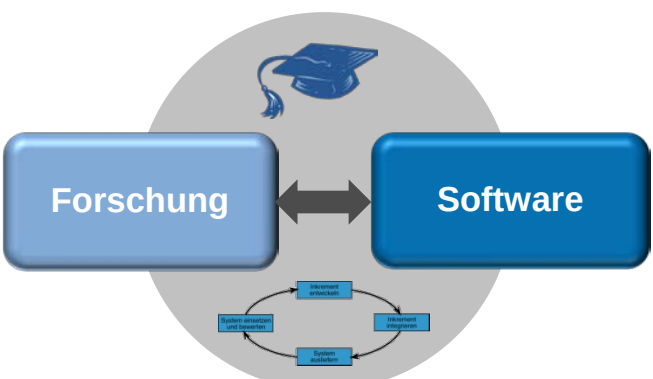
Proceedings des Workshops

Entwicklung und Evolution von Forschungssoftware

Rolduc, November 2011



Berichte der Aachener Informatik

Software Engineering Band 14

Hrsg.: Prof. Dr. rer nat. Bernhard Rumpe
Prof. Dr. rer. nat. Horst Lichter

Inhaltsverzeichnis

Einleitung und Vorwort

Horst Lichter

Lehr- und Forschungsgebiet Softwarekonstruktion
RWTH Aachen University
lichter@swc.rwth-aachen.de

1 Motivation für diesen Workshop

In vielen Forschungsprojekten werden Softwaresysteme erstellt, die gewisse Aufgabenstellungen im Kontext der Forschungsziele bearbeiten. Die Entwicklung dieser Softwaresysteme ist geprägt durch verschiedene Faktoren. So arbeiten typisch viele verschiedene Personen teils eher kurz an der Entwicklung der Software, typisch fehlt auch eine stabile und nutzbare Dokumentation. Während manche Forschungssoftware nur kurzfristig benutzt wird und dann nicht mehr gebraucht wird, gibt es viele Beispiele für Forschungssoftware, die sich über Jahre zu einer wichtigen Forschungsplattform entwickelt haben, bzw. die sogar als Produkte vertrieben werden.

Die forschungsgetriebene Entwicklung von Software muss aufgrund der Randbedingungen und dieser speziellen Probleme offensichtlich grundsätzlich anders organisiert werden, als ein klassisches Auftragsentwicklungsprojekt.

Da die Investitionen in Forschungssoftware sehr erheblich sind (auch, wenn diese selten gemessen und bekannt sind) und Forschungssoftware einen wichtigen Beitrag zu unseren Forschungsergebnissen liefert, entstand die Idee, einen Workshop zum Thema „Entwicklung und Evolution von Forschungssoftware“ (EEFSW) zu organisieren und durchzuführen.

2 Merkmale von Forschungssoftware

Forschungssoftware selbst, aber auch die Projekte, in denen diese entstehen, sind sehr unterschiedlich. Sie unterscheiden sich in der Größe, der Projektdauer, der Anwendungsdomäne und der eingesetzten Technologie.

Die Entwicklung von Forschungssoftware ist mit spezifischen Problemen verbunden. Erste Betrachtungen zu den Problemen bei der Entwicklung von Forschungssoftware hat Segal gemacht [Seg05]. Demnach unterliegt der Software-Entwicklungsprozess in Forschung und Entwicklung besonderen Anforderungen und Beschränkungen. Den klassischen Prozessmodellen der Software-Entwicklung ist gemein, dass zu Beginn der Entwicklung das Ziel – zumindest den Anwendern – grob bekannt ist. Dies ist bei der Entwicklung von Forschungssoftware typischerweise nicht der Fall. Auch ist die Software nicht fertig, wenn sie das erste Mal betrieben wird. Vielmehr entstehen weitere Anforderungen, die durchaus widersprüchlich zu den ursprünglichen Anforderungen sein können.

Allen Entwicklungsprojekten von Forschungssoftware gemeinsam sind nach Hoffmann et al. [HLN10] die beiden folgenden Merkmale

- **Sie sind in eines oder mehrere Forschungsprojekte eingebettet.**
  Dies hat Auswirkungen auf die zu entwickelnde Software, weil sich der Schwerpunkt und die Forschungsziele verschieben können. Es liegt in der Natur von Forschung, dass angestrebte Ziele nicht erreicht werden können, oder dass Ergebnisse, die erzielt wurden, einer Validierung nicht Stand halten. Dies führt dazu, dass in diesen Fällen die dazu entwickelte Forschungssoftware massiv umgearbeitet werden muss.

- **An der Entwicklung von Forschungssoftware sind Studierende massiv beteiligt.**
  Viele Teilfunktionen von Forschungssoftware werden im Kontext von Abschlussarbeiten durch Studierende entwickelt. In vielen Fällen arbeiten weiterhin studentische Hilfskräfte an der Entwicklung mit. Dies führt zu zwei sehr charakteristischen Herausforderungen.
  a) Die Studierenden kennen typischerweise die eingesetzte Technologie, die softwaretechnischen Methoden und Arbeitsweisen sowie die angewandte Vorgehensweise im Entwicklungsprojekt nicht.
  b) Die Mitarbeit der Studierenden im Projekt ist zeitlich beschränkt und in der Regel recht kurz (typisch 6 bis 9 Monate). So haben die Studierenden häufig keine Chance, das Entwicklungsprojekt als Ganzes zu überblicken und sind in erster Linie darauf bedacht, Ergebnisse zu erarbeiten, die ihre Ziele unterstützen (d.h., die Abschlussarbeit erfolgreich abzuschließen). Globale Anforderungen seitens des Projektes können dabei zu kurz kommen.

Diese Merkmale und Herausforderungen muss ein Projekt, in dem Forschungssoftware entwickelt werden soll, kennen und annehmen und es muss angemessen darauf reagieren. Klassische Vorgehensmodelle bieten hier wenige Antworten. Techniken und Organisationsformen der agilen Softwareentwicklung sind jedoch ein Fundus, um angemessene Vorgehensweisen für die Entwicklung von Forschungssoftware zu etablieren. Wie in vielen anderen Bereichen gilt aber auch hier, dass es nicht die eine Vorgehensweise geben kann, die für jede Projektsituation geeignet ist. Gesucht sind also Best-Practices und Vorgehensrahmen, die projektspezifisch konfiguriert werden können.

3 Ziele und zentrale Fragestellungen für den Workshop

Da es sich um den ersten Workshop zu diesem Thema handelt, sollte er weitest gehend zieloffen und interaktiv durch die Teilnehmer bestimmt sein. Auf Basis der formulierten spezifischen Merkmale von Forschungssoftware war jedoch ein denkbares Ziel des Workshops, eine erste Sammlung von Best-Practices-Techniken zu erarbeiten, die bei der Entwicklung von Forschungssoftware verwendet werden kann.

Folgende Fragestellungen sollten u.a. mit diesem Workshop adressiert werden:

- Welche verschiedenen Entwicklungsszenarien gibt es für Forschungssoftware?
- Welche positiven und negativen Erfahrungen mit Techniken etc. gibt es?
- Welche Techniken etc. haben sich unter welchen Randbedingungen bewährt?

- Wie können die Übergänge von Forschungsprototyp zu Forschungssoftware bzw. von Forschungssoftware zum Produkt gestaltet werden?

Als Input und Kommunikationsbasis wurden Projektbeschreibungen sowie Projektpräsentationen durch die Teilnehmer erstellt. Die folgenden Projekte wurden vorgestellt und diskutiert:

Projekt	
HTR Code Package (HCP)	Institut für Nukleare Entsorgung und Reaktorsicherheit, FZ Jülich
libAlf	Lehrstuhl Logik und Theorie diskreter Systeme
MeDIC	LuFG Softwarekonstruktion
SimWeld	Institut für Schweißtechnik und Fügetechnik
NCProfiler	Fraunhofer IPT
OpenFlipper	Lehrstuhl Computergraphik und Multimedia
ViSTA	Forschungsgruppe Virtuelle Realität
AProVE	LuFG Programmiersprachen und Verifikation
Scalasca	Lehrstuhl Parallele Programmierung
Energie Navigator	Lehrstuhl Software Engineering
Score-P	Jülich Super Computing Center (JSC)
SSELab	Lehrstuhl Software Engineering

4 Aufbau des Berichtes

Dieser Bericht dokumentiert den EEFSW-Workshop. Im ersten Teil sind kurze Projektpräsentationen der meisten oben aufgeführten Projekte enthalten. Im abschließenden zweiten Teil haben wir die im Workshop erarbeiteten Ergebnisse zusammengefasst.

Die Organisatoren des EEFSW-Workshops möchten sich ganz herzlich bei den Teilnehmern bedanken, die sehr intensiv und konstruktiv den Workshop mitgestaltet haben.

Außerdem möchten wir dem Forum Informatik der RWTH Aachen danken, das die Kosten für die Ausrichtung des EEFSW-Workshops übernommen hat.

5 Literatur

[HLN10] Hoffmann, V., H. Lichter, A. Nyßen (2010): Processes and Practices for Quality Scientific Software Projects.In H. Kienle (Ed.) Proceedings of 3rd International Workshop on Academic Software Development Tools WASDeTT-3, Antwerp, pp 95-108.

[Seg05] Segal, J.: When Software Engineers Met Research Scientists: A Case Study. In: Empirical Software Engineering, Band 10, Seiten 517–536, Hingham, MA, USA, 2005. Kluwer Academic Publishers.

Teil 1

Projektbeschreibungen

Das HTR Code Package (HCP)

Sarah Scholthaus und Daniela Lambertz

Institut für Nukleare Entsorgung und Reaktorsicherheit (IEK-6)
Forschungszentrum Jülich
Email: s.scholthaus@fz-juelich.de, d.lambertz@fz-juelich.de

1 Zusammenfassung des Projekts

Das HTR Code Package (HCP)[1] ist ein Projekt am Institut für Nukleare Entsorgung und Reaktorsicherheit (IEK-6) des Forschungszentrums in Jülich. In mehr als 40 Jahren Forschung im Bereich der Reaktorsicherheit sind dort eine Vielzahl einzelner etablierter und validierter Simulationsprogramme entstanden. Das HCP vereinigt diese zu einem integrierten Code-System und erlaubt damit detailliertere Untersuchungen auf den Gebieten der Neutronenphysik, Thermohydraulik, Spaltproduktfreisetzung, sowie der Staub- und Spaltproduktverteilung.

Abbildung 1 zeigt das grundlegende Konzept des HCP. Das sogenannte HCP Backbone beinhaltet die Programmsteuerung, das Datenmodell sowie die Ein- und Ausgabe. Nukliddaten, die als Basis für den Aufbau des Datenmodells benötigt werden, werden über die Datenbibliothek zur Verfügung gestellt. An das HCP-Backbone können dann über eine Schnittstelle die einzelnen Programme in Form von Modulen angekoppelt werden.

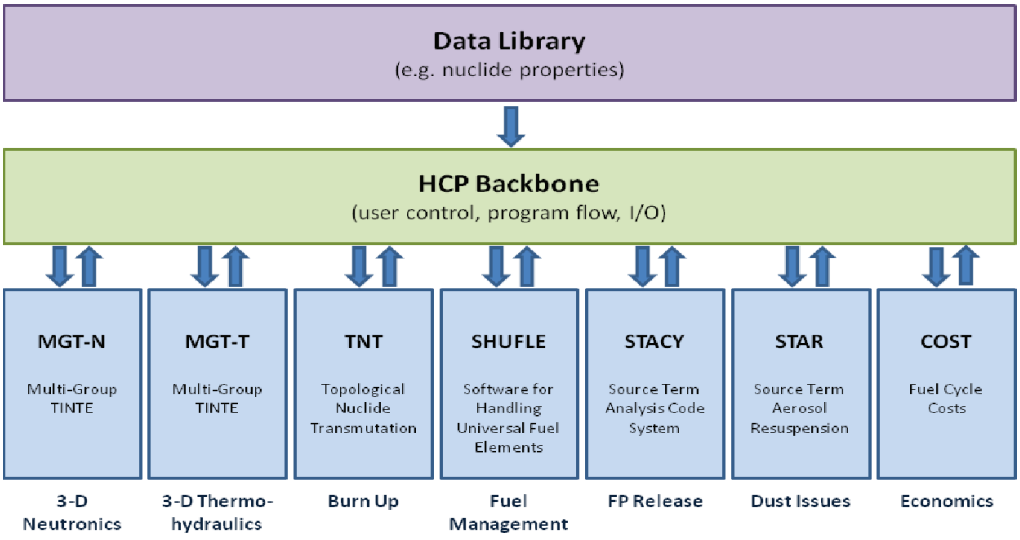


Fig. 1. Konzept des HCP

[1] siehe auch: H.-J. ALLELEIN, S. KASSELMANN, A. XHONNEUX, S.-C. HERBER: *Progress on the development of a fully integrated HTR Code Package*
Forschungszentrum Jülich, IEK-6 und RWTH Aachen, Lehrstuhl für Reaktorsicherheit und -technik, 2011, Paper-Nr.: NED-HTR-25

1.1 Ziele

Durch die Kopplung der bisher separierten Programme wird die Funktionalität erweitert und die Untersuchung neuer Fragestellungen ermöglicht. Zudem werden Code-Dopplungen zurückgebaut um die Wartbarkeit zu erhöhen. Durch die Verwendung aktueller Programmiertechniken und -standards, wie z.B. Objektorientierung und *XML (Extensible Markup Language)*, wird die Lesbarkeit des Quellcodes erhöht. Zusätzlich bewirkt die Ausnutzung moderner Computer-Systeme eine Erhöhung der Performance, v.a. im Hinblick auf Speicherkapazität und Parallelisierung. Wichtig für den Erhalt des in den Programmen enthaltenen Wissens für zukünftige Generationen ist auch die Erweiterung der bestehenden Dokumentation.

1.2 Aktueller Status

In einem ersten Schritt wurden die bestehenden, relativ unflexiblen Eingabedateien aller betroffenen Codes systematisch untersucht und auf Basis dieser Daten ein konsistentes neues XML-basiertes Eingabekonzept entwickelt. Dieses ermöglicht nicht nur die Dateneingabe in objektorientierter Form, sondern auch die Validierung der Eingabedaten durch XML-Schemata.

Zeitgleich wurde ein neues, objektorientiertes Datenmodell implementiert, welches vom HCP unabhängige physikalische Basisklassen, sowie darauf aufbauende komplexere Datentypen zur Verfügung stellt. Die Basisdaten zur Erstellung der Nuklid-Objekte werden vom ebenfalls neu implementierten Werkzeug LibGen eingelesen und daraus eine Datenbibliothek generiert.

Für eine flexible Ausgabe der Ergebnisse wurde ein neues generisches Ausgabekonzept entwickelt, welches objekt- und feldbasierte Datenausgabe in verschiedenen Dateiformaten, sowie eine graphische Darstellung der Daten und die Behandlung von Statusmeldungen vorsieht. Um die Integration von automatisierten Tests in den Entwicklungsprozess zu gewährleisten, wurde eine Test-Suite entwickelt, die Unit-Tests und Regressionstests beinhaltet. Außerdem wurde mit der Vorbereitung der bisher verwendeten, in Fortran implementierten Programme an das HCP begonnen. Dazu musste jeweils der Aufwand für die Beibehaltung des Fortran-Codes gegen eine Neu-Implementierung in C++ abgewogen werden. Im Zuge dessen ist bereits der neue Abbrandcode TNT (Topological Nuclide Transmutation) enstanden.

1.3 Zukünftige Entwicklung

Ein großer Fokus liegt auf der Ankopplung der Fortran-Module an das HCP. Dazu müssen jeweils die Daten von den Algorithmen entkoppelt werden. Für den Datentransfer zwischen HCP und den Modulen wird eine Schnittstelle zwischen C++ und Fortran benötigt. Um eine flexible Interaktion der angekoppelten Module zu ermöglichen, wird zudem eine Hauptsteuerung für das HCP entwickelt und implementiert. Die intuitive Bedienbarkeit des Programms soll durch eine graphische Oberfäche erreicht werden, die basierend auf der Bibliothek QT implementiert werden soll. Zusätzlich soll das HCP im Rahmen einer Masterarbeit auf Parallelisierbarkeit untersucht und umgestellt werden.

2 Technologie und Infrastruktur

2.1 Programmiersprachen

Alle neuentwickelten Programmteile werden in *C++* implementiert. Einen großen Vorteil dieser Programmiersprache stellt die Unterstützung der Objektorientierung dar, auf die im HCP großen Wert gelegt wird. Außerdem bietet die integrierte *STL (Standard Template Library)* zahlreiche nützliche Datentypen, wie z.B. Vektoren oder Hash-Tabellen. Durch die ebenfalls unterstützte Template-Programmierung ist es auch möglich, eigene generische Klassen oder Funktionen zu definieren. Zusätzlich gibt es für C++ eine Vielzahl von externen Bibliotheken, von denen im HCP bisher *boost*, *Eigen* und *Xerces* benutzt werden.

Die bestehenden Simulationsprogramme sind in *Fortran* implementiert. Um die Möglichkeit zu haben, sie in ihrem jetzigen Zustand an das HCP anzukoppeln, wird eine Schnittstelle zwischen Fortran und C++ zum Datenaustausch entwickelt.

Um einen einheitlichen Programmierstil zu gewährleisten und somit die Qualität des Quellcodes zu erhöhen, gibt es Coding Guidelines für C++ und Fortran. Diese enthalten obligatorische Regeln, Richtlinien und Empfehlungen zur Erstellung von übersichtlichem Quellcode.

Neben Fortran und C++ werden noch Shell-Skripte eingesetzt, um an einigen Stellen Abläufe (z.B. auf dem File-System) zu automatisieren.

2.2 Entwicklungsumgebung und Werkzeuge

Die Entwicklungsplattformen für das HCP sind Linux und Windows. Es ist keine feste Entwicklungsumgebung vorgegeben. Unter Linux werden meist *Emacs* oder *Eclipse* verwendet und unter Windows *Visual Studio*.

Das gesamte Projekt unterliegt der Versionskontrolle mit *Subversion (SVN)*. Dies gewährleistet ein sauberes Versions-Management und das automatisierte Zusammenfügen der Änderungen der Projektteilnehmer. Das Werkzeug *GNU Make* sorgt im HCP für ein automatisiertes Build-Management. In einem zentralen Makefile wird modulweises Kompilieren des HCP ermöglicht, wodurch der Kompiliervorgang beschleunigt wird.

2.3 Dokumentation

Um den Entwicklungsprozess auch im Nachhinein nachvollziehbar zu machen, werden Entscheidungen in Form von Sitzungsprotokollen festgehalten. Hintergrundwissen und Beschreibungen einzelner Teile des Programms sind in Manuals und Papers zu finden. Der Aufbau der Software wird mit Hilfe von *UML*-Diagrammen deutlich gemacht. Eine automatisierte Code-Dokumentation in Form von HTML-Seiten wird mit dem Werkzeug *doxygen* erzeugt. Alle Teile der Dokumentation werden über das SVN-Repository und über ein internes Institutswiki zur Verfügung gestellt.

3 Entwicklungsprozess

Die Entwicklung des HTR Code Packages (HCP) begann am 5. Juli 2010 mit der Erstellung des grundlegenden Programm-Konzepts. Die Fertigstellung eines ersten Prototyps, der es ermöglichen soll bereits einzelne Simulationen zu starten

und verschiedene Module nacheinander anzusteuern, ist für 2013 geplant. Für die Umsetzung der Entwicklung sind derzeit bis zu 12 Personen in das Projekt integriert. Diese Gruppe setzt sich aus dem Projektleiter Dr. Stefan Kasselmann (`s.kasselmann@fz-juelich.de`), Wissenschaftlern und Studenten zusammen (siehe Abbildung 2).

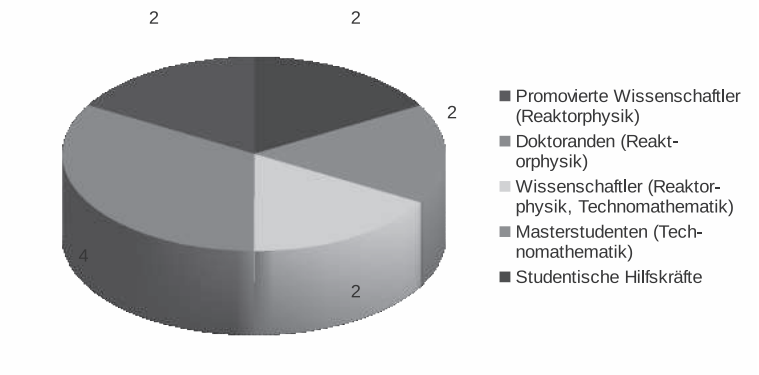


Fig. 2. Aufteilung des HCP-Teams

Die sechs verschiedenen Gruppen von Mitarbeitern befassen sich im Rahmen des Projekts mit verschiedenen Teilaspekten, die auf die entsprechenden Kenntnisse und Fähigkeiten der einzelnen Mitglieder abgestimmt sind. Der Projekt-Leiter befasst sich beispielsweise zum einen mit koordinierenden Maßnahmen, um die Arbeit im gesamten Projekt zu strukturieren und zu leiten. Neben diesen eher organisatorischen Tätigkeiten befasst Herr Kasselmann sich noch mit grundlegenden Design-Entscheidungen bezüglich des Gesamtkonzepts und des Datenmodells sowie mit Implementierungsarbeiten an einzelnen Modulen. Die Gruppe der promovierten Wissenschaftler im Bereich Reaktorphysik befassen sich hauptsächlich mit der Betrachtung von physikalischen Daten und Programmen sowie der Erstellung von Modellen, welche die Grundlage für die Physik innerhalb des Code Pakets bilden.

Die Doktoranden im Bereich der Reaktorphysik hingegen befassen sich mit der konkreten Implementierung bzw. Überarbeitung der einzelnen physikalischen Module, die vom HCP Backbone, dem Hauptprogramm des HCP, angesteuert werden sollen. Mit dem Bereich der eher programmiertechnischen Aspekte des HCP befassen sich vor allem Studenten im Bereich Technomathematik. Dazu gehören unter anderem Aspekte wie Entwicklung des Datenmodell-Designs, Implementierung des Ein- und Ausgabekonzepts sowie einer grafischen Benutzeroberfläche und Analyse des HCP auf Parallelisierbarkeit. Sowohl die Wissenschaftler im Bereich Reaktorphysik, als auch diese, die im Bereich Technomathematik arbeiten, befassen sich zum Teil mit den bestehenden Codes, der Implementierung eines Interfaces zwischen dem in C++ geschriebenen Hauptprogramms und den noch in Fortran vorhandenen Modulen sowie der Bearbeitung anstehender mathematischer Problemstellungen. Abschließend gehören zum HCP noch studentische Hilfskräfte. Da diese meist nur stark begrenzte Zeit am HCP mitarbeiten, werden diese meist mit der Implementierung einzelner

Klassen beauftragt. Gegebenenfalls bietet sich die Bearbeitung einer größeren Teilaufgabe in Form einer Bachelor- oder Diplomarbeit an.

4 Herausforderungen an das Softwareprojekt

4.1 Faktor Mensch

Durch die Heterogenität der Gruppe, die teilweise auch stark schwankt, ist die gute Zusammenarbeit eine der wesentlichen Herausforderungen an ein Softwareprojekt, damit dieses gelingt. Ein besonders wichtiger Aspekt ist deswegen die Kommunikation innerhalb des Teams. Um regelmäßige Treffen und die Organisation geeignet zu gestalten haben wir uns an Aspekten der Agilen Softwareentwicklung bzw. der Scrum-Methode (siehe Abbildung 3) orientiert.

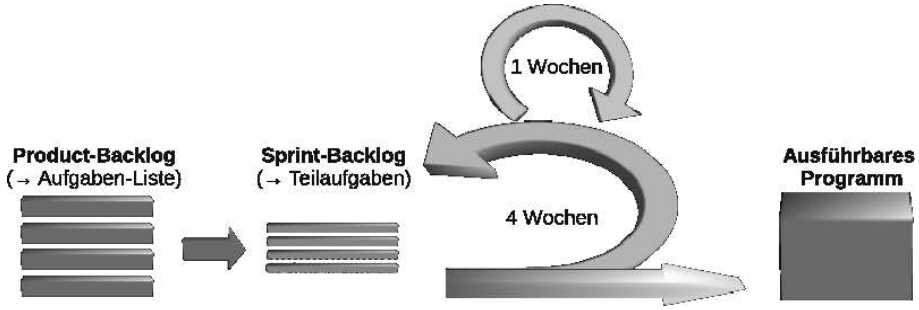


Fig. 3. Für das HCP angepasste Variante des Scrum-Ablaufs

Im Wesentlichen ist dadurch definiert, dass sich das Team des HCP einmal wöchentlich trifft. Dann wird in einem kurzen Treffen besprochen, was in der letzten Woche von jedem bearbeitet wurde, was für Probleme aufgetreten sind und wie fortgefahren wird. Etwa alle vier Wochen findet dann ein etwas größeres bzw. längeres Treffen statt. In diesem geht jeder, der Neuerungen zu präsentieren hat genauer auf deren Details ein, damit jeder in der heterogenen Gruppe einen Überblick hat, wie der gesamte Stand des Projekts ist. Am Ende einer jeden Präsentation werden dann die nächsten Arbeitsschritte, welche in den kommenden vier Wochen erledigt werden sollen, definiert. Diese beschreiben den Sprint-Backlog für jeden Mitarbeiter und geben somit immer eine gute Übersicht über die nächsten Arbeitsschritte und definieren einen bearbeitbaren Rahmen der ansonsten insgesamt umfangreichen Aufgaben, welche im Ganzen den Product-Backlog repräsentieren.

4.2 Faktor Evolution

Auch Änderungen der Anforderungen an das Projekt sind immer wieder eine Herausforderung. Im HCP gab es vor allem zu Beginn immer wieder grundlegende Änderungen, da der genaue Umfang des Projektes noch nicht definiert war und sich nach und nach neue zu behandelnde Aspekte ergaben. So musste sich das Team beispielsweise bei den vorhandenen Fortran-Programmen immer wieder die Frage stellen, ob diese übernommen oder neuimplementiert werden soll. Denn leider ist eine saubere Definition einer Schnittstelle unter Umständen nicht möglich.

5 Zusammenfassung

Bei der Betrachtung des gesamten Projekts lässt sich feststellen, dass sowohl eine gute Kommunikation in der Gruppe, als auch die strukturierte Entwicklung durch geeignete Aufgabenverteilung die Arbeit am HCP gut vorantreiben. Auch durch einen einheitlichen Programmierstil, welcher durch entsprechende Richtlinien definiert wurde, macht das Arbeiten effektiver. Problematisch hingegen ist die Entwicklung der Schnittstelle zwischen C++ und Fortran, wodurch das Projekt durch Neuimplementierungen der vorhandenen Fortran-Module immer weiter wächst. Auch eine zeitnahe Dokumentation der Arbeiten stellt sich durch Zeitmangel als sehr problematisch heraus.

Development of libALF

Daniel Neider

Lehrstuhl für Informatik 7, RWTH Aachen University, Deutschland
neider@automata.rwth-aachen.de

Abstract. Dieser Artikel beschreibt die Entwicklung von libALF, einer Programmbibliothek zum Lernen endlicher Automaten und anderer Maschinen mit endlichem Zustandsraum.

Programmbibliothek , Algorithmisches Lernen, endliche Automaten, Entwicklung

1 Einleitung

libALF ist eine Open Source Programmbibliothek zum Lernen endlicher Automaten und (zukünftig auch) anderer Maschinen mit endlichem Zustandsraum wie etwa Büchi-Automaten, visibly Einzählerautomaten etc. Zurzeit sind in libALF knapp zehn der prominentesten Lernalgorithmen für endliche Automaten implementiert, die aus Tabelle 1 entnommen werden können.

Aktive Lernalgorithmen	Passive Lernalgorithmen
Angluins L^* [1]	RPNI [5]
Angluins L^* (Spalten)	DeLeTe2 [6]
Kearns & Vazirani [2]	Biermann & Feldmann [7]
Rivest & Schapire [3]	Biermann & Feldmann (mittels SAT)
NL^* [4]	

Tabelle 1: In libALF implementierte Lernalgorithmen unterschieden nach aktiven Lernalgorithmen (vgl. Angluin [1]) und passiven Lernalgorithmen (vgl. Biermann [7]).

libALF wurde als eine schlanke und performante Programmbibliothek vor allem (aber nicht ausschließlich) für den Einsatz in wissenschaftlichem Umfeld entwickelt. In der Hoffnung, dass libALF nicht nur intern sondern auch für externe Forschergruppen hilfreich ist, wurde libALF als Open Source unter der LGPL Lizenz in Version 3 veröffentlicht[1]. Wir hoffen, dass sich so auch Externe an der Weiterentwicklung der Bibliothek beteiligen.

Neben libALF besteht und betreut das libALF Projekt weitere Bibliotheken:

- *jALF* ist ein Java Interface für den Zugriff auf libALF, welches mittels JNI (Java Natural Language Interface oder einer Client/Server-Architektur auf libALF zugreift.
- *libAMoRE* ist eine in C geschriebene Programmbibliothek für endliche Automaten, reguläre Ausdrücke und Monoide [8]. libAMoRE wurde ursprünglich nicht vom libALF Projekt entwickelt, wird jedoch zurzeit von dem Projekt betreut.

[1] libALF kann von der Webseite http://libalf.informatik.rwth-aachen heruntergeladen werden

- *libAMoRE++* ist ein C++ Interface für libAMoRE.
- *liblangen* ist eine Programmbibliothek zum Erstellen zufallsgenerierter endlicher Automaten.
- *Finite Automata Tools* ist ein Kommandozeilenprogramm, um zahlreiche Operationen auf endlichen Automaten auszuführen (beispielsweise Komplementierung, Minimierung etc.)

libALF umfasst zur Zeit etwa 25.300 Source Lines of Code C++, 10.600 Source Lines of Code C und 7.800 Source Lines of Code Java Quelltext. Wie bereits erwähnt stammt der C Quelltext nicht ursprünglich vom libALF Projekt, wird aber von uns gewartet und bei Bedarf gepatcht. Obwohl das libALF Projekt mehrere Bibliotheken umfasst, beschränkt sich der Rest dieses Textes ausschließlich mit der libALF und jALF Bibliothek.

2 Designziele

libALF ist modular aufgebaut, wie es in Abbildung 1 beispielhaft dargestellt ist. Unterschiedlich farbige Puzzlestücke repräsentieren dabei verschiedene funktionale Komponenten.

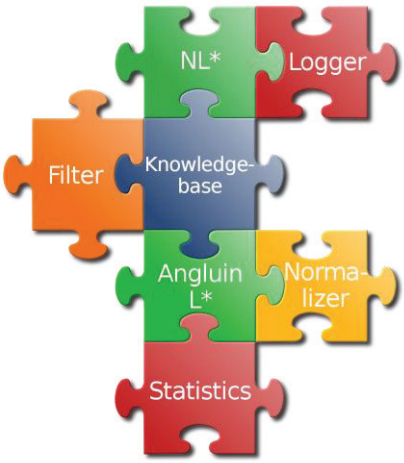


Abbildung 1: Schematische Darstellung des modularen Aufbaus von libALF

Die Hauptkomponente ist die so genannte „Knowledgebase“, die als Informationsspeicher für die Lernalgorithmen dient. An einer Konwledgebase können sich mehrere Lernalgorithmen anmelden (in Abbildung 1 etwa Angluins L^* oder der NL^* Algorithmus), um dann Daten auszutauschen. Eine Knowledgebase kann zusätzlich mit domänenspezifischen Filtern erweitert werden, die „einfache“ Anfragen der Lernalgorithmen selbst beantworten können, ohne rechenintensivere Anfragen auf der Knowledgebase ausführen zu müssen. Darüber hinaus gibt es weitere Komponenten wie z.B. Logger zum Protokollieren von Ereignissen oder diverse Statistiken.

Alle Komponenten sind als Objekte realisiert. Durch Vererbung und Polymorphie wird der oben beschriebene modulare Aufbau der Bibliothek erreicht. Neben diesem modularen Aufbau standen zusätzlich drei weitere Designziele im Vordergrund:

- Wie bereits in der Einleitung erwähnt, soll die Programmbibliothek *schlank und performant* sein. Auf aufwändige Funktionalität haben wir bei der Entwicklung bewusst verzichtet.
- Die Programmbibliothek soll *flexibel* sein. Um in möglichst vielen Umgebungen einsetzbar zu sein, soll libALF auf 32- und 64-bit Systemen sowie auf allen gängigen Betriebssystemen, d.h. Windows, Linux und MacOS, lauffähig sein. Obwohl libALF in C++ entwickelt wurde, soll es dennoch möglich sein, auch von Java auf die Bibliothek zuzugreifen.

- Die Programmbibliothek soll *leicht zu erweitern* sein. Dies bedeutet in erster Linie, dass es mit möglichst geringem Aufwand möglich sein soll, neue Lernalgorithmen und Automatentypen hinzuzufügen.

3 Entwicklung

Abbildung 2 zeigt den zeitlichen Ablauf der Entwicklung von libALF. Die Entwicklung begann im Frühjahr 2009. Ein erster stabiler „Kern“, d.h. die Knowledgebase, einige wenige Lernalgorithmen sowie Logger und Statistiken, wurden im Frühjahr 2010 fertiggestellt. Auf dieser Grundlage wurde libALF im Juli 2010 auf der International Conference on Computer Aided Verification (CAV 2010) präsentiert. Zum jetzigen Zeitpunkt ist die Entwicklung von libALF größtenteils abgeschlossen. Sie wird lediglich um weitere Lernalgorithmen und, bei Bedarf, neue Automatentypen erweitert.

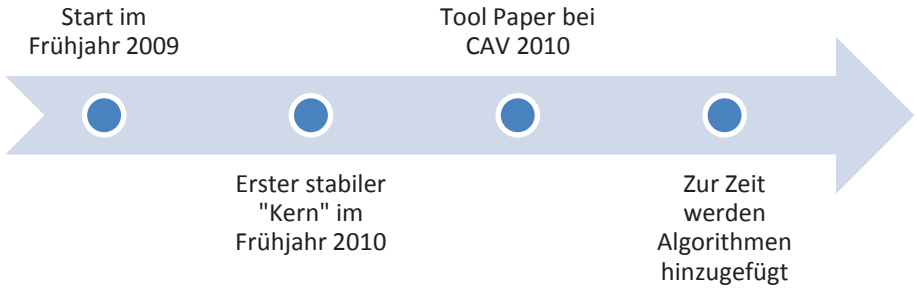


Abbildung 2: Zeitlicher Ablauf der Entwicklung von libALF

libALF entstand mit Unterstützung der Universitätsprofessoren Joost-Pieter Katoen (RWTH Aachen), Benedikt Bollig (ENS de Cachan Paris) und Martin Leucker (TU München, jetzt Universität zu Lübeck). Für die Konzeption und das Design waren in erster Linie die Doktoranden Carsten Kern (Lehrstuhls für Informatik 2) sowie Daniel Neider (Lehrstuhl für Informatik 7 der RWTH Aachen) verantwortlich. Implementiert wurde libALF von einer studentischen Hilfskraft. Zeitweise war eine zweite studentische Hilfskraft mit dem Schreiben der Dokumentation beschäftigt.

Die Entwicklung von libALF war insbesondere durch eine hohe Komplexität hervorgerufen durch die vielen unterschiedlichen Umgebungen (32- und 64-bit, Windows und Linux, C++ und Java) gekennzeichnet. Der Einsatz von Standardwerkzeugen wie z.B. das GNU Build System (auch Autotools genannt), die sowohl für Windows als auch für Linux verfügbar sind, erleichterte die parallele Entwicklung jedoch erheblich. Tabelle 2 gibt einen Überblick über die während der Entwicklung eingesetzten Softwarewerkzeuge. Sie unterteilt sich in Werkzeuge für die Entwicklung in C++ und die in Java. Darüber hinaus wurden einige Werkzeuge wie z.B. das Versionskontrollprogramm Subversion oder das Satzsystem LaTeX eingesetzt.

	C++	Java
Compiler	gcc / MinGW	javac
IDE	Text Editor	Eclipse
Build	make	ant
Tests	eigene Testfälle	JUnit
Dokumentation	doxygen	JavaDoc
	Subversion, LaTeX, …	

Tabelle 2: Während der Entwicklung von libALF eingesetzte Softwarewerkzeuge.

Ein formaler Entwicklungsprozess existierte während der Entwicklung nicht. Änderungen, sowohl am Design als auch an der Funktionalität, erfolgten nach Rücksprache mit den Doktoranden. Dokumentiert wurden Änderungen im Quelltext selbst und im Log von Subversion. Auf gleiche Art wurde und wird beim Beheben von Fehlern verfahren: Zunächst wird der Fehler identifiziert und anschließend nach Rücksprache mit einem der Doktoranden eine entsprechende Korrektur im Subversion vorgenommen. Die Dokumentation fand vorwiegend im Quelltext und in dem Log von Subversion statt.

4 Erfahrungen

Während der Entwicklung von libALF merkten wir schnell, dass der C++ Teil und der Java Teil divergierten. Ursprünglich war geplant, alle Änderungen an libALF sofort auch in jALF zu realisieren. Aufgrund mangelnder Ressourcen war dies aber nur zu einem Teil möglich. Dennoch bietet jALF den Zugriff auf die meisten Funktionen libALFs. In diesem Zusammenhang mussten wir zudem feststellen, dass der Aufwand für die Erstellung einer Dokumentation weit unterschätzt wurde.

Die ausführliche Designphase zu Beginn der Implementierung bewerten wir als äußerst positiv. So war es auch während der Implementierung ohne größere Probleme möglich, Funktionalität zu ändern und hinzuzufügen. Darüber hinaus erlaubten Standardwerkzeuge zur Softwareentwicklung wie z.B. das GNU Build System und Subversion alle Designziele mit vertretbarem Aufwand umzusetzen.

5 Zusammenfassung und Ausblick

libALF ist eine Open Source Programmbibliothek zum Lernen endlicher Automaten und anderer Maschinen mit endlichem Zustandsraum. Die Entwicklung von libALF (und der anderen Bibliotheken) startete im Frühjahr 2009 und wurde von zwei Doktoranden der RWTH Aachen sowie studentischen Hilfskräften durchgeführt. Ein ausführliches Design sowie der Einsatz von Standardwerkzeugen erlaubte eine größtenteils reibungslose Entwicklung. Der Aufwand, insbesondere für die Entwicklung von jALF sowie die Erstellung der Dokumentation, wurde jedoch stark unterschätzt.

Mittlerweile ist die Entwicklung von libALF erfolgreich abgeschlossen. Außer den beabsichtigten funktionalen Erweiterungen durch neue Lernalgorithmen oder Automatentypen ist mittelfristig keine weitere Entwicklung geplant.

6 Literaturverzeichnis

1 D. Angluin (1987): Learning Regular Sets from Queries and Counterexamples. Inf. Comput., Seite 87–106.

2 M.J. Kearns, U.V. Vazirani (1994): An introduction to computational learning theory. The MIT Press.

3 R. Rivest, R. Schapire (1993): Inference of finite automata using homing sequences. Machine Learning: From Theory to Applications, Springer, Seite 51–73.

4 B. Bollig, P. Habermehl, C. Kern, M. Leucker (2009): Angluin-style learning of NFA. Proceedings of the 21st international jont conference on Artifical Intelligence, Morgan Kaufmann Publishers Inc., Seite 1004–1009.

5 P. Gracía, J. Oncina (1992): Inferring regular languages in polynomial update time. Pattern Recognition and Image Analysis, volume 1 of Series in Machine Perception and Artificial Intelligence, Seite 49–61.

6 F. Denis, A. Lemay, and A. Terlutte (2004): Learning regular languages using RFSAs. TCS, Seite 267– 294.

7 A. W. Biermann, J. A. Feldman (1972): On the synthesis of finite state machines from samples of their behavior. IEEE Transactions on Computers, Seite 592–597.

8 O. Matz, A. Miller, A. Potthoff, W. Thomas, E. Valkema (1995): Report on the Program AMoRE, Report 9507, Inst. für Informatik u. Prakt. Mathematik, CAU Kiel.

MeDIC - Eine Infrastruktur zum Verwalten, Dokumentieren und Visualisieren von Metriken

Matthias Vianden

Lehr- und Forschungsgebiet Softwarekonstruktion, RWTH Aachen University
Matthias.vianden@swc.rwth-aachen.de

1 Hintergrund

Es ist weitreichend bekannt, dass Projektleiter und Manager sehr von der Anwendung von Metriken profitieren. Leider zeigen aber viele Forschungsergebnisse, dass es schwer ist, die richtigen Metriken zu finden. Eine bekannte Studie zeigt zum Beispiel, dass es 58% aller Projektleiter und 50% aller Manager schwierig finden, die richtigen Daten zu sammeln und zu analysieren. Auf der einen Seite helfen Rahmenwerke wie GQM dabei Metriken aus abstrakten Zielen für ein Projekt abzuleiten. Auf der anderen Seite ist es für eine große Organisation mit vielen ähnlichen Projekten nicht sinnvoll, die Metriken nur für ein einzelnes Projekt zu definieren. Daher ist es sinnvoll, Metrik-Erfahrung (Metrik-Definitionen, Evaluationen und Modelle) im Rahmen einer großen Organisation mit vielen ähnlichen Projekten wiederzuverwenden.

Obwohl es sehr viele Forschungsergebnisse in den Bereichen Modellierung von Metriken und Metrik-Rahmenwerken gibt, wurde sich weniger damit beschäftigt, zu untersuchen, wie die Ergebnisse dieser Forschung genutzt werden können um ein stimmiges Konzept für Metrik-Anwendung und Wiederverwendung zu entwickeln. Dieses Konzept sollte mit entsprechenden Metrik-Prozessen untermauert werden, die auf der Idee der Metrik-Wiederverwendung aufbauen. Außerdem sollte der Prozess und die Methodik durch spezielle Werkzeuge zur Metrik-Dokumentation, Metrik-Konfiguration und Metrik-Visualisierung unterstützt werden.

2 Systembeschreibung

MeDIC besteht aus einer Sammlung webbasierter Informationssysteme zur Dokumentation, Konfiguration und Visualisierung von Metriken. Abbildung 3 zeigt einen Screenshot der Projektübersichtsseite des Metrik-Konfigurationswerkzeugs MeDIC-Configuration. In dieser kann ein Projektleiter die Metriken, welche in seinem Projekt gesammelt werden, definieren. Neben der Metrik werden ihr Zustand („Vollständig Spezifiziert“ oder „Unvollständig“), ihr Typ („Projektspezifisch“ oder „Unternehmensweite Standardmetrik“) sowie ihr Status („Wird verwendet“, „Neu“, oder „Veraltet“) angezeigt. Hierdurch ist es möglich den Status und die Art der verwendeten Metriken eines Projektes schnell und auf einen Blick z.B. im Rahmen einer CMMI Prüfung zu erkennen.

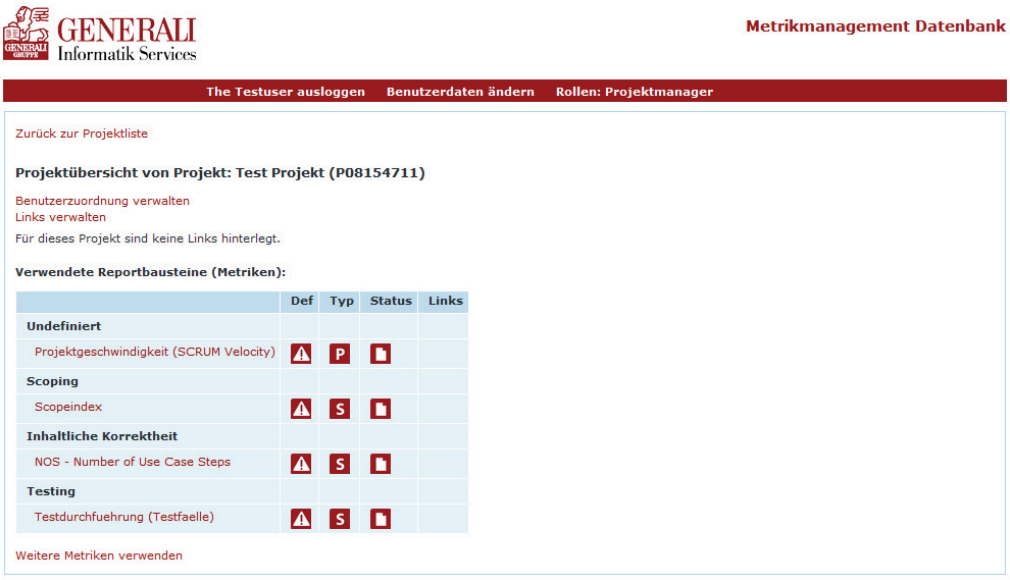


Abbildung 3: Projektübersichtsseite von MeDIC-Configuration

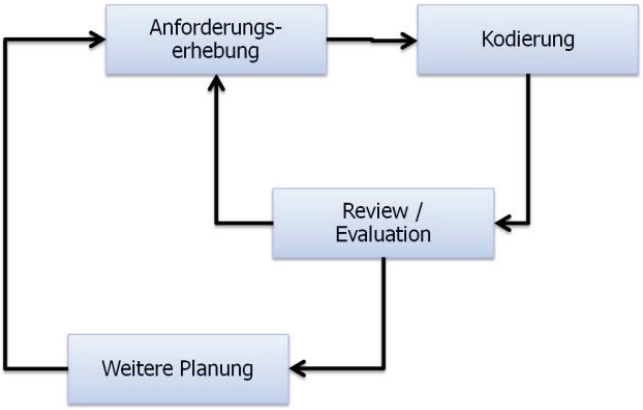


Abbildung 2: Ursprünglicher Enwicklungsprozess

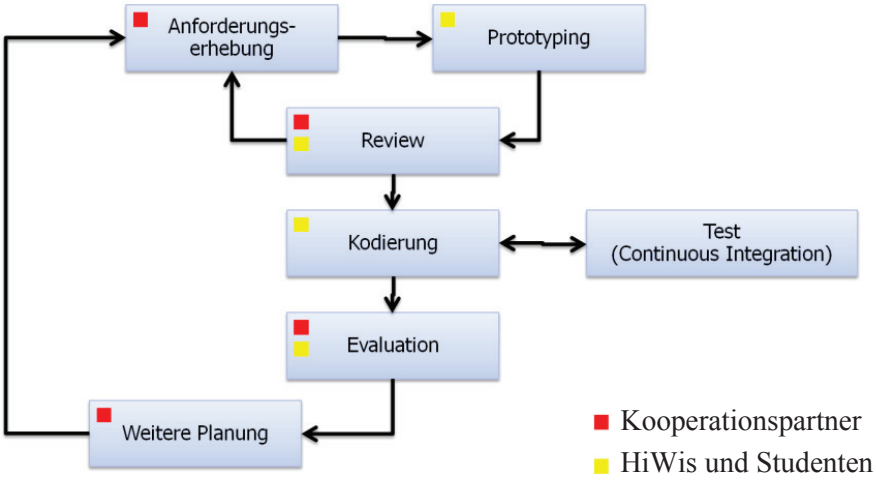


Abbildung 1: Angepasster Entwicklungsprozess

Die Systeme werden in enger Zusammenarbeit mit einem Industriepartner entwickelt und bauen auf der Jave Enterprice Edition auf. Die Datenablage erfolgt in einer relationalen Datenbank (MySQL oder IBM DB2). Die Systeme selbst laufen auf einem Anwendungsserver (Glassfish oder IBM Websphere). Sie sind relativ klein und haben eine Gesamtgröße von ca. 30.000 Lines of Code; circa die Hälfte des Quellcodes ist außerdem durch den Gargoyle-Code-Generator generiert. Der Entwicklungsaufwand lag zwischen Juli 2010 und November 2010 bei 153,7 Assistenten-Stunden sowie 48 Hiwi-PT. Außerdem wurde die Anwendung in dieser Zeit durch eine Praktikantin in 4,5 Monaten Praktikum weiterentwickelt. Insgesamt liefen bisher 8 Diplom-, Master- und Bachelorarbeiten im Umfeld von MeDIC. Ein Großteil der Arbeiten hat sich hierbei allerdings mit der Weiterentwicklung der Konzepte beschäftigt und weniger an den Systemen selbst gearbeitet.

3 Entwicklungsprozess

Das MeDIC System wird in enger Zusammenarbeit mit einem Industriepartner entwickelt. Dieses Vorgehen zeigt sich schon in der ersten Version des Entwicklungsprozesses in **Error! Reference source not found.**. Der Prozess ist ein typischer itterativer und inkrementeller Softwareentwicklungsprozess, der sich am PDCA (Plan, Do, Check, Act) Vorgehen orientiert. Nachdem die Anforderungen für ein Inkrement gesammelt wurden (Plan), wurden diese Kodiert (Do). Anschließend wurden die Ergebnisse beim Kooperationspartner evaluiert (Check) und darauf aufbauend das weitere Vorgehen geplant (Act). Es zeigte sich aber, dass die Umlaufzeiten durch den Prozess (insbesondere durch komplexe Technologien und damit verbundene Aufwände bei der Kodierung) zu lang sind. Hierdurch wurden Probleme in der Konzeption der Realisierung erst relativ spät in der Evaluation festgestellt.

Der Prozess wurde daher wie in Abbildung 1 zu sehen angepasst. Im Gegensatz zur ersten Version des Prozesses werden die Anforderungen in diesem Prozess erst in einem Prototypen realisiert. Dieser Prototyp wird anschließend mit dem Kooperationspartner gereviewed um eventuelle Schwachstellen in den Anforderungen und der Realisierung zu finden. Diese Schleife wird so lange iteriert, bis das Review nur noch wenige Fehler findet. Anschließend wird die Lösung kodiert und dabei kontinuierlich geprüft. Anschließend wird die Lösung analog zum ursprünglichen Prozess beim Kooperationspartner evaluiert um dann darauf aufbauend das weitere Vorgehen zu planen. Der Prozess hat sich bewährt, da die Qualität der Lösung durch die frühen Prototypen und entsprechend viele Reviews deutlich verbessert werden konnte.

4 Herausforderungen im Umfeld von Forschungssoftware

Wie im angepassten Prozess in Abbildung 1 zu sehen, werden Studenten, Praktikanten und Hiwis regelmäßig in das Projekt eingebunden. Neben den typischen Arbeiten im Rahmen von Abschlussarbeiten werden insbesondere die Hiwis auch für Wartungsarbeiten eingesetzt. In der Regel gibt es regelmäßige Treffen, im Sinne von Statusmeetings, in denen die Weiterentwicklung sowie aktuelle Probleme besprochen werden. Die Studenten werden außerdem in einer Art Schulung vor dem Start der

Arbeit mit den technischen Details vertraut gemacht. Wenn viele Arbeiten parallel starten, dann finden diese Schulungen in der Gruppe statt, ansonsten werden die Studenten durch „Pair-Programming-Sessions“ auf ihre Arbeit vorbereitet.

Das System und die damit verbundenen Projekte sind natürlich durch den Forschungscharakter der Entwicklung ständig Änderungen unterworfen. Dies hängt damit zusammen, dass der Fokus für das System bewusst unscharf gehalten wird um auch unkonventionelle Lösungen zuzulassen. Änderungen betreffen häufig die zu realisierenden Features im System und weniger häufig den zugrundeliegenden Prozess. Wir gehen mit diesen häufigen Änderungen so um, dass wir versuchen in der Anforderungsphase den Fokus des Systems durch Prototypen zu schärfen. Hierdurch können unnötige Aufwände in der Kodierung und damit verbundene langwierige Änderungszyklen vermieten werden. Technologisch werden die Änderungen im Quellcode des Systems in ein Versionsverwaltungswerkzeug (SVN) eingespielt. Die Änderungen selbst werden über einen Issue-Tracker (TRAC) verwaltet und zugeordnet. Das System wird kontinuierlich integriert (HUDSON-CI) und über automatisierte Tests geprüft.

5 Zusammenfassung

Der angepasste Entwicklungsprozess hat sich im Rahmen der Entwicklung des Metrik Verwaltungssystems MeDIC bewährt. Insbesondere die technologische Unterstützung über Versionsverwaltung, Issue-Tracker, Build-System und Continuous Integration helfen, die typischen Probleme zu adressieren. Es hat sich außerdem gezeigt, dass es sehr hilfreich ist, die geplanten Lösungen früh durch Prototypen zu realisieren und die dabei entstehenden partiellen Lösungen zusammen mit dem Kooperationspartner zu reviewen.

Probleme entstehen insbesondere durch alte Technologien und degradierte Architekturen. Dies wird in der Regel durch reimplementierung der alten Dinge mit neuen Technologien und neuen Architekturen adressiert. Verwobene und tiefe Hierarchien beim Kooperationspartner können den Transfer und die damit verbundene Evaluation der Ergebnisse behindern. Diese Probleme müssen sobald sie auftreten adressiert werden. Auch die fachliche Einarbeitung von neuen Studenten ist zum Teil problematisch kann aber durch die Menge der bisher gelaufenen Arbeiten abgefangen werden, da in den Arbeiten auch immer fachliche Einführungskapitel zu finden sind. Probleme durch die komplexe Infrastruktur wurden mittlerweile über eine fertige virtuelle Maschine adressiert. Diese enthält bereits alles, was zum Entwickeln des Systems benötigt wird und kann direkt von neuen Studenten verwendet werden.

Score-P

Christian Rössel, Bernd Mohr und Felix Wolf

Forschungszentrum Jülich GmbH
Institute for Advanced Simulation
Jülich Supercomputing Centre

1 Das Score-P Projekt

Score-P [12,3] ist eine Sammlung von Software-Werkzeugen für die Leistungsmessung paralleler Programme. Es ist speziell im Hinblick auf massiv-parallele Systeme im Bereich des wissenschaftlichen Rechnens entwickelt. Die Ergebnisse der Leistungsmessung werden als Ereignisspuren oder Pfadprofile gespeichert oder direkt zur Laufzeit analysiert. Die gespeicherten Daten stehen in offenen Formaten zur Verfügung und werden von verschiedenen, etablierten Analysewerkzeugen (Scalasca [10], Periscope [7], Tau [13] und Vampir [11]) verwendet. Die Werkzeugumgebung Score-P wurde in den Projekten SILC[1] ("Skalierbare Infrastruktur zur Automatischen Leistungsanalyse Paralleler Codes") und PRIMA[2] ("Performance Refactoring of Instrumentation, Measurement, and Analysis Technologies for Petascale Computing") gemeinsam von den Partnern

- Forschungszentrum Jülich GmbH
- German Research School for Simulation Sciences GmbH, Jülich/Aachen
- Gesellschaft für numerische Simulation mbH Braunschweig
- Rheinisch-Westfälische Technische Hochschule Aachen
- Technische Universität Dresden
- Technische Universität München
- University of Oregon, Eugene, USA

entwickelt.

Ziel dieser Kollaboration ist es, offenkundige Probleme in der Entwicklung wie auch der Nutzung von Leistungsmessungs-Werkzeugen für massiv-parallele Systeme zu beheben. Aus Nutzersicht ist z. B. die nur umständlich zu bewerkstelligende Interoperabilität verschiedener, komplementärer Analysewerkzeuge zu bemängeln. Dies liegt hauptsächlich an nicht standardisierten Dateiformaten für Ereignisspuren und Pfadprofildaten. Konvertierungen zwischen Formaten sind prinzipiell möglich und entsprechende Werkzeuge werden bereitgestellt. Bei den auf modernen Supercomputern anfallenden Datenmengen ist die Praktikabilität dieses Ansatzes jedoch stark limitiert. Neben den Datenformaten unterscheiden sich auch die Nutzerschnittstellen zur Vorbereitung einer Leistungsmessung, der sog. Instrumentierung. Nutzer sind also gezwungen, für jede bisher existierende Werkzeugumgebung eigene Befehle zu erlernen, obwohl sich die Funktionalität der Instrumentierung von Werkzeug zu Werkzeug nur marginal unterscheidet. Mit einer einheitlichen Instrumentierungs-Schnittstelle könnte bei den von den

[1] Das SILC Projekt ist gefördert vom BMBF unter dem Förderkennzeichen 01IH08006.
[2] Das PRIMA Projekt ist gefördert vom US DOE unter dem Förderkennzeichen DE-SC0001621.

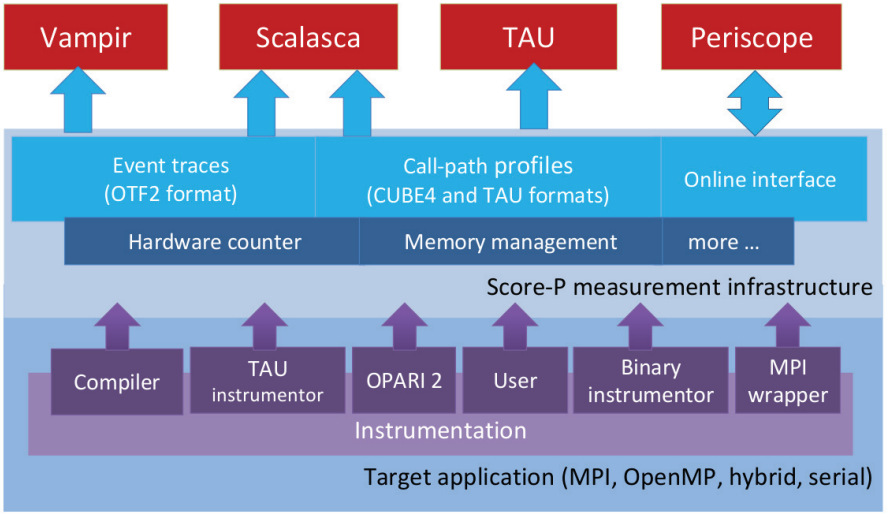


Abb. 1. Score-P Architektur

Projektpartnern regelmäßig durchgeführten VI-HPS [6] Tuning Workshops und Tutorials der Schwerpunkt stärker auf die Analysefunktionalität gelegt werden. Aus Entwicklersicht ist es wünschenswert, die Ressourcen, die durch die redundante Entwicklung ähnlicher Messfunktionalität gebunden werden, in die Verbesserung der sich unterscheidenden Analysefunktionalität umzuleiten, insbesondere im Hinblick auf die extrem hohen Portabilitätsanforderungen während der Entwicklung der Messumgebung.

Das Ergebnis der Kollaboration ist das Softwarepaket Score-P, eine Sammlung von Bibliotheken und Instrumentierungswerkzeugen sowie den offenen Ausgabeformaten OTF2 [8] und CUBE4 [9]. Während der Instrumentierung werden *Sensoren* in die eigentliche Applikation eingebracht, die zur Laufzeit das Messsystem aktivieren um Metriken wie z. B. Zeiten oder Hardware-Counter aufzuzeichnen und in den neuen Datenformaten abzuspeichern. Die bisher unterstützten Programmier-Modelle umfassen umfassen serielle, OpenMP-basierte, MPI-basierte sowie hybride, eine Kombination von OpenMP und MPI, Modelle. Die Architektur von Score-P ist in Abbildung 1 dargestellt. Durch die zwei neuen Datenformate und Adaptionen in den Analysewerkzeugen um diese neuen Formate zu unterstützen stellt sich die Interoperabilität der verschiedenen Werkzeuge wie in Abbildung 2 dar.

Neben der verbesserten Interoperabilität und der Vermeidung von Redundanzen in der Entwicklung ist es das Ziel von SILC und PRIMA der Nutzergemeinde Werkzeuge auch über die Projektlaufzeiten hinaus in Produktionsqualität auf allen gängigen HPC-Systemen zur Verfügung zu stellen. Dies stellt erhöhte Anforderungen an Skalierbarkeit, Portabilität, Robustheit und zu tolerierbarem Overhead.

2 Technologie und Infrastruktur

Um die hoch gesteckten Qualitätsziele zu erreichen wurden von Beginn an viele Ressourcen in den Aufbau einer robusten Infrastruktur investiert.

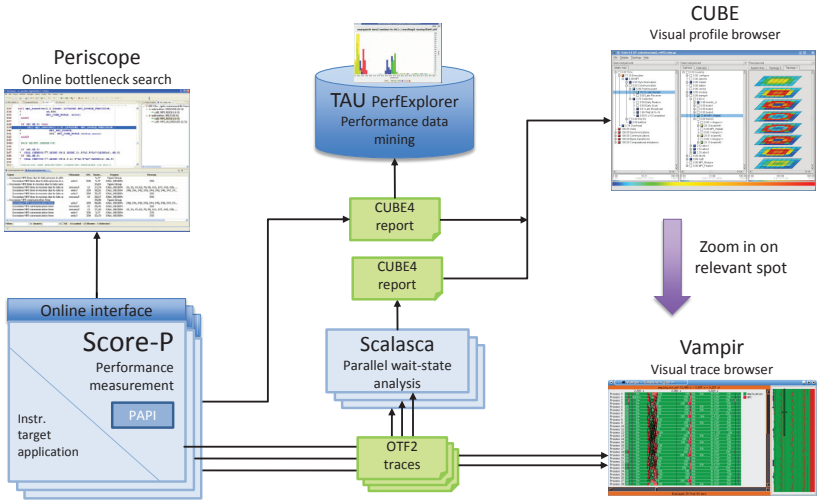


Abb. 2. Integrierte Werkzeugumgebung

2.1 Programmiersprachen und Entwicklungsumgebung

Die Ziel-Plattformen für Score-P sind aktuelle und zukünftige HPC-Systeme. Die Anwendungen, die dort in der Regel anzutreffen sind sind in Fortran, C und C++ geschrieben. Man kann also davon ausgehen, dass C, C++ und Fortran Compiler vorhanden sind. Dies ist für z. B. Java und Python nicht immer gegeben.

Die während der Instrumentierung einzufügenden Sensoren bestehen aus Bibliotheken, die gegen die Applikation gelinkt werden. Um maximale Portabilität zu erreichen wurden die Score-P Bibliotheken deshalb in C99 implementiert. Im MPI-Fall wird das Gros der Funktionalität noch durch eine in Fortran geschriebene Datei ergänzt. Eine Implementierung in C++ kam auch in die engere Wahl – die stärkeren Abstraktionsmöglichkeiten und die von der STL bereitgestellte Funktionalität waren im Hinblick auf Wartbarkeit und Entwicklungstempo ansprechend – wurde aber wegen schwierig zu handhabender Linker-Probleme verworfen.

Die Werkzeuge zur Instrumentierung sind größtenteils in C++ implementiert, rufen ihrerseits jedoch Shell- und AWK-Skripte auf.

Das GNU-autotools basierte Buildsystem wurde durch Shell-, m4- und AWK-Code ergänzt um die im HPC-Bereich häufig anzutreffende Herausforderung der Cross-Compilierung für die Nutzer möglichst transparent zu gestalten.

Es wird keine Entwicklungsumgebung vorgeschrieben, die Entwickler entscheiden sich hier nach persönlichen Vorlieben. Verbreitet werden Varianten von Emacs und Vim verwendet, vereinzelt aber auch integrierte Benutzerumgebungen (IDEs), wie Eclipse oder Code::Blocks.

Jeder Entwickler benötigt allerdings spezielle, teilweise gepatchte, Versionen der GNU autotools und des Quellcode-Formatierer uncrustify. Zur Generierung von Distributionspaketen wird zusätzlich noch doxygen erforderlich.

2.2 Entwicklungsinfrastruktur

Der Score-P Quellcode wird in mehreren, miteinander verbundenen, Subversion-Repositories [4] verwaltet. Jedes dieser Repositories enthält eine eigenständige Komponente. Die Struktur innerhalb der Repositories ist gleich so dass gemeinsam benötigte Funktionalität wie z. B. die Erzeugung von Dokumentation und große Teile des Buildsystems, in ein weiteres Repository ausgelagert werden konnte.

Die Repositories werden ergänzt durch Trac-Umgebungen [5], die als Subversion-Frontend, Wiki und Issue-Tracking-System eingesetzt werden.

Der oben erwähnte Quellcode-Formatierer wird in den Commit-Prozess eingebunden. Neuer Code der nicht den Formatierungs-Richtlinien entspricht gelangt erst gar nicht in das Repository. Somit kann eine einheitliche Formatierung sichergestellt werden. Die Entwickler sehen damit erreichte konsistente Formatierung durchweg als positiv an.

Ein Commit wir auch automatisch verhindert falls bestimmte vorgeschriebene Includes nicht vorhanden bzw. an einer falschen Stelle platziert sind. Dies verhindert mühsam zu findende Bugs.

Um Änderungen an Interfaces unmittelbar publik zu machen werden nach Commits in bestimmten Interface-Verzeichnissen Emails an die Entwickler-Mailingliste verschickt. Dies dient einerseits der Information und andererseits der Detektion unbeabsichtigter Interface-Änderungen.

2.3 Testprozesse und Testumgebungen

Um Syntaxfehler und Kompatibilitätsprobleme auf verschiedenen Plattformen frühzeitig zu erkennen, nutzen wir ein Continuous-Integration-System basierend auf dem Trac-Plugin Bitten [1]. Nach jedem Commit werden auf verschiedenen Plattformen Builds und Tests – Unit, Regression sowie Integration Tests – mit relevanten Compiler-MPI Kombinationen angestoßen. Die Plattformen umfassen nicht nur dedizierte Build-Server sondern auch aktuelle HPC-Systeme.

Nach erfolgreichen Tests wird immer ein Distributions-Paket inklusive Dokumentation erstellt und im Trac-Wiki zum Download angeboten.

Trotz des hohen Automatisierungsgrades und der recht guten Testabdeckung schleichen sich immer wieder Fehler in die Codebasis ein. Um die Anzahl der Fehler weiter zu verringern und neuen Entwicklern frühzeitig Feedback zu geben, planen wir die Einrichtung des Code-Review Systems Review Board [2].

Zur Kommunikation unter den Entwicklern wurden Mailinglisten eingerichtet die den Austausch über das Wiki ergänzen.

3 Entwicklungsprozess

Score-P wird von einer örtlich und zeitlich verteilt operierenden Gruppe entwickelt. Dies stellt erhöhte Anforderungen an Kommunikation und Planung. Um diesen Anforderungen zu genügen werden halbjährliche Treffen anberaumt auf denen die Planung für die nächsten sechs Monate besprochen wird. Dabei stehen die Spezifizierung von Interfaces und die Definition messbarer Ziele im Vordergrund. Davon ausgehend werden im Folgenden Arbeitspakete für die einzelnen Komponenten spezifiziert. In Kleingruppen werden dann technische Aspekte der

Arbeitspakete sowie die Interaktion zwischen Arbeitspaketen erörtert. Die Ergebnisse dieses Prozesses werden im Trac als Tickets, Milestones, Spezifikation oder Dokumentation festgehalten.

Sobald die Interfaces stabil sind erfolgt die Entwicklung neuer Features in der Regel in separaten Branches die bei Fertigstellung in den trunk gemergt werden. Dieses Vorgehen erlaubt die parallele Entwicklung verschiedener Features mit minimalen Reibungsverlusten.

Die während der Implementierung auftretenden neuen Erkenntnisse erfordern u. U. eine Änderung der ursprünglichen Planung. In zwei-wöchentlichen Entwickler-Telefonkonferenzen werden neben Statusmeldungen auch Planungsänderungen besprochen. Daneben dienen die Mailinglisten als Plattform zum Austausch.

In vielen Fällen ist die direkte Interaktion zweier Entwickler die effizienteste Variante, ein Problem anzugehen.

Darüber hinaus finden monatliche Telefonkonferenzen mit weniger technischen Agenden statt.

4 Herausforderungen

Der Hauptteil der Entwicklungsarbeiten wird von vollständig Drittmittel-geförderten Informatikern, Physikern, Ingenieuren und Doktoranden in einem verteilten Team geleistet. Viele der Team-Mitglieder waren anfangs mit der Problemdomäne nicht vertraut, was jedoch durch Unterstützung von erfahrenen Kollegen abgefedert werden konnte. Darüber hinaus waren Programmiererfahrungen der Entwickler stark unterschiedlich verteilt. Das Spektrum reichte von Anfänger bis zehn und mehr Jahren Erfahrung.

Eine große Herausforderung des verteilten Teams waren die unterschiedlichen Interessen die durch die Fixierung auf das eigene Analysewerkzeug zu erklären sind. Es dauerte seine Zeit bis man versuchte, auch die Sicht der anderen Projektpartner während der Planung zu antizipieren. Für die Doktoranden kam erschwerend hinzu dass SILC und PRIMA keine Forschungsprojekte sind, was sich teilweise deutlich in mangelndem Engagement bemerkbar machte.

Des weiteren hat Score-P besonders hohe Portabilitäts-Anforderungen, da es im HPC-Bereich eine große Vielfalt teilweise einzigartiger Plattformen gibt. Der Aufwand um diese Portabilitäts-Anforderungen in einer nachhaltigen Weise hauptsächlich im Buildsystem zu implementieren wurde stark unterschätzt. Auch der Aufwand zum Aufbau der stark automatisierten Infrastruktur wurde zu Projektbeginn nur unzureichend gesehen. Von den Investitionen in diesem Bereich wird die Qualität der Software aber noch lange nach Projektende profitieren.

5 Zusammenfassende Einschätzung

Score-P basiert im Wesentlichen auf etablierten Werkzeugen, wurde aber von Grund auf neu entwickelt. Dabei wurde von Anfang an auf erhöhte Flexibilität und Erweiterbarkeit im Hinblick auf neue Programmiermodelle und Architekturen geachtet. Ein weiteres Augenmerk wurde auf stabile Interfaces gelegt. Die Änderungen in diesem Bereich sind moderat und werden durch automatische Benachrichtigungen angezeigt. Der Umfang des Projektes, zieht man die teilweise

Unerfahrenheit der Entwickler in Betracht, ist etwas zu ambitioniert ausgefallen. Das liegt unter anderem daran, dass Zeitschätzungen durchweg zu positiv ausgefallen sind. Die Eigenheiten von HPC-Systemen und die damit einhergehenden Schwierigkeiten wurden unterschätzt. Ein fertig nutzbares Buildsystem für parallele, cross-compilierbare Projekte existierte nicht und wurde erst im Rahmen dieses Projektes entwickelt. Es findet mittlerweile schon in zwei anderen Projekten Verwendung. Die speziell in diesem Projekt vorhandene starke Kopplung von Komponenten sorgte für Verzögerungen bei den Integrationstests. Viele Implementierungs- und kleinere Design-Fehler, die teilweise erst Monate nach Projektende zu Tage traten, hätten durch die Nutzung eines Code-Review Tools frühzeitig erkannt werden können.

Positiv hervorzuheben sind alle Punkte, die mit Automatisierung verbunden sind. Das Continuous Integration System hat uns bei der frühzeitigen Erkennung von Fehlern und Portabilitätsproblemen sehr geholfen und wird kontinuierlich ausgebaut. Die konsistente Codeformatierung wurde akzeptiert und wird als vorteilhaft betrachtet, trotz der emotionalen Natur diese Themas. Das Verweigern eines Commits bei fehlenden oder fehlerhaft platzierten Includes verhindert das Auftreten von schwierig zu findenden Fehlern. Die Automatische Erzeugung von Dokumentation beim Erzeugen eines Distributionspakets und die Bereitstellung dieses nach jedem erfolgreichen Build runden die Automations-Bemühungen ab. Als voraussichtlich letzte Infrastruktur-Erweiterung ist Einführung eines Code-Review Systems geplant.

Auch die Zusammenarbeit der Entwicklergruppen mit unterschiedlichem Interessenhintergrund ist nach einer Abtastphase als durchweg als positiv zu sehen.

Literatur

1. Bitten – A continuous integration plugin for Trac. http://bitten.edgewall.org.
2. Review Board. http://www.reviewboard.org.
3. Score-P. http://www.score-p.org.
4. Subversion. http://subversion.apache.org.
5. Trac – Integrated SCM and Project Management. http://trac.edgewall.org.
6. Virtual Institute - High Productivity Supercomputing. http://www.vi-hps.org.
7. Shajulin Benedict, Ventsislav Petkov, and Michael Gerndt. PERISCOPE: An online-based distributed performance analysis tool. In Matthias S. Müller, Michael M. Resch, Alexander Schulz, and Wolfgang E. Nagel, editors, *Tools for High Performance Computing 2009*, pages 1–16. Springer, Berlin/Heidelberg, 2010.
8. Dominic Eschweiler, Michael Wagner, Markus Geimer, Andreas Knüpfer, Wolfgang E. Nagel, and Felix Wolf. Open trace format 2: The next generation of scalable trace formats and support libraries. In Koen De Bosschere, Erik H. D'Hollander, Gerhard R. Joubert, David Padua, Frans Peters, and Mark Sawyer, editors, *Applications, Tools and Techniques on the Road to Exascale Computing*, volume 22 of *Advances in Parallel Computing*, pages 481 – 490, 2012.
9. Markus Geimer, Pavel Saviankou, Alexandre Strube, Zoltán Szebenyi, Felix Wolf, and Brian J. N. Wylie. Further improving the scalability of the Scalasca toolset. In *Proc. of PARA 2010: State of the Art in Scientific and Parallel Computing, Minisymposium Scalable tools for High Performance Computing, Reykjavik, Iceland*. Springer, June 2010. (to appear).
10. Markus Geimer, Felix Wolf, Brian J. N. Wylie, Erika Abraham, Daniel Becker, and Bernd Mohr. The Scalasca performance toolset architecture. *Concurrency and Computation: Practice and Experience*, 22(6):702–719, April 2010.
11. Andreas Knüpfer, Holger Brunst, Jens Doleschal, Matthias Jurenz, Matthias Lieber, Holger Mickler, Matthias S. Müller, and Wolfgang E. Nagel. The Vampir Performance Analysis Tool Set. In *Tools for High Performance Computing*, pages 139–155. Springer, July 2008.

12. Andreas Knüpfer, Christian Rössel, Dieter an Mey, Scott Biersdorff, Kai Diethelm, Dominic Eschweiler, Markus Geimer, Michael Gerndt, Daniel Lorenz, Allen D. Malony, Wolfgang E. Nagel, Yury Oleynik, Peter Philippen, Pavel Saviankou, Dirk Schmidl, Sameer S. Shende, Ronny Tschüter, Michael Wagner, Bert Wesarg, and Felix Wolf. Score-p – a joint performance measurement run-time infrastructure for periscope, scalasca, tau, and vampir. In *Proc. of 5th Parallel Tools Workshop*, 2011. (to appear).
13. S. Shende and A. D. Malony. The TAU Parallel Performance System, SAGE Publications. *International Journal of High Performance Computing Applications*, 20(2):287–331, 2006.

OpenFlipper – An Open Source Geometry Processing and Rendering Framework

Jan Möbius, Leif Kobbelt

Department of Computer Science 8
RWTH-Aachen University
moebius@cs.rwth-aachen.de

Abstract. OpenFlipper is an extensible open source geometry processing and rendering framework. It provides a software toolkit and development platform for geometry processing algorithms and is mainly developed in the context of various academic research projects. Nevertheless some companies are already using it as a toolkit for commercial applications. The system is fully modular such that it can be adapted to various research and commercial applications. Currently it consists of about 120 plugins developed in about 12 man years. They provide functions like selection metaphors, rendering, mesh deformation, scripting, file import/export, user interface modifications, input device control and many others. Due to the high flexibility, developers can choose from the existing components and create their own customized OpenFlipper version. The large base of algorithms heavily reduces the implementation effort and therefore the development costs.

Geometry Processing, Framework, Computer Graphics

1 Overview

Currently a lot of work is being done to increase the implementation efficiency in research and development in the field of geometry processing algorithms. Various frameworks have been developed but most of them suffer from two major limitations. On the one hand, they focus on one special type of geometric primitives, namely triangle or polygonal meshes. However there are a lot of applications where multiple types of data are needed, e.g., isosurfaces get extracted from volumetric data or meshes are generated from point clouds.
On the other hand, the applications are usually published under the GPL [1] and are therefore not usable in industrial projects leaving out another large group of potential users and contributors.
Our main goal is to provide a common software development platform for the majority of the geometry processing community. Researchers should be able to use the framework to significantly speed up development and focus on their actual research project as most basic functionality is readily available. Industry should obtain a common platform to build their software upon while end users benefit from a unified look and feel for all algorithms using the framework. Additionally students should be able to use OpenFlipper to learn basic geometry processing techniques while not having to deal with the rendering, file IO or other software architecture components not relevant for a specific thesis topic. This large user community heavily improves the development as existing code from other people can be re-used and combined to new algorithms.

2 Evolution

OpenFlipper started from a variety of different software packages. On the one hand there was a closed source application that was developed in an industry collaboration and which was used by a smaller group of experts.

Additionally a large set of small research applications existed that were developed for special purposes. These applications were always developed from scratch and implemented their own rendering, IO system or user interactions. All of them were usually only usable by their developers and when they left the department, a lot of these programs could not be used anymore.

For lectures there existed small software frameworks that were only used in one specific context. Hence students learned to work with these frameworks but it was not used again for a bachelor and master thesis.

The information and requirements from these various applications have been collected in order to derive the overall design goals for OpenFlipper.

The development was started on Linux with a custom build system that was sufficient for a long time. For porting to other platforms, we required a more flexible build system and finally chose CMake which simplified a lot of porting tasks.

Another big speedup for the development and increased stability was achieved by integrating unit tests and continuous integration.

3 Design Goals

The ultimate motivation for the implementation of a general application framework is that in most research groups various projects are being worked on in parallel and that a considerable percentage of the functionality is shared between those projects. Hence the primary goal of a common framework is to avoid implementation and code redundancy by providing a central repository of modules, e.g., for user interfaces, data file handling, rendering and others.

Another observation is that in research applications different aspects of software engineering become relevant. In the initial basic research phase, the software framework should allow the programmer to focus on the algorithm itself and it should provide a test bed in which different variants of the same algorithm can be evaluated.

At a later stage of the development, the implemented functionality needs to be tested extensively. While the major mode of usage for our framework is that of an interactive application, extensive testing and repetitive execution of similar procedures is more effective in some kind of batch mode. Hence our system comes with a scripting interface that enables the flexible meta-implementation of control procedures that build on top of the available functionality in the C++ implemented modules.

Finally, we want our framework to also support the dissemination and commercialization of geometry processing functionality. This imposes two additional requirements at the technical as well as at the legal level. First, the design of ergonomic and intuitive user interfaces needs to be supported effectively. On the legal level we have to resolve licensing issues (this is why we have put our framework

under the LGPL [2] license) and we have to offer a mechanism to exchange and license functionality without exchanging source code.

Our solution to the latter issue is that individual modules are implemented as plugins that can be loaded (as pre-compiled code) at runtime. The plugins can access the central functionality through a flexible API that also supports the communication between plugins. In addition we included a license checker mechanism into the framework which enables the protection of individual plugins by identifying the computer on which the program is running.

4 Current state of the project

Various research projects use OpenFlipper as their software foundation. Due to the large set of algorithms that have already been integrated, the amount of software development as well as the total time to finish these projects has been reduced. Furthermore some of these projects are already used in closed source projects by using the integrated license management system.

Currently, the OpenFlipper framework is also used intensively in the context of our computer graphics and geometry processing teaching curriculum since it is perfectly suited for student projects. They get familiar with the framework at an early time of their study by writing simple algorithms. Later on this knowledge can be used when starting a bachelor or master thesis and significantly reduce the period of vocational adjustment.

The free version of OpenFlipper is available from the Mac App Store and has been downloaded 6000 times in about 5 months since the software has been published.

As OpenFlipper is an open source software that can be compiled on various software platforms, a community is starting to develop and improve the framework. Students implement new algorithms during practical courses and companies create new plugins and give additional functionality back to the community. Therefore the amount of available functionality rapidly increases and due to the increasing user base, the algorithms are well tested and the stability of the overall framework has been be improved.

5 Technology & Infrastructure

The OpenFlipper framework is written in C++. We chose the QT cross-platform application and User Interface framework [3] as our system interface library. This framework runs on various platforms and allows shipping of versions for Linux, Windows and MacOS. As the build system has to support that flexibility, we chose CMake [4] which supports the same platforms and provides a lot of features for configuring and packaging the software. Furthermore it can generate project files for various integrated development environments. This was required as we develop for different platforms and user groups, where different IDEs are used (e.g. Visual Studio on Windows, XCode on Mac or Eclipse on Linux).

To maintain code stability and to prevent regressions especially when several developers work on the same project, continuous integration as well as unit- and integration tests are a prerequisite. CMake already provides functions to run tests and

collect the results on the different development platforms. Additionally we integrated Google Test [5] for executing unit tests for small components like the basic data structures. The batch mode of OpenFlipper is used for integration tests. OpenFlipper can be started with or without a graphical user interface and executes a given script. The algorithm output can then be checked, for regressions or errors and the result is given back to the developer. As the scripting system does not require interaction, the whole testing process is automated using CMake.

The scripts for OpenFlippers batch system are written in a language that is based on JavaScript(QTScript). QT comes with the integrated QTScript language, which we extend to work with the geometry processing algorithms. Plugins in OpenFlipper can easily provide scriptable functions so there is no additional coding effort for the developers. As the language is based on JavaScript, a lot of developers are already familiar with the syntax and can easily combine and test algorithms in OpenFlipper.

To automate the tests, Jenkins [6] is used as the continuous integration server. OpenFlipper is automatically built when a user checks in new code into the Subversion repository. The code is compiled on all available platforms and if an error occurs, the developer gets an e-mail containing the compiler output and the error. After compilation the tests are run and if they succeed, setup packages are generated for all architectures. A release is then simply created by copying the generated packages to the website.

6 Development Process

The project is managed in a Subversion repository with multiple branches. Each sub project is stored in a separate branch that is automatically built by Jenkins. The developers who could be students, research assistants or others, get write access to their branches.

For lectures, snapshots of OpenFlipper are generated that stay fixed for the semester. These snapshots are typically stripped down versions of the whole framework. As it is plugin based, we could easily remove functionality that is not required for the lecture and hand out a small specialized version. The well tested build system significantly reduces the amount of work required by the students to set up their system. Hence they can immediately start with the actual implementation.

Researchers always get direct access to their branches. As the research process requires a lot of testing and flexibility, the users can completely manage their branch. If new functionality is ready and a project is finished, it will be merged into the main branch after review. A big advantage of the framework is that the researchers can rely on a vast set of algorithms and interactions. Research code is often created in rush to code sessions. This code needs to be revised before publication but as a large percentage of code is used directly from the framework and is of good quality, the amount of code that needs to be cleaned up is usually small.

After research projects are finished, it can happen that the developer leaves the department. The valuable code is than orphaned. But as it is still tested by the continuous integration server and it can be used by all others working with the framework, it is not lost anymore which happens with a lot of standalone software that can only be used by the developer himself.

Patches to the core of the framework need to be reviewed and tested before they are commited to maintain the stability and sometimes the backward compatibility of the system. After an update of the framework, the other branches are then updated using Subversion externals that can be configured separately for every subproject.

7 Conclusion

We developed a system that is helpful for the majority of the geometry processing community. It turned out to be very instrumental when developing research applications and for converting research code into commercial applications.

Nevertheless some problems occur due to the increasing number of developers. The time to manage updates or to dispatch working sets increases. Furthermore, a common documentation style should be maintained among all software components and plugins which turns out to be a growing problem with an increasing size of developers.

By now OpenFlipper is used for many different purposes. Students use it for learning the basics of computer graphics and especially geometry processing and Researchers use it to develop and test algorithms while companies benefit by directly using research code in an end user application with only little effort. All of these users provide updates and extensions to the freely available part of the framework. Therefore the functionality of OpenFlipper rapidly increases and it provides a growing toolbox of basic algorithms, interaction metaphors and rendering functions which have been given back to the community by the various contributors as free software.

All this functionality simplifies development and enables people to focus on their creative work when inventing new algorithms, making OpenFlipper a valuable framework for the community.

A detailed paper about the OpenFlipper framework will be published in [7]. The source code and executables are available at our website [8] or via the Mac App Store.

References

1 GNU General Public License, http://www.gnu.org/licenses/gpl.html
2 GNU Lesser General Public License, http://www.gnu.org/licenses/lgpl.html
3 QT cross-platform application and UI framework, http://qt.nokia.com
4 CMake cross-platform make, http://www.CMake.org
5 Google C++ Testing Framework, http://code.google.com/p/googletest/
6 Jenkins Continuous integration server, http://jenkins-ci.org/
7 OpenFlipper: An Open Source Geometry Processing and Rendering Framework
(Lecture Notes in Computer Science Volume 6920, to appear)
8 OpenFlipper website http://www.openflipper.org

Der Energie Navigator

Thomas Kurpick, Claas Pinkernell, Bernhard Rumpe

Lehrstuhl Software Engineering
{kurpick,pinkernell,rumpe}@se-rwth.de

Abstract. Der Energie Navigator ist ein Softwaresystem für die automatisierte Massendatenerfassung und -aufbereitung von Gebäudesensordaten. Die Software erlaubt es automatische Analysen auf Basis dieser Daten durchzuführen und bietet diverse Visualisierungsarten und Reporting-Mechanismen an. Zusätzlich können Spezifikationen von Gebäuden erstellt und automatisiert überprüft werden.
Nutzer sind Architekten und Energieexperten, die damit Gebäude auf ihren Verbrauch und Einsparpotenzial spezifizieren, analysieren und optimieren. Die Software wir zurzeit mit Partnerunternehmen in einer breit angelegten Feldstudie erprobt.
Es wird ein komplexes (automatisiertes) Buildsystem auf Ant-Basis genutzt. Eine Besonderheit bei dem Projekt ist die Entwicklung einer Backup-Komponente, die eine automatisierte Migration der Produktivdaten nach jedem Release ermöglicht. Die Entwicklung erfolgt agil und in kurzen Release-Zyklen.

Keywords

1 Einleitung

Der Energie Navigator ist ein Softwaresystem für die automatisierte Massendatenerfassung und -aufbereitung von Gebäudesensordaten. Die Software erlaubt es automatische Analysen auf Basis dieser Daten durchzuführen und bietet diverse Visualisierungsarten und Reporting-Mechanismen an. Zusätzlich können Spezifikationen von Gebäuden erstellt und automatisiert überprüft werden.

Die Struktur des vorliegenden Beitrags gliedert sich wie folgt: Zunächst wird das Projekt vorgestellt. Danach wird eine kurze Übersicht über die Architektur des Energie Navigators und die verwendeten Technologien im Projekt skizziert. Im Anschluss wird auf den Entwicklungsprozess im Projekt eingegangen. Als Nächstes wird auf die Herausforderungen auf der menschlichen als auch der technologischen Seite eingegangen und mit einer zusammenfassenden Einschätzung abgeschlossen.

2 Der Energie Navigator

Der Energie Navigator ist ein System für die Spezifikation, Analyse und Optimierung von Gebäuden und technischen Anlagen [1]. Im Mai 2010 wurde die synavision GmbH mit dem Ziel der Weiterentwicklung des Systems zur Produktreife gegründet. Das erste Release (Version 1.0) wurde im Mai 2011 veröffentlicht.

Zum Zeitpunkt der Veröffentlichung dieses Beitrags befindet sich das System bei mehreren Firmen in der Erprobungsphase. Gleichzeitig wird am Release 2.0 des Energie Navigators weitergearbeitet und dieser ständig um neue Konzepte und Funktionen erweitert.

Die Begleitforschung für das System ist durch Mittel des BMWi und der EU gefördert worden. Weitere Förderanträge für neuartige Konzepte sind bereits in Planung.

Der Energie Navigator wird zusammen mit dem Institut für Gebäude- und Solartechnik der TU Braunschweig und der energydesign braunschweig GmbH entwickelt, die für die fachlichen Konzepte der Anwendungsdomäne verantwortlich sind. Die Planung, die Entwicklung und das Management werden in Aachen vom Lehrstuhl Software Engineering der RWTH und der synavision GmbH durchgeführt. Zum aktuellen Zeitpunkt sind an der Entwicklung ca. 20 Personen (4-5 Assistenten, ca. 13-17 Studenten) beteiligt.

Das System wird von den Nutzern positiv bewertet. Es werden regelmäßig Verbesserungsvorschläge und zusätzliche Anregungen mitgeteilt.

3 Architektur der Energie Navigator Plattform

In der Abbildung 1 ist die Architektur des Energie Navigators auf hoher Abstraktionsebene zu sehen. Das System teilt sich in die folgenden Komponenten:

- *Data Import*: Importieren der Massendaten in das System
- *Preprocessing*: Bereinigen der Massendaten von Fehlern
- *Specification*: Anlegen und Verwalten der Gebäudespezifikation
- *Analysis*: Auswertung der Massendaten auf Basis der Spezifikation
- *Reporting*: Erstellen von Berichten mit den Ergebnissen der Analyse

Weitere Informationen über die Energie Navigator Plattform finden sich unter [1, 2, 3, 4].

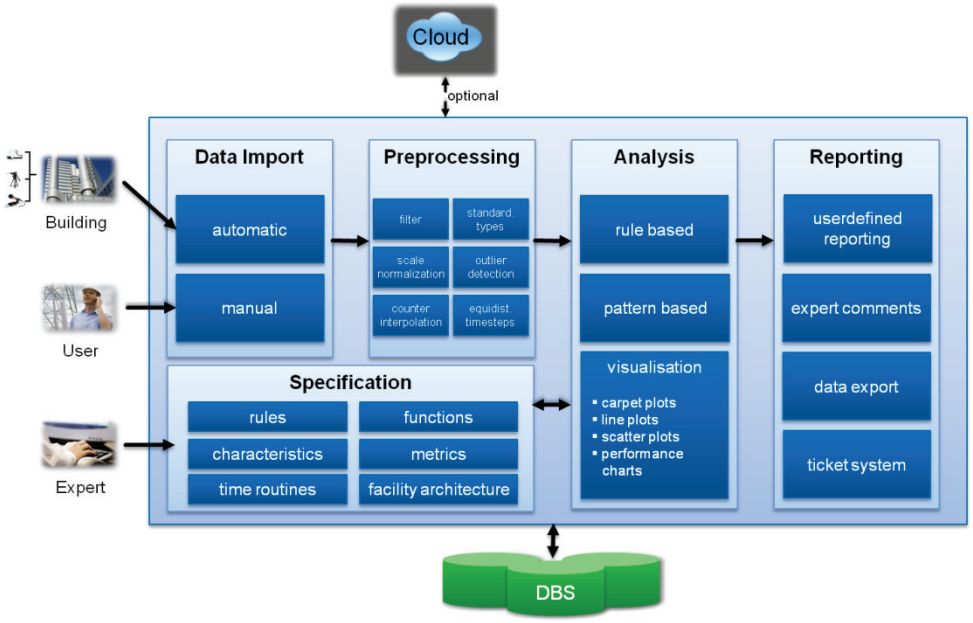


Abbildung 1: Architektur des Energie Navigators

4 Technologie

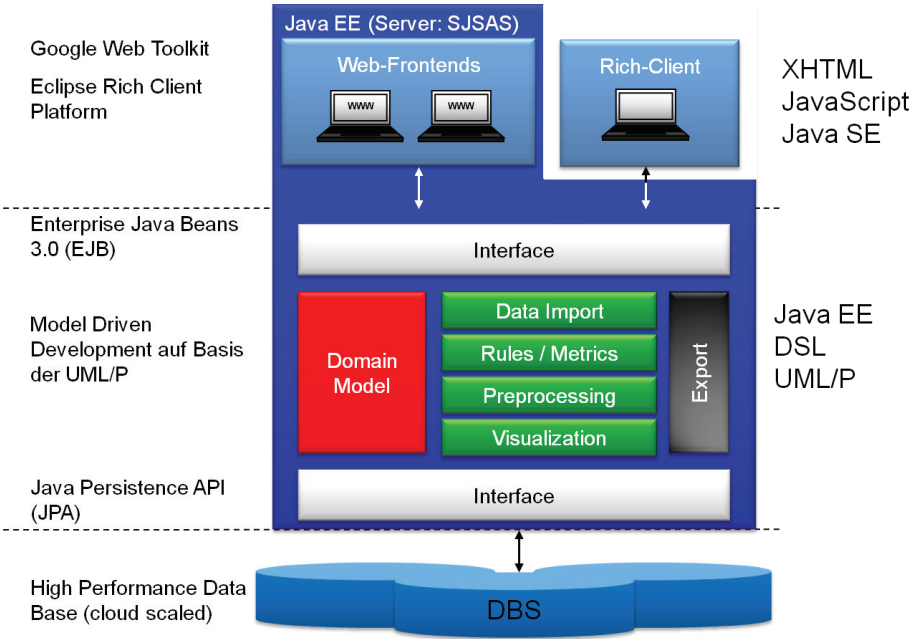


Abbildung 2: Eingesetzte Technologien

In der Laufzeitumgebung des Energie Navigators werden viele verschiedene Technologien eingesetzt. In der Abbildung 2 sind die wichtigsten aufgeführt.

Auf der Serverseite werden Java EE, PostgreSQL und massendatenfähige Datenbanken eingesetzt. Auf der Frontend-Seite werden neben einer Rich Client Platform (RCP) Anwendung auch Web-Applikationen mit dem Google Weg Toolkit (GWT) entwickelt. Die Software wird modellbasiert auf Grundlage eigener DSLs entwickelt, die mit Hilfe des DSL-Frameworks Monticore [5] erstellt wurden.

5 Infrastruktur

Die Entwicklungs- und Erprobungsinfrastruktur des Energie Navigators besteht aus virtuellen Maschinen. Es gibt zwei Server für Produktivsysteme, zwei Server für die kontinuierliche Integration sowie Nightly-Builds und ca. 25 virtuelle Entwicklungsumgebungen. Die virtuellen Entwicklungsumgebungen sind vorkonfiguriert, so dass neue Entwickler sich keine eigene Umgebung für das System erstellen müssen. Der Zugriff erfolgt entweder über einen Remote-Zugang auf den Maschinen oder die virtuelle Maschine wird von den Entwicklern lokal ausgeführt.

Neben den virtuellen Maschinen wird das Projektportal SSELab für die Entwicklung verwendet [6]. Von den angebotenen Services des SSELabs werden die Services Subversion, Trac, Wikis, Mailing-Liste und Feedback-System [7] eingesetzt.

Da die Plattform des Energie Navigators in mehrere Komponenten gegliedert ist, wurde auch viel Aufwand in die Build-Skripte des Systems gesteckt. Mit Hilfe der Build-Skripte ist es möglich, automatisiert die IDE zu installieren sowie das Produktiv-System und das Nightly-Build-System zu erstellen.

Die Buildskripte dienen auch dazu, die Plattform auf mehreren virtuellen Maschinen auszuliefern und zu installieren. Ziel dieser Build-Infrastruktur ist es, den Entwicklern zu ermöglichen, die Plattform zu bauen, auszuliefern und auszuführen, ohne Wissen über das ganze System zu haben.

6 Entwicklungsprozess

Der Entwicklungsprozess des Energie Navigators ist iterativ. Im Moment werden zwei unterschiedliche Entwicklungszweige gepflegt. Der Zweig für die Version 1.0, die schon bei Kunden produktiv eingesetzt wird, und der Zweig 1.1 der neue Funktionen beinhaltet und von einigen Kunden getestet wird. Jeden Monat wird ein neues Release für die Version 1.0 und 1.1 erstellt und ausgeliefert (*Release often*). Der Release-Prozess ist automatisiert, so dass für ein Release des Systems nur ca. eine Stunde gebraucht wird (*Release fast*).

Die Planung des Systems erfolgt über die Releasezyklen. Dafür wird das Ticketsystem Trac als Managementwerkzeug eingesetzt. Im Projekt gibt es verschiedene Rollen, die in der Entwicklung für Aufgaben oder Komponenten verantwortlich sind, z.B. eine studentische Hilfskraft für die Tests des Systems. Dieser Student überwacht die Testausführung im Continous Integration System, erstellt neue Test-Datensätze für die Integrationstests und erhöht die Testfallabdeckung in den einzelnen Komponenten.

Jede Komponente des Systems hat einen wissenschaftlichen Mitarbeiter als Komponentenverantwortlichen. Er überwacht die Qualität der Komponente und fällt Designentscheidungen für die Implementierung und Weiterentwicklung.

Als zusätzliche Aufgaben fallen in dem Projekt das Management der Entwicklungstätigkeiten, das Sammeln und Bewerten von Rückmeldungen der Kunden und das Zuweisen von offenen Tickets an die Entwickler an.

7 Herausforderungen

Das Projekt bietet zwei unterschiedliche Herausforderungen. Auf der einen Seite ist das Management der beteiligten Personen und des Teams eine Herausforderung und wird durch den universitären Kontext zusätzlich erschwert; auf der anderen Seite führt die Evolution des Systems und des Projekts zu technischen Problemen, die aktiv angegangen werden müssen.

7.1 Menschen

Neben den wissenschaftlichen Mitarbeitern, die lange im Projekt arbeiten, sind auch studentische Hilfskräfte, Auszubildende und Studenten, die ihre Bachelor-, Master- und Diplom-Arbeit im Rahmen des Projekts erstellen, beteiligt. Gerade die Studenten, die ihre Studienarbeiten schreiben sind nur sehr kurz im Projekt involviert (ca. 4-6 Monate im Durchschnitt). In dieser Zeit muss sich der Student mit dem System fachlich und technisch vertraut machen, um anschließend seine Aufgabe zu lösen und geeignet in die Plattform zu integrieren.

Um eine möglichst effektive Entwicklung zu gewährleisten, hat das Projekt zwei Projekträume eingerichtet. In diesen Räumen ist Platz für 12 Entwickler. Jeder Platz

hat einen eigenen PC, bietet aber auch Anschlussmöglichkeiten für Laptops. Während der Woche gibt es drei feste Zeiträume in denen sich Mitarbeiter des Lehrstuhls und Studenten in diesen Räumen treffen und zusammen entwickeln oder die nächsten Schritte besprechen. Von jedem Beteiligten wird erwartet mindestens einmal pro Woche an einem Treffen teilzunehmen.

Die gemeinsame Zeit wird hauptsächlich genutzt, um die Architektur der verschiedenen Komponenten und Implementierungsaufgaben zu besprechen. Zusätzlich sollen durch die Treffen technische Probleme schnell behoben werden, die während der Entwicklung entstehen und mit der komplexen Infrastruktur zusammenhängen. Neben den wöchentlichen Treffen finden auch gelegentlich interne Vorträge zu speziellen Themen statt. Als Beispiel sind hier Vorträge zum „Einsatz von Sonar“, „Coding Guidelines“ und Architektur Reviews genannt. Neben den fachlichen Treffen finden aber auch Treffen für das Teambuilding statt; es wurde u.a. eine große Release-Party für das Release 1.0 gefeiert. Außerdem findet jährlich ein Sommergrillen mit allen Entwicklern statt.

7.2 Evolution des Projekts

Die Evolution des Systems und des Projekts muss aktiv gemanagt werden. Am Anfang dauerte ein Release des Systems ca. einen Arbeitstag. Durch mehrere studentische Arbeiten wurden die einzelnen gewachsenen Buildskripte der unterschiedlichen Komponenten auf eine gemeinsame Basis gestellt, und so der Release-Prozess auf ca. eine Stunde reduziert. So wurde auch der Release-Zyklus von 4-5 Monaten zwischen zwei Releases auf einen Monat verkürzt.

Kontrovers wurde die Wahl zwischen Ant und Maven als Buildtool diskutiert. Im Rahmen des Projekts wurde sich für Ant entschieden, da hier die Anpassungen an den Projektkontext leichter zu bewerkstelligen waren als der Umstieg auf Maven.

Durch die Erhöhung des Release-Taktes wurde es nötig, den Kunden auch die Möglichkeit der Datenmigration zu geben. Gerade am Anfang des Projekts wurden noch häufig Änderungen am Datenmodell vorgenommen, so dass Daten nicht ohne weiteres übernommen werden konnten. Deshalb wurde eine Komponente für die Datenmigration zwischen mehreren Releases erstellt, die eine automatische Migration der alten Daten erlaubt.

Durch die verteilte Entwicklung mit den Entwicklern in Aachen und den fachlichen Experten in Braunschweig wurde auch die Feedback-Komponente [7] integriert. Sie ermöglicht dem Endbenutzer im Energie Navigator, Rückmeldungen für die Entwickler zu erstellen. Für die entfernte Kommunikation werden Skype und Teamviewer eingesetzt.

Am Anfang wurden Modelle im Projekt nur in der Analyse genutzt. In den gemeinsamen Treffen wurden Klassen-, Sequenzdiagramme oder Automaten an der Tafel erstellt und diskutiert, um anschließend die Implementierung von Hand zu programmieren. Im weiteren Verlauf des Projekts wurde dann auf eine modellbasierte Entwicklung umgestellt, die aus den Modellen die Implementierung generiert.

Dabei wurde festgestellt, dass der Austausch von handgeschriebenem Code durch generierten Code nicht ohne weiteres möglich ist. Die Generatoren lassen sich gut für neue Komponenten des Systems einsetzten, aber schlecht für die Migration. Grund

sind Sonderfälle, die bei der manuellen Implementierung umgesetzt wurden, nicht aber bei der Generierung.

8 Zusammenfassende Einschätzung

Das Projekt des Energie Navigators erlaubt mit seinem Entwicklungsprozess eine schnelle Entwicklung von neuen Features. Die festen Arbeitszeiten/-treffen führen im Projekt zu einer guten Kommunikation zwischen allen Beteiligten. Das agile Vorgehen ermöglicht Entscheidungen schnell zu treffen und auf geänderte Anforderungen der Kunden zu reagieren.

Es gibt aber noch einige negative Punkte in dem Projekt, die zu verbessern sind. Die mittlerweile komplexe Infrastruktur des Projekts erfordert einen hohen Einarbeitungsaufwand. Die Qualität der Entwicklung leidet unter den vielen studentischen Entwicklern, die selbst nur für kurze Zeit in dem Projekt mitarbeiten. Die Nutzung des Continous Integration Systems für jeden Commit ist nicht sinnvoll, da mittlerweile die Generierungs-/Kompilierungszeit des ganzen Systems länger als 20 Minuten dauert. Im Moment sind deshalb weitere Arbeiten für die Build-Infrastruktur geplant, die beide Punkte erheblich verbessern sollen.

9 References

1 M. N. Fisch, T. Kurpick, C. Pinkernell, S. Plesser, and B. Rumpe. The energy navigator-a web based platform for functional quality mangement in buildings. In Proceedings of the 10th International Conference for Enhanced Building Operations (ICEBO 10), Kuwait City, Kuwait, 2010.

2 M. N. Fisch, M. Look, S. Plesser, C. Pinkernell and B. Rumpe, Der Energie-Navigator - Performance-Controlling für Gebäude und Anlagen. In Technik am Bau (TAB) - Fachzeitschrift für Technische Gebäudeausrüstung, 04/2011:36-41, bau verlag, March 2011.

3 M. N. Fisch, S. Plesser, C. Pinkernell and B. Rumpe, Software für die energieoptimierte Betriebsführung von Gebäuden. In BINE Informationsdienst Projektinfo (ISSN 0937-8367), 14/10:1-4, October 2010.

4 M. N. Fisch, S. Plesser, C. Pinkernell and B. Rumpe, Virtueller Prüfstand für Gebäude: Der Energie Navigator. In XIA Intelligente Architektur (ISSN 0949-2356), Vol. 73, 10-12/10:68-70, October 2010.

5 H. Krahn, B. Rumpe, and S. Völkel. MontiCore: a Framework for Compositional Development of Domain Specic Languages. International Journal on Software Tools for Technology Transfer (STTT), 12(5):353-372, September 2010.

6 C. Herrmann, T. Kurpick, B. Rumpe SSELab: A Plug-In-Based Framework for Web-Based Project Portals In: Proceedings of the 2nd International Workshop on Developing Tools as Plug-Ins (TOPI 2012) at ICSE 2012, June 3, Zurich, Switzerland, 2012.

7 C. Herrmann, T. Kurpick, and B. Rumpe, Agile User-Feedback and its Management through the SSELab Feedback System, RWTH Aachen, Tech. Rep. AIB-2012-11, 2012. [Online]. Available: http://aib.informatik.rwthaachen.de/2012/2012-11.pdf

Scalasca

David Böhme, Marc-André Hermanns und Felix Wolf

German Research School for Simulation Sciences GmbH

1 Das Scalasca-Projekt

Scalasca [1,4] ist eine Sammlung von Software-Werkzeugen für die Leistungsanalyse und -optimierung paralleler Programme. Es ist speziell für den Einsatz auf massiv parallelen Systemen im Bereich des wissenschaftlichen Rechnens entwickelt. Es erlaubt die Identifikation von Aufrufpfaden mit hohem Ressourcenverbrauch mit Hilfe klassischer Laufzeitprofile. Zusätzlich erlaubt es die Aufzeichnung von Event Traces (Ereignisspuren) und unterstützt mit diesen eine automatische Suche nach ineffizienten Kommunikationsmustern sowie die Identifikation der Ursachen von Wartezuständen [3]. Die Spuranalyse ist voll parallelisiert und wurde bereits auf bis zu 294.912 Prozessorkernen erfolgreich eingesetzt. Dies ist eines der Alleinstellungsmerkmale des Scalasca Toolsets im Vergleich zu anderen im Bereich des Hochleistungsrechnens verfügbaren Leistungsanalysewerkzeugen.

Scalasca ist eine gemeinsame Entwicklung des Jülich Supercomputing Centre (Forschungszentrum Jülich) und des Lehrstuhls für Parallele Programmierung (German Research School for Simulation Sciences). Zum einen dient es als Plattform für Forschung im Bereich der Leistungsanalyse paralleler Programme, d.h. es wird durch die Bearbeitung aktueller Forschungsfragen vorangetrieben und erweitert. Zum anderen wurde und wird Scalasca erfolgreich zur Optimierung von wissenschaftlichen sowie industriellen Simulationsanwendungen eingesetzt. Scalasca ist für alle wesentlichen Plattformen verfügbar und auf einer Vielzahl von Hochleistungsrechnern auf der ganzen Welt für Benutzer installiert. Im Rahmen des VI-HPS Projekts [2] werden in regelmäßigen Abständen Workshops und Tutorials durchgeführt, bei denen Entwickler von HPC-Anwendungen den Umgang mit Scalasca lernen. Dadurch wird Scalasca einer breiteren Nutzerbasis bekanntgemacht. Gleichzeitig erhalten wir wertvolle Rückmeldung von den Nutzern.

Der Leistungsanalyseprozess durchläuft verschiedene Phasen. Diese Phasen sind: Instrumentierung und Messung, die Analyse der Messergebnisse sowie die Auswertung der Analyseergebnisse. Abbildung 1 zeigt den üblichen Ablauf im Umgang mit Scalasca. Die Benutzerapplikation, die mit Scalasca analysiert werden soll, wird erst instrumentiert, d.h. es wird zusätzlicher Code eingefügt, der die relevanten Messdaten erzeugt. Scalasca unterstützt die Leistungsbewertung von parallelen Programmen, die das Message Passing Interface (MPI), OpenMP oder die Kombination beider Modelle nutzt. Scalasca instrumentiert auf Quellcode- und Linker-Ebene. Für Benutzerfunktionen und OpenMP-Konstrukte wird der Messcode während des Übersetzungsvorgangs eingefügt. MPI stellt eine spezielle Schnittstelle bereit, um eine vorinstrumentierte Bibliothek zur Link-Zeit dem restlichen Messsystem hinzuzufügen.

Die Messumgebung kann entweder direkt ein Laufzeitprofil erstellen, oder detaillierte Messdaten in der Form von Event Traces aufzeichnen, die durch eine weitere Scalasca-Komponente analysiert werden können. Das Aufrufpfad-Profil

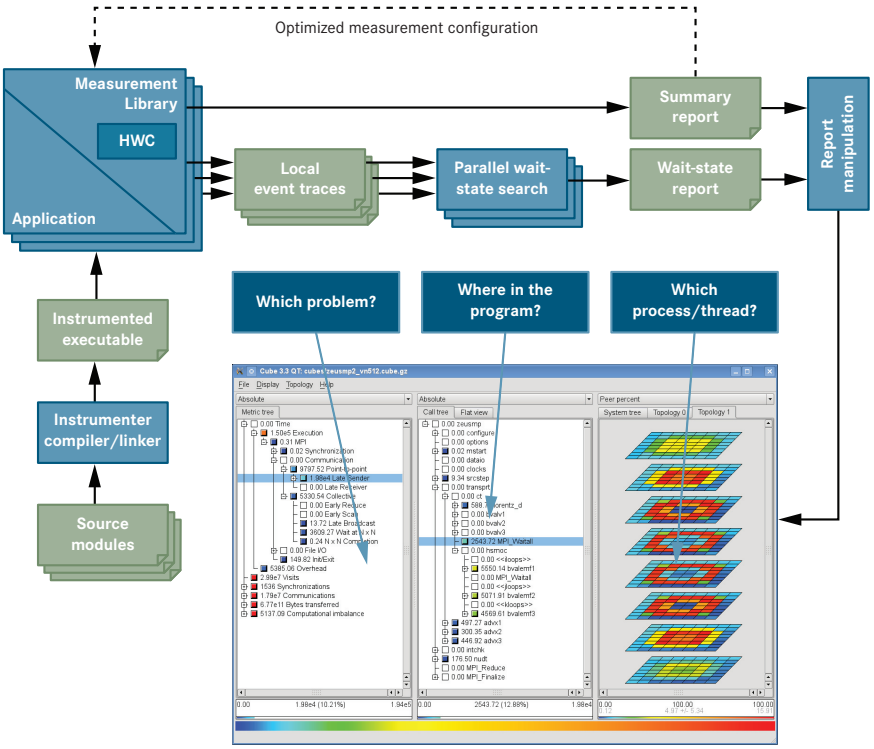


Abb. 1. Scalasca Workflow

beinhaltet weniger Informationen über den Messlauf, benötigt aber im Vergleich zum Tracing erheblich weniger Speicherplatz. Deshalb sollte erst das Aufrufpfad-Profil erzeugt werden, mit Hilfe dessen man nur die für eine detailliertere Messung wichtigen Teile der Applikation auswählt. Sowohl das Laufzeitprofil, als auch der Analysebericht kann zur Auswertung in einer grafischen Benutzeroberfläche angezeigt und interpretiert werden.

Scalasca ist das Nachfolgeprojekt von Kojak [5], welches zwischen 1998 und 2005 entwickelt wurde. Beide Projekte basieren auf dem Prinzip der wissensbasierten Suche nach ineffizienten Kommunikationsmustern. Bei Scalasca liegt der Fokus allerdings auf der Skalierbarkeit der Aufzeichnung und Analyse der Messdaten. Während bei Kojak die Analyse noch seriell verlief, wird die Analyse bei Scalasca von einem hoch skalierenden, parallelen Programm durchgeführt. Dennoch teilten sich die Projekte lange Zeit eine gemeinsame Codebasis. Da Scalasca zum Großteil aus öffentlichen Mitteln finanziert wird und der gesamten HPC-Community zur Verfügung stehen soll, wird Scalasca unter einer Variante der NewBSD-Lizenz veröffentlicht, die das freie Verbreiten, Verwenden und Modifizieren des Codes erlaubt.

2 Technologie und Infrastruktur

2.1 Programmiersprachen und Entwicklungsumgebung

Scalasca besteht aus mehreren wesentlichen Komponenten (Instrumentierung und Messumgebung, Trace-Analyse, und Reporting/Grafische Benutzeroberflä-

che), die entsprechend der jeweiligen Anforderungen in unterschiedlichen Programmiersprachen implementiert sind. Zusammengenommen umfasst das ganze Projekt etwa 140.000 Quellcodezeilen, die sich auf 54% C++, 42% C, 4% Shellskripte und weniger als 1% Fortran verteilen.

Die Instrumentierungs- und Messumgebung umfasst davon insgesamt 70.215 Codezeilen, davon 285 in Fortran, 5.322 in Shellskripten, 8.422 in C++, und 56.186 in C. Die Messumgebung umfasst Bibliotheken, die zu den zu untersuchenden Programmen hinzugelinkt werden. Diese Bibliotheken sind zur Maximierung der Portabilität in C geschrieben. Ein kleiner Teil der Messumgebung für Fortran-Programme ist ebenfalls in Fortran geschrieben. Die Infrastruktur zur Quellcodeinstrumentierung ist zum größten Teil in C++ und als Shellskript implementiert.

Die Traceanalysekomponente umfasst 25.395 Quellcodezeilen in C++. Die Reporting- und GUI-Komponenten für das Erstellen, Bearbeiten und Anzeigen der Analysereports bestehen aus 45.034 Quellcodezeilen, davon 42.422 in C++, 2.256 in C und 336 Zeilen in Shellskripten. Die grafische Benutzeroberfläche nutzt das Qt-Framework und ist daher in C++ implementiert. Die Bibliothek zum Schreiben der Analysereports muss in die in C implementierte Messumgebung eingebunden werden, und ist daher ebenfalls in C geschrieben.

Es wird keine “Default”-Entwicklungsumgebung vorgeschrieben, die Entwickler entscheiden sich hier nach persönlichen Vorlieben. Verbreitet werden Varianten von Emacs und Vim verwendet, vereinzelt aber auch integrierte Benutzerumgebungen (IDEs), wie Eclipse oder Code::Blocks.

2.2 Entwicklungsinfrastruktur

Der Quellcode wird in einem Subversion-Repository verwaltet, das am Forschungszentrum Jülich gehostet wird. Zusätzlich wird eine Trac-Umgebung als Subversion-Frontend, Wiki und Bug-Tracking-System eingesetzt.

Da Scalasca nicht in Binärform, sondern als Quellcodedistribution angeboten wird, verwenden wir ein portables, Makefile-basiertes Buildsystem. Bisher nutzten wir selbstentwickelte Konfigurations- und Buildskripte, die sich einfach auf verschiedene, teilweise exotische (Cross-compilation etc.) Plattformen anpassen ließen. Im Zuge eines größeren, andauernden Umbaus wird das Buildsystem derzeit auf die GNU Autotools umgestellt.

2.3 Testprozesse und Testumgebungen

Um Syntaxfehler und Kompatibilitätsprobleme auf verschiedenen Plattformen frühzeitig zu erkennen, nutzen wir einen automatischen Nightly-Build-Prozess. Hierbei wird der aktuelle Entwicklungshauptzweig einmal am Tag ausgecheckt und in möglichst vielen Konfigurationen übersetzt. Die Entwickler werden per E-Mail über dabei entdeckte Fehler oder Warnungen informiert. Dabei werden bei den verwendeten Compilern hohe Warnlevel eingestellt, um sicherzustellen, dass problematische Code-Teile frühzeitig erkannt werden.

Vor einem für die Öffentlichkeit bestimmten “großen” Release findet darüber hinaus ein umfassender Funktionstest auf verschiedenen Plattformen statt. Da dieser weitestgehend manuelle Release-Test recht personalintensiv ist, wird er derzeit nur vor wichtigen Releases durchgeführt. Darüber hinaus testen Entwickler die von ihnen entwickelten Features in eigener Verantwortung selbst.

3 Entwicklungsprozess

Neue Funktionalität wird zunächst in eigenen, sogenannten Feature-Branches entwickelt. Die Namen dieser Branches folgen einem festen Schema, aus dem sich die dort entwickelte Funktionalität und der zuständige Entwickler ableiten lässt. Der Hauptentwicklungszweig (trunk) bildet die Basis für die regelmäßigen Realeases. Bugfixes und kleine Änderungen können direkt im Hauptzweig vorgenommen werden. Größere Änderungen müssen allerdings zunächst in den Feature-Branches entwickelt werden, und werden dann nach Absprache in den Hauptzweig übernommen.

Welche Funktionalitäten entwickelt werden richtet sich nach verschiedenen Faktoren, unter anderen den wissenschaftlichen Zielen des Projektes selbst, Meilensteinen von geförderten Drittmittel-Projekten, sowie notwendige Anpassungen an neue Plattformen und deren Skalierbarkeitsanforderungen.

Das Scalasca Projekt wird an zwei Standorten entwickelt, dem JSC in Jülich und der GRS in Aachen. Jede der Organisationen hat einen Manager, der sich um Projektanträge und die strategische Ausrichtung des Projekts kümmert. Am JSC gibt es ausserdem zwei festangestellte Senior-Developer, welche den Nutzersupport gewährleisten und die generelle Wartung und Weiterentwicklung des Projektes überwachen. Ein Großteil neuer Funktionalität wird von den derzeit zehn Doktoranden und wissenschaftlichen Mitarbeitern, die über Drittmittelprojekte finanziert sind, implementiert. Im Rahmen von Promotionen werden auch Master- und Bachelor-Arbeiten betreut, die allerdings kleineren Umfangs sind, und meist nicht-kritische Komponenten, allgemeine Verbesserungen oder Konzeptstudien umfassen. Die Entwicklerbasis im Scalasca Projekt kommt größtenteils aus der Informatik, wodurch bereits ein breites generelles technisches Verständnis gewähleistet wird. Für Weiterbildung im Bereich des High Performance Computing können Kurse am JSC in Anspruch genommen werden.

4 Herausforderungen

Ein Hauptteil der Entwicklungsarbeiten wird von Doktoranden und drittmittelgeförderten Mitarbeitern geleistet. Insbesondere bei den Projekten von Doktoranden ergibt sich die Schwierigkeit, dass die Entwicklung der neuen Funktionalität lange in einem eigenen Zweig vorangetrieben wird. Im Spannungsfeld von Software-technischen Punkten, wie Stabilität, Wartbarkeit, oder Portabilität, und dem wissenschaftlichen Beitrag, liegt der Fokus oft auf letzterem. Dies führt dazu, dass am Ende der Promotion umfangreiche neue Funktionalität in einem Seitenzweig bereitsteht, mit dem im Detail allerdings nur der Doktorand selbst vertraut ist. Auch ist nicht zwangläufig gegeben, dass die Qualität der Implementierung weit über den Status einer Konzeptstudie hinausgeht, womit weitere Anpassungen erforderlich werden, um die entsprechende Funktionalität in den Hauptzweig zu integrieren. Dies führt dazu, dass diese Funktionalität oft komplett reimplementiert wird, um direkt auch Verbesserungen an der ersten Umsetzung einarbeiten zu können. Während dieser Reimplementierung kann dann der Fokus auf die Software-technische Qualität gelegt werden. Diese Integration kann allerdings dadurch erschwert und erheblich verzögert werden, dass nach einer Promotion oder Abschlussarbeit der Originalentwickler einer bestimmten Funktionalität aus dem Team ausscheidet, und die Integration nicht

durch wissenschaftliche Drittmittel gedeckt werden kann. Der langwierige Integrationsprozess sorgt leider oft auch für enttäuschte Nutzer, die neue, bereits in wissenschaftlichen Veröffentlichungen vorgestellte Features nicht zeitnah in den aktuellen stabilen Softwarereleases vorfinden.

Die Abhängigkeit von Drittmitteln birgt zudem ein hohes Risiko der Abwanderung von Entwicklern durch fehlende Anschlussfinanzierung. Dies kann insbesondere die längerfristige Planung des Projektes und Unterstützung der Nutzer erschweren.

Spezifische Herausforderungen in Scalasca ergeben sich auch aus der umfangreichen, seit langem gewachsenen Codebasis. Nur wenige Entwickler haben einen umfassenden Überblick über das gesamte Projekt, und neue Entwickler benötigen eine gewisse Einarbeitungszeit. Dies ist insbesondere für Bachelor- oder Masterarbeiten relevant, die in einem kurzen Zeitraum abgeschlossen werden müssen und daher nur wenig Zeit zur Einarbeitung erlauben.

Des weiteren hat Scalasca besonders strenge Anforderungen an die Portabilität, da es im HPC-Bereich eine große Vielfalt teilweise einzigartiger Plattformen gibt. Portabilität ist allerdings selten der Fokus der Drittmittelprojekte. Diese zielen eher auf neue Funktionalität und neue wissenschaftliche Ergebnisse ab, so dass reine Engineering-Aspekte wie die Portabilität und Robustheit aus anderen Bereichen finanziert werden müssen.

Die Zielumgebungen im HPC-Bereich bringen auch einige spezifische organisatorische und technische Herausforderungen mit sich. So wird zum Testen der Skalierbarkeit und der Portabilität Zugriff auf möglichst viele verschiedenen HPC-Plattformen und entsprechend auch Rechenzeit benötigt. Durch gute Kontakte zu Hardwareherstellern und diversen wissenschaftlichen Rechenzentren ist dies glücklicherweise gegeben. Teilweise (wie beispielsweise im Fall der IBM Blue Gene/Q) bekommen wir auch Zugriff auf Prototyp-Systeme, um Scalasca frühzeitig auf neue Plattformen portieren zu können. Rechenzeit auf Produktionsmaschinen für Skalierbarkeitstests muss beantragt werden, was zusätzlichen organisatorischen Aufwand bedeutet. Dabei muss sich Scalasca dem selben Auswahlverfahren stellen wie jedes andere HPC-Projekt auch, was die Gefahr birgt, im Fall einer Ablehnung entsprechenden Einschränkungen bei Skalierbarkeitstests ausgesetzt zu sein.

Einige klassische Software-Engineering Verfahren, wie etwa Unit-Tests, sind für parallele Programme auf HPC-Systemen wie Scalasca nur eingeschränkt anwendbar. So gibt es unseres Wissens nach derzeit kein Unit-Testing Framework für parallele, insbesondere MPI-Programme. Die stark maschinenspezifischen Wege zum Ausführen von Programmen auf HPC-Systemen machen das Erstellen plattformübergreifender automatischer Tests ebenfalls nicht einfach.

5 Zusammenfassende Einschätzung

Im Großen und Ganzen funktioniert der Softwareentwicklungsprozess für Scalasca gut. Durch den Fokus auf Stabilität, Qualität und Portierbarkeit im Hauptentwicklungszweig gelingt es uns, Scalasca als stabiles Werkzeug für Anwendungsentwickler aufzubauen und gleichzeitig als Plattform für neue, experimentelle Funktionalität zu nutzen. Dieser Erfolg ist nicht zuletzt auf den hohen Anteil an Entwicklern mit Software-Engineering Hintergrund im Team zurückzuführen.

Darüber hinaus profitieren wir von zwei Entwicklern ohne Drittmittelfinanzierung, die sich vorrangig der langfristigen strategischen Planung des Projekts und der Software-Qualität widmen können.

Schwierigkeiten bestehen nach wie vor bei der Integration von größeren Forschungsergebnissen in die stabile Plattform. Der Weg vom Prototypen über einen Integrationszweig bis in die Releases ist oft (zu) langwierig. Auch die Testpraxis lässt noch Wünsche offen, was sich zum Teil durch die komplexen Ausführungsumgebungen der HPC-Systeme erklärt, die es erfordern, eigene Testumgebungen zu entwickeln. Um hier Abhilfe zu schaffen, soll in näherer Zukunft ein Continuous-Integration Prozess eingeführt werden.

Literatur

1. Scalasca. http://www.scalasca.org.
2. Virtual Institute - High Productivity Supercomputing. http://www.vi-hps.org.
3. David Böhme, Markus Geimer, Felix Wolf, and Lukas Arnold. Identifying the root causes of wait states in large-scale parallel applications. In *Proc. of the 39th International Conference on Parallel Processing (ICPP, San Diego, CA, USA)*, pages 90–100. IEEE Computer Society, September 2010.
4. Markus Geimer, Felix Wolf, Brian J. N. Wylie, Erika Abraham, Daniel Becker, and Bernd Mohr. The Scalasca performance toolset architecture. *Concurrency and Computation: Practice and Experience*, 22(6):702–719, April 2010.
5. Felix Wolf and Bernd Mohr. KOJAK - A tool set for automatic performance analysis of parallel applications. In *Proc. of the 9th Euro-Par Conference, Klagenfurt, Austria*, volume 2790 of *Lecture Notes in Computer Science*, pages 1301–1304. Springer, August 2003. Demonstrations of Parallel and Distributed Computing.

AProVE – Automated Program Verification Environment

Carsten Otto

LuFG Informatik 2, RWTH Aachen University, Germany

Zusammenfassung. AProVE ist ein automatischer Terminierungsbeweiser für eine Reihe von Programmiersprachen, z.B. Termersetzungssysteme, Prolog, Haskell und Java.

1 Zusammenfassung des Projekts

Das System AProVE wird seit 2001 entwickelt, um die Automatisierbarkeit aktueller Forschungsergebnisse im Bereich der Terminierungsanalyse zu demonstrieren. Die Praktibilität und Nützlichkeit theoretischer Ergebnisse kann so mit Experimenten nachgewiesen werden. Hierfür tritt AProVE an jährlichen Wettbewerben an, bei denen die Stärke der verfügbaren Terminierungs-Tools anhand realer Programme verglichen wird[1]. Hierbei hat AProVE seit 2004 jeden dieser Wettbewerbe gewonnen. Aktuell steht AProVE als Download zur Verfügung, außerdem bieten die Entwickler ein Webinterface[1] an.

1.1 Status

AProVE ist in der Lage Programme verschiedener Programmier-Paradigmen auf Terminierung zu analysieren, so dass die Relevanz der verwendeten Techniken klar erkennbar ist. Seit 2006 ist es möglich, die Terminierung von funktionalen Programmen zu analysieren[2]. Hier wird die funktionale Programmiersprache Haskell 98 unterstützt. Ebenfalls seit 2006 bietet AProVE die Möglichkeit zur Analyse von Logikprogrammen[3] in der Programmiersprache Prolog. Im Jahr 2010 wurde AProVE erweitert, so dass seitdem auch imperative Programme (in Java bzw. Java Bytecode geschrieben) untersucht werden können[4].

Zusätzlich zur Unterstützung für diese drei real benutzten Programmiersprachen enthält AProVE eine große Auswahl an Techniken, die Termersetzungssysteme [7] (inklusive diverser Varianten) auf ihre Terminierung analysieren können. Die hier implementierten Techniken sind der eigentliche Kern des Systems, da die Terminierung der in den oben genannten Programmiersprachen geschriebenen Programme durch eine Umwandlung in Termersetzungssysteme untersucht wird.

AProVE ist auch in der Lage Aussagen zur Komplexität eines Termersetzungssystems zu treffen[8], enthält einen Theorembeweiser[6] und kann als Constraint-Checker[5] eingesetzt werden.

[1] http://aprove.informatik.rwth-aachen.de

Aktuell besteht das System aus ca. 670 000 Zeilen Programmcode, die auf ca. 8 000 Klassen verteilt sind. Im Laufe der ca. 10-jährigen Entwicklung waren ca. 55 Entwickler beteiligt, die ca. 24 000 Commits angelegt haben.

1.2 Ausblick

`AProVE` wird weiterentwickelt, so dass die Ergebnisse verbessert werden und weitere Klassen von Problemen gelöst werden können. Beispielsweise ist die Unterstützung von C-Programmen geplant, wobei der Fokus auf der Behandlung der Pointerarithmetik liegt. Weiterhin ist vorgesehen, zusätzlich zum Terminierungsverhalten auch Komplexitätseigenschaften zu Programmen zu berechnen, die in den erwähnten real benutzten Programmiersprachen geschrieben sind.

Für die Zukunft ist außerdem angedacht, eine für den Endbenutzer intuitiv benutzbare graphische Benutzeroberfläche (GUI) zur Verfügung zu stellen, beispielsweise durch eine Integration in die Entwicklungsumgebung Eclipse.

2 Technologie & Infrastruktur

2.1 Software

Als Programmiersprache wird Java verwendet, wobei die Entwicklung hauptsächlich mit Eclipse stattfindet. Das Versionskontrollsystem unserer Wahl ist git, bis 2009 wurde cvs verwendet. Für Eclipse wurden Einstellungen erarbeitet, die beispielsweise durch den Autoformatter eine für unser Projekt gewünschte Quellcodeformatierung erwirken. Zusätzlich sind Einstellungen für das Plugin Checkstyle vorhanden, die auf typische Probleme im Quellcode hinweisen (fehlende Dokumentation, lange Methoden, etc.). Das Verwenden dieser Einstellungen ist für die Entwickler freiwillig.

Während der Entwicklungsphase ist es nötig (unabhängig von der aktuell implementierten Technik) die Auswirkungen auf eine möglichst große Sammlung von Eingabe-Programmen zu betrachten. Hierbei können beispielsweise Programmierfehler auffallen oder ein Optimierungsbedarf für bestimmte Spezialfälle erkannt werden. Um Tests auf solchen Programmsammlungen zu ermöglichen, wurde ein umfangreiches Benchmark-System entwickelt, das komfortabel über die Kommandozeile und ein Webinterface bedient werden kann. Mit Hilfe des Webinterfaces ist es auch leicht möglich, die Software mit anderen Versionen oder Programmen zu vergleichen oder die Unterschiede alternativer Implementierungen besser zu verstehen.

2.2 Hardware

Als Hardware stehen den Mitarbeitern eigene Rechner zur Verfügung, auf denen lokal gearbeitet werden kann. Für Studierende wird ein großer Rechnerpool zur Verfügung gestellt, wobei die eigentliche Entwicklung per X-Forwarding bzw. VNC auf vier gemeinsam genutzten Servern stattfindet. Das Webinterface sowie

intern benutzte Serverdienste laufen auf einem dieser Server. Einige Mitarbeiter und Studierende nutzen zusätzlich private Rechner zur Entwicklung. Die rechnerisch aufwändigste Arbeit, nämlich das Ausführen diverser Benchmark-Läufe auf tausenden von Beispielen, wird größtenteils auf Blade-Computern durchgeführt, die im Rahmen des Forschungs-Clusters „Ultra High-Speed Mobile Information and Communication“ (UMIC) angeschafft wurden und den `AProVE`-Entwicklern zur Verfügung gestellt werden. Insgesamt können hier 15 Rechner mit zusammen 280 CPU-Cores und 1 136 GByte RAM benutzt werden.

3 Entwicklungsprozess

`AProVE` ist im Kern eine etablierte Plattform, auf deren Basis im Laufe der Zeit verschiedene neue Techniken hinzugefügt werden. Der übliche Entwicklungsprozess ist, dass die Mitarbeiter mit einer neuen Idee auf dem aktuellen Softwarestand experimentieren und so auf ein Ergebnis hinarbeiten. Wenn diese Implementierung weit genug gereift ist, wird der so entwickelte Prototyp für ein konkretes Ziel (beispielsweise eine Veröffentlichung oder ein Wettbewerb) angepasst. Hierbei wird der neue Code in das bestehende System integriert, wobei üblicherweise nur geringe Anpassungen nötig sind. In zeitkritischen Situationen, beispielsweise wenn eine Frist eingehalten werden muss, wird der entwickelte Code mit klarem Fokus auf das jeweilige Ziel umgeschrieben und erweitert. Aus dieser Herangehensweise ergeben sich normalerweise gute und für die Veröffentlichung bzw. den Wettbewerb wichtige experimentelle Ergebnisse, allerdings entsteht hierbei auch suboptimaler Quellcode.

Da `AProVE` hauptsächlich zur Demonstration verschiedener Techniken entwickelt wird, findet für in sich abgeschlossene Programmteile gewöhnlicherweise keine Weiterentwicklung bzw. Codepflege mehr statt. Bei Arbeiten an der Plattform oder an Teilimplementierungen, die auch für weitere Veröffentlichungen benutzt und angepasst werden müssen, wird hingegen schon während der Entwicklung auf verständlichen und wartbaren Programmcode geachtet und auch nach der eigentlichen Fertigstellung Zeit investiert, um die üblicherweise in zeitkritischen Phasen der Entwicklung hinzugefügten „Problemstellen“ zu bereinigen.

4 Faktor Mensch/Mitarbeiter

Da `AProVE` das einzige Softwareprodukt des Forschungsgebietes ist, sind alle assoziierten Mitarbeiter und Studierende an der Entwicklung beteiligt. Hieraus ergibt sich eine stark schwankende Teamgröße und Zusammenstellung.

Für das Projekt findet ungefähr monatlich ein Treffen statt, bei dem alle Mitentwickler über ihren jeweiligen Status berichten. Ein Ergebnis dieser Treffen ist, dass die Mitarbeiter einen Überblick über die Arbeit der Kollegen bekommen und eventuell bei vorgestellten Problemen helfen können. Weiterhin findet auf diesen Treffen üblicherweise ein reger Austausch statt, bei dem spontan neue Ideen für die Weiterentwicklung erarbeitet werden.

Als Nebeneffekt dieser regelmäßigen Treffen erhöht sich auch die Motivation für den einzelnen Mitarbeiter, bis zum jeweiligen Termin vorzeigbare Ergebnisse präsentieren zu können.

Zusätzlich zu den Treffen werden die studentischen Mitarbeiter (Hiwis und solche, die an Abschlussarbeiten schreiben) individuell von einem wissenschaftlichen Mitarbeiter betreut. Da die Studierenden üblicherweise gemeinsam in einem Raum arbeiten, ist es auch üblich und gewünscht, dass diese sich untereinander helfen.

Das Team hat bisher sehr gute Erfahrungen damit gemacht, neue Teammitglieder durch das Ansprechen guter Studierenden aus inhaltlich nahen Vorlesungen oder Seminaren des Forschungsgebietes zu gewinnen. Gute Leistungen in den jeweiligen Veranstaltungen haben sich als guter Indikator für fachliches Verständnis und Interesse erwiesen.

4.1 Evolution

Durch die Entwicklung des *Dependency Pair*-Ansatzes [**?**] hat sich `AProVE` schnell zu einem generellen System zur Terminierungsanalyse entwickelt, das verschiedene Techniken modular benutzt. Die Entwicklung beschränkt sich entsprechend hauptsächlich auf die Pflege dieser Plattform und auf das Hinzufügen neuer Techniken.

Umbauarbeiten an der Plattform sind selten nötig und werden gemeinsamen mit allen Entwicklern besprochen und umgesetzt. Die Entwicklung einzelner Funktionalitäten wird hingegen in kleinen Arbeitsgruppen (bzw. durch einzelne Personen) vorgenommen. Hierfür hat sich die Verwendung von *branches* in GIT etabliert.

In der Anfangsphase des Projektes wurden neue Techniken explizit mit anderen Techniken verknüpft, so dass der Ablauf eines Terminierungsbeweises dadurch vorgegeben war. Der Entwickler hatte also nicht nur die Aufgabe, eine neue Technik zu entwickeln, sondern musste auch direkt dafür sorgen, dass diese sinnvoll angebunden ist. Da im Laufe der Zeit sehr viele Techniken entstanden sind, ist das Finden einer guten Kombination einer neuen Technik mit den vorher existierenden Techniken zunehmend schwerer geworden. Als Lösung für dieses Problem wurde eine Strategiesprache entwickelt, mit deren Hilfe es sehr leicht möglich ist komplizierte Abläufe einfach zu beschreiben, einzelne Details der Ansteuerung zu ändern und Teilstrategien wiederzuverwenden.

Funktionalitäten, die allgemeiner Natur sind und keiner Technik bzw. Veröffentlichung direkt zugeordnet werden können, sind erfahrungsgemäß problematisch in der Entwicklung und Pflege. Ein Beispiel hierfür ist die graphische Oberfläche (GUI) von `AProVE`. In den letzten Jahren wurden bisher drei verschiedene GUIs entwickelt, die alle nach einiger Zeit nicht mehr benutzbar waren. Das Hauptproblem hierbei ist, dass das Erweitern der GUI bei der Entwicklung von neuen Techniken für eine Veröffentlichung nicht im Vordergrund steht und die GUI-Pflege generell eine niedrige Priorität hat.

Weiterhin ist die Codepflege dieser Funktionalitäten problematisch, da der Programmcode üblicherweise über mehrere Jahre benutzt und weiterentwickelt

werden soll, allerdings schwierige Details aus Zeitgründen nur unzureichend dokumentiert sind. Hinzu kommt, dass aus verschiedenen Gründen üblicherweise studentische Hilfskräfte mit der Entwicklung beauftragt werden, da die wissenschaftlichen Mitarbeiter bereits mit anderen Projekten ausgelastet sind. Da diese studentischen Mitarbeiter üblicherweise nur kurz am Projekt beteiligt sind, ist es oft auch nicht mehr möglich, bei Problemen diese um Rat zu fragen.

Ergänzend zu der bereits beschriebenen Dokumentations-Problematik gibt es bei der Entwicklung von `AProVE` das Problem, das der zu dokumentierende Code größtenteils sehr komplex ist. Die nötigen Details der Implementierung sind in entsprechenden wissenschaftlichen Veröffentlichungen beschrieben, weshalb eine zusätzliche Dokumentation des Quellcodes sowohl redundant wäre als auch schwer umzusetzen ist.

Bedingt durch die große Quellcodebasis und den schlechten Dokumentationsstand ist es für den einzelnen Entwickler auch nicht möglich, zu bestehenden Problemene bereits implementierte Lösungen zu finden und zu benutzen. Im Ergebnis passiert es regelmäßig, dass nahezu gleiche Anforderungen zu Neuimplementierungen führen (beispielsweise existierenden mindestens drei Polynom-Implementierungen in `AProVE`). Das Problem wird dadurch verstärkt, dass durch Detailunterschiede der Implementierungen eine Zusammenführung meistens nicht sinnvoll ist.

5 Zusammenfassende Einschätzung

`AProVE` ist sehr gut geeignet, um die praktische Anwendbarkeit von Forschungsergebnissen demonstrieren zu können. Da eine zentrale Plattform vorhanden ist, ist es den Entwicklern auch sehr leicht möglich zusätzliche Funktionalitäten hinzuzufügen. Hierdurch entstehen im Zusammenhang mit dem `AProVE`-Projekt viele Veröffentlichungen, weiterhin kann die Software erfolgreich an den Terminierungs-Wettbewerben teilnehmen. Positiv ist auch, dass Mitarbeiter unterschiedlicher Erfahrungslevel gut zusammenarbeiten können, da die Arbeit für jeden Mitarbeiter größtenteils in eng abgesteckten Teilbereichen stattfindet. Dadurch, dass theoretische Ergebnisse in `AProVE` implementiert werden, liegt außerdem eine interessante Kombination von Theorie und Praxis vor.

Ein negativer Aspekt ist, dass das etablierte Entwicklungsmodell für eine stetig wachsende Programmcode-Basis sorgt, wovon aktuelle Mitarbeiter immer weniger verstehen. Durch fehlende oder mangelhafte Dokumentation ist es auch nicht in vertretbarer Zeit möglich, sofern nötig, diesen Programmcode an aktuelle Bedürfnisse anzupassen. Da die Ergebnisse der Veröffentlichungen immer mit der Implementierung reproduzierbar sein müssen, fallen grundlegende Design-Entscheidungen, die die Plattform direkt betreffen, zunehmend schwerer. Dieses Problem wird dadurch verstärkt, dass die Entwickler der älteren Techniken nicht mehr Teil des Teams sind und dementsprechend keine Hinweise dazu geben können, wie wichtig bestimmte Anpassungen sind.

Für die Zukunft wäre es gut, eine klare Projektrichtung zu kennen und entsprechend zu entwickeln: Je nach den Ansprüchen an `AProVE` ist es hier beispiels-

weise akzeptabel, unwartbaren Programmcode zu produzieren, sofern die für eine Veröffentlichung produzierten Ergebnisse gut sind und der Code anschließend nicht mehr verändert werden muss. Bei anderen Ansprüchen, beispielsweise wenn `AProVE` von anderen Entwicklern aktiv benutzt wird um eigene Programme zu verifizieren oder Fehler zu finden (beispielsweise ermöglicht durch eine Integration von `AProVE` in Eclipse), wäre eine Fokussierung auf Codepflege sinnvoll, so dass man in Zukunft leichter bzw. überhaupt auf der existierenden Implementierung aufbauen kann sowie Fehler schneller finden bzw. direkt vermeiden kann.

Literatur

1. termcomp. Leslie Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, 1994.
2. haskell. Leslie Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, 1994.
3. prolog. Leslie Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, 1994.
4. java. Leslie Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, 1994.
5. constraintchecker. Leslie Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, 1994.
6. theorembeweiser. Leslie Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, 1994.
7. trs. Leslie Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, 1994.
8. komplexitaet. Leslie Lamport, *LaTeX: A Document Preparation System*. Addison Wesley, Massachusetts, 2nd Edition, 1994.

SimWeld - 10 Jahre Schweißprozesssimulation am ISF

U. Reisgen, M. Schleser, O. Mokrov, A. Schmidt

Institut für Schweißtechnik und Fügetechnik (ISF) der RWTH Aachen University
SimWeld@isf.rwth-aachen.de

Abstract. SimWeld ist eine Software für die Simulation von Schweißprozessen dessen Entwicklung vor über 10 Jahren am Institut für Schweißtechnik und Fügetechnik (ISF) begonnen hat. Seitdem wurde es in verschiedenen öffentlich geförderten und privat finanzierten Projekten kontinuierlich weiterentwickelt. In dieser Zeit wurde das Entwicklerteam vor verschiedene Herausforderungen gestellt.
In dem vorliegenden Artikel wird zunächst die Software SimWeld, sowie dessen Evolutionsprozess und Struktur, vorgestellt. Dann wird auf die verwendete Technologie und Infrastruktur, gefolgt von einer Beschreibung des Entwicklungsprozesses, eingegangen. Schließlich werden die auf menschlichen, technologischen und funktionalen Faktoren beruhenden Herausforderungen, sowie die Maßnahmen, die aufgrund dieser Herausforderungen ergriffen worden sind, beschrieben.

Schweißen, Simulation, Softwareentwicklung, Forschungssoftware, Evolution

1 Schweißprozesssimulation mit SimWeld

Die computergestützte Schweißsimulation ist in drei Teilbereiche gegliedert: Struktur-, Prozess- und Werkstoffsimulation (Abbildung 1) nach Radaj [1]. Die Strukturssimulation betrachtet die Auswirkungen des Schweißens auf das Schweißstück, also zum Beispiel die Eigenspannung oder den Verzug. Die Prozesssimulation berechnet die Schweißbadgeometrie, die Prozessstabilität und den Prozesswirkungsgrad. Bei der Werkstoffsimulation werden mikroskopische und makroskopische Einflüsse auf den Werkstoff simuliert.

Am Institut für Schweißtechnik und Fügetechnik (ISF) wird seit 2001 kontinuierlich SimWeld, eine Software für die Schweißprozessimulation, im Rahmen von öffentlich geförderten und privat finanzierten Projekten entwickelt.

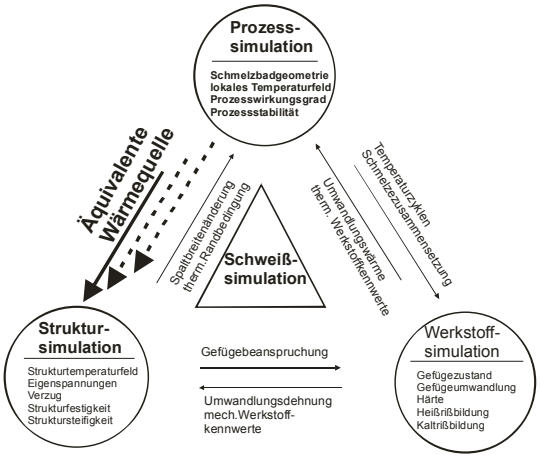


Abbildung 1: Kategorien der Schweißsimulation nach Radaj [1]

1.1 Allgemeine Beschreibung von SimWeld

SimWeld ist eine Desktopanwendung für Microsoft Windows zur Schweißprozesssimulation für das Lichtbogen-Schweißen mit 3 verschiedenen parametrisierbaren Werkstückkonstellationen. Dafür benötigt SimWeld eine Berechnungszeit von einer bis fünf Minuten ohne die Verwendung von Parallelberechnung, wobei die Ergebnisse um maximal 15% von den realen Vergleichsschweißungen abweichen [2]. Im Vergleich dazu benötigt andere Simulationssoftware eine Berechnungszeit in Größenordnungen von Stunden oder Tagen unter Verwendung von parallelen Algorithmen. Der Grund für die verhältnismäßig kurze Berechnungszeit von SimWeld liegt in der Entwicklung effizienter Algorithmen, die das Wissen um die dominanten physikalischen Prozesse und fundierte mathematische Kenntnisse miteinander kombinieren.

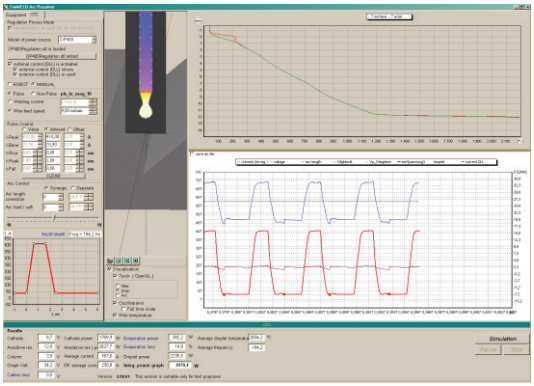


Abbildung 2: Simulation der Tropfenablösung mit SimWeld

Als Eingangsparameter für die Berechnung mit SimWeld dienen unter anderem Materialeigenschaften, geometrische Parameter (z. B. der Drahtvorschub und die Werkstückdicke), Prozessparameter (z.B. die Schweißgeschwindigkeit), insbesondere auch die Steuerungsalgorithmen der digitalen Stromquellen verschiedener Hersteller, die mit den Simulationsalgorithmen, z.B. den Algorithmen der Tropfenablösung des Schweißdrahtes (Abbildung 2), gekoppelt werden können [2]. Ergebnisse der Simulation sind unter anderem der räumliche und zeitliche Temperaturverlauf, die Schweißnahtgeometrie, die eingebrachte Energiemenge und der Anteil der vom Tropfen aufgenommenen Energie (Abbildung 3).

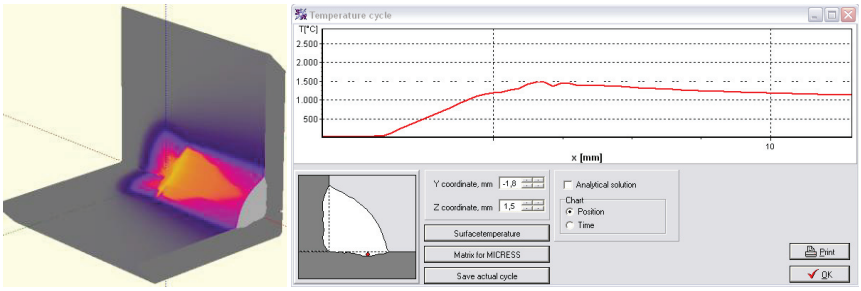


Abbildung 3: Ergebnisse der Simulation mit SimWeld. Schweißnahtgeometrie und Temperaturverteilung in 3D (links) und Temperaturzyklus (rechts).

Neben der kostengünstigen Untersuchung von Schweißungen anstelle von Realversuchen kann SimWeld auch dazu genutzt werden, die Schweißstruktursimulation zur Berechnung von Eigenspannungen und Verzug zu präzisieren. Dazu wird die mit

SimWeld berechnete äquivalente Wärmequelle nach Goldak [3], deren Werte sonst nur anhand der Schliffbilder von Realversuchen ermittelt werden kann, und die Schweißnahtgeometrie, die sonst üblicherweise geschätzt oder vernachlässigt wird, als Eingabeparameter herangezogen.

1.2 Der Evolutionsprozess von SimWeld

Der Grundstein für SimWeld wurde im Jahr 1993 mit MAGSIM 1, einem 16-bit MS DOS Programm mit grafischer Benutzeroberfläche, an der Tula State University (TSU) in Russland, basierend auf verschiedenen Schweißsimulationsmodellen, gelegt [4]. Das ISF war an der Entwicklung von MAGSIM (in allen Versionen) im Rahmen der Durchführung von Kalibrierungsversuchen beteiligt. Die Erstellung von MAGSIM 1 wurde von einem Team aus drei Personen mit der Programmiersprache Turbo Pascal und der Entwicklungsumgebung (IDE) Turbo Vision durchgeführt. Die grafische Darstellung der Benutzungsschnittstelle und der Ergebnisse erfolgte mit Graph Vision. Ab 1995 wurde MAGSIM 2 mit der Möglichkeit von Kehlnahtschweißungen und einer 3D-Ergebnisdarstellung in einem vierköpfigen Team entwickelt [5].

Seit 1997 wurde MAGSIM als Anwendung mit grafischer Benutzeroberfläche für MS Windows unter Verwendung der IDE Delphi weiterentwickelt und es existieren noch Teile des Quelltextes von 1997 in der heutigen Programmversion. Ab 1998 wurde MAGSIM 3 an die 32-bit Rechnerarchitektur angepasst [6], wobei die 3D-Darstellung auf Basis der Open-Source OpenGL-Bibliothek GLScene [7] umgesetzt wurde. Die Umstellung auf eine 32-bit-Software hat durch ihren erheblichen Einfluss auf die hardwarenahen Algorithmen von MAGSIM einen großen Aufwand verursacht. Im Jahr 2000 wurde TSIM in einem Team aus sechs Personen entwickelt. Die für den Tandemschweißprozess erstellten Modelle wurden mit bestehen Simulationsalgorithmen aus MAGSIM in einer neuen Programmstruktur zusammengeführt, in der die Benutzeroberfläche in Prä-, Postprozessor und Simulation getrennt worden ist.

Ab dem Jahr 2001 wurde die Entwicklung von SimWeld in einem Team aus zwei Entwicklern am Institut für Schweißtechnik und Fügetechnik begonnen. Dabei wurde die Programmstruktur an die von TSIM angelehnt und das Oberflächendeformationsmodell aus MAGSIM in angepasster Form übernommen, während alle weiteren Modelle, insbesondere das nichtstationäre, zeitabhängig gelöste, Lichtbogenmodell neu entwickelt wurden. 2003 wurde SimWeld 1 von einem dreiköpfigen Team um die Berücksichtigung der Regelungsalgorithmen realer Schweißstromquellen erweitert. Außerdem wurde die Architektur der Software grundlegend überarbeitet [8]. Die Programmiersprache Object Pascal wurde in Delphi umbenannt und hat umfangreiche Änderungen erfahren, aufgrund derer größere Anpassungen an dem Quelltext nötig waren. Ab 2006 wurde in einem vierköpfigen Team SimWeld 2 entwickelt, das eine Offline-Stromquellenkopplung, in Kooperation mit Schweißstromquellenherstellern (z.B. OTC Daihen, Japan), erlaubte. Des Weiteren wurde eine integrative Simulation des Schweißprozesses (SimWeld) und der Wärmeauswirkung in Bezug auf Eigenspannung und Verzug (SYSWELD) konstruiert [9]. Die in 2003 implementierte Architektur von SimWeld ist um einen nicht unerheblichen, bereits obsoleten Teil reduziert worden.

Im darauf folgenden Zeitraum bis 2011 wurde SimWeld in einem Industrieprojekt weiterentwickelt, mit dem Ziel exklusive Funktionalitäten für die Simulation von

Unterpulverschweißen im Mehrdrahtverfahren mit der Möglichkeit von Lage- und Gegenlageschweißungen zur Verfügung zu stellen. Hierbei wurde die Software wieder in einer einzelnen Anwendung, die Prä- und Postprozessor und Simulation umfasst, vereint. Während des Industrieprojektes wurde der Aufwand teilweise zu gering geschätzt und die Projektplanung nur grob durchgeführt, außerdem wurden während des Projektes zusätzliche Anforderungen an die Software gestellt. Diesen Herausforderungen wurde mit regelmäßigen Projektmeetings mit den Auftraggebern begegnet, in denen die zurückliegende Entwicklung dargestellt, Soll- und Ist-Zustände abgeglichen und die nachfolgende Weiterentwicklung abgestimmt worden sind. Das Projekt konnte so mit sechsmonatigem Verzug erfolgreich abgeschlossen werden.

In der näheren Zukunft ist die weitere Entwicklung von SimWeld in Version 3 geplant, die Strömungen und diskreten Massetransfer nicht nur kompensationsmäßig berücksichtigen soll. Dazu ist eine grundlegende Umstrukturierung der Software unter Weiterverwendung möglichst vieler bestehender Module notwendig. Außerdem ist eine Portierung auf die 64-bit Rechnerarchitektur mit Delphi XE2 geplant.

1.3 Programmstruktur

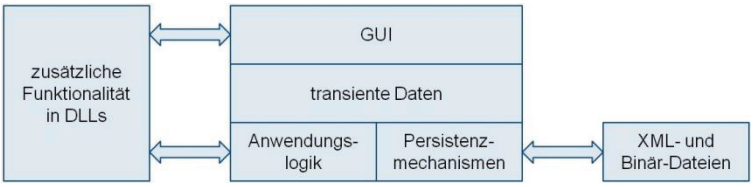


Abbildung 4: Programmstruktur von SimWeld und dessen Umgebung

SimWeld ist modular aufgebaut, wobei sowohl objektorientierte (GUI, transiente Daten, Persistenzmechanismen), als auch prozedurale Konzepte (Anwendungslogik) verwendet werden. Es gibt ein Hauptprogramm für alle Funktionen (Prä-, Postprozessor und Simulation). Exklusive Funktionalitäten werden per Compilerschalter aktiviert/deaktiviert, sodass diese nur in spezielle Versionen kompiliert werden.

SimWeld verfügt über eine Schichtenarchitektur die in Abbildung 4 dargestellt ist. In der obersten Schicht befindet sich die grafische Benutzeroberfläche, die intern auf die transienten Daten (Eingabeparameter, Ergebnisse usw.) und extern auf zusätzliche Funktionalitäten (z.B. Materialdatenbank), in Form dynamischer Einbindung eigener und fremder DLLs, zugreift. Die Anwendungslogik (Simulationsalgorithmen) greift ebenfalls intern auf die transienten Daten und extern auf zusätzliche Funktionalität (z.B. Stromquellensteuerungsalgorithmen) zu. Das Speichern und Laden der „Simulationsexperimente“ erfolgt auf Basis von XML- (Parameter) und komprimierter Binär-Dateien (Ergebnisse) über Persistenzmechanismen, die ebenfalls auf die transienten Daten zugreifen.

2 Technologie und Infrastruktur

Als Programmiersprache für SimWeld wird Delphi mit der IDE Embarcadero Delphi [10] in der jeweils aktuellen Version verwendet, für die Entwicklung von Schnittstellen zu anderer Software auch C++, Python und FORTRAN. Zur Versionierung wird Subversion [11] in Kombination mit Tortoise SVN [12] verwendet. Ein ausführ-

bares Setup wird mit NSIS [13] erstellt. Für das Fehlermanagement wird die Webanwendung „Bug Tracker“ [14] und für die Generierung der Online-Hilfe (chm-Datei) HelpMaker v7 [15] eingesetzt. Als Testumgebung dienen reale oder virtuelle Windows XP 32-bit Systeme unter Berücksichtigung verschiedener Benutzerrechte. Das Entwicklungsteam besteht aus zwei Hauptentwicklern und einem bis drei studentischen Hilfskräften, die im Bereich der Programmierung, Tests und Benutzerdokumentation eingesetzt werden. Die technische Dokumentation besteht hauptsächlich aus „sprechendem“ (also möglichst selbsterklärendem) Quelltext, Quelltextdokumentation (Unit-/Methoden-Header und Inline-Kommentare) und wissenschaftlichen Artikeln bzw. Projektberichten.

3 Entwicklungsprozess

Die wichtigsten Gründe für Änderungen an SimWeld sind die weitergehende Erforschung der physikalischen Prozesse beim Schweißen einerseits und Industriewünsche andererseits. Aber auch die Änderungen der Basistechnologie, also bezüglich der Rechnerarchitektur, des Betriebssystems, der IDE oder der Programmiersprache, verursachen oft umfangreiche Änderungen an der Software.

Ein Änderungsmanagement existiert nur rudimentär. Bei technologisch begründeten Änderungen erfolgt zunächst eine Testmigration mit anschließendem Test und der Dokumentation der Auffälligkeiten, sodass die anschließende Produktivmigration reibungslos verlaufen kann. Diese wird mit Tests und eventuell nötigen Korrekturen abgeschlossen. Für funktionale Änderungen der Software erfolgt zunächst eine grobe Projekt- und Strukturplanung für das gesamte Projekt.

Die Projektplanung basiert auf Expertenabschätzungen, wobei derzeit kein fest definiertes Vorgehensmodell existiert und nur eine geringe Planungstiefe schriftlich dokumentiert wird (Pflichtenheft). Für jede Aufgabe innerhalb eines Projektes erfolgt kurz vor ihrer Umsetzung eine Detailplanung (inkl. Entwurf), die meist gar nicht oder nur handschriftlich dokumentiert wird. Daraufhin erfolgen Umsetzung, Test und Fehlerkorrektur, wobei umfangreiche Änderungen zunächst mit einem Compilerschalter ein- bzw. ausschaltbar sind. Während des Projektverlaufs erfolgen regelmäßig (i.d.R. vierteljährliche) Rücksprachen mit den Auftraggebern und Software-Releases.

Bei der Entwicklung von SimWeld existiert eine klar definierte Arbeitsteilung in Infrastruktur, Schnittstellen (zu Benutzern, anderer Software, XML-Dateien und Datenbanken) und der Anwendungslogik. Es gibt häufige Rücksprachen im Entwicklerteam und Designentscheidungen, die die gesamte Software betreffen, werden nach gemeinsamer Diskussion von einer einzigen verantwortlichen Person getroffen.

4 Herausforderungen

4.1 Faktor Mensch

Das Entwicklerteam setzt sich zusammen aus zwei langfristig beschäftigten Hauptentwicklern, einem wissenschaftlichen (Modellentwicklung und Anwendungslogik) und einem nichtwissenschaftlichen Mitarbeiter (Infrastruktur und Schnittstellen), sowie ein bis drei kurzfristig beschäftigten studentischen Hilfskräften (Programmierung, Tests und Benutzerdokumentation). In der Vergangenheit waren auch Dokto-

randen an der Modellentwicklung und -implementierung beteiligt, was auch für die Zukunft wieder angestrebt ist. Bisher wurden mehrere positive Erfahrungen mit ehemaligen Doktoranden als Kontaktpersonen zur Industrie gemacht. Größere Herausforderungen in diesem Bereich sind bisher ausgeblieben und teambildende Maßnahmen gibt es derzeit keine.

4.2 Funktionale Herausforderungen

Eine funktionale Herausforderung ist die kontinuierliche Weiterentwicklung der Berechnungsmodelle und Benutzungsschnittstelle, der mit einer Schichtenarchitektur, einer iterativen Vorgehensweise und „sprechendem" Code begegnet wird. Eine weitere Herausforderung ist die Einbindung externer Algorithmen (DLLs) und die Anbindung an andere Simulationssoftware, bei der eine möglichst enge Kooperation angestrebt wird. Als dritte funktionale Herausforderung gilt die Abschaltung exklusiver Funktionalitäten unter Beibehaltung nichtexklusiver Funktionalität, wofür momentan Compilerschalter eingesetzt werden.

4.3 Technologische Herausforderungen

Neben den funktionalen gibt es auch signifikante technologische Herausforderungen. So verursachen Änderungen an der zugrundeliegenden Programmiersprache und an den benutzten Komponenten immer wieder Änderungsaufwand. Außerdem ergibt sich durch die Verwendung einer wenig verbreiteten Programmiersprache eine Bibliotheks-, Komponenten- und Werkzeugknappheit. Als Beispiel hierzu sei die Kernkomponente zur 3D-Visualisierung (GLScene) genannt, die der Entwicklung der Programmiersprache derzeit etwa 3 Jahre hinterherhinkt.

5 Zusammenfassung

Aus der über zehnjährigen Erfahrungen mit SimWeld am ISF haben sich insbesondere folgende Aspekte als problematisch herausgestellt: Das korrekte Abschätzen des Aufwandes in der Projektplanung und im Projektverlauf, ein fehlendes Vorgehensmodell und strukturelle Probleme aufgrund historischen Wachstums und unzureichender Dokumentation, sowie eine nur implizit existierende Softwarearchitektur. Derzeit wird diesen Problemen mit einem verbalen Wissenstransfer zwischen den Teammitgliedern, einer kontinuierlichen Ergänzung der Quelltextdokumentation sowie kleinen und großen Refactorings begegnet. Außerdem ist für die Zukunft geplant die Softwarearchitektur und Implementierungsdetails zu dokumentieren sowie das Aufwandsschätzungsverfahren zu verbessern.

Bei dem Evolutionsprozess hat sich als positiv herausgestellt, dass die Verwendung selbsterklärenden Quelltextes und eine Schichtenarchitektur die Erweiterbarkeit und Wartbarkeit der Software wirksam unterstützen, das mit einem kleinen Team gut qualifizierter Spezialisten und einem ausführlichen Projektplan Softwareentwicklung auch für Industrieprojekte möglich ist und das Delphi das Rapid Application Development (z.B. bezüglich der GUI-Entwicklung) sehr gut unterstützt.

6 References

1. **Radaj, D.** *Schweißprozessimulation - Grundlagen und Anwendungen.* Düsseldorf : DVS-Verlag, 1999.

2. **Reisgen, U., et al.** Efficient Simulation of GMA Welding Process by Consideration of Real Power Source Regulation Algorithms. [Buchverf.] H. Cerjak und N. Enzinger. *Mathematical Modelling of Weld Phenomena 9.* Graz, Austria : Verlag der Technischen Universität Graz, 2009.

3. **Goldak, J., Chakravarti, A. und Bibbi, M.** A New Finite Element Model for Welding Heat Sources. *Metallurgical Transactions B.* 1984, Bd. 15B, S. 299-305.

4. **Sudnik, V. A., et al.** MAGSIM program software for analysis, optimisation and diagnostics of the process of consumable electrode welding thin-sheet joints in an active gas. *Welding International.* 1995, Bd. 11, 9, S. 891-896.

5. **Dilthey, U., et al.** *MAGSIM for Windows: GMA pulse welding of fillet welds.* 1998.

6. **Dilthey, U., et al.** *MAGSIM and SPOTSIM - Simulation of GMA- and Spot Welding for Training and Industrial Application.* 2001.

7. **Grange, Eric.** GLScene OpenGL Solution for Delphi. [Online] 22. 11 2011. [Zitat vom: 12. 12 2011.] http://glscene.sourceforge.net/wikka/HomePage.

8. **Pavlyk, V. und Mokrov, O.** *Simulation des MIG-Schweißverfahrens von Aluminium mit SimWeld.* 2004.

9. **Dilthey, U., Pavlyk, V. und Morkrov, O.** *Integrative GMA-Welding Simulation.* 2006.

10. **Embarcadero Technologies, Inc.** Delphi XE2 | RAD Application Development Software. [Online] 2011. [Zitat vom: 13. 12 2011.] http://embarcadero.com/products/delphi.

11. **The Apache Software Foundation.** Apache Subversion. [Online] 2011. http://subversion.apache.org/.

12. **CollabNet, Inc.** *tortoisesvn.tigris.org.* [Online] 2011. [Zitat vom: 13. 12 2011.] http://tortoisesvn.tigris.org/.

13. **Verburg, Joost.** Nullsoft Scriptable Install System. [Online] 12. 02 2011. [Zitat vom: 13. 12 2011.] http://nsis.sourceforge.net/Main_Page.

14. **Wang, Chun-Pin.** *twbsd.org.* [Online] 2011. [Zitat vom: 13. 12 2011.] http://www.twbsd.org/enu/bug_tracker/index.php.

15. **Vizacc Pte Ltd.** *HelpMaker Help Authoring Tool.* [Online] 2011. [Zitat vom: 01. 09 2011.] www.vizacc.com.

Die Entwicklung des Virtual Reality Toolkit ViSTA

Dominik Rausch, Bernd Hentschel und Torsten Kuhlen

Virtual Reality Group
Lehrstuhl für Informatik 12
RWTH Aachen, Germany
Email: {rausch, hentschel, kuhlen}@vr.rwth-aachen.de

Zusammenfassung Seit über zehn Jahren wird an der Virtual Reality Group der RWTH Aachen das Virtual Reality Toolkit ViSTA entwickelt. Dieses bietet Basisfunktionalität und Erweiterungen, die eine schnelle Entwicklung von Virtual Reality-Applikationen ermöglichen. Der vorliegende Bericht beschreibt die Entwicklung von ViSTA und geht auf einige Pläne für zukünftige Entwicklungen ein. Hierbei wird insbesondere der Einfluss des wissenschaftlichen Umfelds auf die Entwicklung diskutiert.

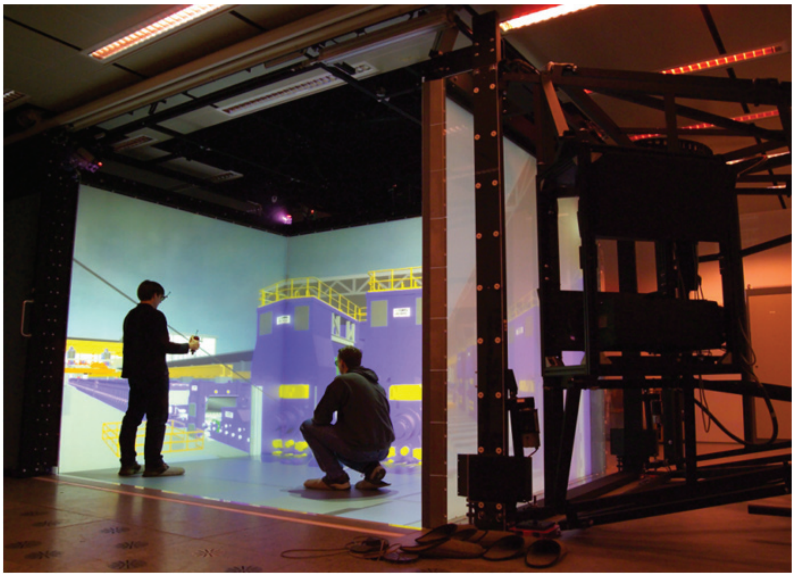


Abbildung 1. Voll-immersives CAVE-Projektionssystem am Rechen- und Kommunikationszentrum der RWTH Aachen

1 Einleitung

Ziel der Virtuellen Realität ist es, eine Virtuelle Umgebung zu erzeugen, in der ein Benutzer natürlich interagieren kann und sich in die Szene hineinversetzt fühlt. Hierfür wird komplexe Hardware benötigt, z.B. stereoskopische Großprojektionssysteme wie eine CAVE (siehe Abb. 1), Kameras zur Echtzeit-Bewegungsverfolgung des Benutzers und weitere Ein- und Ausgabegeräte. Die Virtual Reality Group (VRG) der RWTH Aachen arbeitet in diesem Gebiet und nutzt die Technologien für verschiedenste Anwendungsgebiete, wie z.B. das medizinische Training

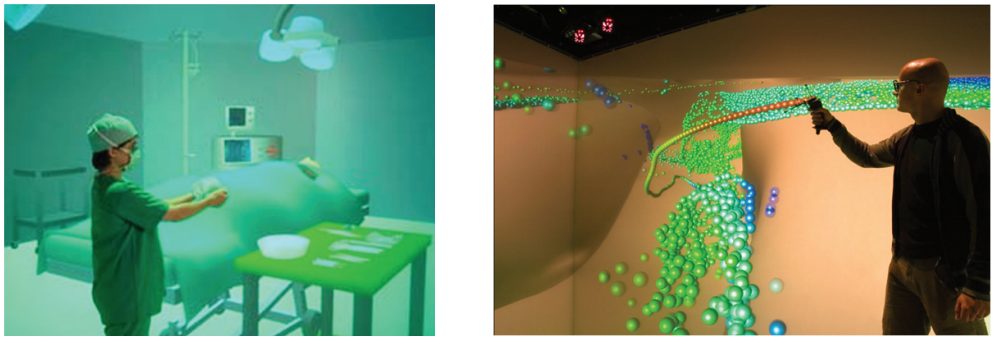


Abbildung 2. Anwendungsbeispiele für VR: Medizinische Trainingssimulatoren (links) und immersive Datenvisualisierung (rechts).

oder die interaktive Datenvisualisierung (siehe Abb. 2), aber auch zur Forschung an Basistechnologien wie Navigations- und Interaktionsmethoden, Akustik oder Haptik. Aufgrund der komplexen Fragestellungen und der hochgradig heterogenen Hardware ist die Entwicklung von VR-Applikationen sehr aufwändig. Diese müssen z.B. auf sehr unterschiedlichen Endsystemen laufen, angefangen bei Desktop-PCs über kleinere 3D-Projektionssysteme bis zu großen, voll-immersiven Systemen wie der CAVE. Hierbei gilt es nicht nur die verschiedene Displaysysteme sondern auch vielfältige Eingabegeräte wie z.B. 3D-Tracker oder haptische Ein-/Ausgabegeräte anzusprechen. Um den Entwicklungsaufwand für neue Virtual Reality (VR)-Prototypen zu reduzieren, müssen daher entsprechende Programmierschnittstellen geschaffen werden, die von den spezifischen Eigenschaften unterschiedlicher Displays und Eingabegeräte abstrahieren. Weiterhin bedarf es grundlegender Interaktionsmethoden, sowie softwaretechnischer Grundbausteine für eine schnelle Entwicklung neuer VR-Prototypen. Hierbei muss eine hohe Performance sichergestellt werden, da VR-Anwendungen konstant in Echtzeit (≥ 30 fps) und mit niedriger Interaktionslatenz (< 100 ms) laufen müssen.

Um eine gemeinsame Basis für die Entwicklungen im Bereich der VR zu haben, wird an der Virtual Reality Group seit über zehn Jahren das Virtual Reality Toolkit ViSTA (Virtual Reality for Scientific and Technical Applications) entwickelt [AK08]. ViSTA ist seit März 2009 als OpenSource-Software unter der LGPL verfügbar.[1] Desweiteren besteht ebenfalls seit 2009 eine Kooperation mit der Abteilung für Simulations- und Softwaretechnik des Deutschen Zentrums für Luft- und Raumfahrttechnik (DLR) in Braunschweig zur gemeinsamen Weiterentwicklung von ViSTA. Diese bietet neue Möglichkeiten für die Weiterentwicklung und -verbreitung von ViSTA. Es ergeben sich aber durch das größere, verteilte Entwicklungsteam auch neue Anforderungen an den gemeinsamen Softwareentwicklungsprozess.

2 Das Projekt ViSTA

Im Umfeld von ViSTA sind über die Jahre diverse Entwicklungen durchgeführt worden. Da generell alle Forschungsentwicklungen der Gruppe zu nutzbaren Modulen weiterentwickelt werden, umfasst ViSTA neben den eigentlichen Kernbi-

[1] http://sourceforge.net/projects/vistavrtoolkit

bliotheken auch Bibliotheken zur interaktiven Datenvisualisierung und weitere Zusatzmodule.

VistaCoreLibs Die VistaCoreLibs bilden die Basis von ViSTA und dienen als Grundlage für alle weiteren Methoden. Sie enthalten gemeinsame Schnittstellendefinitionen, Utility-Klassen und Systemabstraktionen, z.B. für Netzwerkkommunikation und Multi-Threading. Weiterhin bieten die VistaCoreLibs Treiberschnittstellen, um diverse Ein- und Ausgabegeräte ansprechen zu können, sowie ein Datenflussmodell zur Abstraktion und Verarbeitung der Treiberdaten. Basierend auf diesen grundlegenden Bausteinen stellen die VistaCoreLibs Methoden zur Verwaltung komplexer Displaysysteme und eine Szenegraphen-Abstraktion zur graphischen Darstellung bereit. Um VR-Anwendungen auch auf großen, hochauflösenden Multiprojektorsystemen mit der erforderlichen Bildwiederholrate von 30 Hz oder mehr betreiben zu können, ermöglicht ViSTA die synchronisierte Ausführung von Anwendungen auf einem Cluster von Systemen. Aktuell umfassen die VistaCoreLibs zehn Kernbibliotheken sowie 25 Gerätetreiber-Plugins. Diese umfassen etwa 1.000 Klassen, die aus etwa ca. 250.000 Zeilen Quellcode bzw. 80.000 Anweisungen bestehen.

VistaFlowLib Eines der wichtigsten Forschungsgebiete der Virtual Reality Group liegt in der Entwicklung von Methoden zur interaktiven Datenvisualisierung. Diese wurden im Laufe der Jahre zu einer Sammlung von Bibliotheken mit dem Namen ViSTA FlowLib (VFL) zusammengefasst [SGvR+03]. Während das Hauptaugenmerk bei der Entwicklung von VFL auf der Visualisierung zeitvarianter Strömungsphänomene lag, erstreckt sich das Anwendungsgebiet mittlerweile auf verschiedenste Daten aus dem wissenschaftlichen Umfeld. Die VFL gliedert sich in die folgenden drei Teile:

- parallele Visualisierungsalgorithmen, mit deren Hilfe aus Rohdaten darstellbare Visualisierungsdaten gewonnen werden,
- spezielle Rendering-Verfahren, die diese Daten interaktiv darstellen und
- angepasste Interaktionstechniken, die es dem Benutzer ermöglichen, direkt mit seinen Daten in einer virtuellen Umgebung zu interagieren.

Die Erweiterung *Viracocha* erlaubt es zudem, aufwändige Berechnungen neuer Visualisierungen zur Laufzeit auf einen parallelen Hochleistungsrechner auszulagern, um so interaktive Antwortzeiten sicherzustellen [GHW+04]. Der Umfang der VistaFlowLib ist vergleichbar mit dem der VistaCoreLibs; sie enthält etwa 500 Klassen, 180.000 Zeilen und 65.000 Anweisungen.

Weitere Libraries Neben den Hauptbibliotheken existieren eine Reihe von Zusatzbibliotheken, die spezialisierte Funktionalität anbieten. Beispiele hierfür sind Kollisionserkennung und physikalisch-basierte Simulation (unter Verwendung von Bullet[2] und SOFA[3]), Methoden für die haptische Simulation, zusätzliche Interaktionsmethoden, medizinische Simulation, Animationen, Virtual Humans, Audioausgabe, etc.

[2] http://bulletphysics.org
[3] http://www.sofa-framework.org

3 Technologie und Infrastruktur

Die Entwicklung von ViSTA wird von einem verteilten Team in Kooperation zwischen der Virtual Reality Group in Aachen und dem DLR in Braunschweig vorangetrieben. Derzeit arbeiten auf Seite der VRG neun wissenschaftliche Mitarbeiter (WM), fünf nicht-wissenschaftliche Mitarbeiter (NWM), ca. zehn HiWis und fünf weitere Studenten (in Diplom-, Masterarbeiten, etc.) an und mit ViSTA. Auf Seiten des DLR beteiligen sich weitere sechs WM und ein NWM.

Das Team setzt sich primär aus Doktoranden zusammen, die ViSTA im Rahmen von Forschungsvorhaben nutzen und dabei prototypische Funktionalität entwickeln, aber auch bestehenden Code warten und den Funktionsumfang von ViSTA stetig erweitern. Daneben gibt es noch eine Reihe NWM, die wissenschaftliche Datenvisualisierung als Service in der Hochschule und für externe Partner anbieten.

Da der Großteil der ViSTA-Entwickler aus Doktoranden besteht, die in der Regel maximal fünf bis sechs Jahre in der Gruppe verbleiben, besteht eine hohe Personalfluktuation; derzeit arbeiten die dritte und vierte „Generation“ von Doktoranden an ViSTA. Dieser stete Wechsel stellt das Team vor besondere Herausforderungen im Bereich der Dokumentation existierender Methoden und der Weitergabe des Know-Hows, das den einzelnen Entwicklungen zu Grunde liegt.

ViSTA wird vollständig in C++ entwickelt. Wesentliche Gründe dafür liegen in der Anforderung nach hoher System-Performance und den hardwarenahen Schnittstellen. Derzeit wird ViSTA standardmäßig für Microsoft Windows und Linux bereitgestellt. Entwickelt wird vornehmlich auf Desktop-Systemen mit VisualStudio bzw. gcc, wobei CMake[4] als Build-System verwendet wird.

Bei der Verwendung externer Bibliotheken wird versucht, die Zahl der externen Abhängigkeiten niedrig zu halten, insbesondere für die Kernkomponenten. So benötigen die VistaCoreLibs neben nativen C++ und Systembibliotheken nur OpenGL, glut, und ein Szenegraphen-Toolkit – derzeit OpenSG.[5]

Code-Dokumentation erfolgt mittels *doxygen*[6], während abstrakte System-Konzepte und das dynamische Zusammenspiel verschiedener Komponenten in einem Wiki erläutert werden. Hier bestehen jedoch Defizite, da die Dokumentation unvollständig ist und teilweise veraltete Informationen enthält. Für die generelle Einarbeitung in die einzelnen Bibliotheken stehen zahlreiche Demos zur Verfügung, die die Verwendung einzelner Komponenten demonstrieren. Weiterreichende Fragen können über entsprechende Mailinglisten gestellt werden, bzw. werden direkt im Dialog zwischen neu hinzugekommenen und erfahrenen Entwicklern geklärt.

Als Versionskontrollsystem wird Subversion (svn) verwendet. Totalview (Linux) und Intel Parallel Studio (VisualStudio) werden als Laufzeit-Debugger verwendet, Letzteres zusätzlich zum Profiling und Thread-Checking. Bei der Koordination der verteilten Entwicklung wird derzeit verstärkt Mantis[7] verwendet. Weiterhin wird mittels Jenkins/Hudson ein Nightly Build durchgeführt.

[4] http://www.cmake.org
[5] http://www.opensg.org
[6] http://ww.doxygen.org
[7] http://www.mantisbt.org/

4 Entwicklungsprozess

Die Arbeiten im Umfeld von ViSTA lassen sich grob in die Entwicklung neuer Funktionalität und die Wartung vorhandener Komponenten gliedern.

Für jede bestehenden Komponenten existiert ein Maintainer, der die grundsätzliche Funktionstüchtigkeit sicherstellt und die Weiterentwicklung koordiniert. Bei Uneinigkeit über Entwicklungen hat er ein Vetorecht. Der Maintainer wechselt typischerweise mit jeder „Doktorandengeneration“. Es wird versucht, diesen Wechsel nach Möglichkeit langfristig vorzubereiten, so dass eine möglichst lange Zeitspanne besteht, in der alter und neuer Maintainer zusammenarbeiten.

Neue Funktionalität geht oft direkt aus Forschungsvorhaben der einzelnen Mitarbeiter hervor. Arbeitsergebnisse werden nach Abschluss der prototypischen Entwicklung in eine vorhandene Bibliothek eingepflegt; wahlweise wird auf Basis der Ergebnisse ein neues Modul erstellt. Wenn gezielte Entwicklungen in einem bestimmten Bereich durchzuführen sind, wird für diese eine „Task Force“ gebildet. Diese Arbeitsgruppen bestehen aus etwa zwei bis vier Entwicklern und arbeiten gezielt auf den Entwurf und die Implementierung des gewünschten Features hin. In der Vergangenheit kam es des Öfteren dazu, dass Arbeiten in Task Forces sich über zu lange Zeit hinzogen und wenig koordiniert abliefen. Daher wird derzeit versucht, die Arbeiten in kurzen Iterationen zu organisieren, an deren Ende konkrete Ergebnisse stehen sollen.

Während kleinere Entwicklungen direkt im svn-Trunk des jeweiligen Moduls durchgeführt werden, erfolgt die Entwicklung größerer Veränderungen in eigenen Branches. Weiterhin werden, zumindest für die größeren Module, unregelmäßig Releases erstellt. In der Gruppe besitzen alle Mitarbeiter und Studenten svn-commit-Rechte, so dass jeder Einzelne kleinere Änderungen – insbesondere Bug Fixes – sofort durchführen kann und soll. Generell wird von jedem Entwickler verlangt, alle Eigenentwicklungen einzuchecken, so dass der Code zumindest als prototypische Implementierung für alle verfügbar ist. Dies gilt insbesondere für die Forschungsergbnisse der Doktoranden. Weiterhin existieren Coding-Styleguides, die ein einheitliches Erscheinungsbild der Quelldateien sicherstellen sollen.

Zur Koordination der Entwicklung wird 14-tägig ein Software-Meeting in der Gruppe durchgeführt. Bei diesem werden geplante zukünftige Entwicklungen diskutiert oder abgeschlossene Arbeiten präsentiert. In diesem Rahmen werden auch neue Task Forces zusammengestellt und über den Verlauf von existierenden Task Forces berichtet. Einmal monatlich finden die SW-Meetings als Videokonferenz mit allen Teilnehmers der Virtual Reality Group und des DLR statt.

5 Spezielle Herausforderungen

Die Entwicklung von Software im wissenschaftlichen Umfeld der Hochschule bringt eine Reihe zusätzlicher Herausforderungen mit sich. Da ein Großteil der Entwickler eine Promotion anstrebt, liegt deren primärer Fokus auf der Forschung. Dies birgt einen Konflikt mit dem Ziel der nachhaltigen Softwareentwicklung, da der Aufwand, einen Forschungsprototypen in ein funktionierendes, allgemein nutzbares Modul weiterzuentwickeln, sehr hoch ist.

Ein weiteres Problem besteht darin, dass die meisten Mitarbeiter auf befristeten Stellen arbeiten und somit eine hohe Mitarbeiterfluktuation herrscht. Obwohl

vor dem Abgang eines Mitarbeiters in den meisten Fällen eine Übergabe an einen designierten Nachfolger durchgeführt wird, tritt dennoch ein Verlust an Wissen und Erfahrung auf. So ist es in der Vergangenheit immer wieder dazu gekommen, dass einzelne Bibliotheken oder Teile davon mit der Zeit nicht weiter gepflegt werden und dadurch unbrauchbar werden. Um dieses Problem zu reduzieren wurde einerseits ein langjähriges Mitglied der Gruppe nach erfolgter Promotion auf eine Dauerstelle übernommen. Auf diese Weise wird versucht, langfristig ein Mindestmaß an konzeptioneller Integrität zu wahren. Andererseits wäre die Unterstützung der Softwareentwicklung durch langfristig beschäftigtes nicht wissenschaftliches Personal hilfreich. Allerdings hat es sich als schwierig erwiesen, im wissenschaftlichen Umfeld Gelder für eine entsprechende Stelle zu akquirieren. Somit hängt der Erfolg zumeist von der Zahl und dem Engagement der einzelnen befristet angestellten Mitarbeiter ab.

Aufgrund der bestehenden Kooperation mit dem DLR besteht neuerdings zusätzlich eine räumliche Trennung des Teams, die zusätzliche Herausforderungen bedeutet. Zwar werden regelmäßige Videokonferenzen abgehalten, dennoch ist es schwierig, Informationen über aktuelle Entwicklungen zuverlässig zu kommunizieren.

Da ViSTA in seiner – im Vergleich zu ähnlichen Bibliotheken – langen Lebensdauer stark gewachsen ist und mittlerweile eine hohe Komplexität erreicht hat, ist die Einarbeitung neuer Mitarbeiter relativ langwierig. Im Schnitt dauert es etwa drei Monate, bis ein Neuzugang sich so weit eingearbeitet hat, dass er oder sie selbständig produktiv zur Weiterentwicklung beitragen kann. Dies ist insbesondere im Hinblick auf Studierende ein Problem, vor allem beim Schreiben einer Abschlussarbeit, bei der die Einarbeitungszeit aufgrund der zeitlich begrenzten Bearbeitungszeit eine zusätzliche Hürde darstellt.

6 Aktuelle Entwicklungen

In den letzten Jahren ist die Zahl der Entwickler, die an und mit ViSTA arbeiten, stark gestiegen. Dadurch ist es schwerer geworden ist, die gemeinsamen Arbeiten zu koordinieren und jeden über aktuell laufende Entwicklungen zu informieren. Dies spiegelt sich auch in geändertem Vorgehen bei der gemeinsamen Softwareentwicklung wider. Beispielsweise wurden die Software-Meetings umstrukturiert. Früher wurden hier technische Entwicklungen mit dem gesamten Entwicklerteam diskutiert. Mit wachsender Teamgröße stellte sich allerdings heraus, dass dieses Vorgehen nicht mehr praktikabel ist. Daher wurden technische Detaildiskussionen in die Taskforces verlagert. Die Ergebnisse dieser Diskussionen werden dann im Softwaremeeting vorgestellt und deren Umsetzung dort beschlossen.

Darüber hinaus wird inzwischen über die schrittweise Einführung neuer Entwicklungspraktiken, z.B. dem Vorgehen in kurzen Iterationen, Continuous Integration mit mindestens täglichem Check-In und formale Softwaretests nachgedacht. Diese Umstellungen erfordern teilweise ein deutliches Umdenken und stehen häufig, zumindest subjektiv empfunden, dem schnellen Vorankommen in der Forschung im Weg.

7 Fazit

Mit ViSTA wurde in der Virtual Reality Group ein Toolkit für das Erstellen von Prototypen, Anwendungen und Bibliotheken entwickelt, das auf eine über zehn Jahre währende Entwicklungsgeschichte zurückblickt. Die dadurch erreichte Nachhaltigkeit der Entwicklungen hilft, neue Funktionalität, sowie Forschungsprototypen schneller zu entwickeln. Insgesamt hat sich das gemeinsame Vorgehen bewährt, auch wenn neuerdings durch die wachsende Teamgröße und das verteilte Entwicklerteam neue Herausforderungen hinzugekommen sind. Diese zu lösen, ist eine wesentliche Herausforderung der aktuellen Weiterentwicklung der Kommunikations- und Arbeitsabläufe.

Literatur

[AK08] Ingo Assenmacher and Torsten Kuhlen. The ViSTA Virtual Reality Toolkit. In *Proceedings of the IEEE VR 2008 Workshop Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, pages 23–28. Shaker Verlag, 2008.

[GHW+04] Andreas Gerndt, Bernd Hentschel, Marc Wolter, Torsten Kuhlen, and Christian Bischof. Viracocha: An Efficient Parallelization Framework for Large-Scale CFD Post-Processing in Virtual Environments. In *Proceedings of the 2004 ACM/IEEE Conference on Supercomputing*, 2004. Published on CD-ROM.

[SGvR+03] Marc Schirski, Andreas Gerndt, Thomas van Reimersdahl, Torsten Kuhlen, Philipp Adomeit, Oliver Lang, Stefan Pischinger, and Christian Bischof. ViSTA FlowLib - A Framework for Interactive Visualization and Exploration of Unsteady Flows in Virtual Environments. In *Proceedings of the 7th International Immersive Projection Technology Workshop (IPT) and 9th Eurographics Workshop on Virtual Environment (EGVE)*, pages 77–85, 2003.

Entwicklung für das Softwarewerkzeug »NCProfiler«

Mario Pothen, Meysam Minoufekr, Lothar Glasmacher

Aachen
mario.pothen@ipt.fraunhofer.de
meysam.minoufekr@ipt.fraunhofer.de
lothar.glasmacher@ipt.fraunhofer.de

Abstract. Die Software »NCProfiler« zur Analyse, Visualisierung und Optimierung von NC-Daten wird kontinuierlich seit ca.15 Jahren am Fraunhofer IPT entwickelt und um neue Funktionalitäten erweitert. Das Softwarewerkzeug bietet effiziente Methoden zur Optimierung von NC-Daten für verschiedene Fertigungsprozesse. Eine besondere Herausforderung ist das dynamische und interdisziplinäre Forschungsumfeld, in dem diese Software entwickelt wird, welche sowohl Expertise im Software-Engineering als auch prozesstechnologische Kenntnisse erfordert.

Keywords: Automatisierung; Computer Aided Engineering; Dynamik; Fertigungssimulation; Fertigungstechnik; Fräsen; Freiformfläche; Machine Tools; Machining; Maschinenbau; Modellierung; Optimierung; Production Process; Produktion; Produktionstechnik; Prozessanalyse; Prozessoptimierung; Qualitätssicherung; Simulation; Softwareentwicklung; Steuerungstechnik; Ultrapräzisionszerspanung; Werkzeubau; Werkzeug- und Formenbau; Werkzeugbau; Werkzeugmaschinen; Zerspanen;

1 Einleitung

Softwaresysteme sind eine wichtige Voraussetzung für die wissenschaftliche Forschung. Sie werden eingesetzt, damit sie die Relevanz von Forschungsergebnissen und ihre Anwendbarkeit in der Praxis demonstrieren. Außderdem können teure Fallstudien im wissenschaftlichen sowie im industriellen Umfeld durch Simulationen reduziert werden. Entwicklungsprozesse bilden einen wichtigen Erfolgsfaktor bei der Realisierung und der Qualitätssicherung von Softwaresystemen. Hier spielen sowohl die fachliche Qualifikation der beteiligten Mitarbeiter als auch die Vorgehensweise, die Methoden und Werkzeuge, die eingesetzt werden, eine wichtige Rolle [1].

Im Rahmen von Forschungsprojekten und durch diverse bilaterale Projekte wurde am Fraunhofer IPT die Software »NCProfiler« zur Analyse und Optimierung von Fertigungsprozessen entwickelt. Über den Entwicklungszeitraum wurden die Anforderungen an das System kontinuierlich angepasst und erweitert, was auf Grund des wachsenden Funktionsumfangs oftmals Strukturanpassungen erforderte. Während des Entwicklungsprozesses der Software zeigte sich, dass die konsequente Umsetzung einer definierten Vorgehensweise unter allen beteiligten Programmierern – wie bei wissenschaftlichen Softwareprojekten üblich – schwer erreichbar ist. So berücksichtigen die klassischen Entwicklungsprozessmodelle wie das Wasserfallmodell, RUP und XP [2] die spezifischen Abhängigkeiten wissenschaftlicher Softwareentwicklung kaum oder nicht im ausreichendem Maße. Im Folgenden werden nach einer Beschreibung des Softwarewerkzeugs »NCProfiler« wichtige Problemstellungen, die während

der Entwicklung der Software in den Fokus rückten, sowie die eingesetzten Lösungsansätze und Techniken erläutert.

2 »NCProfiler« – NC-Datenanalyse und -optimierung

Der »NCProfiler« erlaubt die Analyse, Bewertung und Optimierung eines technologischen Bearbeitungsprozesses wie Fräsen oder Schleifen bereits während der Planung am Rechner. Der »NCProfiler« bildet dabei die Steuerungseigenschaften und die Kinematik der eingesetzten Werkzeugmaschine ab, auf der die CNC-Bearbeitung stattfinden soll [3]. Die Modellierung des Steuerungsverhaltens sowie des kinematischen Aufbaus der Werkzeugmaschine ermöglicht eine Evaluierung der Bahnführung und Erzeugung abgestimmter Vorschubprofile [4]. Weiterhin erfolgt eine Abschätzung der Bearbeitungsdauer unter Betrachtung der Randbedingungen der jeweiligen Werkzeugmaschine zwecks weiterer Optimierung, wodurch Fehlerquellen und Verbesserungspotenziale in den NC-Programmen im Vorfeld erkannt werden [5].

Ein weiterer wichtiger Aspekt bei der Bearbeitung von Bauteilen, die als Freiformflächen definiert sind, ist die Erzeugung einer harmonischen Bahnführung [6], wodurch eine höhere Oberflächengüte bei der Fräsbearbeitung eines Werkstücks erreicht wird. Im »NCProfiler« sind Funktionen zur Bahnharmonisierung unter Einhaltung von Vorgabetoleranzen realisiert. Dies bedeutet eine Verbesserung der Ausgangssituation in der Nachbearbeitung und führt folglich zu einer Reduktion kostenintensiver manueller Nachbearbeitung [7].

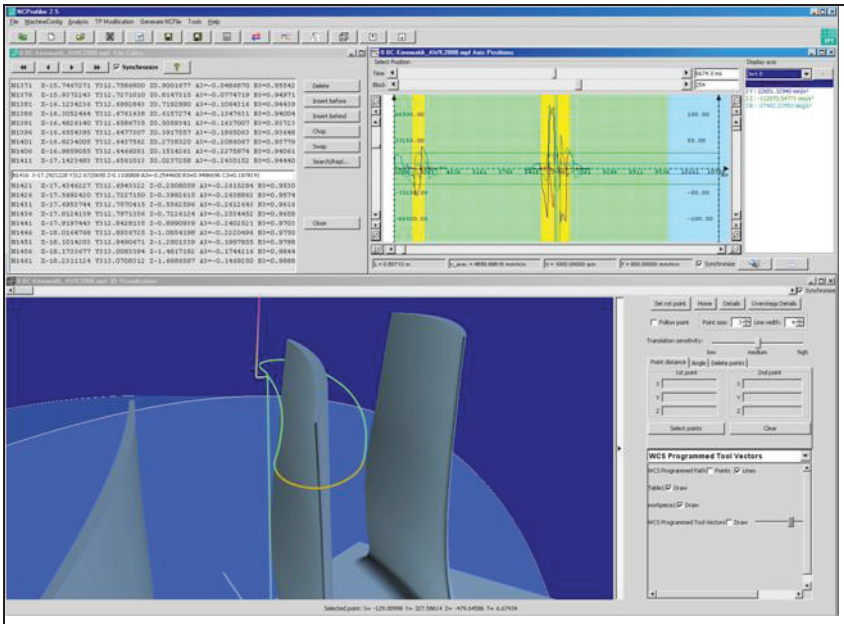


Abbildung 1: Benutzeroberfläche des Softwarewerkzeugs »NCProfiler«

3 Entwicklungsprozess am Fraunhofer IPT

Der »NCProfiler« zeichnet sich als wissenschaftliche Software [1] dadurch aus, dass er seit ca. 15 Jahren im Rahmen von mehreren Forschungsprojekten entwickelt wurde, in denen er zur Lösung der jeweils aktuellen Aufgaben eingesetzt wurde. Dabei sind Forschungschwerpunkte häufigen Wechseln unterworfen, die je nach erhaltenen Ergebnissen, verfügbaren Forschungsmöglichkeiten oder vorgegebenem Finanzierungsrahmen variieren können, insbesondere da es hierbei um die Entwicklung neuer, experimenteller Ansätze innerhalb einer interdisziplinären Forschungsdomäne geht. In der Praxis stellt sich heraus, dass neue Ansätze in prototypischen Implementierungen getestet und entweder als fester Bestandteil in die Software übernommen oder wieder verworfen werden. Der Lebenszyklus des »NCProfiler« muss daher von Beginn an anders als ein klassischer Zyklus ausgelegt werden. Die Modelle, die von den Forschern permanent überarbeitet und verfeinert werden, ziehen Anpassungen im Softwaremodell nach sich. Dies wirkt sich ebenfalls auf die Anforderungsanalyse und -spezifikation aus, da die Anforderungen nie vollständig vorliegen, bevor mit dem Design und der Implementierung begonnen werden muss. Somit sind die Anforderungen an das Produkt einem ständigen Änderungs- und Evaluationsprozess ausgesetzt. Praktische Versuche können ebenfalls Gesetzmäßigkeiten in vorangegangenen Forschungsarbeiten bestätigen. Auf der anderen Seite können Sie zu der Erkenntnis beitragen, dass bereits realisierte Lösungsansätze im »NCProfiler« sich im Nachhinein nicht als sinnvoll und praktikabel erweisen. Somit kann die Entwicklung der »NCProfiler«-Software nicht als abgeschlossen betrachtet werden, da die mit der Software erworbenen Erkenntnisse neue Fragen aufwerfen, die es zu beantworten gilt. Der Entwicklungsprozess des »NCProfiler« ist als wissenschaftliche Softwareentwicklung charakterisiert, in der sich die beteiligten Entwickler regelmäßig mit veränderten Anforderungen und Rollbacks auseinandersetzen müssen [1].

Eine weitere Besonderheit ergibt sich aus den personellen Rahmenbedingungen im Forschungsumfeld, bei der die Softwareentwicklung durch hohe Mitarbeiterfluktuation betroffen sein kann. Dieses zeigte sich auch in der Entstehungsgeschichte des »NCProfiler« am Fraunhofer IPT. Ein beachtlicher Teil der Arbeit wurde durch studentische Hilfskräfte oder im Zuge von Studienleistungen erbracht, beispielsweise als Diplom-, Master- oder Studienarbeiten. So stammen die Mitarbeiter häufig aus dem universitären Umfeld, sind auf Teilzeitbasis angestellt oder stehen nicht permanent zur Verfügung. Der Softwareentwicklungsprozess darf außerdem keine dauernde Anwesenheit der Beteiligten voraussetzten, da Studenten sich nur für einen sehr begrenzten Zeitraum an der Entwicklung der Software beteiligen und selten ein fester Bestandteil des Entwicklerteams sind. Darüber hinaus müssen Abschlussarbeiten als eigenständige Projekte definiert werden, welche im Anschluss in den Gesamtkontext des Produkts integriert werden können. Dabei haben die studentischen Mitarbeiter oft nur eine unzureichende Übersicht über das gesamte »NCProfiler«-Projekt als Ganzes. Hinzu kommt, dass aufgrund der fehlenden Erfahrung selten fundiertes Know-how im Software-Engineering vorausgesetzt werden kann. Bei einem Wechsel der Mitarbeiter muss der Transfer der Arbeiten ohne Informationsverluste erfolgen können. Daher nimmt die Dokumentation eine besondere Rolle ein, um das Wissen sichern zu können und um die Einarbeitung in den Prozess zu unterstützen.

3.1 Kommunikation und Dokumentation

Die Generierung von Wissen und neuen Erkenntnissen ist eines der wichtigsten Ziele im Forschungsumfeld. Dies gilt auch für die eingesetzten Softwaresysteme in der Forschung. Daher ist es essentiell, den erreichten Entwicklungsfortschritt und das dabei gewonnene Know-how zu dokumentieren. Der Vorgang der Wissenskonservierung ist insbesondere aufgrund der im vorherigen Kapitel beschriebenen Entwicklerstruktur notwendig. Dies geschieht zum größten Teil mit einer eigenständigen Anwenderdokumentation, welche die Funktionalität dem Benutzer zugänglich macht, da sie eng an den aktuellen Entwicklungsstatus gekoppelt ist. Wie bei jeder Softwareentwicklung setzt die Wartung und Pflege der Software eine weitere Dokumentation direkt im Quellcode voraus, die spezielle Variablen, Klassen und Methoden beschreibt. Durch die hohe Volatilität der Anforderungen unterliegt die Dokumentation regelmäßigen Änderungen, wobei sichergestellt werden muss, dass sie zeitnah aktualisiert wird. Während der Aufwand bei der Erstellung der Anwendungsdokumentation und Versionsverwaltung hoch ist und laufend anfällt, zeigt sich der Nutzen bei der Auslieferung der Software. Ein eingesetztes technisches Hilfsmittel ist das Wiki-System »Atlassian Confluence« [8] mit eingebautem Versionsverwaltungssystem, da es eine Bearbeitung der Dokumentation sowohl auf der Entwicklerseite als auch auf der Benutzerseite ermöglicht. In diesem Tool wird eine Beschreibung der Features in Form von User-Stories bereitgestellt. Andererseits erlaubt das Wiki-System eine Referenzierung der Funktionsbeschreibung, wodurch textuelle Erläuterungen mit Quellcodestellen assoziiert werden und somit zum Wissenstransfer innerhalb des Entwicklerteams beitragen.

3.2 Aufwandsabschätzung und Release-Management

Obwohl es sich bei dem »NCProfiler« um eine Software handelt, die im Forschungsumfeld entwickelt wird, muss dennoch bedacht werden, dass Fraunhofer-Institute praxisnahe Lösungen für Problemstellungen aus dem industriellen Alltag anbieten. Diese Ausrichtung erweckt bei den Kooperationspartnern eine Erwartungshaltung, durch ein Forschungsprojekt einen für den Produktionseinsatz geeigneten Softwareprototypen erlangen zu können. Diese Anforderung erfordert eine präzise Use-Case-Definition und Einsatzumfeldbeschreibung, welche die Erprobungsszenarien und Abnahmeprozedur der prototypischen Software präzisiert. Trotz der oft prototypischen Umsetzung muss der aktuelle Softwarestand im sicherheitskritischen Umfeld einsatzfähig sein, da die dadurch erzeugten NC-Programme die Steuerung von komplexen Maschinensystemen übernehmen. Durch eine fehlerhafte Software können daher hohe Kosten entstehen, die beispielsweise aus Werkzeug- oder Spindelschäden resultieren, was im schlimmsten Fall zum Produkstionsstillstand führen kann.

Ein weiterer Aspekt des Release-Prozesses ist die Kosten- und Zeitabschätzung für die Umsetzung eines neuen Features. Aufgrund meist wager Anforderungsspezifikationen durch den Endbenutzer und unvorgesehener Fragestellungen, die oft erst während der Projektlaufzeit identifiziert werden können, wird eine zuverlässige Aufwandschätzung erschwert. Eine Möglichkeit zur Unterstützung bieten Issue-Management-System wie zum Beispiel das am IPT eingesetzte »Atlassian Jira« [9]. In diesen Systemen kann neben der Planung von Aufgaben auch eine Dokumentation des tatsächlich gebrauchten Arbeitsaufwands zu der jeweiligen Aufgabe vermerkt

werden. Bei neuen Projekten kann auf diese Aufwandserfassung zurückgegriffen werden, um eine grobe Abschätzung von Aufwänden zu ermöglichen und ein passendes Projektbudget festzulegen. Die beteiligten Entwickler sollten hierbei informiert werden, dass die Aufwandserfassung nicht zu ihrer Überwachung oder Bewertung, sondern als Instrument zur Projektplanung verwendet wird.

3.3 Zusätzliche Softwareunterstützung zur Qualitätssicherung

Als Hilfsmittel zur Entwicklung des in Java programmierten »NCProfiler« werden einige zusätzliche Softwaretools verwendet, die im folgenden vorgestellt werden. Als IDE wird NetBeans 7.1.1 [10] verwendet, da die Software für die Java-Entwicklung sehr gut angepasst ist und der Entwickler beim Programmieren mit verschiedenen Hilfsmitteln gut unterstützt wird. Als Versionskontrollsystem kommt SVN [11] zum Einsatz, welches optimal in NetBeans integriert ist. Dieses System wird benötigt um bei mehreren Entwicklern die Änderungen am Code gut protokollieren und nachvollziehen zu können. Für größere Modifikationen am Projekt werden alternative Entwicklungszweige, so genannte »Branches« verwendet, damit Änderungen am Projekt nicht direkt am Hauptentwicklungszweig sichtbar werden. Erst nach Abschluss der Implementierung einer umfangreichen Änderung wird der Entwicklungszweig mit dem Hauptzweig wieder vereinigt, so dass zu jeder Zeit eine auslieferbare Version vorliegt.

In naher Zukunft wird der Build-Server »Bamboo« [12] zur kontinuierlichen Kompilierung und Integration in Betrieb genommen. Dieser Server führt bei jedem Eincheckvorgang von neuem Code einen kompletten Rebuild durch und weist den Entwickler so auf mögliche Probleme hin. Dadurch wird sichergestellt, dass der eingecheckte Code jederzeit kompilierbar ist. Gleichzeitig werden diverse vorher spezifizierte Tests durchgeführt, wobei beispielsweise der »NCProfiler« eine Optimierung an einem standardisierten NC-File durchführt.

Vollständige Tests im Forschungsumfeld zu generieren, ist extrem schwierig, da die Ausgabedaten einen Teil der Realität simulieren, die allerdings nie exakt erreicht wird. Die Software wird dabei laufend angepasst, um die Ausgabedaten zu verbessern. Somit können nur grobe Fehler festgestellt werden, wie z. B. Ausnahmen oder Divisionen durch Null. Für diesen Zweck wurden standardisierte Eingangsdateien entwickelt, die möglichst viele Testfälle abdecken. Diese werden automatisiert mit dem »NCProfiler« verarbeitet und bei auftretenden Fehlern erfolgt eine Benachrichtigung der zuständigen Entwickler.

4 Anwendung existierender Software-Prozessmodelle

Ein wichtiges Ziel des Entwicklungsprozesses für den »NCProfiler« ist es, eine Software-Struktur zu erhalten, die wartbar und verwaltbar ist. Hierbei spielt die Verzahnung der eingesetzten Software-Tools zur Entwicklungsunterstützung sowie eine definierte Vorgehensweise eine überragende Rolle. Während der Einsatz von mehreren Werkzeugen zur Entwicklungsunterstützung bereits stattfindet und erfolgreich erprobt wurde, unterliegt die praktizierte Vorgehensweise keinem klassischen Entwicklungsmodell. In [2] wurde detailliert erläutert, dass die Entwicklung des »NCProfiler« durch die verfügbaren Modelle nicht erfasst und abgebildet werden

kann. [1] stellt ein »Prozessrahmenwerk« für die Entwicklung wissenschaftlicher Software vor, das die Anforderungen des »NCProfiler« berücksichtigt. In diesem Modell werden Entwicklungsaktivitäten in drei separaten, aber miteinander verbundenen Teilprozessen, nämlich einen Plattformprozess, einen Feature-Prozesse und einen zugrundeliegenden Koordinationsprozess gegliedert.

Der Plattformprozess beinhaltet die ständige Wartung der allgemeinen Software-Architektur, um die Verwendbarkeit der Software bei neuen Forschungsprojekten zu gewährleisten. Da hierbei die Kernfunktionalität des »NCProfiler« kontinuierlich entwickelt wird, sollten in diesem Prozess festangestellte und erfahrene Entwickler für eine langfristige Begleitung involviert sein. In Forschungsprojekten und bilateralen Projekten wird der Funktionsumfang des »NCProfiler« durch eigenständige Entwicklerteams über den gesamten Lebenszyklus der Software als sogenannter Feature-Prozess parallel erweitert. Die einzelnen Feature-Prozesse können je nach Forschungsziel stark variieren und sind u. a. abhängig von der Art des Forschungsvorhabens. Allerdings muss bei allen Feature-Prozessen sichergestellt werden, dass die Anforderungen und Funktionen des zu entwickelnden Features sowie seine Implementierung und Integration in den »NCProfiler« genau dokumentiert werden. Obwohl jeder Feature-Prozess seinen eigenen spezifischen Software-Lebenszyklus besitzt, müssen alle Feature-Prozesse sowie der Plattformprozess über den Koordinationsprozess abgestimmt werden. Der Koordinationsprozess definiert die Entwicklungsinfrastruktur, die Dokumentationsanforderungen sowie die entsprechenden Freigaben und Meilensteine, die für alle Feature-Prozesse verbindlich sind. Darüber hinaus wird im Koordinationsprozess ein globaler Zeitplan mit Synchronisationspunkten für die Plattform des »NCProfiler« und alle darauf aufbauenden Feature-Projekte bestimmt.

Die vorgestellte Vorgehensweise ermöglicht einerseits die effiziente Entwicklung von neuen Software-Features auf der Basis einer Basisplattform und einer gemeinsamen Infrastruktur. Auf der anderen Seite kann die Qualitätssicherung und die Wartung einer Plattform gewährleistet werden. Hierzu sind allerdings Management- und Koordinationsmaßnahmen notwendig, auf die sowohl in [2] als auch in [1] eingegangen wird, damit die dezentralen Feature-Prozesse koordiniert werden können. Die vorgestellte Vorgehensweise kann zusätzlich durch die gewählte Software-Architektur unterstützt werden.

5 References

1 Hoffmann, V; Lichter, H.; Nyßen, A.: Processes and Practices for Quality Scientic Software Projects, 2011.
2 Gran, A.: Entwicklung eines Prozessmodells für die evolutionäre Software Entwicklung im Forschungsumfeld, Diplomarbeit, 2008.
3 Klocke, F. et al: Dynamikorientierte NC-Programme für die simultane 5-Achs-Hochgeschwindigkeitsbearbeitung, Fraunhofer-Institut für Produktionstechnologie IPT, Aachen, 2001.
4 Meinecke, M. et al.: Model Based Optimization Trochoidal Roughing of Titanium, 11th CIRP International Conference on Modeling of Machining Operations, 2008.
5 Markworth, L.; Glasmacher, L.: Simultaneous five-axis-machining of complex shaped parts. Proceedings of the 3rd International Conference on Machining and Measurements of Sculptured Surfaces, 55-65, 2003.
6 Cerit, E., Lazoglu, I., A CAM-based path generation method for rapid prototyping applications, The International Journal of Advanced Manufacturing Technology, 2011.
7 Pothen, M.; Minoufekr, M.; Huwer, T.; Glasmacher, L.: NC-Datenanalyse und –optimierung - Einsatz des »NCProfilers« im Werkzeug und Formenbau, wt Werkstattstechnik online, 2011.
8 Atlassian Confluence, http://www.atlassian.com/software/confluence/overview, 2012.
9 Atlassian JIRA, http://www.atlassian.com/software/jira/overview, 2012.
10 NetBeans IDE, http://netbeans.org/downloads/index.html, 2012.
11 Subversion, http://subversion.tigris.org, 2012.
12 Atlassian, Bamboo, http://www.atlassian.com/software/bamboo/overview, 2012.

Teil 2

Ablauf des Workshops
-
Ergebnisse und Ausblick

Ablauf des Workshop

Matthias Vianden

Lehr- und Forschungsgebiet Softwarekonstruktion
RWTH Aachen University
vianden@swc.rwth-aachen.de

Um einen Überblick über die breite Spanne der vertretenen Projekte zu bekommen, präsentierte sich jedes Projekt am ersten Tag des Workshops. In dieser Präsentation sollte nicht nur die im Projekt entwickelte Software vorgestellt, sondern es sollten auch schon auf Vorgehensweisen, Techniken und Erfahrungen eingegangen werden. Die Ziele des Workshops wurden dadurch sehr früh adressiert; dies erlaubte einen schnellen Einstieg in das Thema. Nach jeder Präsentation blieb Raum für eine kurze Diskussion, um spezifische und interessante Eigenschaften der einzelnen Vorgehensweisen oder Techniken zu besprechen. Hierbei wurden typische Probleme bei der Entwicklung von Forschungssoftware aufgenommen und auf einer Metaplanwand gesammelt. Neben bekannten Problemen bei der Erstellung von Software wie „Anforderungserhebung und Verwaltung“ oder „Aufwandsschätzungen“ wurden hier auch insbesondere forschungstypische Probleme genannt. Diese waren zum Beispiel:

- Einfluss von Kooperationspartnern
- Ablenkung durch das Tagesgeschäft (Lehre)
- Veröffentlichungsdruck
- Codequalität
- Ausbildungsstand der Entwickler
- Finanzierung von Wartungsaufwänden
- Ungewisser Erfolg

Alle diese Punkte zeigten noch einmal deutlich, dass die Entwicklung von Forschungssoftware andere bzw. angepasste Vorgehensweisen benötigt als typische Softwareentwicklungsprojekte in der Industrie.

Anschließend wurde im Plenum das weitere Vorgehen abgestimmt. Hierbei wurde insbesondere die inhaltliche Ausrichtung der einzelnen Arbeitsgruppen, welche für den nächsten Tag geplant waren, besprochen. Dazu wurden die oben genannten Probleme in Gruppen zusammengefasst. Diese Gruppen waren:

- Produkt- / Business-Modell
- Management und Vorgehensmodelle
- Werkzeuge
- Menschen und Human Factors
- Langlebigkeit

Nach einer kurzen Abstimmung über die einzelnen Themen zeigte sich, dass ein großer Teil der Anwesenden (14) mehr über den Bereich „Werkzeuge“ erfahren wollte bzw. darüber diskutieren wollte. Danach folgten die beiden Bereiche „Management und Vorgehensmodelle“ und „Menschen und Human Factors“ jeweils mit 11 Stimmen. Es wurde daher entschieden, die beiden Tracks am nächsten Tag so

zu füllen, dass zunächst zwei Arbeitsgruppen jeweils getrennt die Themen „Management und Vorgehensmodelle“ und „Menschen und Human Factors“ diskutierten. Nachdem danach die Ergebnisse der Arbeitsgruppensitzungen dem Plenum vorgestellt wurden, sollte das Thema „Werkzeuge“ gemeinsam bearbeitet werden. Hier integrierte sich dann auch eine Demonstration der Werkzeuge des SSE-Labs, das am Lehrstuhl Software Engineering betrieben und weiter entwickelt wird.

Da in der Arbeitsgruppe „Management und Vorgehensmodelle“ insbesondere ausführlich der Aspekt Entwicklungsprozess diskutiert wurde, wurde entschieden anschließend gemeinsam den Rahmen eines Plattform-Prozesses zur Softwareentwicklung im Forschungsumfeld gewidmet. Neben den Aktivitäten des Prozesses wurden auch Best Practices notiert und besprochen. Insbesondere die nebenläufige Entwicklung von Abschlussarbeiten und Baselines wurde als eine der wichtigsten Eigenschaften identifiziert, die ein solcher Prozess adressieren muss.

Abgeschlossen wurde der Workshop dann mit einer Feedbackrunde, in der sich alle Teilnehmer ausgesprochen positiv über den Ablauf, die Organisation und das Thema des Workshops aussprachen. Es wurde daher beschlossen, die inhaltliche Arbeit des Workshops in regelmäßigen „After Work“ Meetings jeweils am zweiten Mittwoch im Monat fortzuführen. Zu diesen Meetings soll es jeweils Schwerpunktthemen geben. Thema des ersten Treffens war „Aufwandsschätzung“, beim zweiten Treffen wurde das Thema „Architektur und Architekturmanagement“ diskutiert.

<table>
<tr><td rowspan="2">Einführung</td><td rowspan="5"></td><td>AG: Management und Vorgehensmodelle</td><td>AG: Menschen und Human Factors</td></tr>
<tr><td colspan="2">Präsentation der Ergebnisse</td></tr>
<tr><td rowspan="2">Präsentation der Projekte</td><td colspan="2">AG: Werkzeuge</td></tr>
<tr><td rowspan="2">Planung Folgetag</td><td colspan="2">Entwicklung eines Plattformprozesses</td></tr>
<tr><td colspan="2">Feedbackrunde</td></tr>
</table>

Tabelle 1: Ablauf des Workshops

Ergebnisse der Arbeitsgruppen und Ausblick

Andreas Ganser, Horst Lichter, Matthias Vianden

Lehr- und Forschungsgebiet Softwarekonstruktion
RWTH Aachen University
{ganser, lichter, vianden}@swc.rwth-aachen.de

In den folgenden Abschnitten stellen wir detailliert die Ergebnisse der einzelnen Arbeitsgruppen des Workshops vor. Diese fanden (wie oben beschrieben) am zweiten Tag des Workshops statt.

1 AG: Menschen und Human Factors

Diese Arbeitsgruppe hatte sich zum Ziel gesetzt, Probleme und Best Practices beim Umgang mit Menschen (hier insbesondere mit Studierenden, die im Kontext der Projekte eine Abschlussarbeit erstellen) bei der Softwareentwicklung im Forschungsumfeld zu identifizieren. Nach einem ersten Austausch über das Thema wurden die einzelnen Punkte im Wesentlichen basierend auf den Erfahrungen der Teilnehmer diskutiert. Als Ergebnis sind eine Reihe von *Dos* und eine kleine Liste von *Don´ts* entstanden, die im Folgenden kurz wiedergegeben und erläutert werden.

1.1.1 Don´ts

Es wurde versucht, möglichst viele positive Hilfen im Sinne von Dos zu formulieren. Dies führte umgekehrt dazu, dass die Liste der Don´ts eher kurz war, sie enthält die folgenden drei Punkte.

1. *Fachliche und persönliche Angelegenheiten werden nicht voneinander getrennt!*
2. *Zwischenmenschliche Probleme werden in der Gruppe eskaliert!*
3. *Es werden Studierende betreut, mit denen der Betreuer befreundet ist oder die er persönlich nicht schätzt!*

Insbesondere der letzte Punkt wurde rege diskutiert, da es hierzu sowohl positive als auch negative Erfahrungen bei den Teilnehmern gab. Allgemein wurde aber festgehalten, dass in der Regel die Betreuung von unbekannten Studierenden einfacher ist, wobei sich die Beziehung natürlich im Laufe der Betreuung ändert.

Als wichtige Hilfe bei Problemen wurde das sogenannte „4-Augen-Gespräch“ genannt, in dem frei über Themen und Probleme gesprochen werden kann, ohne dass einer der beiden Gesprächspartner das Gesicht verliert oder an den Pranger gestellt wird. Dadurch sollen Probleme möglichst schnell angesprochen und gelöst werden, bevor sie eskalieren.

1.1.2 Dos

Wie schon erwähnt, wurde versucht, möglichst viele Best Practices im Sinne von *Dos* zu formulieren. Die folgenden Punkte wurden aufgenommen (siehe Abbildung 1) und intensiv diskutiert:

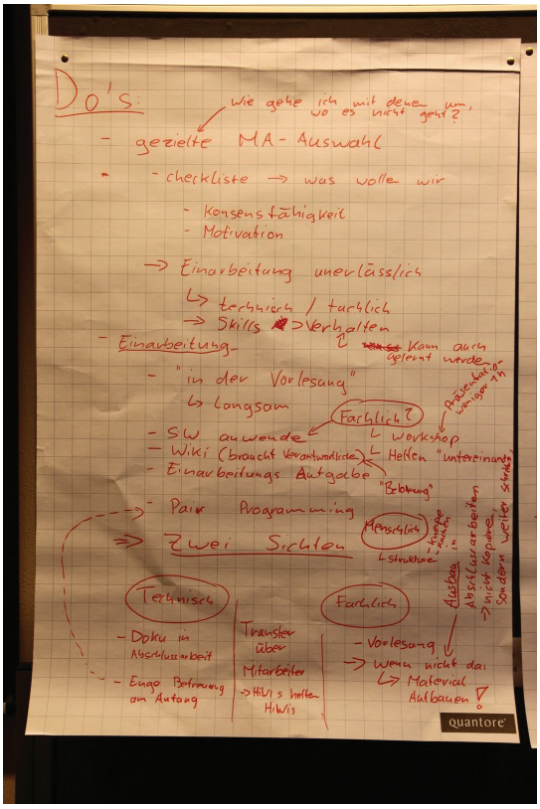


Abbildung 1: Die im AK identifizierten Dos

1. Gezielte (Mitarbeiter)-Auswahl

Natürlich möchte jeder Verantwortliche, dass in seinem Projekt nur „gute Mitarbeiter“ tätig sind, beispielsweise als wissenschaftliche Hilfskraft oder dadurch, dass eine Abschlussarbeit erstellt wird. Es stellt sich in diesem Zusammenhang die Frage, wie diese am besten ausgewählt werden können. Dazu wurde eine initiale Checkliste erstellt, die insbesondere die Punkte Konsensfähigkeit und Motivation enthielt.

Desweiteren wurde auch generell über wünschenswerte Skills und Verhalten diskutiert. In der Diskussion wurde klar, dass diese beiden Eigenschaften im Laufe der Tätigkeit oder der Abschlussarbeit erlernt werden können. Dazu ist jedoch eine intensive Einarbeitung und Betreuung unerlässlich. Als offene Frage wurde aufgenommen, wie agiert oder reagiert werden kann, wenn menschliche Problemsituationen entstehen, die schwerwiegend sind und das Projekt negativ beeinträchtigen können.

2. Intensive Einarbeitung neuer Mitarbeiter

Damit bei einer Tätigkeit oder Abschlussarbeit gute und verwertbare Ergebnisse erzielt werden können, ist es aus Sicht der Teilnehmer unerlässlich, neue Mitarbeiter intensiv einzuarbeiten. Nach einiger Diskussion wurde festgehalten, dass es bei der Einarbeitung zwei Dimensionen gibt: die technische und die fachliche Einarbeitung.

Bei der technischen Einarbeitung wurde insbesondere eine enge Betreuung am Anfang als wichtig angesehen. Hier bietet sich *Pair Programming* an, bei dem ein neuer Mitarbeiter dem Betreuer oder einem anderen erfahrenen Mitarbeiter beim Lösen typischer Aufgabenstellungen zuschaut. Auch spezielle Einarbeitungsaufgaben wurden als hilfreich genannt.

Die fachliche Einarbeitung wurde als eine der wichtigsten Voraussetzungen für eine erfolgreiche Tätigkeit oder Abschlussarbeit genannt, da hier in der Regel auch die größten Defizite bei neuen Mitarbeitern zu finden sind. Zur fachlichen Einarbeitung kann insbesondere Material aus entsprechenden Vorlesungen heran gezogen werden. Leider ist dieses aber nicht immer oder nicht in der dafür benötigten Qualität vorhanden. Eine entsprechende Aufbereitung des Materials ist in diesen Fällen sinnvoll und eine gute Investition in die Zukunft. Auch vorangegangene Abschlussarbeiten können zur fachlichen (und technischen) Einarbeitung genutzt werden. Es sollte allerdings darauf geachtet werden, dass die aufeinander aufbauenden Arbeiten nicht ständig sich wiederholende Passagen haben (insbesondere bei den Grundlagen), sondern das erarbeitete fachliche Wissen ergänzen. Eine weitere Möglichkeit, neue Mitarbeiter mit der Fachlichkeit vertraut zu machen, besteht darin, vorhandene Software-Werkzeuge zu verwenden. Dies kann sehr effizient mit sinnvollen kleineren Aufgaben (zum Beispiel Pflege eines vorhandenen Datenbestandes oder Durchführen von Experimenten) kombiniert werden.

Weitere Maßnahmen, die sowohl zur fachlichen als auch zur technischen Einarbeitung genutzt werden können, sind Wikis. Damit diese erfolgreich aufgebaut und genutzt werden können, muss es zumindest einen Verantwortlichen geben, der darauf achtet, dass die Inhalte weiter gepflegt werden und aktuell gehalten werden. Damit regelmäßig neue Inhalte eingebracht werden, sollte eine Art von „Belohnung“ etabliert werden. Hier reicht möglicherweise schon eine positive Erwähnung bei regelmäßigen Meetings.

Kurze, thematisch fokussierte Workshops unterstützen sowohl die fachliche als auch die technische Einarbeitung. Hier sollte darauf geachtet werden, dass diese auch wirklich kurz sind. Als sinnvoll wurde hier weniger als eine Stunde angesehen, damit die Aufnahmefähigkeit der Teilnehmer nicht überschritten wird und das erlernte Wissen verarbeitet werden kann. Im speziellen Fällen können aber auch 1,5 Stunden sinnvoll sein.

Als letzte hilfreiche Maßnahme für beide Dimensionen wurde „Mitarbeiter helfen Mitarbeitern“ diskutiert. Erfahrene Mitarbeiter haben in der Regel sowohl ein tiefgreifendes fachliches als auch technisches Wissen, welches sie an neue Mitarbeiter weitergeben können. Wichtig ist hier insbesondere, dass sich die erfahrenen Mitarbeiter noch an ihre eigene Einarbeitung erinnern und so typische Probleme initiativ ansprechen.

Neben der fachlichen und technischen Einarbeitung ist aber auch die Integration der neuen Mitarbeiter in das Projektteam wichtig. Dadurch lernen sie vorhandene

Strukturen kennen und gliedern sich in diese ein. Als sehr hilfreich hierzu wurden gemeinsame „Social Events“ genannt (beispielsweise Kneipenbesuche, gemeinsames Mittagessen oder „Kuchen“ nach Meetings).

3. Geplante Übergabe der erzielten Ergebnisse

Neben den beiden oben genannten wichtigen Punkten wurde die Übergabe der von Studierenden erzielten Ergebnisse genannt. Diese muss immer explizit bei jeder Tätigkeit oder Abschlussarbeit eingeplant werden (denn das benötigt Zeit und Aufwand von Seiten des Betreuers). Dies ist insbesondere wichtig, um einen festen Projektabschluss zu realisieren. Analog zu den oben genannten Punkten sollte auch für die Übergabe bzw. den Projektabschluss eine individuelle Checkliste ausgearbeitet werden.

2 AG: Management und Vorgehensmodelle

Entwickeln mehrere Beteiligte in einem Forschungsprojekt gleichzeitig Ergebnisse, so muss auch auf der nichtfachlichen Ebene koordiniert werden. Diese Ebene umfasst Management und Vorgehensaspekte und unterstützt die Entwicklung in steuernder Art. Dazu gibt es beginnend von Richtlinien über Projektdefinitionen bis hin zu Architekturrichtlinien viele unterstützenden Hilfsmittel, aber auch Herausforderungen und Probleme. Besonders die Probleme sind im gegebenen Kontext von spezieller Natur.

Zunächst ist in vielen Fällen ein "Definition of Done" schwer erzielbar, da es sich um Forschung und unterstützende Software handelt. Ist Software als Artefakt schon relativ zu den Anforderungen und damit schwer quantifizierbar, so fügt der Forschungskontext einen weiteren Unsicherheitsfaktor hinzu. Darüber hinaus unterliegen Beiträge, die in Abschlussarbeiten entwickelt werden, auch einer Prüfungssituation. Das bedeutet, dass Studierende, motiviert durch das Ziel eine gute Note zu erreichen, möglicherweise zu viel Funktionalität hinzufügen und damit die Anforderungen übererfüllen. Dieser "Feature Creep" wird in vielen Forschungsprojekten gleichzeitig als Fluch und als Segen empfunden. Sind die zusätzlichen Funktionen anfänglich meist hilfreich, entwickeln sie sich über die Zeit, wie alle anderen Teile, zu Altlasten, die der Wartung unterliegen und damit Aufwand erzeugen.

Besonders dramatische Auswirkungen hat ein "Feature Creep" dann, wenn große Teile in einem Schritt integriert werden müssen. Dieser so genannte "big bang" kommt in vielen Forschungsprojekten vor und verschleiert wegen Überhangfunktionalität die Kernfunktionalität der Software. Durch angemessenes Management und angemessene Werkzeugunterstützung mit Continuous Integration lässt sich diese Problematik mildern. Ein weiterer Lösungsansatz ist eine entsprechende Architektur. Wird eine Architektur so entwickelt und gepflegt, dass sie die Integration von Features unterstützt, werden kurze Integrationszyklen möglich. Dabei ist darauf zu achten, dass ein Architekt über die entsprechenden Erweiterungspunkte entscheidet und zu starke Kopplung vermeidet. Ganz besonders sollten unerfahrene Entwickler keine architektonischen Entscheidungen treffen und Entwurfsentscheidungen und Schnittstellen regelmäßig im Team abstimmen.

Insgesamt ließ sich feststellen, dass in den meisten Projekten ein geeigneter Entwicklungsprozess fehlte. Zwei der Projekte haben Scrum teilweise erfolgreich eingesetzt und darauf hingewiesen, dass leichtgewichtige Prozesse bei hoher Mitarbeiterfluktuation der einzige Ausweg sei. Darüber hinaus kristallisierte sich ein Rahmen heraus, wie Teilprojekte oder Features entwickelt werden könnten. Dieser Rahmen ist auch der Tatsache geschuldet, dass in Forschungsprojekten gelegentlich Beiträge sehr unterschiedlicher Qualität abgeliefert werden und der zeitliche Rahmen für Beiträge oft nicht ausreichend ist.

2.1 Vorschlag für einen Vorgehensrahmen

Als Rahmen für Teilprojekte wurde besonders für neue Mitarbeiter ein leichtgewichtiges Vorgehen entwickelt. Von einer ersten Idee bis hin zur Realisierung wird dabei besonders auf das Commitment geachtet. Deshalb wird anfänglich keine genaue Teilprojektbeschreibung vorgelegt, sondern nur der Projektkontext und eine Idee kommuniziert. Auf letztere aufbauend ist dann von dem neuen Mitarbeiter eine Vision zu entwickeln, die in mehreren Iterationsschritten überarbeitet wird. Dadurch entwickelt der Mitarbeiter eine enge Bindung zur Vision und dem angestrebten Ziel. Andererseits ist von Auftragsseite sicherzustellen, dass die Problemgröße angemessen bleibt.

Liegt eine formulierte Vision für das Teilprojekt vor, so sind daraus die tatsächlichen Anforderungen abzuleiten und mit potentiellen Benutzern zu diskutieren. Je nach Natur des Teilprojektes bieten sich verschiedene etablierte Techniken an. Oft kann in diesem Zusammenhang auf User Interface Paper-Prototyping zurückgegriffen werden. Durch Benutzerstudien werden dann die essentiellen Use Cases festgelegt und die Benutzeroberfläche bleibt nah an den Erwartungen der späteren Benutzer. Außerdem verhindert diese Herangehensweise, dass unnötige Funktionen realisiert werden. Aufbauend auf die extern sichtbare Funktionalität lässt sich dann auf Datenebene ein Prototyping durchführen, damit so viele Daten wie nötig, aber so wenig wie möglich erhoben werden. Verschiedene Software-Prototypingtechniken können diesen Schritt erleichtern. Ein folgendes algorithmisches Prototyping und der Einbau in das Projekt schließen den Rahmen ab.

Ein zentraler Aspekt des Rahmens besteht aus den begleitenden Maßnahmen. Dazu gehört das Zerlegen in kleinere und handhabbare Arbeitspakete und Meilensteine. Sie sollen besonders bei der Projektplanung helfen und mindestens als Gantt Charts vorliegen. Die Meilensteine und Arbeitspakete sollten dann in ein Ticketsystem überführt werden, damit eine Fortschrittsverfolgung und Versionstransparenz im SCM ermöglicht werden. Eine Risikobetrachtung gehört ganz explizit nicht zu dem Rahmen. Denn Risiken sind so projektspezifisch, dass ihre allgemeine Betrachtung keinerlei Mehrwert erzeugt.

Je nach Projekttyp kann es vorkommen, dass die Anforderungen an die Realisierung nicht besonders umfangreich sind. Das kann zum Beispiel bei Optimierungen oder Anforderungen mit rein mathematisch funktionalem Charakter vorkommen. Beispielsweise könnte ein schnellster Algorithmus zu finden sein. Doch diese Projekte sind experimenteller Natur und folglich nur schwach mit Vorgehensmodellen zu unterstützen.

3 AG: Werkzeuge

Die Entwicklung von Forschungssoftware benötigt aus vielen Gründen zunehmend komplexe Werkzeuginfrastruktur. Einerseits das den stetig wachsenden und doch selten gewarteten Quelltexten und weiteren Dokumentationen geschuldet. Andererseits ändert sich gelegentlich der Forschungsschwerpunkt eines Projektes, sodass Altlasten entstehen, die aber aus Dokumentationsgründen weiter archiviert werden müssen; beispielsweise, weil eine Publikation diese Dokumente referenziert. Folglich ist eine langfristig stabile Werkzeuginfrastruktur in den meisten Fällen zwingend.

Diese Werkzeuginfrastruktur umfasst ein breites Spektrum beginnend vom Dokumentenmanagement über Entwicklungsumgebungen und Build-Infrastruktur, bis hin zu Deployment- und Server-Infrastruktur. Dennoch muss diese Werkzeuginfrastruktur langfristig stabil und bedienbar sein.

3.1 Build- und Deployment-Infrastruktur

Als besondere Herausforderungen haben sich deshalb die Build- und die Deployment-Infrastruktur gezeigt. Aktuelle Technologien unterstützen hier aber soweit, dass nightly builds inzwischen in vielen Projekten eingesetzt werden. Dabei werden auch automatisierte Tests eingesetzt und ein automatisches Deployment durchgeführt. Allerdings hat das bei zwei Projekten dazu geführt, dass die Build-Infrastruktur noch deutlich komplexer wurde und jetzt als problematisch eingestuft wird.

Ein Ausweg aus schwer verständlicher Build-Infrastruktur können virtuelle Entwicklungsmaschinen darstellen, die "out of the box" alle notwendigen Installationen vorweisen. So wird auch jungen und unerfahrenen Mitarbeitern ermöglicht, schnell ihre Beiträge in das Projekt einzubringen. Die Lernkurve dieser Mitarbeiter ist also vor allem in der Anfangsphase steil. Nachteilig ist an diesem Ansatz, dass "rush to code" unterstützt wird, womit auch eines der Projekte Probleme hat. Denn eine direkte Konsequenz ist mangelnde Dokumentation; vor allem in der Anforderungserhebung.

3.2 Autoformatierer

Abschließend wurde noch kontrovers diskutiert, ob Autoformatierungen und Checkstyles von Quelltexten nützlich oder hinderlich seien. Auf der einen Seite wurden Autoformater, auch "pretty printer" genannt, positiv gesehen, da sie sicherstellen, dass die Layout-Konventionen des Projektes eingehalten werden. Gibt es Abweichungen, können diese gewöhnlich automatisch gelöst werden. Andererseits verliert ein potentiell qualitativ schwieriger Quelltext ein Qualitätsmerkmal. Denn schlechte Quelltextqualität war oft auch mit schlechter Formatierung korreliert. Folglich ließen sich qualitativ minderwertige Abschnitte schlechter finden. Anders verhält es sich mit "Checkstyle" Konfigurationen, die Konventionen für Dokumentation, Attributierung und Zugriffe definieren. Diese sind oft Best Practices, die zwar gefunden, aber nicht automatisch korrigiert werden können. Sollen z.B. Attribute grundsätzlich privat sein, lässt sich eine Warnung oder Fehlermeldung

ausgeben. Die Sichtbarkeit allerdings manuell zu ändern, wäre fatal. Für unerfahrene Mitarbeiter haben sich "Checkstyle" Konfigurationen als sehr hilfreich herausgestellt, da sie auch architekturelle Regeln enthalten können.

4 Ein Plattform-Prozess zur Softwareentwicklung im Forschungsumfeld

Abbildung 2 zeigt die Struktur des entworfenen Plattformprozesses. Dieser beschreibt im Wesentlichen ein typisches feature-getriebenes Vorgehen. Ein wichtiges Unterscheidungsmerkmal im Gegensatz zu einem analogen Vorgehen im industriellen Umfeld ist hierbei allerdings, dass nicht jedes Feature zwangsläufig in die Plattform übernommen wird, da es aufgrund des Forschungscharakters des Projekts vorkommen kann, dass ein geplantes und entwickeltes Feature nicht den erhofften Nutzen / Erfolg zeigt.

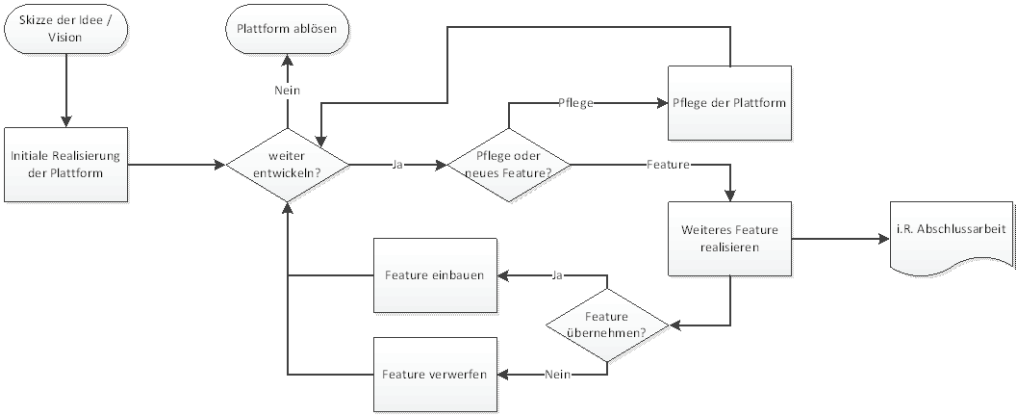


Abbildung 2: Plattformprozess zur Software Entwicklung im Forschungsumfeld

Der Prozess startet analog zur industriellen Softwareentwicklung mit einer ersten Skizze bzw. Vision des Projektes. Hierbei sollten insbesondere zentrale Ziele und Ergebnisse des Projektes festgelegt werden, aus denen nachher die verschiedenen Features gewonnen werden können. Als Ergebnis steht nach der ersten Realisierung dann eine initiale Plattform, welche in der Regel ebenfalls in einer (Abschluss-) Arbeit entwickelt wird. Hier muss dann entschieden werden, ob die Plattform weiterentwickelt werden soll, wodurch ein iterativer und inkrementeller Erweiterungsprozess eingeleitet wird. Nachdem ein neues Feature realisiert wurde, sollte in einem Entscheidungsprozess (am besten mit einem formalen Review qualitätsgesichert) entschieden werden, ob das Feature in die Plattform übernommen oder verworfen wird. Wie oben schon angesprochen, ist insbesondere das Verwerfen eines Features natürlich für Forschungssoftware. Häufig ist ein vermeintlicher Fehlschlag dann der Anstoß für weitere Forschung. Neben der Weiterentwicklung der Plattform muss diese allerdings auch konstant gepflegt werden. Hierdurch kann eine Qualitätskultur entstehen, die positive Auswirkungen auf die Realisierung von weiteren Features hat.

4.1 Best Practices

Neben der Beschreibung dieses zentralen Prozessrahmens wurde auch eine Menge an Best Practices (insbesondere an Tools) gesammelt, welche bei der Entwicklung einer solchen Plattform hilfreich sein können.

4.1.1 Organisatorische Best Practices

- Klare Verantwortlichkeiten im Projekt (wer macht was?)
- Sinnvolle Organisationsstruktur
- Einheitliche Methodik
- Kollaborationsprozess
- Etablierte Schnittstellen Prozesse
- Etabliertes Change Management
- Klarer „Feature-Freeze“

4.1.2 Best Practices im Bereich Checklisten und Dokumente

- Plattform Glossar
- Qualitäts-Richtlinien
- Infrastruktur-Richtlinien
- Architektur-Richtlinien
- „Prototype to Feature“ Checkliste

4.1.3 Technische Best Practices

- Continuous Integration
- Versionskontrolle
- Automatisierte Tests der Plattform
- Konstante Weiterentwicklung der Werkzeuge und Infrastruktur

5 Ausblick

Der Workshop „Entwicklung und Evolution von Forschungssoftware“ ist der erste, der sich mit dieser Thematik beschäftigt hat. Naturgemäß konnten bei weitem nicht alle Fragen und Herausforderungen, die in diesem Zusammenhang interessant sind, diskutiert werden. So war die Liste der offenen Fragen länger als die der Themen, zu denen erste Ergebnisse erzielt werden konnten.

Wichtig war, dass sich Verantwortliche und Entwickler, die in solchen Projekten tätig sind, austauschen konnten und so erkennen konnten, dass „ihre“ Probleme möglicherweise nicht projektspezifisch sind, sondern auch in anderen Projekten zentral sind.

In den Diskussionen hat sich heraus gestellt, dass oft schon einfache Hilfsmittel (beispielsweise Checklisten) einen sehr großen Nutzen für die Projekte erbringen können.

Basierend auf den Diskussionsbeiträgen und den erzielten Ergebnissen könnte zukünftig eine Agenda aufgestellt werden, die zentrale Aspekte dieser speziellen Art der Software-Entwicklung enthält.