Agile Programmierung in der Praxis

— Seminararbeit —

Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Seminar Softwareentwicklung in der Wissenschaft
Sommersemester 2014

Vorgelegt von:	Tatyana Yotsova
E-Mail-Adresse:	tatyana_yotsova@web.de
Matrikelnummer:	6195176
Studiengang:	Wirtschaftsinformatik

Betreuer: Christian Hovy

Hamburg, den 30.08.2014

Inhaltsverzeichnis

1 Vorwort

Nachdem sich der Einsatz von agilen Prozessmodellen in vielen Bereichen der Wirtschaft als gut geeignet erwiesen hat, versucht man agile Praktiken auch in der wissenschaftlichen Softwareentwicklung anzuwenden.
In dieser Seminararbeit werden Softwareentwicklungsprojekte aus drei großen Forschungsbereichen vorgestellt. Betrachtet werden Projekte in den Forschungsbereichen Bioinformatik, Klimaforschung sowie Luft- und Raumfahrttechnik.
In diesen Forschungsbereichen stellt die Einführung von agilen Prozessmodellen, eine große Veränderung der Entwicklungsprozesse dar. Die Erkenntnisse zur Anwendung von XP Praktiken werden teilweise aus Beobachtungsstudien zu Projekten gewonnen (Paper 1, Paper 3) oder sind als Berichte zu Pilotprojekten (Paper 2, Paper 4, Paper 5) verfasst. Die Arbeiten zeigen einen großen Unterschied im Grad der Vorkenntnisse und Erfahrungen, die die Organisationen mit XP Praktiken hatten.
Auch die Organisationen selbst, unterscheiden sich stark in ihren Strukturen und Abläufen. So gibt es Organisationen deren Strukturen und Abläufe von vornherein ganz gut mit XP harmonieren müssten, andererseits gibt es aber auch Organisationen bei denen die Einführung von XP einen regelrechten Kulturschock bedeutete (Paper 4, NASA Luft und Raumfahrt).
Bei der Zusammenfassung und Verdichtung der Berichte wurde versucht die Inhalte so wiederzugeben, dass sie der Perspektive der jeweiligen Autoren entsprechen. Die Autoren gehen unterschiedlich stark auf die Darstellung von Prozessen oder Strukturen, auf das eigentliche Forschungsgebiet, oder auf die Verwendung agiler Praktiken ein.
Meine Zusammenfassungen sind so gestaltet, dass sie den Charakter dieser Ausarbeitungen beizubehalten versuchen. Ich beabsichtige hierdurch die wesentlichen, individuellen Erfahrungen der Studien zu erhalten, was bei einem schablonenhaften Vergleich der Ergebnisse allein, nicht möglich wäre.

2 Grundlagen des agilen Programmierens

2.1 Agile Manifesto

Das agile Manifesto ist das Fundament der agilen Programmierung. In diesem sind die Werte der agilen Programmierung zusammengefasst.

- Menschen und Zusammenarbeit sind wichtiger als Prozesse und Werkzeuge
- Lauffähige Software ist wichtiger als umfangreiche Dokumentation
- Zusammenarbeiten mit Auftraggebern ist wichtiger als Vertragsverhandlungen
- Reagieren auf Änderungen ist wichtiger als das sture Befolgen eines Plans

Auf Basis des agilen Manifesto ist eine Vielzahl an agilen Prinzipien definiert worden. Das Verfolgen dieser Prinzipien wird mit Hilfe von agilen Methoden (Praktiken) möglich. Die unterschiedliche Zusammensetzung von agilen Methoden führt zum Entstehen von verschiedenen agilen Vorgehensmodellen. Relevant für diese Seminararbeit sind Extreme Programming (XP) und SCRUM.

2.2 XP

Extreme Programming ist ein Prozessmodell, das die besten Softwarepraktiken in den Vordergrund stellt. Die zwölf grundlegenden Praktiken sind: Pair-Programming, testgetriebene Entwicklung, Refactoring, kollektives Eigentum, permanente Integration, Kundeneinbeziehung, kurze Iterationen, keine Überstunden, Metapher, Coding-Standards, einfaches Design und Planning-Game. XP eignet sich gut für die Realisierung von komplexen Projekten, bei denen die Bereitstellung von Funktionalität während der Gesamtprojektdauer besonders wichtig ist. Dank der testgetriebenen Entwicklung wird sicher gestellt, dass potentielle Veränderungen oder Erweiterungen des Projekts keine negative Auswirkungen auf die geschaffene Funktionalität haben werden. Kurze Iterationen und die Einbeziehung des Kunden sind ebenfalls besonders wichtig für die erfolgreiche Umsetzung eines Projekts.

2.3 SCRUM

Scrum ist auch ein Prozessmodell, das das agile Manifesto implementiert. Im Gegensatz zu XP, wird hier nicht die Umsetzung von den einzelnen agilen Praktiken in den Vordergrund gestellt, sondern der Projektmanagementprozess als Ganzes. Die Teammitglieder werden in drei Rollen unterteilt: Product Owner, Entwicklungsteam und Scrum Master. Jede dieser Rollen bringt spezifische Anforderungen mit sich, die durch die Anwendung von agilen Praktiken erfüllt werden. Der Product Owner hält den Kontakt zum Auftraggeber und entscheidet darueber welche User Stories in der nächsten Iteration bearbeitet werden. Das Entwicklungsteam entscheidet selbst, in welcher Reihenfolge und mittels welcher Techniken, die User Stories umgesetzt werden. Der Scrum Master ist für den reibungslosen Ablauf des Scrums zuständig, moderiert die Meetings und betreut das Team in schwierigen Situationen.
Scrum eignet sich gut für Projekte bei denen die Weiterentwicklung schwer planbar ist, und natürlich für neu gestartete Projekte.

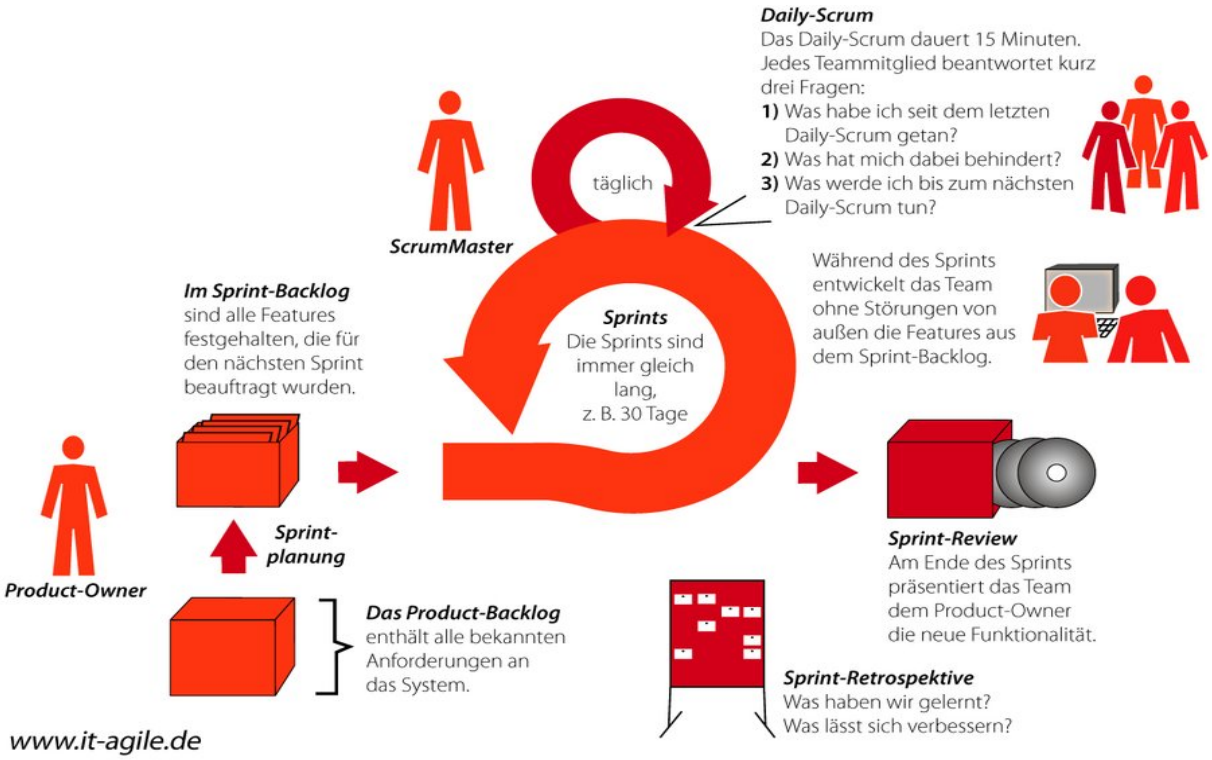

Abbildung 2.1: SCRUM

3 Umsetzung in der Praxis

In diesem Kapitel werden fünf Softwareentwicklungsprojekte aus der Praxis vorgestellt, die auf Basis von agilen Prozessmodellen oder durch den Einsatz agiler Praktiken umgesetzt wurden. Die Beispiele stammen aus unterschiedlichen, wissenschaftlichen Forschungsgebieten.

3.1 Paper 1 - Engineering the Software for Understanding Climate Change

Die Klimaforschung stellt sehr hohe Anforderungen an die Softwareentwicklung und die Rechnerleistungen, da sie sehr komplexe Berechnungen und Simulationsmodelle verwendet. Aus vorangehenden Untersuchungen ist bereits bekannt, dass wissenschaftliche Software in diesem Bereich eine sehr lange Lebensdauer hat, während der sie permanent an neue wissenschaftliche Erkenntnisse und neue Computertechniken angepasst wird.

Es wurde eine 8 wöchige Beobachtungsstudie am Met Office Hadley Centre (UK) durchgeführt, die zur Identifizierung der Konzepte und Arbeitspraktiken diente, die von den Klimaforschern angewendet werden, und zum Verstehen ihrer Beweggründe.
Das Hadley Centre wurde ausgewählt, da es international führend in der Klimaforschung ist, und unter Klimaforschern für seine, in Teilbereichen als gut eingeschätzten Softwareentwicklungsprozesse, bekannt ist.
Met Office Unified Model ist eine gemeinsame Sammlung von Fortran Routinen für die NWP (Numerical Weather Predition) und Klima Modelle. Die Code-Basis wurde über einen Zeitraum von 30 Jahren entwickelt, und umfasst aktuell ca. 830.000 Zeilen Fortran Sourcecode. Diese Software kann für unterschiedliche Wettervorhersagemodelle verwendet werden.
Die Berechnung dieser Vorhersagemodelle ist in der Laufzeit abhängig von der gewünschten Vorhersagedauer, der Auflösung und der Komplexität der verwendeten Modelle.
So dauert bspw. die Berechnung einer 100 jährigen Klimasimulation auf einem aktuellen Supercomputer (bspw, NEC-SX8) mehrere Monate, wohingegen die Berechnung mehrerer Dekaden nur (!) einige Wochen in Anspruch nimmt.
Neben dieser zentralen Software, kommen noch Anwendungen zum Einsatz, die für die Bereitstellung der Wetterdaten, die Konfiguration der Modelle, sowie Analyse und Grafikaufbereitung der Daten angewendet werden. Treiber für die Veränderung der Software waren hauptsächlich neue wissenschaftliche Erkenntnisse, die dazu führten dass der Verlauf der physikalischen Prozesse in den Modellen anders abgebildet wurden.

Des Weiteren führten Software-Reengineering Maßnahmen und Operationale Veränderungen (Hardware, Konfigurationen etc.) zu Veränderungen an der Software. Diese Treiber waren interner Natur, und nicht von Kunden beeinflusst.

Das Met Personal übernimmt spezielle Rollen, die einer Art Zwiebel-Modell entsprechen, wie man es öfter in Open Source Projekten beobachten kann. Dabei übernimmt ein Kernteam von ungefähr 12 Personen die Anforderungsüberprüfung in einen trunk für das UM. Auf der nächsten Schale gibt es ungefähr 20 Senior Wissenschaftler, die als Code Owner agieren und für spezielle Teile des UM verantwortlich sind. Als Code Owner, sind sie dabei Domänen Experten, die die wissenschaftlichen Fortschritte in ihrem Bereich verfolgen und die Übersicht für Entwicklungen in ihrem Teilbereich des Modells haben. Auf den äußeren Schalen arbeiten Wissenschaftler, die das Modell als Teil ihrer wissenschaftlichen Arbeit nutzen. Ein Konfigurationsmanager ist für jedes Klimamodell benannt, hierbei handelt es sich, üblicherweise, um einen Nachwuchswissenschaftler.
Releases von UM werden in einem, sich wiederholenden Zyklus von 4-5 Monaten geplant. Alle Anforderungen werden als Tickets erfasst. Es existiert eine Deadline für Anforderungen an das aktuelle Release, die ungefähr im 3. Monat des Releasezyklus beginnt, und auf die, einen Monat später der Codefreeze folgt. Ein weiterer Monat ist für das Team vorgesehen, um alle Konfigurationen vorzunehmen, und alle offenen Bugs zu beseitigen. Bevor eine Änderung Eingang in den aktuellen trunk findet, muss sie zwei Qualitätsstufen durchlaufen. Zuerst prüft der verantwortliche Code Owner nach wissenschaftlichen Gesichtspunkten, dann überprüft ein IT-Teammitglied die Einhaltung von Coding Standards, Code Hygiene, potenzielle Performanz-Auswirkungen, und auf Integrationstests zu unterschiedlichen Modell-Konfigurationen. Bei Übernahme in den trunk werden automatisierte, nächtliche Testläufe durchgeführt.
Die Validierung der Parameter, und die Überprüfung der Ergebnisse, wird, während der Entwicklungsphase, laufend vorgenommen. So wird kontinuierlich überprüft, ob die umgesetzten Anforderungen den erwarteten Ergebnissen, aus den vorgenommenen Modellveränderungen, entsprechen. Die Wissenschaftler führen also ein continous integration testing durch, ohne es so zu bezeichnen, da sie es als natürlichen Teil ihrer wissenschaftlichen Arbeit betrachten.
Eine weiterer Qualitätsmaßnahme ist die Durchführung von Regressionstests. Dadurch kann festgestellt werden, dass eines Softwareänderung nichts verändert , was sie nicht verändern sollte.
Das gemeinsame Verständnis wird durch unterschiedliche Strategien gefördert.
Die direkte Kommunikation stellt dabei den wichtigsten Aspekt dar. Dies wird durch die räumliche Gestaltung gefördert, ein großes flaches Büro , in dem man eine großen Teil der Beteiligten erreichen kann, ohne eine Tür zu durchqueren, oder das Stockwerk wechseln zu müssen. Die Teams setzen außerdem sehr stark elektronische Medien zur Kommunikation und Koordination ein.
Besonders hervorzuheben scheint die Tatsache, dass die Entwicklung des Wettermodells an einem gemeinsamen Standort betrieben wurde. Aus bisherigen Untersuchungen ist bekannt, dass alle komplexen Wettermodelle immer an einem gemeinsamen Standort entwickelt wurden.

Die Wissenschaftler haben nur wenig formale Ausbildung in der Softwareentwicklung. Sie stehen dem Einsatz von Softwareengineering Tools eher kritisch gegenüber, machen aber gern davon Gebrauch, wenn sie ihre Bedürfnisse gut erfüllen. Die Software hat eine lange Lebensdauer, die Entwickler haben einen stark ausgeprägten, gemeinsamen wissenschaftlichen Hintergrund. Sie pflegen einen informellen, kollegialen Arbeitsstil, mit einer Kultur, die die Beteiligten einbezieht und die Verantwortlichkeit teilt.
Das beständige Anwachsen des UM in den letzten 15 Jahren (siehe Abbildung: 3.1), widerspricht anderen Studien, die für große kommerzielle Systeme eine Verlangsamung der Weiterentwicklung vorhersagen, wenn sich der Umfang und die Komplexität erhöhen.

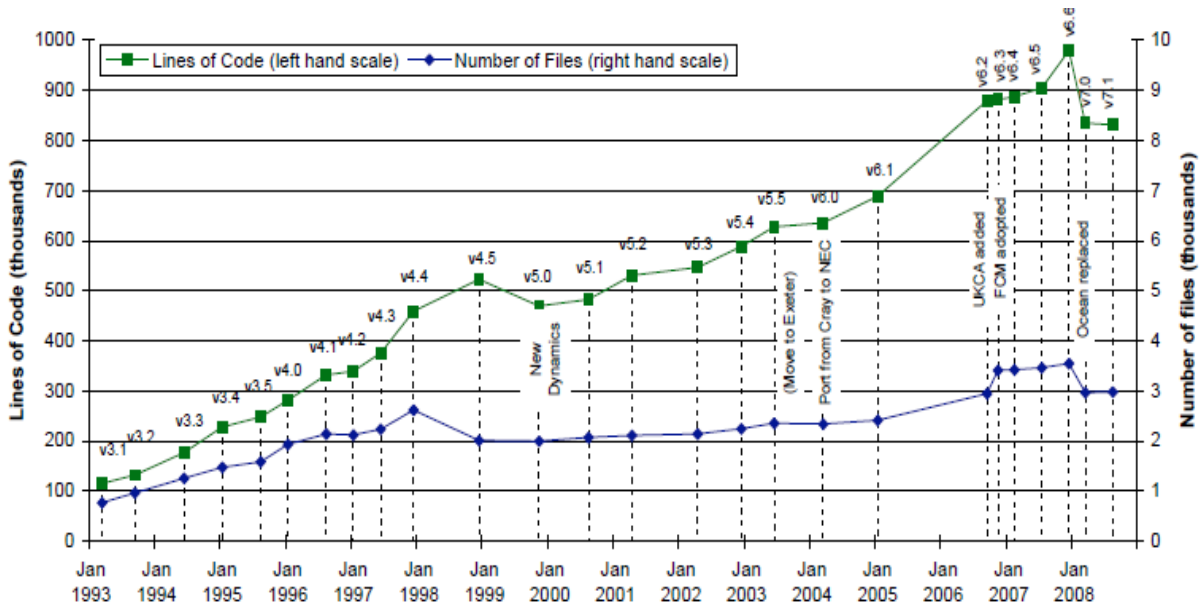

Abbildung 3.1: A Summary of the Organisations

Vielmehr lassen sich hier eher Parallelen, zu Weiterentwicklungsmustern von Open Source Projekten erkennen. Dies wird auf die vielen Gemeinsamkeiten mit Open Source Projekten zurückgeführt, die bereits diskutiert wurden. Die Organisation verhält sich teilweise wie eine agile Software Firma. Alle Entwickler befinden sich in einem großen gemeinsamen Büro, mit sehr ausgepägten, informellen Kommunikationskanälen.
Es werden viele agile Praktiken angewendet, einschließlich Releaseplanung, Einbeziehung des Kunden, gemeinsame Ownership, continous integration, und Risiko Management. Es ist aber kein agiles Prozess-Modell eingeführt. Außerdem wird in einer viel größeren Anzahl zusammengearbeitet, als dies bislang in der Literatur zu agilen Teams beschrieben wurde.
Diese Beobachtungen bestätigen, dass die agile Entwicklung eine Anzahl best practices beinhaltet, die von Entwicklern angewendet werden, da sie in ihrem Arbeitsumfeld gut funktionieren. Des Weiteren stimmen die Beobachtungen mit anderen Studien überein,

die feststellten, dass erfolgreiche, ehemalige Software Startups, dadurch charakterisiert werden konnten, dass sie eine Anzahl agiler Praktiken angewendet haben, jedoch jeweils im Rahmen eines eigenen, gewachsenen Prozessmodells.
Im Laufe der Zeit entwickeln diese Organisationen Prozesse, die hochgradig auf ihr Geschäftsfeld ausgerichtet sind.
Diese erfolgt jedoch möglicherweise nur in stabilen, gut etablierten Teams, mit einer langen Erfahrung in der Zusammenarbeit.
Dies trifft bei den erwähnten Studien, und im Falle des Met Office zu. Allerdings zeigen diese Studien auch, dass domänenunabhängige Prozessmodelle wie XP, SCRUM und RUP für diese Organisationen nicht relevant sind.
Gemeinsam bleibt allen drei Organisationen (open source, agile, wissenschaftlich), dass ihre Entwickler über ein sehr tiefes Verständnis und einen hohen Grad an Selbstorganisation verfügen. Als Hypothese kann man unterstellen, dass die Entwickler unter solchen Umständen, ein hochgradig auf ihren Arbeitskontext angepasstes Prozessmodell entwickeln werden, unberücksichtigt der Ratschläge, die die Software Engineering Literatur vorgibt.

3.2 Paper 2 - Chaste: Using Agile Programming Techniques to Develop Computational Biology Software

Die Herzmodellierung stellt jenen Teil der physiologischen Modellierung dar, in welchem die Software wahrscheinlich am Weitesten entwickelt ist. In der vorliegenden Studie, wurden Einige, der am weitesten fortgeschrittenen Softwarepakete zur Simulation der elektrischer Herzaktivität, als Ausgangspunkt zur Untersuchung von Software-Entwicklungsmethoden verwendet.
Hierbei wurden die Software Entwicklungsmethoden in Bezug auf die Anzahl numerischer Algorithmen, die relative Effizienz der Berechnungsmethoden, die Benutzerfreundlichkeit, die Robustheit, und die Erweiterbarkeit hin, untersucht.
Im Weiteren, haben die Autoren dann eine Gruppe von Software Entwicklungsmethoden beschrieben, die unter dem Begriff "Testgetriebene agile Entwicklungsmethoden" bekannt sind. Als Fallstudie, führten die Autoren daraufhin ein eigenes Projekt unter dem Namen Cancer (Krebs), Heart (Herzkreislauf) and Soft Tissue (Weichgewebe), kurz Chaste durch. Im Rahmen dieses Projekts wurde eine Library für biologische Berechnungsfunktionen unter Verwendung agiler Programmiermethoden entwickelt.
Mit Hilfe eines Reviews wurden dann die Vorteile und Nachteile des neuen Ansatzes, mit jenen Entwicklungsmethoden verglichen, die in herkömmlichen Paketen verwendet werden.
Die Autoren Schlussfolgern, dass zur besseren Berücksichtigung von Änderungsanforderungen, die verwendete Software-Kodierung (gemeint Programmierung), flexibler sein muss. Die Kodierung soll es ermöglchen, sowohl detaillierte mathematische Modelle, als auch effizientere numerische Verfahren einzubinden, die zur Zeit von vielen Forschungs-

gruppen weltweit entwickelt werden.

Die entscheidenden Herausforderungen für die Software von Herzaktivitäts -Modellierungen, liegt sowohl im Software Design, als auch in den verwendeten numerischen Algorithmen. Darüber hinaus müssen solche Softwarepakete auch robust, erweiterbar und verläßlich sein. Das gewählte Softwareentwicklungs- Paradigma stellt hierbei einen wichtigen Faktor, für die Erstellung hochperformanter Kardiologischer Software, dar.
Die Autoren stellten fest, dass es einige Widrigkeiten gibt, die der Einführung guter Software Engineering Praktiken im wissenschaftlichen Bereich entgegenstehen.
Ein typisches akademisches Szenario ist, dass bspw. eine einzelne Person den Code entwickelt der für ihren eigenen Forschungsbedarf notwendig ist. Nachdem der Forscher seine Arbeit in dieser Abteilung beendet, bleibt der Code ungenutzt oder er wird bestenfalls als Blackbox ohne jegliche Wartung weiter genutzt, da niemand den Code versteht. Das Hauptproblem, auch bei größeren Anwendungen an denen Teams von Forschern arbeiten, stellt die Wartbarkeit dar. Solange Jeder, seinen eigenen esoterischen Programmierstil verfolgt, oder seinen Code nicht dokumentiert, oder Jemand einen ganz speziellen Bereich des Codes bearbeitet und dann das Team verlässt, bleibt der erbrachte Beitrag undurchsichtig für zukünftige Entwickler-Generationen.

Um den genannten Herausforderungen in der Software Entwicklung von kardiologischen Modellen zu begegnen, setzten die Autoren das Projekt Chaste auf. Sie erkannten hierbei, dass sich ihre Bedürfnisse sehr gut mit agilen Methoden abdecken ließen. Sie entschieden sich deshalb dafür die agilen Techniken in einem vierwöchigen Trainingskurs anzuwenden. Da dies erfolgreich war, wurde daraus eine 3-jährige Anwendung agiler Methoden, bei der ein Set von wartbaren Bibliotheken entstand.
Chaste verwendete den XP Ansatz (siehe Abbildung: 3.2). Dabei wurden, folgende, agile Praktiken angewendet,

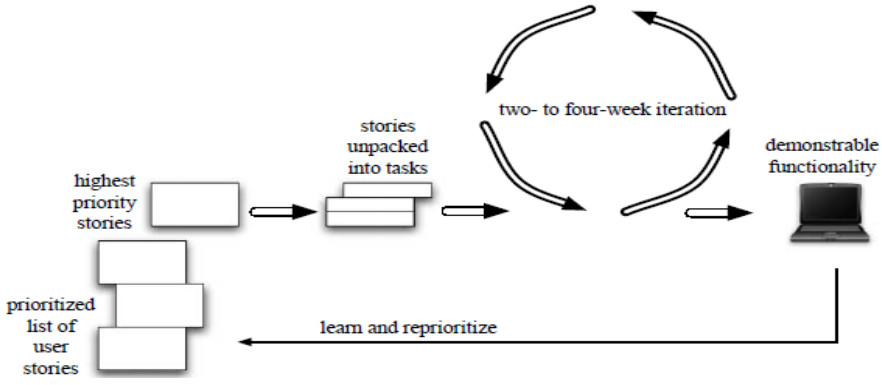

Abbildung 3.2: Extreme Programming

von denen allerdings einige, wenige Praktiken, auf das akademische Umfeld angepasst wurden (diese sind kommentiert):

- Customer and User stories
  Im Kontext des Chase Projekts waren die Forscher/Programmierer auch die Customer
- Release planning
  Die Releases wurde an die jeweilige Veröffentlichung gebunden.
- Iteration planning
- Accepatance tests
- Test-drive development
  Die Tests beinhalteten teilweise unbekannte Ergebnisse und numerische Toleranzen
- Refactoring
- Collective code ownership
  Teilweise wurde Code nur von einer Person gehalten
- Pair programming
  Unterschiedliche Ziele und unterschiedliche Einsatzorte führten zu einigen Code Stellen die nur von Spezialisten betrachtet wurden.
- Continous integration
- Stand-up meetings
- Whole team

Der wechselnde Personaleinsatz im akademischen Bereich, war ein Hauptgrund für den agilen Ansatz.
Der agile Ansatz trägt diesem Problem Rechnung, sowohl beim Weggang von erfahrenen Entwicklern, als auch bei der Integration neuer Team-Mitglieder.
Der pair programming Ansatz passt auch gut zu interdisziplinären Projekten, wobei Beide vom Wissen des jeweils Anderen profitieren. Dieser wechselseitige Effekt wird noch verstärkt, wenn das Team während der Coding-Sitzungen in gemeinsamen Büroräumen sitzt.
Eine solche Umgebung fördert die sogenannte “osmotische“ Kommunikation. Agile Ansätze basieren auf den Erkenntnissen, dass Projektanforderungen selten in Stein gemeißelt sind, sondern sich im Laufe der Zeit verändern. Dies ist besonders bei akademischen Forschungsprojekten der Fall, da es in der Natur der Forschung liegt, neue Richtungen zu verfolgen, die bislang nicht betrachtet wurden. Dies scheint ein Schlüssel für die Favorisierung agiler Methoden bei der Entwicklung akademischer Software zu sein.
Eines der wichtigsten Merkmale einer Programmier-Methodik, ist die Frage, ob sie es ermöglicht, schnell Code zu entwickeln. Auch dies trifft auf den akademischen Bereich zu. Den Erfahrungen der Autoren folgend, ist der Einsatz agiler Methoden hilfreich, um hochwertige wissenschaftliche Software zu entwickeln, die einfach an aktuelle Forschungsanforderungen angepasst werden kann und signifikante Vorteile mit sich bringt. Die Eigenschaften des Forschungsumfelds, die von einem agilen Entwicklungsprozess gut abgedeckt werden, lassen sich folgendermaßen zusammenfassen:

- eine große Fluktuation bei den Beteiligten mit kurz laufenden Arbeitsverträgen
- Ein weiter Bereich wissenschaftlicher Kenntnisse und Codierungserfahrungen
- Die Notwendigkeit einer wartbaren und erweiterbaren Codebasis
- Anforderungsänderungen, die durch neue wissenschaftliche Erkenntnisse ausgelöst werden

Schließlich zwei weitere Punkte, die von jedem beachtet werden sollten, der darüber nachdenkt, den agilen Ansatz für seine Software Projekte anzuwenden: Es ist notwendig eine gute Infrastruktur bereitzustellen. Außerdem sollte frühstmöglich geprüft werden, welchem Lizenztyp das Projekt unterliegt.
Da es sonst im weiteren Fortschritt des Projekts praktisch unmöglich ist, festzustellen welche Institutionen welche Rechte am Code haben.
Für Chaste wurde eine GNU LPGL Lizenz festgelegt, da diese im Besonderen den industriellen Partnern eine größere Flexibilität, als eine open source Distribution, erlaubt.

3.3 Paper 3 - Agile Methods in Biomedical Software Dewelopment: A Multi-Site Experience Report

Es handelt sich hierbei um eine systematische Untersuchung, zur Verwendung agiler Methoden in der biomedizinischen Softwareentwicklung.
Die Autoren stellen zunächst einige grundlegende Informationen zu agilen Methoden zusammen, gehen auf das Agile Manifest ein, und erläutern die Bedeutung iterativer Entwicklungsprozesse als Antwort auf sich ändernde Anforderungen. Agile Prinzipien unterstellen, dass sich die Anforderungen ändern und die Entwicklung und Lieferung von Software zu neuen Erkenntnissen führt.
Sie führen weiter aus, dass aus agiler Sichtweise das beste Feedback für die Softwareentwicklung vom Benutzer der Software kommt, der mit der Software interagiert. Hier haben agile Methodiken Gemeinsamkeiten mit älteren Methodiken wie Rapid Application Development (RAD), die ebenfalls eine schnelle Visualisierung von Software für den Benutzer anstreben. Im Unterschied zu diesen Methoden legen agile Ansätze jedoch Wert darauf, dass nicht nur Visualisierung entsteht, sondern sofort komplette Features entstehen, die zudem mit automatisierten Tests unterlegt sind. Im Gegensatz zu traditionellen Ansätzen, sind die Zyklen für die Software-Entwicklung üblicherweise nur noch wenige Wochen kurz.
Um die Untersuchung der Projekte durchführen zu können, begannen die Autoren damit, Organisationen zu identifizieren, die agile Methoden zur Entwicklung von Biomedizinischer Software einsetzen. Die Gruppen wurden im Rahmen informeller Treffen und Konferenzen identifiziert und in einer Tabelle zusammengefasst (siehe Abbildung: 3.3).

Organization	Type	Application(s) and Users	Team Size	Previous Approach
Applied Biosystems	Commercial	A custom workflow engine as a component to be used by developers of products.	Two developers, a part time project manager, and a customer.	Approach based on the Rational Unified Process (RUP)
Fred Hutchinson Cancer Research Center	Academic	A community tool to collect, analyze, report, and share genetic sequence data.	Four engineers for both developing this application and maintaining legacy systems.	Approach similar to the Rational Unified Process (RUP)
Memorial Sloan-Kettering Cancer Center	Academic	A freely available, open source cancer pathway database with a growing array of public users.	One scientific lead, and one architect/developer.	None. Agile-like practices used since inception of project.
National Cancer Institute	Government (supported by a commercial contractor)	A variety of tools to integrate and visualize integromic data set that are made available to the public.	Three engineers and a bioinformatics analyst.	No explicit process
Northwestern University Center for Functional Genomics	Academic	Two projects written at the Center, with users onsite and at two other institutions participating in a consortium.	Three to five developers, domain and quality assurance staff.	No explicit process
Vanderbilt University Medical Center	Academic	A clinical support application.	Three developers and additional quality assurance and configuration management support staff.	A plan driven development approach that emphasized extensive up front design

Abbildung 3.3: A Summary of the Organisations

Um die Situation jedes Projektes zu erfassen wurden die Erfahrungen jedes Projekts im Detail qualitativ erhoben. Dies war notwendig um interessante Themen, in Bezug auf die Anwendung agiler Methoden, identifizieren zu können.
Als weitere Massnahme wurde eine Liste mit offen formulierten Fragen mit den Teilnehmern durchgearbeitet, um Details herauszuarbeiten, die bei direkteren Fragen nur schwierig herauszuarbeiten wären.
Der Frageteil war dabei in drei Abschnitte gegliedert.
Der erste Teil befasste sich mit Fragen des Projektumfelds.
Im zweiten Teil wurde die historische Entwicklung der Praktiken des Projekts beleuchtet, und die Teilnehmer gebeten diese zu beschreiben. Sie sollten dann erklären, wie sich die Praktiken im Laufe der Zeit änderten.
Im dritten Teil wurden den Teilnehmern dann offenere Fragen gestellt, bspw. inwieweit die angenommenen Werte agiler Entwicklung mit ihren Projekterfahrungen übereinstimmen. Es wurde auch gefragt, ob nicht agile Praktiken angewendet wurden. Bei der Auswertung der Fragen wurde nach gemeinsamen Mustern der Projekte gesucht und nach berücksichtigungswürdigen individuellen Erfahrungen.

Practice\Organization	Applied Biosystems	Fred Hutchinson Cancer Research Center	Memorial Sloan-Kettering Cancer Center	National Cancer Institute	Northwestern University Center for Functional Genomics	Vanderbilt University Medical Center
Customer Collaboration						
Onsite Customer		•		•		
Automated Acceptance Testing						•
Use Cases	•		•			
User Stories	•	•	•	•	•	•
Planning						
Iterative Development	•	•	•	•	•	•
Feature/Product Backlog	•	•	•	•	•	•
Mid-range Planning		•	•	•	•	
Scrum Meeting	•			•	•	•
Developer Task Self-Selection	•	•		•	•	•
Building						
Automated Unit Tests	•	•	•	•	•	•
Collective Code Ownership	•	•		•	•	•
Continuous Integration	•	•	•	•	•	•
Nightly Builds & Tests	•	•	•		•	
Refactoring	•	•	•	•	•	•
Shared/Open Workspace/ Team Room	•	•	•	•	•	•
Adoption						
Sponsor	•	•	•	•	•	•
Champion	•	•	•	•	•	•

Abbildung 3.4: The commonality of key Practices and Attributes across the Projects

Die Ergebnisse in Abbildung 3.4 zeigen, dass die Projekte, über unterschiedliche Organisationen hinweg, viele Gemeinsamkeiten bezüglich des Einsatzes agiler Praktiken aufzeigen. Die Praktiken, die die Projekte anwenden, wurden detailliert in der Studie aufgelistet. Die Projekte wurden dann auf die Verwendung oder die Anpassung agiler Praktiken hin untersucht. Dies beinhaltete den Abgleich mit folgenden Praktiken:

- das Vorgehen beim Erfassen von Anforderungen
- die automatisierten Akzeptanztests
- gemeinsamen offenen Arbeitsräume
- die Projektplanung und Priorisierung
- die Iterationsplanung
- den Schätzprozess
- die Releaseplanung
- die Entscheidungsfindung bei der der Anwender einbezogen ist
- die Kodierungs Praktiken

- das aumomatisierte Testen
- das Refactoring
- das pair programming

Folgende Schlüsse konnten die Autoren aus ihrer Studie ziehen:

1. Agile Praktiken stellen eine entscheidende Veränderung zu Entwicklungsprozessen dar, die früher in den Organisationen verwendet wurden. Alle Organisationen erkannten wesentliche Unterschiede in den Gruppen nachdem diese agile Praktiken verwendeten. Dies wirkte sich auch, auf die Art der Zusammenarbeit der Teams mit den Fachleuten aus.

2. Agile Methoden sind wertvoll für Bioinformatik Projekte. Die untersuchten Projekte verfügten über eine gute Felderfahrung. Alle untersuchten Projekte berichteten über viele Vorteile, die sie aus agilen Entwicklungstechniken ziehen. Die Gruppen berichteten über eine verbesserte Qualität, höhere Flexibilität und Wartbarkeit. Die Entwickler fühlten sich wohler mit diesen Praktiken, weil dadurch das dynamische Anwendungsfeld handhabbarer wurde.

3. Auch wenn die untersuchten Projekte in unterschiedlicher Art von agilen Methoden und Praktiken Gebrauch machen, so ist doch allen die Anwendung nachfolgender Praktiken gemeinsam:
    - autmatisierte unit Tests
    - Continuous Integration
    - Feature Backlog
    - Refactoring
    - Open Workspace

Bei einigen Projekten konnten Praktiken erkannt werden, die nicht im Standard-Portfolio agiler Entwicklungspraktiken vorhanden sind. Diese wurden vor allem deshalb eingesetzt, da die vorhandenen agilen Praktiken nicht ausreichend waren, bspw. die Einbeziehung von internen und externen Gruppen oder von Mitwirkenden, die nicht bei der Entwicklungsgruppe vor Ort untergebracht waren. Die Autoren fassen abschließend zusammen, dass agile Methoden inzwischen signifikant in unterschiedlichen Anwendungsgebieten der Softwareentwicklung angenommen werden. Die in der Studie aufgezeigten Erfahrungen zeigen, dass viele Vorteile der Agiliät die in anderen Anwendungsbereichen festgestellt wurden, auf die Bioinformatik ausgedehnt werden können. Diese Methoden stimmen passgenau mit dem Forschungscharakter, dem iterativen Vorgehen, und der Art der Zusammenarbeit im wissenschaftlichen Umfeld überein.

3.4 Paper 4 - Exploring XP for Scientific Research

Die Autoren sind Forscher am NASA Langley Forschungszentrum und setzten XP für ein Pilot-Projekt im Anwendungsbereich der Luft- und Raumfahrtforschung ein. Auslöser hierfür war eine Ausschreibung des Langley Creative and Innovation Office (C&I), die zum Zweck hatte, festzustellen, ob durch Anwendung nicht traditioneller Entwicklungsmethoden, außerordentliche Produktivitätsgewinne erzielbar sind, oder völlig neuartige Applikationen entwickelt werden können.
Gegenstand des Projekts war die Entwicklung einer Software, die als Testfeld zur Performanz-Untersuchung von numerischen Verfahren genutzt werden kann. Bei den numerischen Verfahren handelt es sich um spezielle Runge Kutta Verfahren, die zur Lösung von Anfangswertproblemen in der Strömungslehre eingesetzt werden. Während der Entwicklung wurden das GNU/Linux Betriebssystem, die integrierte Entwicklungsumgebung Emacs, und Ruby als Programmiersprache verwendet.
Zu Beginn des Projekts mussten die Autoren erkennen, dass sie zuerst einige kulturelle Konflikte lösen mussten, um die 12 XP Praktiken überhaupt einsetzen zu können. Kent Beck (Vater von XP) listet neun Umstände auf, die nicht mit XP zusammenpassen, davon trafen sechs auf das kulturelle Umfeld des NASA Langley Forschungscenters zu. Einschränkend erwähnt Beck, dass seine Liste auf rein persönlichen Erfarhungen beruht: "Ich habe noch nie Software für Raketenköpfe entwickelt, ... wenn sie welche schreiben, müssen Sie selbst entscheiden ob Sie XP anwenden oder nicht". Die Software, die die Autoren entwickeln, wird eingesetzt um Vorhersagen zur Hitzeströmung an Spitzen von Körpern mit Übergeschwindigkeit (i.a. $>$ 3.000 m/sec) zu machen! An diesem Punkt entschieden sich die Autoren dafür, anhand der sechs kritischen Punkte zunächst zu untersuchen, ob XP in einem solchen Umfeld überhaupt funktionieren kann.

Punkt 1: Das nach Beck größte Hinderniss, ist die Beharrlichkeit, mit der auf ein komplettes Voraus-Design bestanden wird, anstatt einfach das Projekt auf den Weg zu bringen. So hat die NASA in 2002 bspw. 23,5 Millionen Dollar dafür ausgegeben, ihre Fähigkeiten zum Erstellen zuverlässiger Software zu erhöhen.
Daraufhin wurden zweiwöchige Trainings im SEI (Software Engineering Institute) durchgefuehrt, und sogenannte "Team and Personal Software Processes" TSP eingeführt, die mit einer Einführungsliteratur von 750 Seiten ausgestattet sind. Der TSP weist 2/3 der Projektzeit der Ermittlung der Anforderungen, Dokumentation, und dem Design zu. Die Kodierung ist bis zum letzten Drittel des Projekts nicht erlaubt. Die Autoren entscheiden sich dazu, diesen Standard Prozess nicht zu nutzen und XP exklusiv zu betreiben.

Punkt 2: Den zweiten Konflikt stellten die Autoren bezüglich des Vorgehens bei der Spezifikation fest. Die ISO 9001 Implementierung des Langley beinhaltet ein 45 seitiges Ablaufdiagramm zu Software-Qualitäts-Anforderungen und ein 17 seitiges Ablaufdiagramm zur Planung und Entwicklung, wovon sich nur 1 Kästchen (Tätigkeit) von 48 Kästchen, um die Kodierung und den Test kümmert, die 75% des Engpasses ausmachen. Die Autoren entschieden sich deshalb dafür, das Risiko in Kauf zu nehmen, nicht ISO konform zu sein.

Punkt 3: Beck stellte fest, dass Hervorragende Programmierer sich manchmal mit XP sehr schwer tun, weil sie dazu tendieren das “Ich habe Recht“-Spiel in der Kommunikation zu spielen, um ihren Experten Status zu beweisen.
Die Autoren mussten also ihre eigenen Bedürfnisse nach individueller Anerkennung unterdrücken und fest daran glaubten, dass zwei Personen gemeinsam mit XP produktiver sein würden, als die Summe ihrer individuellen Anstrengungen.

Punkt 4: Das zwei Mann Team schien für XP zu klein zu sein, galt es doch die Rollen Programmierer, Kunde, Protokollant und Coach zu besetzen. Die Autoren entschieden sich deshalb dafür, nur bei der Testgetriebenen Entwicklung neuer Funktionen pair programming einzusetzen und erlaubten auch Solo Refactorings.

Punkt 5: Da ein staatliches Forschungszentrums oft langfristige revolutionäre Vorhaben verfolgt, kann die Rückmeldung für ein Projekt manchmal Jahre dauern. Die Autoren konnten jedoch mit Hilfe der XP Design Praktik, ihre technischen Anforderungen in kleine Iterationsblöcke zerlegen.

Punkt 6: Beck warnt vor Seniors in Eckbüros (gemeint erfahrene Mitarbeiter), wegen den hieraus entstehenden Kommunikationsbarrieren.
Am Forschungszentrum haben Senior Engineers üblicherweise Einzelbüros und Kollegen sind über unterschiedliche Gebäude auf dem Campus verteilt. Für das Projekt machten die Autoren deshalb aus ihrem Büro eine gemeinsame Entwicklungsnische.

Die Autoren konnten dann die 12 XP Praktiken in ihrem Projekt einsetzen. Abbildung: 3.5 zeigt den Erfüllungsgrad und ihre Kommentare.

XP practice	Degree of adoption	Comments
Planning game	Full	We followed it by the book.[1]
Small releases	Full	Two-week iterations worked well for a project of this scope.
Metaphor	Full	We used a naive metaphor because both players spoke the same jargon.
Simple design	Full	We accepted this with skepticism, but it made future optimization easier than expected.
Test-driven development	Full	Comprehensive test coverage is the key to agility.
Refactoring	Full	This was integral to test-driven development; we followed *Refactoring: Improving the Design of Existing Code*,[6] often verbatim.
Pair programming	Full	We added new functionality only when working in pairs, improving source code readability.
Collective ownership	Full	We used CVS code control with no access restrictions.
Continuous integration	Full	We implemented four levels of automated tests with feedback ranging from seconds to hours.
Sustainable pace	Partial	Concurrent duties and nonoverlapping schedules made the pace difficult to sustain.
On-site customer	Partial	Our most difficult practice was very beneficial when implemented diligently. It can lead to loss of focus if not followed.
Coding standards	Full	Code had to be mutually understandable to both members of the pair. Clarity, over consistency, was the guideline.

Abbildung 3.5: Experience with XP Practices

Das Pilot-Projekt umfasste 2 Release-Zyklen. Ein Release-Zyklus war in 3 Iterationsschritten mit je 2 Wochen Dauer eingeteilt, was zu einer Gesamtprojektdauer von 12 Wochen führte.
Die gemessenen Arbeitsgeschwindigkeiten streuten zwischen doppelt so schnell als geschätzt und halb so schnell.
In der ersten Iteration 1.1 wurde die geschätzte Entwicklungszeit zur Umsetzung der Funktionen ungefähr erreicht.
In den beiden folgenden Iterationen 1.2 und 1.3 wurden die Aufgaben in der Hälfte der veranschlagten Zeit erledigt. Daraufhin wurde die höherere Arbeitsrate für die Iteration 2.1 bereits eingeplant, um die Geschwindigkeit wieder auf eins zu adjustieren. In Iteration 2.2 viel die Geschwindigkeit jedoch dramatisch auf die Hälfte ab. Dies führten die Autoren darauf zurück, dass sie selbstzufrieden und schlampig geworden waren. Die Aufgaben wurden im Planning Game nicht mehr gut genug festgelegt und die Zeit war nicht gut eingeschätzt. Der Prozess, während des Planning Games, die Anforderungen in einzelne Aufgaben zu zerlegen, ist ein Schlüsselbestandteil von XP. Allerdings konnten die Autoren bereits während der Iteration den Produktivitätsrückgang feststellen. Das Planning Game wurde für die letzte Iteration ernsthafter betrieben und die Produktivität verbesserte sich.
Es wurden insgesamt 2545 Zeilen Code produziert, dies entspricht 27 Zeilen code je Paar-Stunde. Es wurden im Schnitt alle 45 Minuten eine Methode mit zugehörigen 6 Testasserts entwickelt. Dies beinhaltet das Design, mit vollständiger Integration, Refactoring, Test und dem Debugging.
Aus vergleichbaren Projekten ohne XP konnten die Autoren bislang eine Produktivität von 12 Zeile Code je Programmierer je Stunde, bzw. 24 Zeilen für zwei Programmierer je

Stunde ermitteln, allerdings ohne getesteten Code!
Noch deutlicher für die Verwendung von XP spricht, dass mit XP ein viel leichter lesbarer Code entstanden ist, was sich auch durch die Tatsache ausdrückt, dass der gesamte Produktivcode nur 912 Zeilen umfasst, bzw. 120 Methoden. Für ein vorheriges nicht XP Projekt, das einen vergleichbaren Funktionsumfang abbildete, benötigten die Autoren noch 2144 Zeilen Code, das entspricht ungefähr dem doppelten Umfang des aktuellen Produktivcodes.
Abbildung: 3.6 zeigt die Produktivitätsergebnisse.

	Iteration						Total
	1.1	1.2	1.3	2.1	2.2	2.3	
Estimated hours	19	14	15	8	17	29	102
Actual hours	22	8	8	8	30	18	94
Velocity	1	2	2	1	0.5	1.5	1

Abbildung 3.6: Work effort fot two-weeks iterations over two releace cycles

Die Ergebnisse zeigen, dass der XP Ansatz ungefähr um das doppelte produktiver ist, als dies vergangene Projekte waren die die Autoren durchführten. Die eigentliche Funktionalität wurde ungefähr mit der selben Umsetzungsrate wie in der Vergangenheit implementiert. Allerdings wurden in gleichem Umfang Tests mitentwickelt, die in der historischen Produktivitätsrate nicht enthalten waren.
Außerdem umfasst der entwickelte Produktivcode nur die Hälfte der Zeilen und die Lesbarkeit ist deutlich besser. Kontinuierliches Refactoring, sich entfaltendes Design, und ständiges Codereview, wie sie von XP vorgegeben werden, sind die Hauptgründe für die verbesserte Ästhetik des Codes.

Die Evaluierung des Pilot-Projekts hat dazu beigetragen, dass XP im NASA Langley Forschungscenter für geschäftskritische Software Entwicklungen eingesetzt wird.
Aktuell werden XP Praktiken vom High-Energy Flow Solver Synthesis Projekt übernommen. Dieses Projekt wurde vor 3 Jahren begonnen und beschäftigt zwischen 10 und 15 Personen. Der HEFSS Code umfasst ca. 500.000 Zeilen Fortran90 Code. XP Techniken werden umfänglich eingesetzt um mit Ruby eine Vielzahl an Aufgaben im Bereich Wartung, Support und Tests zu lösen. Dies beinhaltet ein Fortran Templating, die automatische Code Generierung, ein verallgemeinertes Fortran Unit Test Framework, die laufende Regressionstest Automatisierung, das Release-, Installations-, und Konfigurations-Scripting.

3.5 Paper 5 - Introducting Agile Development into Bioinformatics: an Experience Report

In letzten Jahren stieg die Zahl der Krebserkrankungen weltweit deutlich an. Studien zufolge wird die Zahl der Neuerkrankten im Jahr 2030 bei 21,6 Millionen liegen, im Vergleich zu 14 Millionen Erkrankten im Jahr 2012. Das ist Grund dafür, dass der Einsatz von Software im Bereich der Krebsforschung immer wichtiger wird. Die Anforderungen an die Software sind sehr hoch. In vorliegenden Fall soll sie in der Lage sein möglichst früh potentielle Gen- oder Proteinveränderungen zu erkennen und präzise zu analysieren. Wichtig im Krebsforschungsbereich ist die Visualisierung der Ergebnisse und die Nachweisbarkeit dieser, durch mathematische und statistische Analysen.
Ein proof of concept wurde an der Genomics and Bioinformatics Group im Laboraty of Molecular Pharmacologie of the National Cancer Institut in Bethesda, Maryland, zusammen mit einem SRA International Softwareentwicklerteam durchgeführt. Ziel war es zu überprüfen, ob die Entwicklung von Softwaretools, zur Auswertung, zur Analyse und zur Visualisierung von Genen und Proteinen, möglich wäre. Beim Erfolg der Studie sollte die vollständige Entwicklung der Tools folgen.
GoMiner und MatchMiner sind Tools, mit denen komplexe Analysen von menschlichen Genen und Proteinen durchgeführt werden können. Die Komplexität liegt darin, dass im menschlichen Körper zwischen 30.000 und 60.000 Gene vorkommen und jedes von diesen Genen verschiedene mRNA Strukturen produzieren kann. Die verschiedenen mRNA Strukturen repräsentieren verschedene Proteine. Eine Manipulation der mRNA kann zu Veränderungen in den Proteinen führen. Diese Studie wurde ausgewählt, da es ein Beispiel dafür ist, wie agile Vorgehenstechniken in den Arbeitsablauf von wissenschaftlichen Softwareentwicklern integriert werden können.

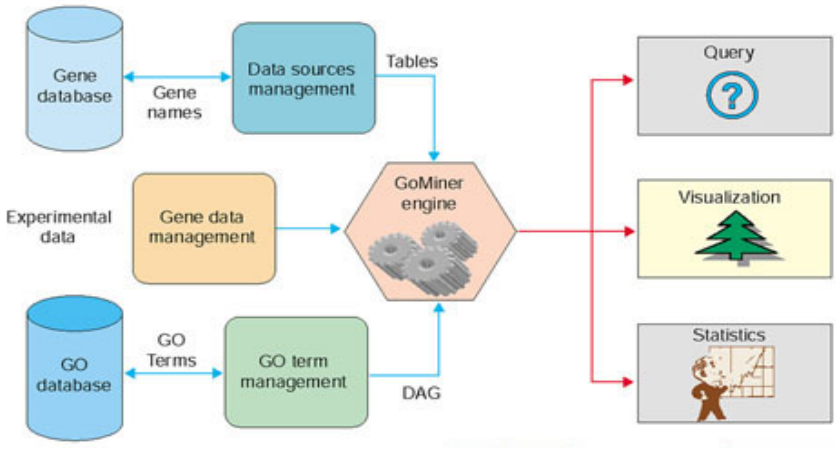

Abbildung 3.7: Schematic of GoMiner architecture and data flow

Das Projekt startete im Oktober 2000 mit einem Team von SRA Softwareentwicklern. Zuerst wurde ein proof of concept für ein neues Visualisierungstool durchgeführt.

Der Kunde wollte zudem zwei Visualisierungstechniken SVG und Sportfire untersuchen lassen. Für die Entwicklung wurde als Programmiersprache Java ausgewählt, da diese für die eingesetzten Betriebs- und Entwicklungssysteme geeignet ist und die Integration von bereits mit PERL und S PLUS entwickelten Applikationen unterstützt.
Die im Projekt angewendeten Praktiken, wurden vom Kunden übernommen. Es gab eine allgemeine Vorstellung zu dem neuen Tool, aber die Anforderungen wurden erst untersucht als das Tool entwickelt wurde. Die frühen Fortschritte waren durch viele Vorfuehrungen des Tools gekennzeichnet. Allerdings benötigten die jeweiligen Vorbereitungen aufgrund mangelnder Konfigurations- und Integrationspraktiken sehr viel Zeit.
Nach der erfolgreichen Durchfuehrung des proof of concept wurde entschieden, dass für die vollständige Ausarbeitung des Projekts agile Praktiken eingeführt werden. Dieses konnte nur Schrittweise passieren, da das Vorankommen des Projekts im Vordergrund stand. Im Zeitraum von 2001 bis 2003 wurde eine Kombination aus XP und Scrum Praktiken umgesetzt. Als erstes wurde die Möglichkeit zur direkten Kommunikation im Team und zur Einbeziehung des Kunden geschaffen, in dem ein gemeinsames Büro als Arbeitsraum eingerichtet wurden.
Für die Erleichterung der gemeinsamen Arbeit wurden im August 2001 Tools integriert, mit denen die Versionsverwaltung von Dateien (CVS) und der automatisierte Build-Prozess (Apache ANT) gesteuert wurde. Hinzu kam im September 2001 die Einführung von Coding-Standards, die die Qualität, die Verständlichkeit und Wartbarkeit von Software unterstützen sollten.
Zur Verbesserung der Teamzugehörigkeit wurde Collective Ownership angewendet somit hatte jeder Teammitglied im Projekt die gleiche Rechte Veränderungen am Code vorzunehmen, Fehler zu beheben, Refactoring durchzuführen und Funktionalität hinzuzufügen.
Im Bereich der Krebsforschung der Einsatz muss die Software besonders intensiv auf Richtigkeit geprüft werden, deswegen wurden im Projekt im Oktober 2001 automatische Tests wie JUnit und MaxQ eingebunden. Im nächsten Schritt wurde ein Refactoring mit Hilfe der Refactoringtools (built-in und IntelliJ) durchgeführt. Im März 2002 wurden die Praktiken “The Planning Game“ und “Scrum Meetings“ vom Team eingeführt. Für die die Steigerung der Softwarequalität wurde die kontinuierlichen Integration mittels Cruise Control tool durchgeführt. Als letztes wurden im Januar 2003 das Pair Programming und die Code Reviews übernommen.
Die wissenschaftlichen Softwareentwickler haben nur wenig Erfahrung mit der allgemeinen Softwareentwicklung. Sie haben viel mehr Kenntnisse in ihren Spezialbereichen der Krebsforschung und deswegen stehen sie den Einsatz bereits vorhandener Softwareengeneering tools, kritisch gegenüber. Durch die Zusammenarbeit mit dem SRA International Team haben sie einen Überblick über die allgemein angewendeten Softwaretechniken und agilen Praktiken bekommen. Die schrittweise Einführung der agilen Praktiken im Projekt führt dazu, dass der Sinn der Praktiken besser verstanden wird. Die geschickte Abfolge der Techniken motiviert die Team-Mitglieder weitere Praktiken zu integrieren. Es werden viele agile Praktiken angewendet, die Einbeziehung des Kunden, der gemeinsame Arbeitsraum, gemeinsame Ownership, continous integration, Coding Standards, Pair Programming usw. Es wurde aber keine agiles Prozess-Modell eingeführt. Es wird erneut bestätigt, dass die agile Entwicklung eine Anzahl best practices beinhaltet, die von Entwicklern

angewendet wird, da sie in ihrem Arbeitsumfeld gut funktionieren. Allerdings wird auch hier kein bestimmtes Prozessmodell wie XP oder Scrum angewendet.
Als Hypothese kann man unterstellen, dass im Kontext der wissenschaftlichen Softwareentwicklung die agilen Praktiken so umgesetzt werden, dass ein hochgradig spezifisches und an den Anforderungen des Forschungsbereichs angepasstes Prozessmodell entstehen wird.

4 Fazit

Nach kritischer Betrachtung der Softwareentwicklungsprojekte in den Forschungsbereichen der Bioinformatik, der Klimaforschung sowie der Luft- und Raumfahrttechnik konnte gezeigt werden, dass agile Ansätze für die Entwicklung von Software im wissenschaftlichen Umfeld sehr gut eingesetzt werden können.
In der Natur der Forschung liegt es, neue Richtungen zu verfolgen, die bislang nicht betrachtet wurden. Dies bringt neue Projekte mit sich, die sehr schnell erste Ergebnisse liefern müssen, um den Forschern Rückmeldung über den eingeschlagenen Weg zu liefern. Andererseits ändern sich durch neue Erkenntnisse häufig Anforderungen für bereits in Arbeit befindliche Projekte, was es erforderlich macht, dass die Prozessmodelle denen die Projekte folgen, in der Lage sind mit diesen Änderungen umzugehen, ohne dass die Vorhaben davon bezüglich ihrer Lieferung von Ergebnissen in zeitlicher und inhaltlicher Hinsicht stark beeinflusst werden. Die Arbeiten haben gezeigt, dass agile Praktiken für beide Herausforderungen gute Antworten liefern. Da bei Forschungsvorhaben ebenso die Transparenz und das Aufsetzen auf bereits vorhandenes Wissen und die vorhandene Software eine immer wichtigere Rolle spielt, werden auch die Software Engineering Techniken immer wichtiger um hochwertigen, wartbaren Code zu entwickeln. Dies haben die Wissenschaftler erkannt und versuchen deshalb Verfahren und Techniken zu adaptieren, die ihnen ermöglichen einen gemeinsamen Codebereich für ihre Anwendungen einzurichten, von dem auch Andere profitieren können. Gleichzeitig erhöhen sie damit auch die Chancen ihre Software um neue Funktionen zu erweitern, die sie nicht selbst erstellen und warten müssen. Die Anwendung agiler Praktiken trägt zu einem besseren Sourcecode und zu besserem Testen bei. Auch hier zeigen die Arbeiten, dass Zuwächse an Produktivität und Qualität erzielbar sind, die ohne agile Praktiken bislang nicht erzielbar waren.
Allerdings wird aus den Arbeiten auch deutlich, dass die Wissenschaftler, bezüglich der Adaption der agilen Praktiken auf ihr Aufgabenfeld, fast immer eigene Prozesse, unter Verwendung agiler Praktiken entwickeln, die hochgradig auf ihr Umfeld und ihre Bedürfnisse zugeschnitten sind. Dass dies einfach möglich ist, kann ebenso als Pluspunkt der Agilität angesehen werden.
In den Projekten wurde auch deutlich, dass zunächst mit überschaubaren Aufgabenstellungen begonnen wurde, und relativ schnell Erfahrungen aufgebaut werden konnten. So konnten agile Praktiken erst in nicht kritischen Bereichen der jeweiligen Forschungsgebiete etabliert werden, um anschließend in größerem Maßstab auch in den Kernbereichen eingesetzt zu werden.

5 Literaturverzeichnis

- Magnus Sletholt; Jo Hannay; Dietmar Pfahl; Hans Benestad; Hans Langtangen
"*A Literature Review of Agile Practices and Their Effects in Scientific Software Development*"

- Steve M. Easterbrook; Timothy C. Johns
"*Engineering the Software for Understanding Climate Change*"

- Joe Pitt-Francis; Miguel O Bernabeu; Jonathan Cooper; Alan Garny and Co.
"*Chaste: using agile programming techniques to develop computational biology software*"

- David W. Kane; Moses m. Hohman; Ethan G. Cerami and Co.
"*Agile methods in biomedical software development: a multi-site experience report*"

- William A. Wood; William L. Kleb
"*Exploring XP for Scientific Research*"

- Davide Kane
"*Introducting Agile Development into Bioinformatics: An Experience Report*"

- Leffingwell, Dean
"*Agile Software Requirements*"

- it-agile Die Experten für agile Softwareentwicklung
`www.it-agile.de`

- Die Zeit
`www.zeit.de/wissen/gesundheit/2014-02/who-studie-krebserkrankungen-weltweit`