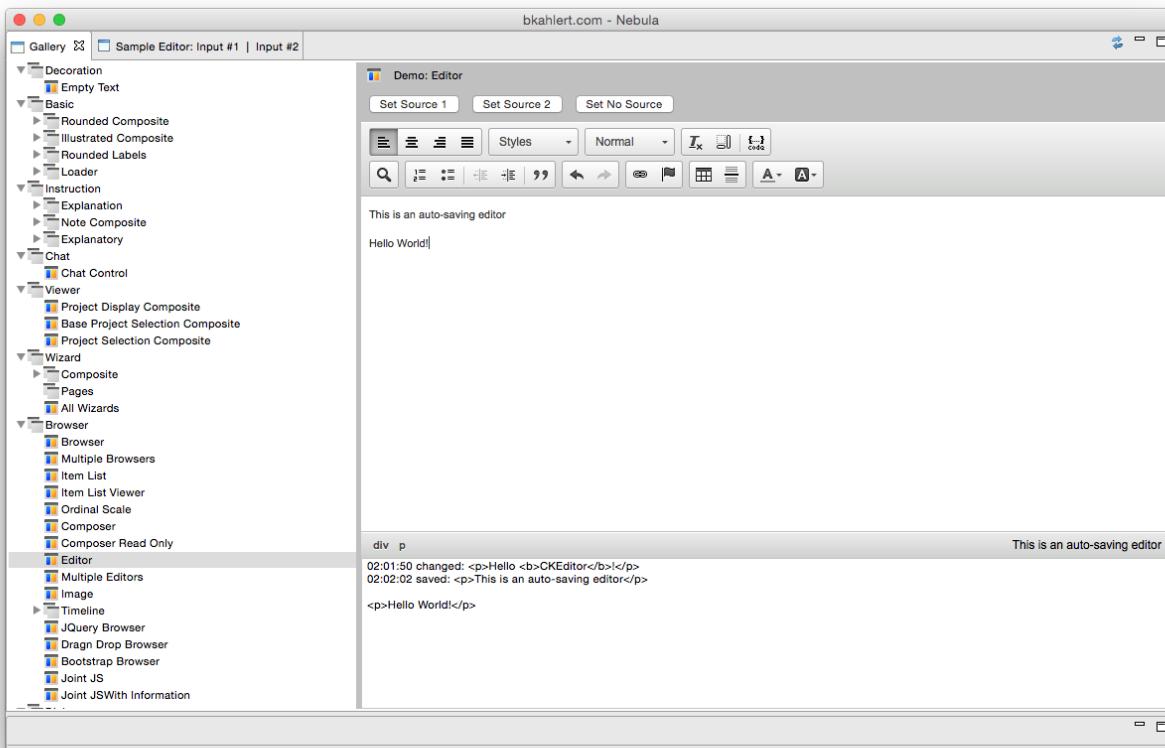


ABBILDUNG A.1: Im rechten Bereich der *Widget Gallery* wird ein auf dem *Nebula*-Browser basierender Rich-Text-Editor dargestellt. Er verwendet den Open-Source-Web-Editor *CKEditor* (<http://ckeditor.com>).

Abbildung A.3 zeigt, wie auf diese Weise die Hintergrundfarbe von [wikipedia.org](http://wikipedia.org) geändert werden kann. Abbildung A.4 hingegen zeigt eine Reihe von Knöpfen, deren Gestalt und Verhalten durch das Laden des *Bootstrap-Frameworks* verändert wurde.

Der Lebenszyklus ist ausdifferenziert. Die folgenden Methoden können überschrieben werden: `beforeLoad`, `afterLoad`, `beforeCompletion`, `afterCompletion`, `scriptAboutToBeSentToBrowser`, `scriptReturnValueReceived` und `dispose`.

### Asynchrone Schnittstelle

SWT erzwingt, dass Code, der mit einer  $UI^G$ -Komponente interagiert, in einem bestimmten Thread laufen muss. Hält sich der Entwickler nicht daran, wird eine Ausnahme zur Laufzeit geworfen. Eine Webseite wird von einem Browser allerdings immer asynchron geladen. Darum stellt bereits der SWT-Browser einen *ProgressListener* bereit, der aufgerufen wird, sobald eine Seite geladen wurde. Allerdings kann die Benachrichtigung des Listeners zu einem ungewollten Moment geschehen, da die Komponente nicht wissen kann, wann eine Seite tatsächlich geladen wurde. JavaScript-Code, könnte den tatsächlichen "Fertig geladen"-Moment verzögern.

Der *Nebula*-Browser erlaubt beim Laden die Angabe eines JavaScript-Ausdrucks. Erst wenn dieser Ausdruck `true` zurückgibt, gilt die Seite als geladen.

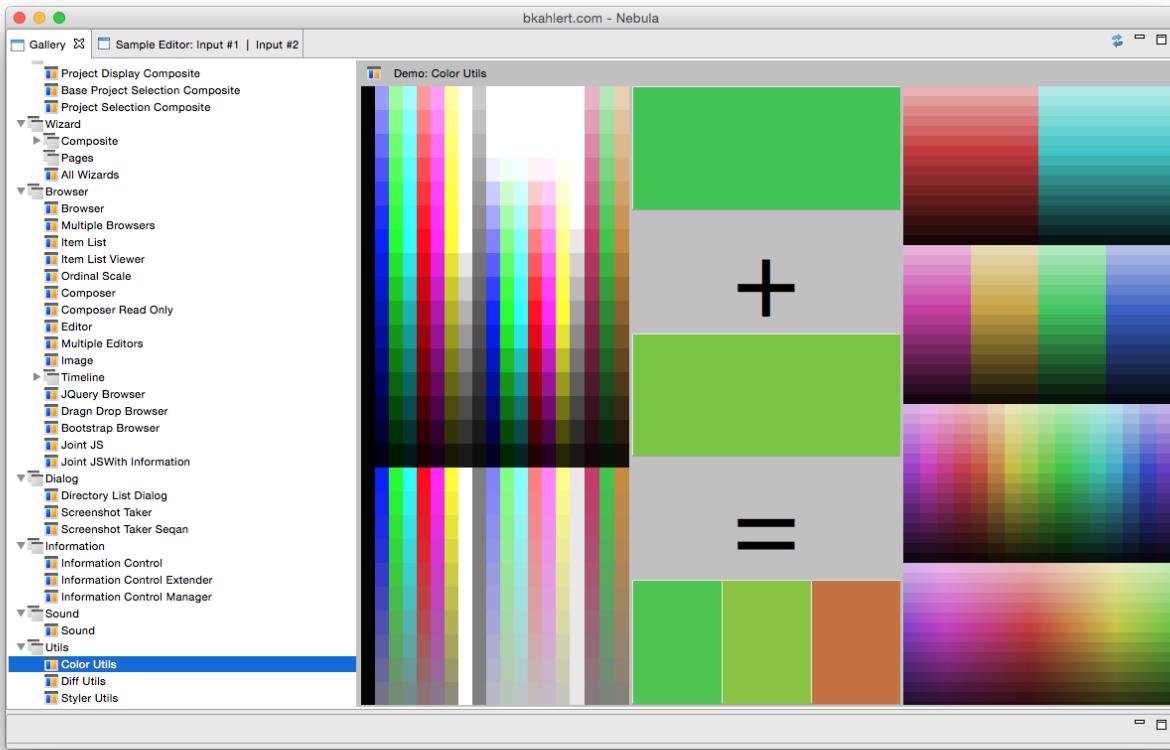


ABBILDUNG A.2: Im rechten Bereich der *Widget Gallery* wird eine auf dem *Nebula*-Browser basierende Funktionsdemonstration der *Nebula*-Utility-Klasse dargestellt.

Ein weitaus größeres Problem erwächst auf der `execute`-Funktion des SWT-Browsers. Diese gibt den Kontrollfluss an den nativen Browser weiter und kehrt erst nach Ausführung zurück. Aufwendige oder hochfrequente Aufrufe können so schnell den main-/UI<sup>G</sup>-Thread blockieren.

Um die API konsistent zu halten, aber dem Entwickler die Programmierung einfach zu machen, lassen sämtliche Methoden des *Nebula*-Browsers Aufrufe aus beliebigen Threads zu und geben ein `Future`-Objekt zurück. Der Entwickler kann dann also selbst entscheiden, wann er auf die Rückgabe zurückgreift, ohne die weitere Ausführung zu blockieren.

## JavaScript-API

Die Möglichkeiten, mit JavaScript zu interagieren sind vielfältig. Es existieren Methoden für die jeweils synchrone und asynchrone Ausführung von JavaScript-Code und Injizierung von CSS-Dateien. Liegen die Quellen als Datei vor, kann gewählt werden, ob die Dateien verlinkt oder ihr Inhalt inkludiert werden soll.

Bereitgestellt werden außerdem eine Reihe von Methoden, u.a. zum Scrollen oder der Manipulation des *Document Object Models*.

Der Aufruf von JavaScript-Methoden mit Parametern ist im SWT-Browser sehr aufwendig, da er als Zeichenkette unter Beachtung des korrekten Escapens zusammengebaut werden muss. Der *Nebula*-Browser bietet die Utility-Klasse `BrowserUtils` an, die diese Aufgabe übernimmt.

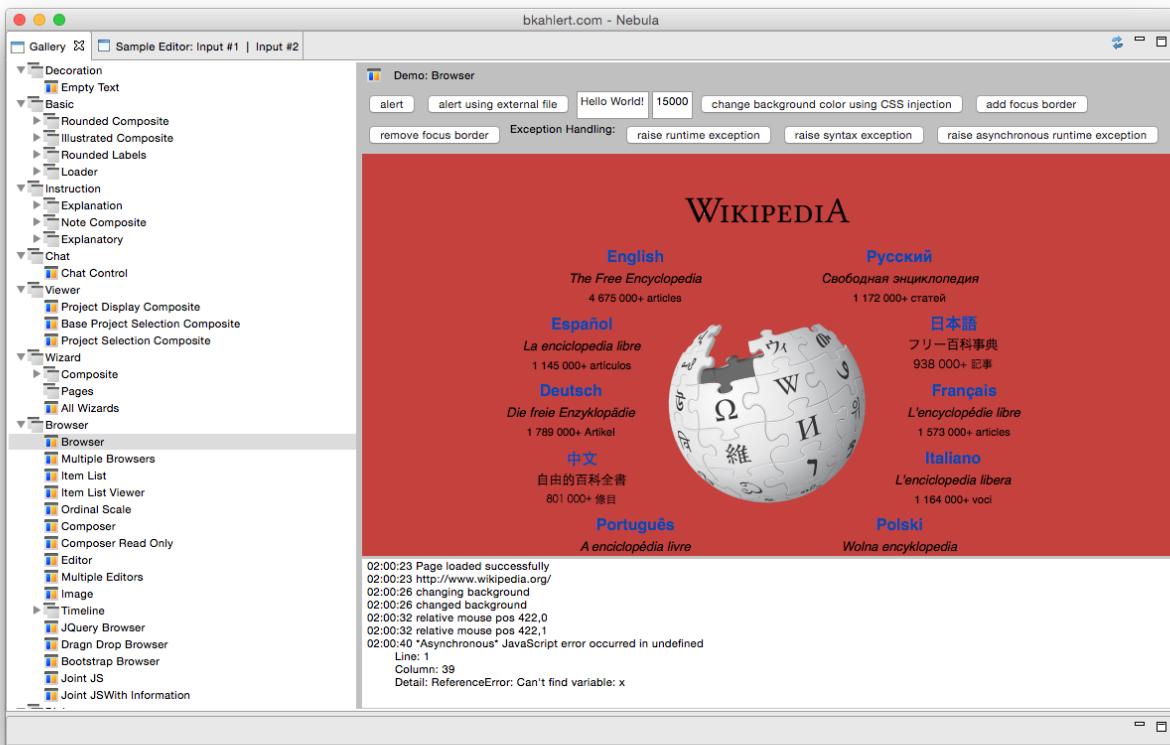


ABBILDUNG A.3: Im rechten Bereich der *Widget Gallery* wird ein *Nebula*-Browser dargestellt, der mittels einer Erweiterung die Hintergrundfarbe der geladenen Wikipedia-Startseite ändert. In dem darunter liegenden Verlauf, wird gezeigt, wie eine testweise geworfene Ausnahme gefangen wurde.

Rückgaben wiederum sind nicht auf boolesche Werte, Zahlen oder Zeichenketten beschränkt. Es werden auch Listen derselben unterstützt. Datentypen, die nicht unmittelbar unterstützt werden, werden bei der Rückgabe an Java in einen JSON<sup>3</sup>-String konvertiert.

### Debugging

Da der native Browser als Release-Build und nicht als Debug-Build vorliegt, gibt es auch keine Möglichkeit ihn zur Laufzeit im klassischen Sinn zu debuggen.

Allerdings erlaubt der *Nebula*-Browser die folgenden Aktionen:

- Ausnahmen in JavaScript, die auf einen Aufruf von Java aus zurückgehen, erzeugen eine *RuntimeException* mit entsprechender JavaScript-Zeilenummer. Die Ausnahme wird geworfen, sobald die `get`-Methode auf dem von `Browser.run` zurückgegebenen `Future`-Object aufgerufen wurde.
- Ausnahmen in JavaScript, die auf eine Aktion innerhalb des Browser zurückgehen (z.B. Klick auf einen Knopf) können mit einem zuvor registrierten `JavaScriptExceptionListener` behandelt werden.

<sup>3</sup> Dabei handelt es sich um sehr einfaches Datenaustauschformat

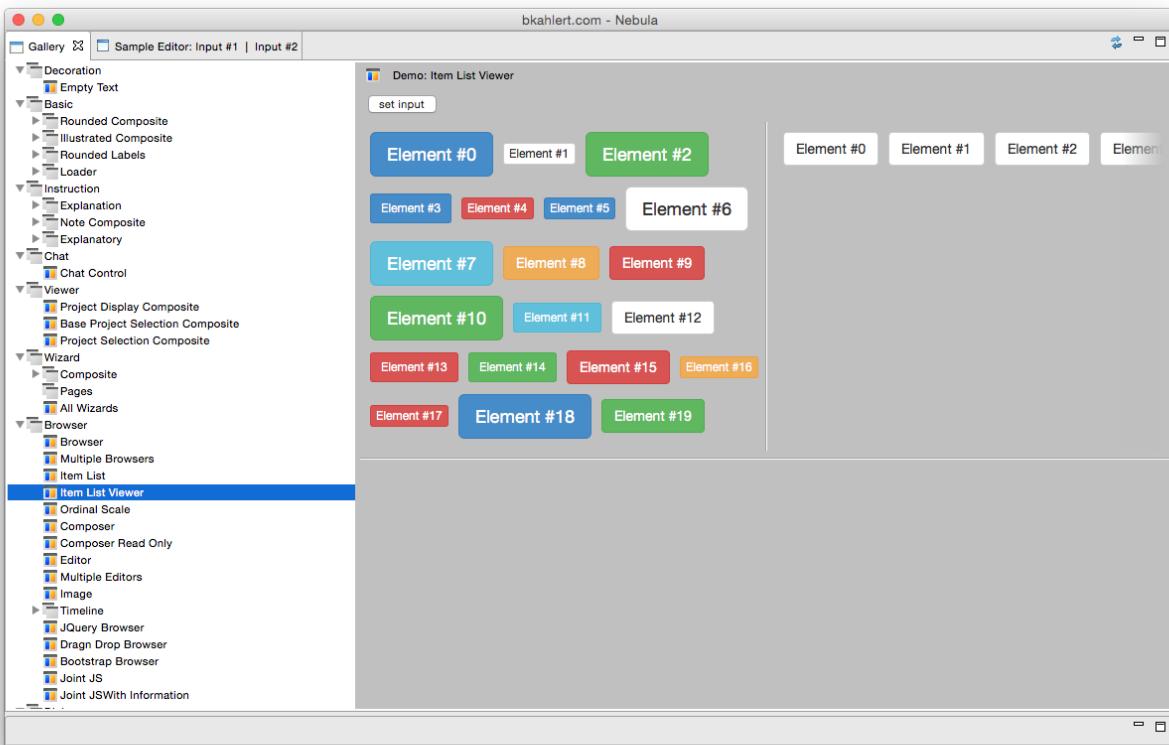


ABBILDUNG A.4: Im rechten Bereich der *Widget Gallery* werden zwei *Nebula*-Browser dargestellt, die mit Hilfe des *Bootstrap-Frameworks* (<http://getbootstrap.com>) erweiterte Knöpfe darstellen.

- Ausnahmen, die auf Syntaxfehler zurückgehen werden ebenfalls abgefangen. Die Ausnahme wird in Java geworfen, sobald die `get`-Methode auf dem von `Browser.load` zurückgegebenen `Future`-Object aufgerufen wurde.
- Aufrufe der JavaScript-Funktionen `console.log` bzw. `console.error` werden an `System.out.println` bzw. `System.err.println` weitergeleitet.

Abbildung A.3 zeigt, wie eine testweise geworfene Ausnahme gefangen wurde.

Der von den Browsern *Safari*, *Firefox* und *Chrome* bekannte Shortcut **Ctrl+Alt+I** (unter Windows) bzw. **Cmd+Alt+I** (unter Mac OS) zum Öffnen der browsereigenen Entwicklungswerkzeuge existiert auch im *Nebula*-Browser, wo er den aktuellen Zustand der eingebundenen Seite im Standard-Browser des Betriebssystems öffnet und dort effizient inspiziert werden kann.

## Drag'n'Drop

Drag'n'Drop-Operationen für textuelle Inhalte werden massiv vereinfacht, denn entsprechende Ereignisse werden auf Java-Seite an registrierte `DNDListener` weitergegeben. Ziehbare Elemente können direkt in HTML mittels der beiden Attribute `data-dnd-mime` und `data-dnd-data` deklariert werden. Das Element `<span data-dnd-mime="text/plain" data-dnd-data="Überraschung!">Hallo Welt!</span>` würde bei einer Drag'n'Drop-Operation dem Drop-Element den Text "Überraschung!" anbieten.

## Listeners

Weil der SWT-Browser eine SWT-Komponente ist, erlaubt er die Registrierung von *MouseListenern* und *FocusListenern*. Allerdings werden diese nie aufgerufen, weil die entsprechenden Ereignisse im Browser von der SWT-Komponente nicht weitergeleitet werden. Der *Nebula*-Browser hingegen überwacht die entsprechenden Ereignisse und leitet diese entsprechend weiter. Abbildung A.3 zeigt im unteren Bereich zwei mitgeschnittene Mausbewegungen.

## Zwischenablage

Einfügeoperationen ('Rechtsklick, Einfügen') werden abhängig vom gekapselten Browser unterschiedlich gehandhabt. Eine weitere Einflussgröße ist, ob die Daten direkt vorliegen oder nur die auf sie verweisende URI. Die verschiedenen Kombinationen können dazu führen, dass beim Einfügen eines Bildes im Browser entweder die *data-URI*<sup>4</sup>, die URI des Bildes selbst, der Dateiname oder sogar nichts eingefügt wird. Der *Nebula*-Browser vereinheitlicht das Verhalten bei Einfüge-Operationen mit Bildern, indem stets eine Base64<sup>5</sup>-*data-URI* eingefügt wird.

## Patches

Bestimmte auf dem WebKit<sup>6</sup> basierende Browser (hier: der *Safari*-Browser) verhalten sich besonders eigenwillig. Sie wandeln einzufügende Bilddaten in eine *webkit-fake-url*<sup>7</sup> um. Solch eine *webkit-fake-url* verweist auf die einzufügenden Bilddaten, hat aber nur Gültigkeit für den Prozess, in dem sie erzeugt wurde und bleibt auch nur für die Laufzeit des Prozesses erreichbar. Der APIUA Memo-Editor verwendet einen HTML-basierten Rich-Text-Editor, der seinen Inhalt ebenfalls als HTML-Dokument speichert. Würde ein Anwender ein Bild einfügen, würde er erst bei seiner nächsten Forschungssitzung das Fehlen seiner eingefügten Bilder feststellen. Dieses Verhalten ist derart ungewöhnlich, dass das Internet voll mit dies bezüglichen Beschwerden<sup>8</sup> ist, ein unbestätigter Bug<sup>9</sup> existiert und Software-Projekte sogar entsprechende Workarounds in ihre Anwendung implementieren<sup>10</sup>.

Erschwerend kommt hinzu, dass der *Safari*-Browser unter *Mac OS X* Bilddaten in dem exotischen *PICT*-Format<sup>11</sup> bereitstellt.

Gelöst habe ich das *webkit-fake-url*-Problem durch die Implementierung eines speziellen Handlers für Einfügeoperationen von Bildern. Dieser bezieht selbstständig, konvertiert es gegebenenfalls in das verlustfreie PNG-Format und fügt es schließlich ein.

SWT verwendet so genannte *Layouts* um die Größe und Position von UI<sup>G</sup>-Elementen innerhalb eines definierten Bereichs zu berechnen. Dazu müssen die zu formatierenden UI<sup>G</sup>-Elemente ihre

<sup>4</sup> Eine *data-URI* (spezifiziert in RFC 2397) lokalisiert keine entfernte Ressource sondern enthält diese Ressource in der *data-URI* selbst kodiert. Eine 16x16 Pixel große schwarze GIF-Grafik wird durch die folgende *data-URI* beschrieben: `data:image/gif;base64,R0lGODlhCgAKAIAAAAAAAP//yH5BAAAAAAALAAAAAKAAoAAAIhI+py+OPYysAOw==`. Nur zu! Diesen Link kann man tatsächlich in die Adresszeile des Browsers einfügen (ohne den abschließenden Satzpunkt)!

<sup>5</sup> <https://tools.ietf.org/html/rfc4648>

<sup>6</sup> <https://www.webkit.org>

<sup>7</sup> Fügt der Anwender ein Bild in einen HTML-basierten Rich-Text-Editor wird auf HTML-Ebene folgender Code eingefügt: ``

<sup>8</sup> <https://xenforo.com/community/threads/missing-images-with-webkit-fake-url-in-the-url.34752/> and <http://dev.ckeditor.com/ticket/8881>

<sup>9</sup> [https://bugs.webkit.org/show\\_bug.cgi?id=49141](https://bugs.webkit.org/show_bug.cgi?id=49141)

<sup>10</sup> <https://github.com/jejacksOn/mercury/issues/179>

<sup>11</sup> <http://www.fileformat.info/format/macpict/egff.htm>

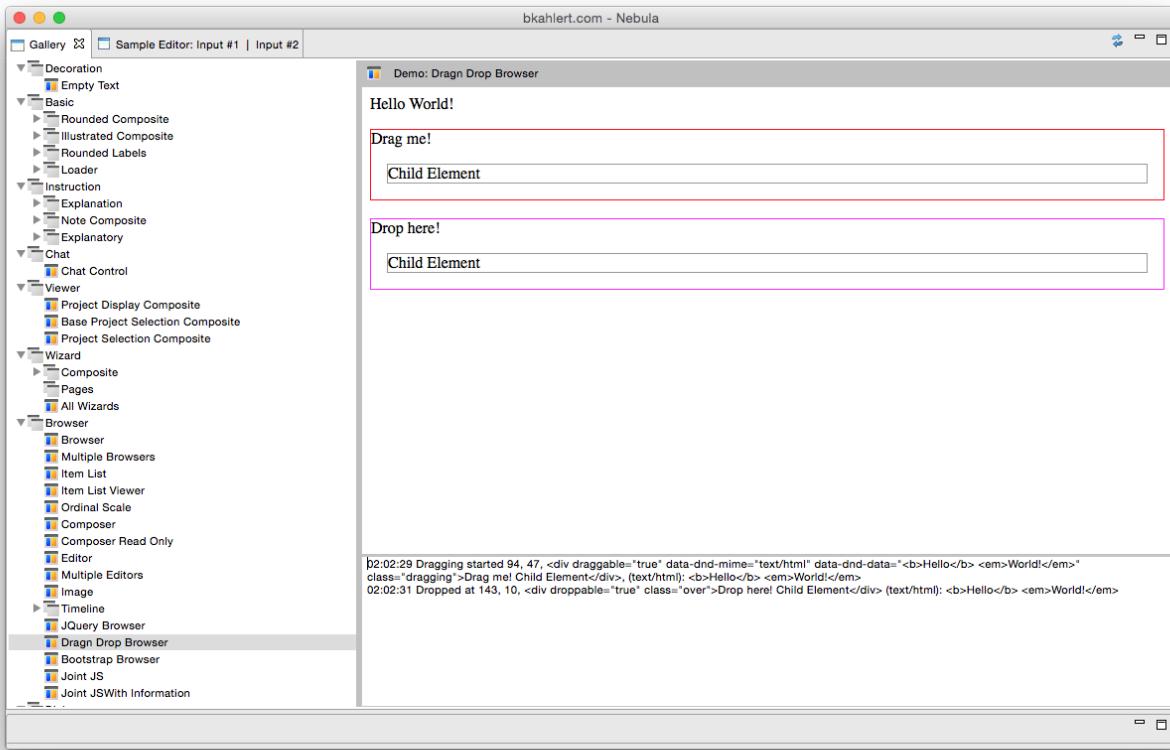


ABBILDUNG A.5: Im rechten Bereich der *Widget Gallery* wird die Drag'n'Drop-Funktionalität des *Nebula*-Browsers demonstriert.

eigene Idealmaße berechnen können. Dies wird vom SWT-Browser nicht unterstützt. Der *Nebula*-Browser hingegen injiziert ein JavaScript-Programm, das die notwendige Darstellungsgröße ereignisbasiert berechnet und an den *Nebula*-Browser weiterleitet.

Die `setBackground`-Methode wird unterstützt, indem der Hintergrund des `html`- und `body`-Tags mittels einer injizierten CSS-Regel gesetzt wird.

Implementiert wird ein Großteil der oben beschriebenen Funktionen durch ein zu Beginn des Laufvorgangs injizierten JavaScript-Codes, der eine große Anzahl an Ereignissen an die Java-seitige *Nebula*-Browser-Komponente weiterleitet. Das Fangen von Fehlern wiederum wird ermöglicht, indem sämtlicher von Java-Seite ausgeführter JavaScript-Code entsprechend gekapselt wird.

## Schwerwiegendes Problem mit dem Editor für axiales Kodieren

Beim Endspurt für diese Arbeit — genauer gesagt beim Generieren der GT<sup>G</sup> — arbeitete ich mit meinem APIUA-internen, Browser-basierten Editor für axiales Kodieren. Technisch greife ich dabei

auf die JavaScript-basierte JointJS-Bibliothek<sup>12</sup> zurück. Diese verwendet wiederum SVG zum Zeichnen meiner Graphen.

Als ich meiner Theorie ein weiteres Konzept hinzugefügt hatte, reagierte APIUA nicht mehr. Ich musste APIUA neu starten und bei jedem Versuch, den problematischen Theoriegraphen zu öffnen, wiederholte sich das Problem. Ich konnte nicht mehr arbeiten.

Der Eclipse-Debugger gab darüber Aufschluss, dass der `main`-Thread nicht mehr terminierte und es sich um die Ausführung nativen Codes handelte. Für das Debuggen musste ich also in den Browser wechseln und kleinschrittig das Problem reproduzieren, indem ich im Browser den Graphen lud und unter den letzten 478 JavaScript-Aufrufen den finden musste, der das Versagen provozierte.

Es handelte sich um eine Anweisung zur Erzeugung einer Kante zwischen zwei Knoten. Aber auch hier gab die interne Konsole des Browsers keine Fehlermeldungen aus, sondern terminierte nicht mehr. Nach unzähligen Zyklen von Debuggen-Versuchen, Browser-Abstürzen und -Neustarts stellte ich fest, dass JointJS zur Positionierung von Kantenbezeichnern die Funktion `SVGPathElement.getTotalLength()` zur Berechnung der Kantenlänge nutzte. Leider verfügt diese Funktion über einen Defekt, der bei bestimmten Bezierkurven zum besagten Versagen führt, bekannt ist<sup>13</sup> und bis dato nicht behoben wurde.

Ich konnte das Problem verifizieren. Bereits das bloße Verschieben einer der verbundenen Knoten um einen Pixel führte nicht mehr zum Versagen. Eine Lösung, die nicht in Frage kam, denn das Versagen hätte bei anderen Konstellationen erneut auftreten können.

Glücklicherweise ist JavaScript eine Prototypen-basierte Sprache, was die Möglichkeit gibt, Funktionen überschreiben zu können, ohne mit Vererbung arbeiten zu müssen. Dadurch konnte ich die defekte Funktion `SVGPathElement.getTotalLength` überschreiben, ohne sämtliche Aufrufe auf Seiten von JointJS anpassen zu müssen.

Die JavaScript-Bibliothek Raphaël<sup>14</sup> besitzt ebenfalls eine Implementierung für die Berechnung von Längen beliebiger Kurven. Diese konnte ich verwenden, um die Funktion korrekt zu überschreiben, was das Problem löste — mich jedoch Stunden und Nerven kostete.

---

12 <http://www.jointjs.com>

13 u.a. [https://bugzilla.mozilla.org/show\\_bug.cgi?id=1044355](https://bugzilla.mozilla.org/show_bug.cgi?id=1044355)  
und <https://code.google.com/p/chromium/issues/detail?id=349873>

14 <http://raphaeljs.com>



## ANHANG

### B

## LITERATURERGÄNZUNGEN

### B.1 Heuristiken der Heuristischen Evaluation (Nielsen u. a. 1990)

*Übersetzung (Schweibenz u. Thissen 2003, S. 101)*

#### H1 Sichtbarkeit des Systemstatus

Das System soll die Benutzer ständig darüber informieren, was geschieht, und zwar durch eine angemessene Rückmeldung in einem vernünftigen zeitlichen Rahmen.

#### H2 Übereinstimmung zwischen dem System und der realen Welt

Das System sollte die Sprache des Benutzers sprechen, und zwar nicht mir systemorientierter Terminologie, sondern mit Worten, Phrasen und Konzepten, die den Benutzern vertraut sind. Dabei soll die natürliche und logische Reihenfolge eingehalten werden.

#### H3 Benutzerkontrolle und -freiheit

Benutzer wählen Systemfunktionen oft fälschlicherweise aus und benötigen einen 'Notausgang', um den unerwünschten Zustand wieder zu verlassen. Dazu dienen Undo- und Redo-Funktionen.

#### H4 Konsistenz und Standards

Benutzer sollten sich nicht fragen müssen, ob verschiedene Begriffe oder Aktionen dasselbe bedeuten. Deshalb sind Konventionen einzuhalten.

#### H5 Fehlerverhütung

Noch besser als gute Fehlermeldungen ist ein sorgfältiges Design, das Fehler verhüttet.

H6 Wiedererkennen, statt sich erinnern

Objekte, Optionen und Aktionen sollten sichtbar sein. Die Benutzer sollten sich nicht an Informationen aus einem früheren Teil des Dialogs mit dem System erinnern müssen. Instruktionen sollen sichtbar oder leicht auffindbar sein.

H7 Flexibilität und Effizienz der Benutzung

Häufig auftretende Aktionen sollten vom Benutzer angepasst werden können, um Fortgeschrittenen eine schnellere Bedienung zu erlauben.

H8 Ästhetik und minimalistisches Design

Dialoge sollten keine irrelevanten Informationen enthalten, da die Informationen um die Aufmerksamkeit des Benutzers konkurrieren.

H9 Hilfe beim Erkennen, Diagnostizieren und Beheben von Fehlern

Fehlermeldungen sollten in natürlicher Sprache ausgedrückt werden (keine Fehlercodes), präzise das Problem beschreiben und konstruktiv eine Lösung vorschlagen.

H10 Hilfe und Dokumentation

Jede Information der Hilfe oder Dokumentation sollte leicht zu finden sein, auf die Aufgabe abgestimmt sein und die konkreten Schritte zur Lösung auflisten. Außerdem sollte sie nicht zu lang sein. (Schweibenz & Thissen, 2003, S. 101)

## B.2 Sämtliche Fragen des “Cognitive Dimensions Questionnaire Optimised for Users” (Blackwell u. Green 2000)

- Visibility and Juxtaposability

- How easy is it to see or find the various parts of the notation while it is being created or changed? Why?
- What kind of things are more difficult to see or find?
- If you need to compare or combine different parts, can you see them at the same time? If not, why not?

- Viscosity

- When you need to make changes to previous work, how easy is it to make the change? Why?
- Are there particular changes that are more difficult or especially difficult to make? Which ones?

- Diffuseness

- Does the notation a) let you say what you want reasonably briefly, or b) is it long-winded? Why?

- What sorts of things take more space to describe?
- Hard Mental Operations
  - What kind of things require the most mental effort with this notation?
  - Do some things seem especially complex or difficult to work out in your head (e.g. when combining several things)? What are they?
- Error Proneness
  - Do some kinds of mistake seem particularly common or easy to make? Which ones?
  - Do you often find yourself making small slips that irritate you or make you feel stupid? What are some examples?
- Closeness of Mapping
  - How closely related is the notation to the result that you are describing? Why? (Note that in a sub-device, the result may be part of another notation, rather than the end product).
  - Which parts seem to be a particularly strange way of doing or describing something?
- Role Expressiveness
  - When reading the notation, is it easy to tell what each part is for in the overall scheme? Why?
  - Are there some parts that are particularly difficult to interpret? Which ones?
  - Are there parts that you really don't know what they mean, but you put them in just because it's always been that way? What are they?
- Hidden Dependencies
  - If the structure of the product means some parts are closely related to other parts, and changes to one may affect the other, are those dependencies visible? What kind of dependencies are hidden?
  - In what ways can it get worse when you are creating a particularly large description?
  - Do these dependencies stay the same, or are there some actions that cause them to get frozen? If so, what are they?
- Progressive Evaluation
  - How easy is it to stop in the middle of creating some notation, and check your work so far? Can you do this any time you like? If not, why not?
  - Can you find out how much progress you have made, or check what stage in your work you are up to? If not, why not?
  - Can you try out partially-completed versions of the product? If not, why not?
- Provisionality

- Is it possible to sketch things out when you are playing around with ideas, or when you aren't sure which way to proceed? What features of the notation help you to do this?
- What sort of things can you do when you don't want to be too precise about the exact result you are trying to get?

- Premature Commitment

- When you are working with the notation, can you go about the job in any order you like, or does the system force you to think ahead and make certain decisions first?
- If so, what decisions do you need to make in advance? What sort of problems can this cause in your work?

- Consistency

- Where there are different parts of the notation that mean similar things, is the similarity clear from the way they appear? Please give examples.
- Are there places where some things ought to be similar, but the notation makes them different? What are they?

- Secondary Notation

- Is it possible to make notes to yourself, or express information that is not really recognised as part of the notation?
- If it was printed on a piece of paper that you could annotate or scribble on, what would you write or draw?
- Do you ever add extra marks (or colours or format choices) to clarify, emphasise or repeat what is there already? [If yes: does this constitute a helper device? If so, please fill in one of the section 5 sheets describing it]

- Abstraction Management

- Does the system give you any way of defining new facilities or terms within the notation, so that you can extend it to describe new things or to express your ideas more clearly or succinctly? What are they?
- Does the system insist that you start by defining new terms before you can do anything else? What sort of things?
- If you wrote here, you have a redefinition device: please fill in one of the section 5 sheets describing it.

## ANHANG

### C

## ROHDATEN

### C.1 Programmierfortschritte

Die umfangreichen Programmierfortschritte-Daten stehen vollständig unter den folgenden URIs zur Verfügung:

**Workshop'11** <https://github.com/bkahlert/seqan-research/tree/master/raw/workshop11>

**Workshop'12** <https://github.com/bkahlert/seqan-research/tree/master/raw/workshop12>

**Workshop'13** <https://github.com/bkahlert/seqan-research/tree/master/raw/workshop13>

**PMSB'12** <https://github.com/bkahlert/seqan-research/tree/master/raw/pmsb12>

**PMSB'13** <https://github.com/bkahlert/seqan-research/tree/master/raw/pmsb13>

### C.2 Notizen zum offenen Interviews zum Thema “ATLAS.ti”

Um eine valide Einschätzung von den Stärken und Schwächen der qualitativen Datenanalysesoftware *ATLAS.ti* zu erhalten, habe ich jeweils ein offenes Interview mit meinen Arbeitsgruppen-Kollegen geführt. Die dabei angefertigten Notizen sind im Folgenden aufgeführt.

### C.2.1 Notizen zum offenen Interview mit Franz Zieris am 10.10.2014

#### Aufbau

- Hermeneutische Einheiten (hermeneutic unit, HU)
- z.B. Liste von Primärdokumenten, (z.B. P19)
- Globale Zeitleiste
- Gezoomte Zeitleiste
- Code Manager
- Phänomen / Quotations = Markiertes Datum / Bereich
- Code = Konzept, wird an Phänomen gebunden

#### Pro

- Multimonitor
- Eigenschaften sind Codes
- Memos müssen nicht an Phänomene oder Codes gebunden sein
- Phänomen kann mehrere Codes haben [in APIUA nicht möglich]
- Memos haben einen Typ (z.B. Forschungsmemo)
- Verschiedenste Medientypen möglich
- Rich text editor für Memos (mit OLE Support)

#### Negativ

- Click auf Quotation → Zeitleiste springt an den Anfang des Phänomens (auch wenn man schon in der Mitte war) → verändert den Fokus
- Zu viele Quotations führen durch die Anordnung nebeneinander dazu, dass sie nicht mehr auf den Bildschirm passen → horizontales Scrollen
- Positionierung kann nicht verändert werden
- Kein automatisches Clustering
  - 1x Diagramm pro Code
  - keine Gesamtübersicht
  - Pfeilverbindungen können nicht ausgeblendet / beeinflusst werden
  - Strategie: in Visio machen
- Schlechter Umgang mit großen Datenmengen [langsam?]
- Keine Sortierung von Codes möglich
- Keine Gruppierung von Codes möglich (bzw. nur 1-2 zusätzliche Levels mittels Familien und Superfamilien; schwierig zu handhaben: furchtbar komplizierte Syntax, IDs werden wiederverwendet [bei APIUA kommen IDs auf eine Sperrliste])
- Kein Filtern von Quotations
- Kein Überblick beim Open Coding
- Fokus verrutscht leicht

- Keine Referenzen zwischen Memos möglich [aber bei APIUA mittels Hyperlinks]
- Code-Eigenschaften sind nur in einer separaten Übersicht zu sehen. Folge: schwer den Überblick zu behalten
- Lösung: bessere Darstellung; Eigenschaft allgemein und Eigenschaftswert an Phänomen gebunden
- Constant comparision schwierig; viele Ansicht sind exklusiv und nur Ausschnitte → zeitraubender Wechsel zwischen Ansichten (z.B. Daten- und Phänomenansicht, Doppelklick auf Phänomen schließt Phänomenansicht und öffnet Datenansicht; beides soll offen bleiben)
- Backup-Zeiten unbestimmt [bei APIUA nach jeder Aktion]
- Suche sehr Umständlich wegen komplizierter Syntax
- Quotationrahmen können nicht verschoben werden
- Inkonsistente Icons (Speichern = Diskette und manchmal Haken)
- Keine Codiersupport für mehr als einer Person

## C.2.2 Notizen zum offenen Interview mit Julia Schenk am 03.07.2014

### Positiv

- Frei gestaltbare UI (Videos abdicken und auf zweiten Bildschirm ziehen)
- Drag'n'Drop-Support
- Co-Occurrence tables

### Negativ

- Co-Occurrence tables: Konfiguration kann nicht gespeichert werden
- Zu großer Platzbedarf für die Anzeige von Quotations (führt zu horizontalem Scrollen)
- Quotation nicht mit Entfernen löschen
- Der Versuch, ein Netzwerk zu überschreiben, wird als mit "Unique name requested" quittiert.

## C.3 Ausgefüllte Cognitive-Dimensions-Fragebögen Workshop'13 am 15.09.2013

Die ausgefüllten Cognitive-Dimensions-Fragebögen können online unter <https://github.com/bkahlert/seqan-research/tree/master/raw/workshop13/workshop2013-data-20130926/cd> abgerufen werden.

## C.4 Transkript der Gruppendiskussion am 06.09.2014

Die Gruppendiskussion kann online abgerufen werden:

**Transkript** [https://rawgit.com/bkahlert/seqan-research/master/raw/workshop12/workshop2012-data-20120906/group-discussions/workshop'12%20-%20Interview%20Gruppendiskussion%20\(2012-09-06T13-01-28+0200\).html](https://rawgit.com/bkahlert/seqan-research/master/raw/workshop12/workshop2012-data-20120906/group-discussions/workshop'12%20-%20Interview%20Gruppendiskussion%20(2012-09-06T13-01-28+0200).html)

**Audioaufzeichnung** [https://github.com/bkahlert/seqan-research/blob/master/raw/workshop12/workshop2012-data-20120906/group-discussions/workshop'12%20-%20Interview%20Gruppendiskussion%20\(2012-09-06T13-01-28%2B0200\).mp4](https://github.com/bkahlert/seqan-research/blob/master/raw/workshop12/workshop2012-data-20120906/group-discussions/workshop'12%20-%20Interview%20Gruppendiskussion%20(2012-09-06T13-01-28%2B0200).mp4)

## ANHANG

D

# FORSCHUNGSERGEBNISSE: HEURISTISCHE EVALUATION

Die Ergebnisse der im Abschnitt 3.2 vorgestellten Ergebnisse der Heuristischen Evaluation können online abgerufen werden:

**Ergebnisse** <https://rawgit.com/bkahlert/seqan-research/master/raw/HE-Analyse.pdf>

**Maßnahmen** <https://rawgit.com/bkahlert/seqan-research/master/raw/HE-Massnahmen.pdf>



ANHANG

E

FORSCHUNGSDOKUMENTE

## E.1 Einverständniserklärung zur Datenerhebung



GEFÖRDERT VOM



SPONSORED BY THE



### Einverständnis- erklärung

Hiermit erkläre ich mich einverstanden, dass die im Rahmen des SeqAn – BioStore Workshops 2012, im Folgenden genannten Daten zu akademischen Zwecken pseudonymisiert gespeichert und verwendet werden dürfen.

Die gespeicherten Daten umfassen:

- Quellcode-Änderungen im lokalen SeqAn-Arbeitsverzeichnis
- Compiler-Ausgaben innerhalb des SeqAn-Arbeitsverzeichnisses
- diverse Aktionen (Seitenaufzüge, Scrollen, ...) auf seqan.de
- Betriebssystem-Version
- CMake-Einstellungen bzgl. des SeqAn-Arbeitsverzeichnisses

### Declaration of Consent

I hereby agree that in the context of the SeqAn – BioStore workshop 2012 the following mentioned data may be pseudonymously used and stored for academic purposes.

The stored data include:

- source changes in the local SeqAn working directory
- compiler output within the seqan working directory
- various actions (page views, scrolling, ...)
- operating system version
- CMake settings regarding the working directory

Datum / Date

Name

Unterschrift / Signature

## E.2 Online-Umfrage

### Vorerfahrung

- What experience do you have with C, C++, C# and Java?

- Besides SeqAn, what other applications / libraries do you use?
- To what extend did you already work with SeqAn before the workshop?

## Installation

- How difficult was the installation of SeqAn? (The installation includes downloading the sources, the CMake call(s) and the first run of your chosen IDE.)
- What did you particularly like in the installation process?
- What did you particularly dislike in the installation process?

## SeqAn-Techniken

- SeqAn heavily relies on the following four C++ / SeqAn techniques (Standard Template Library (STL) and template programming, Metafunctions, Template subclassing, Global function interface).
- How good was your understanding and ability to correctly use each technique before the workshop?
- How good is your understanding and ability to correctly use each technique now that you have absolved the workshop?

## Workshop-Tutorials

(Die folgenden Fragen wurden zu jedem Tutorial gestellt, das der Proband besucht hat.)

- Did you work on this tutorial during the workshop?
- How would you rate your SeqAn skills concerning the tutorial topic before you have participated this workshop?
- Was this tutorial helpful?
- Could you have finished this without help by the SeqAn team?
- Do you think that the documentation web pages ([www.seqan.de](http://www.seqan.de)) and [trac.mi.fu-berlin.de/seqan](http://trac.mi.fu-berlin.de/seqan)) were helpful to solve the exercises of this tutorial?
- While working on this tutorial: What did you particularly like? What did you particularly dislike?

**Abschließende Fragen**

- What did you particularly like about SeqAn?
- What did you particularly dislike about SeqAn?
- How would you rate the workshop altogether?
- How would you rate the organisation of the workshop?
- How helpful was the assistance by the SeqAn staff?
- What do you think about the duration of the workshop?
- What did you expect from the workshop? Did it meet and fulfil your expectations?
- Will you use SeqAn in the future?
- What do / did you study?
- What is your currently highest degree?

## E.3 Feedback-Zettel

### Kickoff-Feedback-Zettel

SeqAn — BioStore Workshop, 4th - 6th Sept. 2012

 BioStore 

## Kick-Off Survey

**1** Please give us the first 4 characters of your ID: \_\_\_\_\_ . . . . .

**2** How much experience do you have with the shell and the following languages?

	No Experience	Basic Skills, Few Experience	Ordinary Experience	Much Experience	Extraordinary Experience
Shell	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
C	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
C++	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
C#	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Java	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

**3** Do you currently use SeqAn? If not yet, why?

Yes       No, why? →  

**4** Besides SeqAn what other applications / libraries do you use?

**5** To what extend did you already work with SeqAn before this workshop?

I never worked with SeqAn.       I hardly worked with SeqAn.       I use SeqAn apps.       I use SeqAn as a software library.       I develop pure SeqAn apps.       I'm a SeqAn core developer.

**6** How difficult was the installation of SeqAn altogether and how difficult the different phases?

	Very Easy	Easy	Average	Difficult	Very Difficult
All together	<input type="checkbox"/>				
Step 1: download	<input type="checkbox"/>				
Step 2: CMake	<input type="checkbox"/>				
Step 3: IDE	<input type="checkbox"/>				

**7** How good was your understanding and ability to correctly use each technique?

	Unknown Technique	Basic Skills, Few Experience	Ordinary Experience	Much Experience	Extraordinary Experience
Standard Template Library	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Metafunctions	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Template Subclassing	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

## Tutorial-Feedback-Zettel

SeqAn – BioStore Workshop, 4th - 6th Sept. 2012

# Feedback

## Input/Output and Writing Parsers



**1**

Please give us the first  
4 characters of your ID: \_\_\_\_\_ . . . . .

**2**

How would you rate your SeqAn skills concerning  
the topic addressed in this tutorial before having done it.

No Experience

Basic Skills,  
Few Experience

Ordinary Experience

Much Experience

Extraordinary  
Experience

**3**

What was your motivation to do this tutorial? How helpful was the tutorial for you?

**4**

Could you have finished this tutorial without help by the SeqAn team?

No

Yes, but only with a  
significant amount of  
more time

Yes, with slightly to  
no more additional  
time

**5**

Do you think that the documentation web pages  
([www.seqan.de](http://www.seqan.de) and [trac.seqan.de](http://trac.seqan.de)) were helpful to solve the exercises of the tutorial?

Hardly

Average

Above Average

Extraordinary

**6**

What did you particularly like about this tutorial?

**7**

What did you particularly dislike about this tutorial? What posed the biggest problems?

## Abschluss-Feedback-Zettel

SeqAn — BioStore Workshop, 4th - 6th Sept. 2012



### Final Survey

**1** Please give us the first 4 characters of your ID: \_\_\_\_\_ . . . . .

**2** What did you particularly **like** in the installation process?

**3** What did you particularly **dislike** in the installation process?

**4** What did you particularly **like** about SeqAn?

**5** What did you particularly **dislike** about SeqAn?

**6** How would you rate the workshop altogether?

- |                          |          |                          |     |                          |         |                          |      |                          |           |
|--------------------------|----------|--------------------------|-----|--------------------------|---------|--------------------------|------|--------------------------|-----------|
| <input type="checkbox"/> | Very bad | <input type="checkbox"/> | Bad | <input type="checkbox"/> | Average | <input type="checkbox"/> | Good | <input type="checkbox"/> | Very good |
|--------------------------|----------|--------------------------|-----|--------------------------|---------|--------------------------|------|--------------------------|-----------|

**7** How would you rate the organization of the workshop?

- |                          |          |                          |     |                          |         |                          |      |                          |           |
|--------------------------|----------|--------------------------|-----|--------------------------|---------|--------------------------|------|--------------------------|-----------|
| <input type="checkbox"/> | Very bad | <input type="checkbox"/> | Bad | <input type="checkbox"/> | Average | <input type="checkbox"/> | Good | <input type="checkbox"/> | Very good |
|--------------------------|----------|--------------------------|-----|--------------------------|---------|--------------------------|------|--------------------------|-----------|

**8** How helpful was the assistance by the SeqAn staff?

- |                          |                |                          |                         |                          |                   |                          |                         |                          |                         |
|--------------------------|----------------|--------------------------|-------------------------|--------------------------|-------------------|--------------------------|-------------------------|--------------------------|-------------------------|
| <input type="checkbox"/> | Hardly helpful | <input type="checkbox"/> | Below averagely helpful | <input type="checkbox"/> | Averagely helpful | <input type="checkbox"/> | Above averagely helpful | <input type="checkbox"/> | Extraordinarily helpful |
|--------------------------|----------------|--------------------------|-------------------------|--------------------------|-------------------|--------------------------|-------------------------|--------------------------|-------------------------|

**9** What do you think about the duration of the workshop?

- |                          |               |                          |           |                          |      |                          |          |                          |              |
|--------------------------|---------------|--------------------------|-----------|--------------------------|------|--------------------------|----------|--------------------------|--------------|
| <input type="checkbox"/> | Far too short | <input type="checkbox"/> | Too short | <input type="checkbox"/> | Good | <input type="checkbox"/> | Too long | <input type="checkbox"/> | Far too long |
|--------------------------|---------------|--------------------------|-----------|--------------------------|------|--------------------------|----------|--------------------------|--------------|

SeqAn — BioStore Workshop, 4th - 6th Sept. 2012



## Final Survey

**10**

How would you rate the pure online tutorials (not their integration in the workshop)?

<input type="radio"/>	Very bad	<input type="radio"/>	Bad	<input type="radio"/>	Average	<input type="radio"/>	Good	<input type="radio"/>	Very good
-----------------------	----------	-----------------------	-----	-----------------------	---------	-----------------------	------	-----------------------	-----------

**11**

Why did you attend to this workshop?

<input type="checkbox"/>	I'm currently working on a project where I'm expected to use SeqAn	<input type="checkbox"/>	I'm currently working on a project where I think SeqAn does the job	<input type="checkbox"/>	I was just curious
--------------------------	--	--------------------------	---	--------------------------	--------------------

Other reason

**12**

Did the workshop meet your expectations?

**13**

What functionality of SeqAn are you using?

What functionality do you miss?

What is / would be your preferred way to develop with SeqAn?

<input type="checkbox"/>	As a framework (create apps)	<input type="checkbox"/>	As a library (include SeqAn headers in own program)	<div style="border: 1px solid #0070C0; width: 100px; height: 30px; margin-left: 10px;"></div>
--------------------------	---------------------------------	--------------------------	--	---

Other

**14**

Will you use SeqAn in the future?

Yes, why? →

No, why? →

**15**

What do / did you study?

**16**

What is your currently highest degree?

## E.4 Cognitive-Dimensions-Fragebogen

Der näher im Abschnitt 3.3.4 vorgestellte Cognitive-Dimensions-Fragebogen kann online abgerufen werden:

**Deutsch** <https://rawgit.com/bkahlert/seqan-research/master/raw/workshop13/cd-fragebogen-de.pdf>

**Englisch** <https://rawgit.com/bkahlert/seqan-research/master/raw/workshop13/cd-fragebogen-en.pdf>

Die Fragen aus Teil 3 des Fragebogens beziehen sich auf jeweils eine CD<sup>G</sup> und waren für die Befragten nicht sichtbar. Sie konnten lediglich die Fragen selbst sehen. Die erfragten CDs des Fragebogens, die, wenn nicht anders angegeben, von Clarke u. Becker (2003) stammen, lauten:

1. Anpassungsfreundlichkeit *API Viscosity*
2. Arbeitsschrittgröße / Codedichte (*Work-Step Unit*)<sup>1</sup>
3. Arbeitsgedächtnisanforderungen (*Hard Mental Operations*)<sup>2</sup>
4. Fehleranfälligkeit (*Error Proneness*)<sup>3</sup>
5. Fachbezogenheit (*Domain Correspondence*)
6. Rollenerkennbarkeit (*Role Expressiveness*)
7. Fortschreitende Evaluation (*Progressive Evaluation*)
8. Vorläufigkeit (*Provisionality*)<sup>4</sup>
9. Verfrühter Entscheidungzwang (*Premature Commitment*)
10. Konsistenz (*Consistency*)
11. Abstraktionsebenen (*Abstraction Level*)
12. Lernweise (*Learning Style*)

---

<sup>1</sup> Diese CD stammt von Clarke u. Becker (2003) und ist eine Spezialisierung der CD *Diffuseness* von Blackwell u. Green (2000).

<sup>2</sup> Dabei handelt es sich um eine CD von Blackwell u. Green (2000). Von Clarke u. Becker (2003) wird sie als *Working Framework* bezeichnet.

<sup>3</sup> Dabei handelt es sich um eine Dimension nach Blackwell u. Green (2000).

<sup>4</sup> Dabei handelt es sich um eine Dimension nach Blackwell u. Green (2000).



# GLOSSAR

**API** application programming interface. 27, 33–35, 60, 89, 93, 94, 97, 101, 102, 105, 108, 111, 122, 129, 134, 135, 231, 341, 353, 395, *siehe:* application programming interface

**APIUA** API Usability Analyzer. 146, 148, 193, 194, 199, 202–222, 225–227, 231, 235, 237, 241, 243, 245, 336, 342, 343, 346, 347, 357, 359, 364–366, 371, 395, *siehe:* API Usability Analyzer

**API Usability Analyzer** wird ausführlich im Abschnitt 3.4, ab Seite 193 besprochen. 146, 193, 227, 231, 343, 346, 395

**application programming interface** wird ausführlich im Abschnitt 1.2, Seite 33 definiert. 395

**axiale Kodierung** ist das Ergebnis der Phase *axiales Kodieren* basierend auf Phänomenen (vgl. *axiales Kodiermodell*). 51, 195, 203, 220, 236, 360, 395

**axiales Kodiermodell** ist das Ergebnis der Phase *axiales Kodieren* basierend auf Kodes (vgl. *axiale Kodierung*). 51, 195, 203–205, 220, 225, 237, 238, 360, 362, 395

**Bedienoberfläche** ist die Schnittstelle, durch die ein Anwender mit einem technischen System interagiert. 396, 398

**Bundle** ist eine grundlegende Komponente der OSGi Service Platform (OSGi) — vergleichbar mit einem Plugin. 397

**CD** kognitive Dimension. 79, 87, 94, 116, 125, 179, 180, 330, 353, 393, 395, *siehe:* kognitive Dimension

**CDF** Cognitive Dimensions Framework. 78–80, 83, 85, 86, 95, 96, 117, 179, 353, 395, 397, *siehe:* Cognitive Dimensions Framework

**Cognitive Dimensions Framework** ist ein Diskussionswerkzeug, das eine gemeinsame Terminologie — nämlich ein Dutzend so genannter kognitiver Dimensionen — für die Diskussion über den Umgang mit Interaktions- und Kommunikationsstrukturen bereitstellt (siehe Abschnitt 2.3.2). 395, 397

**Eclipse** ist eine populäres Entwicklungswerkzeug, das ursprünglich von IBM als integrierte Entwicklungsumgebung (englisch *integrated development environment*) (IDE) für Java entwickelt wurde. 163, 173, 189, 207, 210, 211, 346

**Eclipse Rich Client Platform** ist der Teil von Eclipse, mit denen individuelle Rich-Client-Anwendungen entwickelt werden können. 397

**end-user software engineering** (deutsch *Endanwender Softwaretechnik*) bezeichnet das Gebiet der Softwaretechnik, bei dem der Entwickler sich gezwungener Maßen, also mangels Alternativen, im Bereich der Softwaretechnik bewegt. Die bekannteste Form ist der Endanwender-Programmierer (engl. *end-user programmer*), der — ohne typischerweise über eine entsprechende Ausbildung zu verfügen — programmatisch Lösungen entwickelt. 251, 396

**EUSE** end-user software engineering. 55, 58, 99, 108, 111, 138, 251, 396, *siehe:* end-user software engineering

**Git** ist ein populärer verteiltes Versionsverwaltungssystem, das in seiner ersten Version von Linus Torvalds entwickelt wurde und unter der GNU General Public License (GNU GPL) steht. 358, 396

**GitHub** ist ein populärer Onlinedienst, das die Entwicklung von Softwareprojekten ermöglicht und besonders für sein auf Git basierendes Versionsverwaltungssystem bekannt ist. 207

**GNU General Public License** ist eine weit verbreitete Software-Lizenz, die das kostenlose Studium, die kostenlose Nutzung, Änderung und Weitergabe von Software erlaubt. 396

**GNU GPL** GNU General Public License. 396, *siehe:* GNU General Public License

**grafische Benutzeroberfläche** ist die grafische Variante einer Bedienoberfläche (UI). 396

**Grounded Theory** Ergebnis einer empirischen Studie basierend auf der Methode der Grounded Theory. 222, 225, 242, 396, 397

**GT** Grounded Theory. 47, 49, 50, 52, 148, 199, 201, 204, 206, 222, 225, 226, 237, 238, 242, 247, 249, 251, 292, 325, 329, 333, 339, 342, 343, 349, 354, 356, 357, 372, 396, 397, *siehe:* Grounded Theory

**GTM** Methode der Grounded Theory. 19, 25, 34, 47–50, 52–55, 57, 141, 144, 145, 148, 151, 154, 157, 163, 170, 173, 176, 179, 181, 188, 191, 193, 195, 199, 201, 202, 206, 212, 213, 216, 217, 222, 225–227, 229–231, 236, 239, 241, 243, 245, 249, 251, 275, 278, 303, 310, 325–327, 329, 336, 339, 342, 345–347, 349, 350, 354, 356–358, 360, 396, *siehe:* Methode der Grounded Theory

**GUI** grafische Benutzeroberfläche. 27, 396, *siehe:* grafische Benutzeroberfläche

**HE** Heuristische Evaluation. 59, 60, 113–115, 141, 151, 153, 154, 174, 178, 229, 230, 243, 245, 326, 336, 342, 352, 397, *siehe*: Heuristische Evaluation

**Heuristische Evaluation** ist ein ursprünglich von Nielsen (1993) entwickelte Usability-Evaluationsmethode (siehe Abschnitt 3.2.2). 141, 153, 174, 229, 397

**IDE** integrierte Entwicklungsumgebung (englisch *integrated development environment*). 112, 357, 396, 397, *siehe*: integrierte Entwicklungsumgebung (englisch *integrated development environment*)

**integrierte Entwicklungsumgebung (englisch integrated development environment)** ist eine Sammlung von Werkzeugen, mit denen im optimalen Fall sämtliche Aufgaben der Softwareentwicklung gelöst werden können. 396, 397

**Java** ist eine populäre objektorientierte Programmiersprache. 102, 104, 105, 366, 370, 397

**kognitive Dimension** ist ein integraler Bestandteil des Cognitive Dimensions Framework (CDF) (siehe Abschnitt 2.3.2). 395

**LET** Sprachentitätstyp. 397, *siehe*: Sprachentitätstyp

**Methode der Grounded Theory** Eine qualitative Forschungsmethode, die - basierend auf empirisch erfassten Daten - eine Grounded Theory hervorbringt. Eine ausführlichere Beschreibung befindet sich im Abschnitt 1.4.. 47, 141, 226, 229, 396

**OOBE** Out-Of-Box-Experience. 117, 151, 342, 397, *siehe*: Out-Of-Box-Experience

**OSGi** OSGi Service Platform. 210, 226, 395, 397, *siehe*: OSGi Service Platform

**OSGi Service Platform** spezifiziert eine Java-basierte Laufzeitumgebung, die das Laden, Aktualisieren und Entladen von Bundles erlaubt. 395, 397

**Out-Of-Box-Experience** beschreibt die ersten Erfahrungen, die ein Anwender mit einem Produkt macht. Dabei hat der Anwender — abhängig von der Art des Produkts — Kontakt mit Artefakten wie der Produktverpackung oder einer Installationsprozedur. Etwas ausführlicher wird dieser Begriff im Abschnitt 3.2.1.1 auf Seite 151 behandelt. 151, 397

**Plugin** auch Plug-In genannt, ist ein Softwaremodul, das von einer bestehenden Software geladen werden kann. 203, 208, 210–212, 366, 395, 397

**RCP** Eclipse Rich Client Platform. 207, 209, 210, 226, 346, 397, *siehe*: Eclipse Rich Client Platform

**Saros** ist ein in der Arbeitsgruppe Software Engineering, des Fachbereichs Informatik der Freien Universität Berlin entwickeltes Eclipse-Plugin, dass die gemeinsame Arbeit an Softwareprojekten erlaubt. Ein Plugin für die IDE *IntelliJ* ist in Arbeit.<sup>5</sup>. 357

---

<sup>5</sup> <http://www.saros-project.org>

**Sprachentitätstyp** (englisch: language entity type), siehe Abschnitt 4.1.4, ab Seite 270. 397

**Standard Widget Toolkit** ist eine Bibliothek der Eclipse Foundation, die die native plattformunabhängige Verwendung von UI-Elementen eines Betriebssystems erlaubt.. 398

**SWT** Standard Widget Toolkit. 210, 357, 358, 366–368, 371, 372, 398, *siehe:* Standard Widget Toolkit

**UI** Bedienoberfläche. 210, 357, 358, 366–368, 371, 396, 398, *siehe:* Bedienoberfläche

**Uniform Resource Identifier** ist ein Identifikator für eine Ressource. 398

**URI** Uniform Resource Identifier. 185, 204, 205, 210–212, 215, 217, 226, 343, 346, 359, 362, 365, 371, 398, *siehe:* Uniform Resource Identifier

# LITERATURVERZEICHNIS

## Aggarwal 2009

AGGARWAL, J C.: *Essentials Of Educational Psychology*. 2. Edition. Vikas Publishing House Pvt Ltd, 2009

## Aguiar 2000

AGUIAR, Ademar: A minimalist approach to framework documentation. In: *OOPSLA '00: Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. New York, New York, USA : ACM Request Permissions, Januar 2000, S. 143–144

## Alexander et al. 2012

ALEXANDER, Erik K. ; KENNEDY, Giulia C. ; BALOCH, Zubair W. ; CIBAS, Edmund S. ; CHUDOVA, Darya ; DIGGANS, James ; FRIEDMAN, Lyssa ; KLOOS, Richard T. ; LiVOLSI, Virginia A. ; MANDEL, Susan J. ; RAAB, Stephen S. ; ROSAI, Juan ; STEWARD, David L. ; WALSH, P S. ; WILDE, Jonathan I. ; ZEIGER, Martha A. ; LANMAN, Richard B. ; HAUGEN, Bryan R.: Preoperative Diagnosis of Benign Thyroid Nodules with Indeterminate Cytology. In: *New England Journal of Medicine* 367 (2012), Nr. 8, S. 705–715

## Aschwanden u. Crosby 2006

ASCHWANDEN, Christoph ; CROSBY, Martha E.: Code Scanning Patterns in Program Comprehension. In: *HICSS: 39th Hawaii International Conference on System Sciences*, 2006

## Bager 2013

BAGER, Jo: *Werbenetzwerk Zanox setzt auf Browser-Fingerprinting.* [http://www.heise.de/newsticker/meldung/Werbenetzwerk-Zanox-setzt-auf-Browser-Fingerprinting-1962527.html?wt\\_mc=rss.ho.beitrag.atom](http://www.heise.de/newsticker/meldung/Werbenetzwerk-Zanox-setzt-auf-Browser-Fingerprinting-1962527.html?wt_mc=rss.ho.beitrag.atom). Version: September 2013

## Bamberg et al. 2011

BAMBERG, Eva ; MOHR, Gisela ; BUSCH, Christian: *Arbeitspsychologie*. Hogrefe Verlag, 2011

**Barth 2011**

BARTH, Michael: *API Evaluation*. <http://www.michaelbarth.net/files/publications/api-evaluation.pdf>. Version: Dezember 2011

**Basili et al. 2000**

BASILI, V R. ; LANUBILE, F ; SHULL, F: Investigating reading techniques for object-oriented framework learning. In: *IEEE Transactions on Software Engineering* 26 (2000), Nr. 11, S. 1101–1118

**Beaton et al. 2008a**

BEATON, Jack ; JEONG, Sae Young S. ; XIE, Yingyu C. ; STYLOS, Jeffrey ; MYERS, Brad A.: Usability challenges for enterprise service-oriented architecture APIs. In: *VLHCC '08: Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, IEEE Computer Society, September 2008

**Beaton et al. 2008b**

BEATON, Jack ; MYERS, Brad A. ; STYLOS, Jeffrey ; JEONG, Sae Young S. ; XIE, Yingyu C.: Usability evaluation for enterprise SOA APIs. In: *RSSE '10: Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*. New York, New York, USA : ACM Press, 2008, S. 29

**Berglund 2003**

BERGLUND, Erik: Designing electronic reference documentation for software component libraries. In: *Journal of Systems and Software* 68 (2003), Oktober, Nr. 1, S. 65–75

**Blackwell u. Green 1999**

BLACKWELL, A F. ; GREEN, TRG: Investment of attention as an analytic approach to cognitive dimensions. In: *Collected Papers of the 11th Annual Workshop of the Psychology of Programming Interest Group (PPIG '11)* (1999), S. 24–35

**Blackwell u. Green 2003**

BLACKWELL, Alan F. ; GREEN, Thomas: Notational Systems—The Cognitive Dimensions of Notations Framework. In: CARROLL, John M. (Hrsg.): *HCI Models, Theories, and Frameworks*. San Francisco : Morgan Kaufmann, 2003, S. 103–133

**Blackwell u. Green 2000**

BLACKWELL, Alan F. ; GREEN, Thomas R G.: A Cognitive Dimensions Questionnaire Optimised for Users. In: *PPIG 2000: Proceedings of the Twelfth Annual Meeting of the Psychology of Programming Interest Group*, 2000, S. 137–152

**Blais u. White 2015**

BLAIS, Christopher M. ; WHITE, Janet L.: Bioethics in practice - a quarterly column about medical ethics: ebola and medical ethics - ethical challenges in the management of contagious infectious diseases. In: *The Ochsner journal* 15 (2015), Nr. 1, S. 5–7

**Blinman u. Cockburn 2005**

BLINMAN, Scott ; COCKBURN, Andy: Program Comprehension: Investigating the Effects of Naming Style and Documentation. In: *AUIC* (2005), S. 73–78

**Bloch 2006a**

BLOCH, Joshua: How to design a good API and why it matters. In: *OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* (2006), Oktober, S. 506

**Bloch 2006b**

BLOCH, Joshua: *How to design a good API and why it matters*. <http://lcsd05.cs.tamu.edu/slides/keynote.pdf>. Version: 2006

**Bloch 2008**

BLOCH, Joshua: *Effective Java*. Pearson Education, 2008 (Java Series)

**Boehm et al. 2003**

BOEHM, Andreas ; LEGEWIE, Heiner ; MUHR, Thomas: *Kursus Textinterpretation: Grounded Theory*. [http://www.qualitative-forschung.de/information/publikation/modelle/kurs\\_ti.pdf](http://www.qualitative-forschung.de/information/publikation/modelle/kurs_ti.pdf). Version: November 2003

**Boehm 1984**

BOEHM, Barry W.: Software Engineering Economics. In: *IEEE Transactions on Software Engineering* SE-10 (1984), Januar, Nr. 1, S. 4–21

**de Bouvet et al. 2006**

BOUVET, Armelle de ; DESCHAMPS, Claude ; BOITTE, Pierre ; BOURY, Dominique: Bioinformatics: The philosophical and ethical issues at stake in a new modality of research practices. In: *Medicine, health care, and philosophy* 9 (2006), Nr. 2, S. 201–209

**Breckenridge et al. 2012**

BRECKENRIDGE, J ; JONES, D ; ELLIOTT, I ; NICOL, M: Choosing a methodological path: Reflections on the constructivist turn. In: *Grounded Theory Review* 11 (2012), Juni, Nr. 1, S. 64–71

**Briand et al. 1997**

BRIAND, Lionel C. ; BUNSE, Christian ; DALY, John W.: An experimental evaluation of quality guidelines on the maintainability of object-oriented design documents. In: *ESP '97: Papers presented at the seventh workshop*. New York, New York, USA : ACM Press, 1997, S. 1–19

**Brooks 1980**

BROOKS, Ruven E.: Studying programmer behavior experimentally: the problems of proper methodology. In: *Communications of the ACM* 23 (1980), April, Nr. 4, S. 207–213

**Brooks 1983**

BROOKS, Ruven E.: Towards a Theory of the Comprehension of Computer Programs. In: *International Journal of Man-Machine Studies* 18 (1983), Nr. 6, S. 543–554

**Bruch et al. 2010**

BRUCH, M ; MEZINI, M ; MONPERRUS, M: Mining subclassing directives to improve framework reuse. In: *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, IEEE, Mai 2010, S. 141–150

**Bruch et al. 2006**

BRUCH, Marcel ; SCHÄFER, Thorsten ; MEZINI, Mira: FrUiT: IDE support for framework understanding. In: *eclipse '06: Proceedings of the 2006 OOPSLA workshop on eclipse technology eXchange*. New York, New York, USA : ACM, Oktober 2006, S. 55–59

**Bryant u. Charmaz 2010**

BRYANT, Antony ; CHARMAZ, Kathy: *The SAGE Handbook of Grounded Theory: Paperback Edition*. SAGE Publications, 2010

**Buse u. Weimer 2012**

BUSE, Raymond P L. ; WEIMER, Westley: Synthesizing API usage examples. In: *ICSE '12: Proceedings of the 34th International Conference on Software Engineering*, IEEE Press, Juni 2012

**Canning et al. 1989**

CANNING, Peter ; COOK, William ; HILL, Walter ; OLTHOFF, Walter ; MITCHELL, John C.: F-bounded polymorphism for object-oriented programming. In: *FPCA '89: Proceedings of the fourth international conference on Functional programming languages and computer architecture*. New York, New York, USA : ACM Request Permissions, November 1989, S. 273–280

**Cao et al. 2010**

CAO, Jill ; RICHE, Yann ; WIEDENBECK, Susan ; BURNETT, Margaret M. ; GRIGOREANU, Valentina: End-user mashup programming: through the design lens. In: *CHI '10: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM Request Permissions, April 2010

**Card et al. 1983**

CARD, Stuart K. ; NEWELL, Allen ; MORAN, Thomas P.: *The Psychology of Human-Computer Interaction*. Hillsdale, NJ, USA : L. Erlbaum Associates Inc., 1983

**Carroll et al. 1990**

CARROLL, John M. ; SINGER, Janice A. ; BELLAMY, Rachel K E. ; ALPERT, Sherman R.: A view matcher for learning Smalltalk. In: *CHI '90: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM Request Permissions, März 1990

**Charmaz 2006**

CHARMAZ, Kathy: *Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis*. SAGE Publications, 2006 (Constructing grounded theory)

**Clarke 2005a**

CLARKE, Adele ; CLARKE, Adele (Hrsg.): *Situational Analysis*. London : Sage Publications, 2005 (Grounded Theory After the Postmodern Turn)

**Clarke 2003**

CLARKE, Steven: *API usability and the cognitive dimensions framework - So what is a developer experience anyway?* <http://blogs.msdn.com/b/stevencl/archive/2003/10/08/57040.aspx>. Version: Oktober 2003

**Clarke 2004**

CLARKE, Steven: *Measuring API Usability.* <http://www.drdobbs.com/windows/measuring-api-usability/184405654#>. Version: Mai 2004

**Clarke 2005b**

CLARKE, Steven: Describing and measuring API usability with the cognitive dimensions. In: *Cognitive Dimensions of Notations 10th Anniversary Workshop*, 2005

**Clarke 2007**

CLARKE, Steven: What is an End User Software Engineer?. In: *End-User Software Engineering 2007* (2007)

**Clarke u. Becker 2003**

CLARKE, Steven ; BECKER, C: Using the cognitive dimensions framework to evaluate the usability of a class library. In: *Proceedings of the First Joint Conference of EASE & PPIG 2003*, 2003, S. 359–366

**Cooper 2004**

COOPER, Alan: *The Inmates Are Running the Asylum: Why High-tech Products Drive Us Crazy and how to Restore the Sanity.* Sams Publishing, Februar 2004

**Corbin u. Strauss 2014**

CORBIN, Juliet M. ; STRAUSS, Anselm L.: *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory.* SAGE Publications, 2014

**Correia et al. 2009**

CORREIA, Filipe F. ; AGUIAR, Ademar ; FERREIRA, Hugo S. ; FLORES, Nuno: Patterns for consistent software documentation. In: *PLoP '09: Proceedings of the 16th Conference on Pattern Languages of Programs*. New York, New York, USA : ACM Request Permissions, August 2009, S. 1

**Corritore u. Wiedenbeck 1999**

CORRITORE, Cynthia ; WIEDENBECK, Susan: Mental representations of expert procedural and object-oriented programmers in a software maintenance task. In: *International Journal of Human-Computer Studies* 50 (1999), Januar, Nr. 1, S. 61–83

**Crosby et al. 2002**

CROSBY, Martha E. ; SCHOLTZ, Jean ; WIEDENBECK, Susan: The Roles Beacons Play in Comprehension for Novice and Expert Programmers. In: *PPIG 2002 - Programmers, 14th Workshop of the Psychology of Programming Interest Group* Brunel University, 2002, S. 18–21

**Cwalina u. Abrams 2008**

CWALINA, Krzysztof ; ABRAMS, Brad: *Framework design guidelines: conventions, idioms, and patterns for reusable. net libraries.* Pearson Education, 2008

**Dagenais u. Ossher 2008**

DAGENAIS, Barthelemy ; OSSHER, Harold: Automatically locating framework extension examples. In: *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering.* New York, New York, USA : ACM Request Permissions, November 2008, S. 203–213

**Dahotre et al. 2011**

DAHOTRE, Aniket ; KRISHNAMOORTHY, Vasanth ; CORLEY, Matt ; SCAFFIDI, Christopher: Using intelligent tutors to enhance student learning of application programming interfaces. In: *Journal of Computing Sciences in Colleges* 27 (2011), Oktober, Nr. 1, S. 195–201

**Daughtry et al. 2009a**

DAUGHTRY, John M. ; FAROOQ, Umer ; MYERS, Brad A. ; STYLOS, Jeffrey: API usability: report on special interest group at CHI. In: *SIGSOFT Softw. Eng. Notes* 34 (2009), Juli, Nr. 4, S. 27

**Daughtry et al. 2009b**

DAUGHTRY, John M. ; FAROOQ, Umer ; STYLOS, Jeffrey ; MYERS, Brad A.: API usability: CHI'2009 special interest group meeting. In: *CHI '09 Extended Abstracts on Human Factors in Computing Systems* (2009), April, S. 2771–2774

**Dekel 2011**

DEKEL, Uri: *Increasing Awareness of Delocalized Information to Facilitate Api Usage.* Proquest, UMI Dissertation Publishing, 2011

**Dekel u. Herbsleb 2009**

DEKEL, Uri ; HERBSLEB, James D.: Improving API documentation usability with knowledge pushing. In: *ICSE '09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering,* IEEE Computer Society, Mai 2009, S. 320–330

**Dohnert et al. 2014**

DOHNERT, Christian ; FAULRING, Andrew R. ; MYERS, Brad A.: EUKLAS. In: *PLATEAU '14: Proceedings of the 5th Workshop on Evaluation and Usability of Programming Languages and Tools.* New York, New York, USA : ACM Request Permissions, Oktober 2014, S. 13–20

**Dresing 2006**

DRESING, Thorsten: *MAXqda und Grounded Theory?* <http://www.maxqda.de/support/forum/viewtopic.php?f=6&t=291>. Version: Dezember 2006

**Duala-Ekoko u. Robillard 2011**

DUALA-EKOKO, Ekwa ; ROBILLARD, MartinP: Using Structure-Based Recommendations to Facilitate Discoverability in APIs. In: MEZINI, Mira (Hrsg.): *Lecture Notes in Computer Science.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2011, S. 79–104–104

**Eagan u. Stasko 2008**

EAGAN, James R. ; STASKO, John T.: The buzz: supporting user tailorability in awareness applications. In: *CHI '08: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, New York, USA : ACM Request Permissions, April 2008, S. 1729

**Eisenberg et al. 2010a**

EISENBERG, Daniel S. ; STYLOS, Jeffrey ; FAULRING, Andrew R. ; MYERS, Brad A.: Using Association Metrics to Help Users Navigate API Documentation. In: *2010 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, IEEE, 2010, S. 23–30

**Eisenberg et al. 2010b**

EISENBERG, Daniel S. ; STYLOS, Jeffrey ; MYERS, Brad A.: Apatite: a new interface for exploring APIs. In: *CHI '10: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, New York, USA : ACM Request Permissions, April 2010, S. 1331

**Ellis et al. 2007**

ELLIS, B ; STYLOS, Jeffrey ; MYERS, B: The Factory Pattern in API Design: A Usability Evaluation. In: *ICSE '07: Proceedings of the 29th international conference on Software Engineering* (2007), Mai, S. 302–312

**Fairbanks et al. 2006**

FAIRBANKS, George ; GARLAN, David ; SCHERLIS, William: Design fragments make using frameworks easier. In: *OOPSLA '06: Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. New York, New York, USA : ACM Request Permissions, Oktober 2006, S. 75–88

**Farooq et al. 2010**

FAROOQ, Umer ; WELICKI, Leon ; ZIRKLER, Dieter: API usability peer reviews: a method for evaluating the usability of application programming interfaces. In: *CHI '10: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, New York, USA : ACM Request Permissions, April 2010, S. 2327–2336

**Farooq u. Zirkler 2009**

FAROOQ, Umer ; ZIRKLER, Dieter: Scaling API Design Knowledge in Large Software Organizations. In: *CSCW '10: Proceedings of the 2010 ACM conference on Computer supported cooperative work*, 2009

**Farooq u. Zirkler 2010**

FAROOQ, Umer ; ZIRKLER, Dieter: API peer reviews: a method for evaluating usability of application programming interfaces. In: *CSCW '10: Proceedings of the 2010 ACM conference on Computer supported cooperative work*. New York, New York, USA : ACM Request Permissions, Februar 2010, S. 207

**Faulkner 2003**

FAULKNER, Laura: Beyond the Five-User Assumption: Benefits of Increased Sample Sizes in Usability Testing. In: *Behavior Research Methods* 35 (2003), April, Nr. 3

**Feilkas u. Ratiu 2008**

FEILKAS, M ; RATIU, D: Ensuring Well-Behaved Usage of APIs through Syntactic Constraints. In: *The 16th IEEE International Conference on Program Comprehension, 2008. ICPC 2008*, IEEE, Juni 2008, S. 248–253

**Fouts 2000**

FOUTS, Jason W.: On site: an “out-of-box” experience. In: *Communications of the ACM* 43 (2000), November, Nr. 11, S. 28–29

**Fowler 1999**

FOWLER, Martin: *Refactoring: improving the design of existing code*. Reading, MA : Addison-Wesley, 1999

**Fu et al. 2002**

FU, Limin ; SALVENDY, Gavriel ; TURLEY, Lori: Effectiveness of user testing and heuristic evaluation as a function of performance classification. In: *Behaviour & Information Technology* 21 (2002), Januar, Nr. 2, S. 137–143

**Furnas et al. 1987**

FURNAS, G W. ; LANDAUER, T K. ; GOMEZ, L M. ; DUMAIS, S T.: The vocabulary problem in human-system communication. In: *Communications of the ACM* 30 (1987), November, Nr. 11, S. 964–971

**Gagné 1985**

GAGNÉ, Robert M.: *The Conditions of Learning and Theory of Instruction*. Holt, Rinehart and Winston, 1985

**Gasparich u. Wimmers 2014**

GASPARICH, Gail E. ; WIMMERS, Larry: Integration of Ethics across the Curriculum: From First Year through Senior Seminar. In: *Journal of microbiology & biology education* 15 (2014), Dezember, Nr. 2, S. 218–223

**Gellenbeck u. Cook 1991**

GELLENBECK, Edward M. ; COOK, Curtis R.: An Investigation of Procedure and Variable Names as Beacons During Program Comprehension / Oregon State University Corvallis, OR, USA. 1991. – Forschungsbericht

**Gerken et al. 2011**

GERKEN, Jens ; JETTER, Hans-Christian ; ZÖLLNER, Michael ; MADER, Martin ; REITERER, Harald: The concept maps method as a tool to evaluate the usability of APIs. In: *CHI '11 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, New York, USA : ACM Request Permissions, Mai 2011, S. 3373

**Gibas u. Jambeck 2002**

GIBAS, Cynthia ; JAMBECK, Per: *Einführung in die praktische Bioinformatik*. O'Reilly, 2002

**Gieselmann 2014**

GIESELMANN, Hartmut: *Spiele-Entwicklungs-Umgebung Unity 5 bringt u.a. bessere Grafik und 64-Bit-Support.* [http://m.heise.de/developer/meldung/Spiele-Entwicklungs-Umgebung-Unity-5-bringt-u-a-bessere-Grafik-und-64-Bit-Support-2298586.html?wt\\_mc=rss-developer.beitrag.atom&from-classic=1](http://m.heise.de/developer/meldung/Spiele-Entwicklungs-Umgebung-Unity-5-bringt-u-a-bessere-Grafik-und-64-Bit-Support-2298586.html?wt_mc=rss-developer.beitrag.atom&from-classic=1). Version: August 2014

**Glaser 1978**

GLASER, Barney G.: *Theoretical Sensitivity: Advances in the Methodology of Grounded Theory.* Sociology Press, 1978 (Advances in the methodology of grounded theory)

**Glaser u. Strauss 1967**

GLASER, Barney G. ; STRAUSS, Anselm L.: *The Discovery of Grounded Theory: Strategies for Qualitative Research.* New York, NY : Aldine de Gruyter, 1967

**Glaser u. Strauss 2005**

GLASER, Barney G. ; STRAUSS, Anselm L.: *Grounded theory: Strategien qualitativer Forschung.* 2. Bern : Hans Huber, 2005

**Glass u. Vessey 1992**

GLASS, Robert L. ; VESSEY, Iris: Toward a taxonomy of software application domains: History. In: *Journal of Systems and Software* 17 (1992), Februar, Nr. 2, S. 189–199

**Glass et al. 2002**

GLASS, Robert L. ; VESSEY, Iris ; RAMESH, V: Research in software engineering: an analysis of the literature. In: *Information and Software Technology* 44 (2002), Juni, Nr. 8, S. 491–506

**Gogol-Döring 2009**

GOGOL-DÖRING, Andreas: *SqAn - A Generic Software Library for Sequence Analysis.* Freie Universität Berlin, Diss., November 2009

**Gogol-Döring u. Kahlert 2013**

GOGOL-DÖRING, Andreas ; KAHLERT, Björn: *Templatemetaprogrammierung und OOP in SqAn.* Dezember 2013

**Gogol-Döring u. Reinert 2009**

GOGOL-DÖRING, Andreas ; REINERT, Knut: *Biological Sequence Analysis Using the SqAn C++ Library.* CRC Press, 2009 (Chapman & Hall/CRC Mathematical and Computational Biology)

**Good et al. 1984**

GOOD, Michael D. ; WHITESIDE, John A. ; WIXON, Dennis R. ; JONES, Sandra J.: Building a user-derived interface. In: *Communications of the ACM* 27 (1984), Oktober, Nr. 10, S. 1032–1043

**Green 1989**

GREEN, Thomas R G.: Cognitive Dimensions of Notations. In: *People and Computers V* (1989), S. 443–460

**Green 1996**

GREEN, Thomas R G.: *An Introduction to the Cognitive Dimensions Framework*. MRC Applied Psychology Unit, 15 Chaucer Road, Cambridge CB2 2EF, UK, November 1996

**Green u. Blackwell 1998**

GREEN, Thomas R G. ; BLACKWELL, Alan F.: *Cognitive Dimensions of Information Artefacts: a tutorial* . <http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDtutorial.pdf>. Version: Oktober 1998

**Green et al. 1991**

GREEN, Thomas R G. ; PETRE, M ; BELLAMY, Rachel K E.: *Comprehensibility of Visual and Textual Programs*. Open University, Computer Assisted Learning Research Group, 1991

**Green et al. 1980**

GREEN, Thomas R G. ; SIME, M E. ; FITTER, M J.: The problems the programmer faces. In: *Ergonomics* 23 (1980), Nr. 9, S. 893–907

**Green u. Petre 1995**

GREEN, TRG ; PETRE, M: Usability Analysis of Visual Programming Environments: A ‘Cognitive Dimensions’ Framework. In: *Journal of Visual Languages & Computing* 7 (1995), Dezember, Nr. 2, S. 131–174

**Grill et al. 2012**

GRILL, Thomas ; POLACEK, Ondrej ; TSCHELIGI, Manfred: Methods towards API Usability: A Structural Analysis of Usability Problem Categories. In: *Human-Centered Software Engineering*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2012, S. 164–180

**Gross et al. 2011**

GROSS, P ; YANG, J ; KELLEHER, C: Dinah: An interface to assist non-programmers with selecting program code causing graphical output. In: *CHI ’11 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2011

**Gross u. Kelleher 2009**

GROSS, Paul ; KELLEHER, Caitlin: Non-programmers identifying functionality in unfamiliar code: strategies and barriers. In: *Journal of Visual Languages & Computing* 21 (2009), Dezember, Nr. 5, S. 263–276

**Gross u. Kelleher 2010**

GROSS, Paul ; KELLEHER, Caitlin: Toward transforming freely available source code into usable learning materials for end-users. In: *PLATEAU ’10: Evaluation and Usability of Programming Languages and Tools*. New York, New York, USA : ACM Press, 2010, S. 1–6

**Gruber 1993**

GRUBER, Thomas R.: A translation approach to portable ontology specifications. In: *Knowledge Acquisition* 5 (1993), Juni, Nr. 2, S. 199–220

**Hansen 2013**

HANSEN, A: *Bioinformatik: Ein Leitfaden für Naturwissenschaftler*. Birkhäuser Basel, 2013

**Hartmann 1991**

HARTMANN, Petra: Soziale Erwünschtheit im Interview. In: *Wunsch und Wirklichkeit*. Wiesbaden : Deutscher Universitätsverlag, 1991, S. 35–172

**Haselhoff 2010**

HASELHOFF, Vanessa: Konzeption der empirischen Untersuchung. In: *Patientenvertrauen in Krankenhäuser*. Wiesbaden : Gabler, 2010, S. 85–114

**Hayes u. Simon 1975**

HAYES, J R. ; SIMON, Herbert A.: *Psychological Differences Among Problem Isomorphs*. Carnegie-Mellon University. Department of Psychology, 1975

**Henning 2007**

HENNING, Michi: API design matters. In: *Queue* 5 (2007), Mai, Nr. 4, S. 24–36

**Henning 2009**

HENNING, Michi: API design matters. In: *Communications of the ACM* 52 (2009), Mai, Nr. 5, S. 46–56

**Holmes u. Murphy 2005**

HOLMES, Reid ; MURPHY, Gail C.: Using structural context to recommend source code examples. In: *ICSE '05: Proceedings of the 27th international conference on Software engineering*. New York, New York, USA : ACM Request Permissions, Mai 2005, S. 117

**Horie u. Chiba 2010**

HORIE, Michihiro ; CHIBA, Shigeru: Tool support for crosscutting concerns of API documentation. In: *AOSD '10: Proceedings of the 9th International Conference on Aspect-Oriented Software Development*. New York, New York, USA : ACM Press, 2010, S. 97–108

**Hou u. Pletcher 2010**

HOU, Daqing ; PLETCHER, David M.: Towards a better code completion system by API grouping, filtering, and popularity-based ranking. In: *RSSE '10: Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering*. New York, New York, USA : ACM Press, 2010, S. 26–30

**Hou et al. 2005**

HOU, Daqing ; WONG, Kenny ; HOOVER, H J.: What Can Programmer Questions Tell Us About Frameworks? In: *IWPC '05: Proceedings of the 13th International Workshop on Program Comprehension*, IEEE Computer Society, Mai 2005

**Hülsenbusch 2014**

HÜLSENBUSCH, Ralph: Cloud Computing. In: *iX* 2 (2014), Februar, Nr. Verlag Heinz Heise GmbH & Co. KG, S. 20

**Jeong et al. 2009**

JEONG, Sae Young S. ; XIE, Yingyu C. ; BEATON, Jack ; MYERS, Brad A. ; STYLOS, Jeffrey ; EHRET, Ralf ; KARSTENS, Jan ; EFEOGLU, Arkin ; BUSSE, Daniela K.: Improving Documentation for eSOA APIs through User Studies. In: *Human-Centered Software Engineering*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2009, S. 86–105

**Kadoda 2000**

KADODA, Gada: A Cognitive Dimensions view of the differences between designers and users of theorem proving assistants. In: *PPIG 2000: Proceedings of the Twelfth Annual Meeting of the Psychology of Programming Interest Group*, 2000, S. 33–44

**Kahlert 2011**

KAHLERT, Björn: *Verbesserung der Out-Of-Box-Experience in Saros mittels Heuristischer Evaluation und Usability-Tests*. Berlin, Germany, Freie Universität Berlin, Diplomarbeit, Januar 2011

**Kamkar 2010**

KAMKAR, Samy: *evercookie – never forget*. <http://samy.pl/evercookie/>. Version: Oktober 2010

**Keenan et al. 1999**

KEENAN, Susan L. ; HARTSON, H R. ; KAFURA, Dennis G. ; SCHULMAN, Robert S.: The Usability Problem Taxonomy: A Framework for Classification and Analysis. In: *Empirical Software Engineering* 4 (1999), Februar, Nr. 1, S. 71–104

**Kelle 1994**

KELLE, U: *Empirisch begründete Theoriebildung: zur Logik und Methodologie interpretativer Sozialforschung*. Deutscher Studien Verlag, 1994

**Kemerer u. Paulk**

KEMERER, C F. ; PAULK, M C.: The Impact of Design and Code Reviews on Software Quality: An Empirical Study Based on PSP Data. In: *IEEE Transactions on Software Engineering* 35, Nr. 4, S. 534–550

**Khajouei et al. 2011**

KHAJOUEI, R ; PEUTE, L W P. ; HASMAN, A ; JASPERS, M W M.: Classification and prioritization of usability problems using an augmented classification scheme. In: *Journal of Biomedical Informatics* 44 (2011), November, Nr. 6, S. 948–957

**Kintsch 1988**

KINTSCH, Walter: The role of knowledge in discourse comprehension: A construction-integration model. In: *Psychological Review* 95 (1988), Nr. 2, S. 163–182

**Ko u. Riche 2011**

KO, A J. ; RICHE, Y: The role of conceptual knowledge in API usability. In: *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2011* (2011), S. 173–176

**Ko et al. 2011**

KO, Andrew J. ; ABRAHAM, Robin ; BECKWITH, Laura ; BLACKWELL, Alan F. ; BURNETT, Margaret M. ; ERWIG, Martin ; SCAFFIDI, Christopher ; LAWRENCE, Joseph ; LIEBERMAN, Henry ; MYERS, Brad A. ; ROSSON, Mary B. ; ROTHERMEL, Gregg ; SHAW, Mary ; WIEDENBECK, Susan: The state of the art in end-user software engineering. In: *ACM Computing Surveys (CSUR)* 43 (2011), April, Nr. 3

**Ko u. Myers 2004**

KO, Andrew J. ; MYERS, Brad A.: Designing the whyline: a debugging interface for asking questions about program behavior. In: *CHI '04: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, New York, USA : ACM Request Permissions, April 2004, S. 151–158

**Ko u. Myers 2005**

KO, Andrew J. ; MYERS, Brad A.: Human factors affecting dependability in end-user programming. In: *SIGSOFT Softw. Eng. Notes* 30 (2005), Juli, Nr. 4, S. 1–4

**Ko et al. 2004**

KO, Andrew J. ; MYERS, Brad A. ; AUNG, Htet H.: Six Learning Barriers in End-User Programming Systems. In: *VLHCC '04: Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing*, IEEE Computer Society, September 2004

**Lange u. Moher 1989**

LANGE, B M. ; MOHER, T G.: Some strategies of reuse in an object-oriented programming environment. In: *ACM SIGCHI Bulletin* 20 (1989), März, Nr. SI, S. 69–73

**LaToza et al. 2007**

LA TOZA, Thomas D. ; GARLAN, David ; HERBSLEB, James D. ; MYERS, Brad A.: Program comprehension as fact finding. In: *ESEC-FSE '07: Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*. New York, New York, USA : ACM Request Permissions, September 2007, S. 361

**Layman et al. 2008**

LAYMAN, Lucas M. ; WILLIAMS, Laurie A. ; AMANT, Robert S.: MimEc: intelligent user notification of faults in the eclipse IDE. In: *CHASE '08: Proceedings of the 2008 international workshop on Cooperative and human aspects of software engineering*. New York, New York, USA : ACM, Mai 2008, S. 73–76

**Layzell u. Champion 1993**

LAYZELL, P J. ; CHAMPION, R: DOCKET: Program comprehension-in-the-large. In: *Proceedings of the IEEE Second Workshop on Program Comprehension, 1993*. Capri, Juli 1993, S. 140–148

**Lee u. Stepanov 1994**

LEE, Meng ; STEPANOV, Alexander: *The Standard Template Library*. <http://www>.

[stepanovpapers.com/Stepanov-The\\_Standard\\_Template\\_Library-1994.pdf](http://stepanovpapers.com/Stepanov-The_Standard_Template_Library-1994.pdf). Version: März 1994

### Letondal 2006

LETONDAL, Catherine: Participatory Programming: Developing Programmable Bioinformatics Tools for End-Users. In: *Human-Computer Interaction Series*. Dordrecht : Springer Netherlands, 2006, S. 207–242

### Lewins u. Silver 2007

LEWINS, Ann ; SILVER, Christina: *Using Software in Qualitative Research: A Step-by-Step Guide*. SAGE Publications, 2007

### Li et al. 2012

LI, Jing-Woei ; SCHMIEDER, Robert ; WARD, R M. ; DELENICK, Joann ; OLIVARES, Eric C. ; MITTELMAN, David: SEQanswers: an open access community for collaboratively decoding genomes. In: *Bioinformatics* 28 (2012), Mai, Nr. 9, S. 1272–1273

### Mayer et al. 2012

MAYER, Clemens ; HANENBERG, Stefan ; ROBBES, Romain ; TANTER, Éric ; STEFIK, Andreas: An empirical study of the influence of static type systems on the usability of undocumented software. In: *OOPSLA '12: Proceedings of the ACM international conference on Object oriented programming systems languages and applications*. New York, New York, USA : ACM Request Permissions, November 2012, S. 683–702

### Mayring 2002

MAYRING, Philipp: *Einführung in die qualitative Sozialforschung: Eine Anleitung zu qualitativem Denken*. Weinheim : Beltz Verlag, 2002

### McKeogh u. Exton 2004

MCKEOUGH, John ; EXTON, Chris: Eclipse plug-in to monitor the programmer behaviour. In: *eclipse '04: Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*. New York, New York, USA : ACM, Oktober 2004, S. 93–97

### McLellan et al. 1998

MCLELLAN, Samuel G. ; ROESLER, Alvin W. ; TEMPEST, Joseph T. ; SPINUZZI, Clay I.: Building More Usable APIs. In: *IEEE Software* 15 (1998), Mai, Nr. 3, S. 78–86

### Mey u. Mruck 2007

MEY, Günter ; MRUCK, Katja: Grounded Theory Methodologie - Bemerkungen zu einem prominenten Forschungsstil. In: *Historical Social Research, Supplement* (2007), Nr. 19, S. 11–39

### Mey u. Mruck 2011

MEY, Günter ; MRUCK, Katja ; MEY, Günter (Hrsg.) ; MRUCK, Katja (Hrsg.): *Grounded Theory Reader*. Wiesbaden : VS Verlag für Sozialwissenschaften, 2011

**Meyers 1992**

MEYERS, Scott: *Effective C++: 50 specific ways to improve your programs and designs.* Reading, MA : Addison-Wesley, 1992 (Addison-Wesley Professional Computing Series)

**Meyers 2001**

MEYERS, Scott: *Effective STL: 50 Specific Ways to Improve Your Use of the Standard Template Library.* Indianapolis, IN : Addison-Wesley, 2001

**Miller 1956**

MILLER, George A.: The magical number seven, plus or minus two: some limits on our capacity for processing information. In: *Psychological Review* 63 (1956), Nr. 2, S. 81–97

**Monperrus et al. 2011**

MONPERRUS, Martin ; EICHLBERG, Michael ; TEKES, Elif ; MEZINI, Mira: What should developers be aware of? An empirical study on the directives of API documentation. In: *Empirical Software Engineering* 17 (2011), Dezember, Nr. 6, S. 703–737

**Mühlmeyer-Mentzel 2011**

MÜHLMAYER-MENTZEL, A: The logical structure of data in ATLAS. ti and its advantage for grounded theory studies. In: *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research* 12 (2011), Januar, Nr. 1

**Neal 1989**

NEAL, L R.: A system for example-based programming. In: *CHI '89: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.* New York, New York, USA : ACM Request Permissions, März 1989, S. 63–68

**Needleman u. Wunsch 1970**

NEEDLEMAN, S B. ; WUNSCH, C D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. In: *Journal of molecular biology* 48 (1970), März, Nr. 3, S. 443–453

**Ng et al. 2007**

NG, T H. ; CHEUNG, S C. ; CHAN, W K. ; YU, Y T.: Do Maintainers Utilize Deployed Design Patterns Effectively? In: *ICSE '07: Proceedings of the 29th international conference on Software Engineering* (2007), Mai, S. 168–177

**Nielsen 1993**

NIELSEN, Jakob: *Usability Engineering.* 1st. Morgan Kaufmann, 1993

**Nielsen 2005**

NIELSEN, Jakob: *Heuristic Evaluation.* <http://www.useit.com/papers/heuristic/>. Version: 2005

**Nielsen u. Mack 1994**

NIELSEN, Jakob ; MACK, Robert L.: *Usability Inspection Methods.* New York : Wiley John + Sons, 1994

**Nielsen u. Molich 1990**

NIELSEN, Jakob ; MOLICH, Rolf: Heuristic evaluation of user interfaces. In: *CHI '90: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, New York, USA : ACM Request Permissions, März 1990, S. 249–256

**Noelle-Neumann 1989**

NOELLE-NEUMANN, Elisabeth: Die Theorie der Schweigespirale als Instrument der Medienwirkungsforschung. In: *Massenkommunikation*. Wiesbaden : VS Verlag für Sozialwissenschaften, 1989, S. 418–440

**Nykaza et al. 2002**

NYKAZA, J ; MESSINGER, R ; BOEHME, F: What programmers really want: results of a needs assessment for SDK documentation. In: *SIGDOC '02: Proceedings of the 20th annual international conference on Computer documentation*, 2002

**Oates 2006**

OATES, Briony J.: *Researching Information Systems and Computing*. SAGE Publications, 2006

**O'Callaghan 2010**

O'CALLAGHAN, Portia: The API walkthrough method. In: *PLATEAU '10: Evaluation and Usability of Programming Languages and Tools*. New York, New York, USA : ACM Press, 2010, S. 1–6

**Omar et al. 2012**

OMAR, Cyrus ; YOON, Young S. ; LATOZA, Thomas D. ; MYERS, Brad A.: Active code completion. In: *ICSE '12: Proceedings of the 34th International Conference on Software Engineering*. Zürich : IEEE, Juni 2012, S. 859–869

**Oney u. Brandt 2012**

ONEY, Stephen ; BRANDT, Joel: Codelets: linking interactive documentation and example code in the editor. In: *CHI '12: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, New York, USA : ACM Request Permissions, Mai 2012, S. 2697

**Oppermann u. Reiterer 1992**

OPPERMANN, Reinhart ; REITERER, Harald: Der Evaluationsleitfaden EVADIS II. In: *Ergonomie Informatik* 15 (1992), S. 25–29

**Parnas 2011**

PARNAS, David L.: *Precise Documentation: The Key to Better Software*. Springer Berlin Heidelberg, 2011

**Parnin u. Treude 2011**

PARNIN, Chris ; TREUDE, Christoph: Measuring API documentation on the web. In: *Web2SE '11: Proceedings of the 2nd International Workshop on Web 2.0 for Software Engineering*. New York, New York, USA : ACM Request Permissions, Mai 2011, S. 25–30

**Pennington 1987**

PENNINGTON, Nancy: Stimulus structures and mental representations in expert comprehension of computer programs. In: *Cognitive Psychology* 19 (1987), Juli, Nr. 3, S. 295–341

**Piccioni et al.**

PICCIONI, Marco ; FURIA, Carlo A. ; MEYER, Bertrand: An Empirical Study of API Usability. In: *2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, IEEE, S. 5–14

**Plauger 2001**

PLAUGER, P J.: *The C++ Standard Template Library*. Bd. 42. Prentice Hall, 2001

**Polya 1957**

POLYA, George: *How to Solve it: A New Aspects of Mathematical Method*. 2nd edition. Princeton University Press, 1957

**Pruitt u. Grudin 2003**

PRUITT, John ; GRUDIN, Jonathan: Personas. In: *DUX '03: Conference on Designing for User Experiences*. New York, New York, USA : ACM Press, 2003, S. 1–15

**Pugh 2006**

PUGH, Ken: *Interface-Oriented Design (Pragmatic Programmers)*. <http://www.amazon.de/Interface-Oriented-Design-Pragmatic-Programmers-Pugh/dp/0976694050>. Version: Juli 2006

**Rarey 2010**

RAREY, Matthias: *Was ist Informatik?* <http://www.zbh.uni-hamburg.de/informationen-fuer-schueler-innen/was-ist-bioinformatik.html>. Version: 2010

**Reardon 2008**

REARDON, Martin: *Planning for Phases of Learning*. [http://www.people.vcu.edu/~rmreardon/706/Models%20of%20Teaching/Gagne\\_phases.htm](http://www.people.vcu.edu/~rmreardon/706/Models%20of%20Teaching/Gagne_phases.htm). Version: September 2008

**Rehfus 2003**

REHFUS, Wulff D.: *Handwörterbuch Philosophie*. Göttingen : Vandenhoeck & Ruprecht, 2003

**Reid 2014**

REID, John E.: STEME: Suffix arrays for probabilistic motif finding. (2014)

**Reinert 2011**

REINERT, Knut: *Pressemitteilung: Ein App-Store für DNA-Analyse: BMBF fördert Informatikprojekt der Freien Universität Berlin*. Freie Universität Berlin, 2011

**Reinert et al. 2011**

REINERT, Knut ; WEESE, David ; GRÖPL, Clemens: Algorithmen und Datenstrukturen für Bioinformatik. In: *Freie Universität Berlin*, 2011, S. 28

**Reinert et al. 2014**

REINERT, Knut ; WEESE, David ; HOLTEGREWE, Manuel ; SINGER, Jochen ; KRAKAU, Sabrina ; KAHLERT, Björn: *Abschlussbericht VIP-Projekt BioStore*. <https://docs.google.com/document/d/1A8ekjPMsRr-Iag2UiS97Mz2XMc0GpZc6jcT3Mbf1dTI/edit?usp=sharing>. Version: 2014

**Roast et al. 2000**

ROAST, C R. ; KHAZAEI, B ; SIDDIQI, J I.: Formal comparisons of program modification. In: *VL 2000 IEEE Symposium on Visual Languages*, IEEE Comput. Soc, 2000, S. 165–171

**Roberts u. Johnson 1997**

ROBERTS, Don ; JOHNSON, Ralph E.: Patterns for evolving frameworks. In: MARTIN, Robert C. (Hrsg.) ; RIEHLE, Dirk (Hrsg.) ; BUSCHMANN, Frank (Hrsg.): *Pattern languages of program design* 3. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1997, S. 471–486

**Robertson 2007**

ROBERTSON, Scott M.: Postsecondary Education & Autism: Developing an Online Community. In: *VLHCC '07: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, IEEE Computer Society, September 2007, S. 50–60

**Robillard 2009**

ROBILLARD, Martin P.: What Makes APIs Hard to Learn? Answers from Developers. In: *IEEE Software* 26 (2009), Dezember, Nr. 6, S. 27–34

**Robillard u. DeLine 2010**

ROBILLARD, Martin P. ; DELINE, Robert: A field study of API learning obstacles. In: *Empirical Software Engineering* 16 (2010), Nr. 6, S. 703–732

**Rosson et al. 2005**

ROSSON, M B. ; BALLIN, J ; RODE, J: Who, what, and how: a survey of informal and professional Web developers. In: *VLHCC '05: Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, 2005, S. 199–206

**Rosson u. Carroll 1996**

ROSSON, Mary B. ; CARROLL, John M.: The reuse of uses in Smalltalk programming. In: *ACM Transactions on Computer-Human Interaction* 3 (1996), September, Nr. 3, S. 219–253

**Rosson u. Carroll 2001**

ROSSON, Mary B. ; CARROLL, John M.: *Usability Engineering: Scenario-Based Development of Human-Computer Interaction (Interactive Technologies)*. 1. Morgan Kaufmann, 2001

**Salinger 2013**

SALINGER, Stephan: *Ein Rahmenwerk für die qualitative Analyse der Paarprogrammierung*. Berlin, Diss., 2013

**Sarodnick u. Brau 2006**

SARODNICK, Florian ; BRAU, Henning: *Methoden der Usability Evaluation: Wissenschaftliche Grundlagen und praktische Anwendung*. 1. Auflage. Bern : Verlag Hans Huber, Hogrefe AG, 2006

**Schenk 2014**

SCHENK, Julia: *Besprechung ATLAS.ti mit Julia Schenk*. Juli 2014

**Schiffer 1998**

SCHIFFER, S: *Visuelle Programmierung*. Addison-Wesley-Longman, 1998

**Schmidt u. Buschmann**

SCHMIDT, D C. ; BUSCHMANN, F: Patterns, frameworks, and middleware: their synergistic relationships. In: *25th International Conference on Software Engineering, 2003. Proceedings.*, IEEE, S. 694–704

**Schmidt 2014**

SCHMIDT, Julia: *C++17 soll Großes bringen*. [http://www.heise.de/newsticker/meldung/C-17-soll-Grosses-bringen-2300999.html?wt\\_mc=rss.ho.beitrag.atom](http://www.heise.de/newsticker/meldung/C-17-soll-Grosses-bringen-2300999.html?wt_mc=rss.ho.beitrag.atom). Version: August 2014

**Schweibenz u. Thissen 2003**

SCHWEIBENZ, Werner ; THISSEN, Frank: *Qualität im Web*. Springer, 2003 (Benutzerfreundliche Webseiten durch Usability-Evaluation)

**Shaft u. Vessey 1998**

SHAFT, Teresa M. ; VESSEY, Iris: The Relevance of Application Domain Knowledge: Characterizing the Computer Program Comprehension Process. In: *Journal of Management Information Systems* 15 (1998), Juni, Nr. 1, S. 51–78

**Shi et al. 2011**

SHI, Lin ; ZHONG, Hao ; XIE, Tao ; LI, Mingshu: An Empirical Study on Evolution of API Documentation. In: GIANNAKOPOULOU, Dimitra (Hrsg.) ; OREJAS, Fernando (Hrsg.): *Lecture Notes in Computer Science*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2011, S. 416–431–431

**Shneiderman 1977**

SHNEIDERMAN, Ben: Measuring computer program quality and comprehension. In: *International Journal of Man-Machine Studies* 9 (1977), 1977, S. 465–478

**Shneiderman u. Mayer 1979**

SHNEIDERMAN, Ben ; MAYER, Richard: Syntactic/semantic interactions in programmer behavior: A model and experimental results. In: *International Journal of Computer & Information Sciences* 8 (1979), Juni, Nr. 3, S. 219–238

**de Souza u. Bentolila 2009**

SOUZA, C R B. ; BENTOLILA, D L M.: Automatic evaluation of API usability using complexity metrics and visualizations. In: *ICSE-Companion 2009. 31st International Conference on Software Engineering - Companion Volume, 2009*, 2009, S. 299–302

**de Souza et al. 2004**

SOUZA, Cleidson R B. ; REDMILES, David ; CHENG, Li-Te ; MILLEN, David ; PATTERSON, John: Sometimes you need to see through walls: a field study of application programming interfaces.

In: *CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work.* New York, New York, USA : ACM Request Permissions, November 2004, S. 63–71

### Spinellis 2006

SPINELLIS, Diomidis: *The Bad Code Spotter's Guide*. <http://www.informit.com/articles/article.aspx?p=457502>. Version: April 2006

### Spoehr et al. 1984

SPOEHR, K T. ; MORRIS, M E. ; SMITH, E E.: Comprehension of Instructions for Operating Devices. 1984. – Forschungsbericht

### Steinke 1999

STEINKE, Ines: *Kriterien qualitativer Forschung : Ansätze zur Bewertung qualitativ-empirischer Sozialforschung*. Weinheim : Juventa Verlag, 1999 (Juventa Paperback)

### Stiemerling et al. 1997

STIEMERLING, Oliver ; KAHLER, Helge ; WULF, Volker: How to make software softer—designing tailorable applications. In: *DIS '97: Proceedings of the 2nd conference on Designing interactive systems: processes, practices, methods, and techniques*. New York, New York, USA : ACM Request Permissions, August 1997, S. 365–376

### Strauss 1987

STRAUSS, Anselm L.: *Qualitative Analysis for Social Scientists*. Cambridge University Press, 1987

### Strauss u. Corbin 1990

STRAUSS, Anselm L. ; CORBIN, Juliet M.: *Basics of qualitative research: grounded theory procedures and techniques*. Sage Publications, 1990

### Strauss u. Corbin 1996

STRAUSS, Anselm L. ; CORBIN, Juliet M.: *Grounded Theory: Grundlagen qualitativer Sozialforschung*. Beltz, Psychologie-Verlag-Union, 1996

### Strübing 2005

STRÜBING, Jörg: *Pragmatistische Wissenschafts- und Technikforschung*. Campus Verlag, 2005 (Theorie und Methode)

### Stylos 2006

STYLOS, Jeffrey: Informing API Design through Usability Studies of API Design Choices: A Research Abstract. In: *VLHCC '06: Proceedings of the 2006 IEEE Symposium on Visual Languages - Human Centric Computing*, IEEE, September 2006, S. 246–247

### Stylos 2009

STYLOS, Jeffrey: *Making apis more usable with improved api designs, documentation and tools*. Pittsburgh, PA, USA : Carnegie Mellon University, 2009

### Stylos u. Clarke 2007

STYLOS, Jeffrey ; CLARKE, Steven: Usability Implications of Requiring Parameters in Objects'

Constructors. In: *ICSE '07: Proceedings of the 29th international conference on Software Engineering*, IEEE Computer Society, Mai 2007, S. 529–539

### Stylos et al. 2006

STYLOS, Jeffrey ; CLARKE, Steven ; MYERS, B: Comparing API design choices with usability studies: A case study and future directions. In: *Proceedings of the 18th Annual PPIG - PPIG 2006*, 2006

### Stylos et al. 2009a

STYLOS, Jeffrey ; FAULRING, Andrew ; YANG, Zizhuang ; MYERS, Brad A.: Improving API documentation using API usage information. In: *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2009), S. 119–126

### Stylos et al. 2008

STYLOS, Jeffrey ; GRAF, B ; BUSSE, D K. ; ZIEGLER, C ; EHRET, R ; KARSTENS, J: A case study of API redesign for improved usability. In: *IEEE Symposium on Visual Languages and Human-Centric Computing. Proceedings* (2008), September, S. 189–192

### Stylos u. Myers 2006

STYLOS, Jeffrey ; MYERS, Brad A.: Mica: A Web-Search Tool for Finding API Components and Examples. In: *VLHCC '06: Proceedings of the 2006 IEEE Symposium on Visual Languages - Human Centric Computing*, IEEE, September 2006, S. 195–202

### Stylos u. Myers 2008

STYLOS, Jeffrey ; MYERS, Brad A.: The implications of method placement on API learnability. In: *SIGSOFT '08/FSE-16: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. New York, New York, USA : ACM Press, 2008, S. 105–112

### Stylos et al. 2009b

STYLOS, Jeffrey ; MYERS, Brad A. ; YANG, Zizhuang: Jadeite. In: *CHI '09 Extended Abstracts on Human Factors in Computing Systems*. New York, New York, USA : ACM Press, April 2009, S. 4429

### Suddaby 2006

SUDDABY, Roy: From the Editors: What Grounded Theory is Not. In: *Academy of Management Journal* 49 (2006), August, Nr. 4, S. 633–642

### Sunshine et al. 2014

SUNSHINE, Joshua ; HERBSLEB, James D. ; ALDRICH, Jonathan: *Searching the State Space: A Qualitative Study of API Protocol Usability*. <http://www.cs.cmu.edu/~jssunshi/pubs/icse14-searching.pdf>. Version: 2014

### Teasley 1993

TEASLEY, Barbee E.: The effects of naming style and expertise on program comprehension. In: *International Journal of Human-Computer Studies* 40 (1993), Dezember, Nr. 5, S. 757–770

**Tenny 1988**

TENNY, T.: Program Readability: Procedures Versus Comments. In: *IEEE Transactions on Software Engineering* 14 (1988), September, Nr. 9, S. 1271–1279

**Tisdall 2001**

TISDALL, James: *Why Biologists Want to Program Computers*. [http://www.oreillynet.com/pub/a/oreilly/news/perlbio\\_1001.html](http://www.oreillynet.com/pub/a/oreilly/news/perlbio_1001.html). Version: Oktober 2001

**Vayena et al. 2015**

VAYENA, Effy ; BROWNSWORD, Roger ; EDWARDS, Sarah J. ; GRESHAKE, Bastian ; KAHN, Jeffrey P. ; LADHER, Navjoyt ; MONTGOMERY, Jonathan ; O'CONNOR, Daniel ; O'NEILL, Onora ; RICHARDS, Martin P. ; RID, Annette ; SHEEHAN, Mark ; WICKS, Paul ; TASIOULAS, John: Research led by participants: a new social contract for a new kind of research. In: *Journal of Medical Ethics* (2015), März

**Watson 2009**

WATSON, Robert B.: Improving software API usability through text analysis: A case study. In: *2009 IEEE International Professional Communication Conference*, IEEE, 2009, S. 1–7

**Watson 2012**

WATSON, Robert B.: Development and application of a heuristic to assess trends in API documentation. In: *SIGDOC '12: Proceedings of the 30th ACM international conference on Design of communication*. New York, New York, USA : ACM Press, 2012, S. 295

**Wharton et al. 1994**

WHARTON, Cathleen ; RIEMAN, John ; LEWIS, Clayton ; POLSON, Peter: The cognitive walkthrough method: a practitioner's guide. In: *Usability Inspection Methods*. New York : Wiley John + Sons, 1994, S. 105–140

**Wiedenbeck 1986**

WIEDENBECK, Susan: Beacons in computer program comprehension. In: *International Journal of Man-Machine Studies* 25 (1986), Dezember, Nr. 6, S. 697–709

**Wiedenbeck 1991**

WIEDENBECK, Susan: The Initial Stage of Program Comprehension. In: *International Journal of Man-Machine Studies* 35 (1991), Nr. 4, S. 517–540

**Wiedenbeck 2005**

WIEDENBECK, Susan: Facilitators and Inhibitors of End-User Development by Teachers in a School Environment. In: *VLHCC '05: Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, IEEE Computer Society, September 2005

**Wightman et al. 2012**

WIGHTMAN, Doug ; YE, Zi ; BRANDT, Joel ; VERTEGAAL, Roel: SnipMatch: using source code context to enhance snippet retrieval and parameterization. In: *UIST '12: Proceedings of the 25th annual ACM symposium on User interface software and technology*. New York, New York, USA : ACM Request Permissions, Oktober 2012, S. 219

**Wingchen 2014**

WINGCHEN, Jürgen: *Kommunikation und Gesprächsführung für Pflegeberufe*. 3. Edition. Schlütersche Verlagsgesellschaft, 2014 (Grundlagen & Umsetzung. Modelle & Strategien. Für Lehre & Praxis.)

**Wolf 2014**

WOLF, Nicholas: *Using Quotation Names for Coding: An Illustration From Grounded Theory*. <https://atlastiblog.wordpress.com/2014/03/26/1608/>. Version: März 2014

**Yamashita 2012**

YAMASHITA, Aiko: *Measuring the outcomes of a maintenance project: Technical details and protocols*. <https://www.simula.no/publications/measuring-outcomes-maintenance-project-technical-details-and-protocols>. Version: Mai 2012

**Yamashita u. Moonen**

YAMASHITA, Aiko ; MOONEN, Leon: Do developers care about code smells? An exploratory survey. In: *2013 20th Working Conference on Reverse Engineering (WCORE)*, IEEE, S. 242–251

**Yamashita u. Moonen 2013**

YAMASHITA, Aiko ; MOONEN, Leon: Exploring the impact of inter-smell relations on software maintainability: an empirical study. In: *Proceedings of the International Conference on Software Engineering* (2013), S. 682–691

**Ye u. Fischer 2002**

YE, Yunwen ; FISCHER, Gerhard: Supporting reuse by delivering task-relevant and personalized information. In: *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*. New York, New York, USA : ACM, Mai 2002, S. 513

**Young 2014**

YOUNG, Susan: *Industrieprodukt Genomsequenz*. <http://www.heise.de/tr/artikel/Industrieprodukt-Genomsequenz-2084085.html>. Version: Januar 2014

**Zhang et al.**

ZHANG, Cheng ; YANG, Juyuan ; ZHANG, Yi ; FAN, Jing ; ZHANG, Xin ; ZHAO, Jianjun ; OU, Peizhao: Automatic parameter recommendation for practical API usage. In: *ICSE '12: Proceedings of the 34th International Conference on Software Engineering*, IEEE, S. 826–836

**Zibran et al. 2011**

ZIBRAN, M F. ; EISHITA, F Z. ; ROY, C K.: Useful, But Usable? Factors Affecting the Usability of APIs. In: *Working Conference on Reverse Engineering. Proceedings* (2011), September, S. 151–155

**Zibran 2008**

ZIBRAN, Minhaz F.: **What Makes APIs Difficult to Use?** In: *IJCSNS International Journal of Computer Science and Network Security* 8 (2008), April, Nr. 4, S. 255–261

**Ziegler 2015**

ZIEGLER, Peter-Michael: *Biotechnologie statt Software: SAP-Gründer Hopp über seine Life-Sciences-Investments.* <http://www.heise.de/newsticker/meldung/Biotechnologie-statt-Software-SAP-Gruender-Hopp-ueber-seine-Life-Sciences-Investments-2596015.html>.

Version: April 2015

**Zieris 2014**

ZIERIS, Franz: *Besprechung ATLAS.ti mit Franz Zieris.* Oktober 2014