Horst Lichter, Martin Glinz (Hrsg.) (2001): Software Engineering im Unterricht der Hochschulen
SEUH 7 – Zürich, dpunkt.verlag, Heidelberg.

# XP lehren und lernen

*Martin Lippert, Stefan Roock, Henning Wolf, Heinz Züllighoven*
Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik
Email: {lippert | roock | wolf | zuelligh@informatik.uni-hamburg.de}
http://swt-www.informatik.uni-hamburg.de

## Zusammenfassung

*Der Arbeitsbereich Softwaretechnik der Universität Hamburg führt einmal im Jahr ein Grundstudiumspraktikum (GSP) durch, in dem Java, Objektorientierung und softwaretechnische Grundprinzipien vermittelt werden. In diesem Jahr haben wir das Vorgehen im GSP stark modifiziert, um den Anteil softwaretechnischer Grundprinzipien zu erhöhen. Dafür haben wir das GSP als XP-Projekt aufgesetzt.*

*Dieser Artikel beschreibt das Vorgehen im GSP und die daraus gezogenen Lehren, die sich unserer Meinung nach auch auf die Weiterbildung und Vermittlung von XP im kommerziellen Umfeld übertragen lassen. Das Vorgehen wird jeweils aus Sicht der Betreuer der jeweiligen Gruppe beschrieben.*

## 1 eXtreme Programming

Das Praktikum sollte neben praktischen Erfahrungen mit softwaretechnischen Konzepten und der Programmiersprache Java die Grundzüge des eXtreme Programming (XP) vermitteln. Wir skizzieren an dieser Stelle sehr knapp, welche Konzepte wir dabei berücksichtigt haben (mehr zu XP in [Beck00]).

- Story-Cards, Kundenrolle: In XP werden Karteikarten verwendet, um Kundenanforderungen auf Ebene einzelner Features aufzuschreiben. Die Karten werden dann vom Kunden priorisiert und von den Programmierern umgesetzt. Dabei wird nur das an Änderungen und Erweiterungen erledigt, was für dieses Feature gerade benötigt wird. (siehe auch KISS und YAGNI).
- Engineering-Cards: Neben Story-Cards werden zu erledigende Umstellungen (sog. Refactorings) auf Engineering-Cards vermerkt, so dass die Aufgaben nicht vergessen werden, aber nicht sofort erledigt werden müssen.
- Refactoring: Auf Grundlage der Engineering-Cards soll das bereits lauffähige Programmsystem durch Refactoring im softwaretechnischen Sinne gesäubert werden. Voraussetzung für einfaches Refactoring sind Testfälle, da so sehr leicht überprüft werden kann, ob das System seine Funktionalität noch erbringt. Didak-

Horst Lichter, Martin Glinz (Hrsg.) (2001): Software Engineering im Unterricht der Hochschulen
SEUH 7 – Zürich, dpunkt.verlag, Heidelberg.



tisch setzen wir das Refactoring auch ein, um dem weit verbreiteten „never change a running system“ entgegenzuwirken.

- Integrationsrechner / Continuous Integration: Die Entwickler sollen die verschiedenen Ergebnisse so häufig wie möglich zu einem lauffähigen System integrieren, um größere Integrationsprobleme am Ende von mehreren Tagen Programmierarbeit zu vermeiden und möglichst die ganze Zeit auf dem aktuellen Stand des Systems zu arbeiten. Dies haben wir mehrfach täglich mit den 8 parallel arbeitenden Gruppen gemacht. Dabei wurde ein im Arbeitsbereich entwickelter Integrationsrechner benutzt.
- Testklassen: Für jede Klasse soll es auf der Ebene von Unit-Tests eine Testklasse geben. Als Test-Framework haben wir JUnit (siehe [junit]) verwendet.
- „Keep it simple stupid“ (KISS) und „You aren’t gonna need it“ (YAGNI) sind zwei XP-Konzepte, die verhindern sollen, dass die praktisch entwickelten Programme unnötig kompliziert werden. Dies passiert häufig, weil Entwickler gerne Änderungen antizipieren, die potenziell auftreten können. Es soll aber genau das realisiert werden, was auf der Story-Card steht und nicht mehr.
- Pair-Programming: Jeweils zwei Entwickler programmieren gleichzeitig an einem Rechner. Derjenige, der gerade die Tastatur hat, erläutert ständig was er gerade warum macht und der zweite beurteilt und kommentiert dies im Sinne eines ständigen Reviews. Dabei sollte häufig (mehrmals pro Stunde) die Tastatur gewechselt werden. Wir wollten den Studenten erfahrbar machen, dass dieses Entwickeln in Paaren große Vorteile bringt.

## 2 Das Grundstudiumspraktikum

Ein GSP ist in Hamburg verbindlicher Teil des Grundstudiums und wird im wesentlichen von Studierenden im dritten Semester besucht. Diese haben Lehrveranstaltungen über Konzepte der funktionalen und logischen sowie der objektorientierten Programmierung mit Java besucht.

Die Erfahrungen zeigen, dass die Studierenden nach den ersten beiden Semestern einen sehr unterschiedlichen Kenntnisstand zu Java und Objektorientierung mitbringen. Daher kombinieren wir in unserem GSP softwaretechnische Lernziele mit einer Vertiefung in den Bereichen Java und Objektorientierung.

Zum GSP wurden 32 Studierende zugelassen, die in zwei Gruppe à 16 Studierende aufgeteilt wurden. Die eine Gruppe wurde von einem wissenschaftlichen Mitarbeiter die andere von zwei Mitarbeitern im Wechsel betreut. Das GSP dauerte insgesamt fünf Tage (Montag bis Freitag, jeweils 9 bis 17 Uhr) und entspricht damit einer Lehrveranstaltung mit 4 Semesterwochenstunden.

In den vorangegangenen Jahren war jeder Tag des GSP in einen Plenumsteil und einen Übungsteil aufgeteilt. Morgens wurde ein Vortrag (1-2 Stunden) zu einem Thema (z.B. Swing) gehalten. Anschließend haben die Studierenden feste Paare ge-

bildet, die dann festgelegte Aufgaben übernommen haben. Wenn die Aufgaben für den jeweiligen Tag erledigt waren, konnten die Studierenden gehen.

In diesem Semester erhielt jede Gruppe (à 16 Studierende) eine gemeinsame Projektaufgabe: Die Studierenden sollten ein Multi-User-Dungeon programmieren. Dazu musste ein Labyrinth programmiert werden, in dem sich der Spieler bewegen und Geldsäcke einsammeln kann. Ausbaustufen betreffen vom Computer gesteuerte Spieler, Multi-User-Fähigkeit etc.

Da im GSP die XP-Praktiken und die Arbeit im Team vermittelt werden sollten, mussten die Studierenden die Paare möglichst häufig wechseln (mind. täglich). Plenumsvorträge aus dem traditionellen GSP wurden ersetzt durch das Angebot der Betreuer, situativ zu beliebigen Themen Vorträge zu halten. Wir machten deutlich, dass Vorträge nur dann gehalten werden, wenn dies von den Studierenden selbst gefordert wird. Dazu waren die Betreuer ständig mit Notebooks und Beamern ausgerüstet.

Das GSP wurde von Martin Lippert, Stefan Roock und Henning Wolf betreut. Alle drei haben langjährige Praxis in Entwicklungsprojekten mit Java und XP (siehe auch [Lip00]). Außerdem haben sie reichhaltige Vortragserfahrung in Lehrveranstaltungen sowie kommerziellen Schulungen und Beratungen.

## 3 Bericht über das Grundstudiumspraktikum

Im folgenden fassen wir die Aufzeichnungen der Betreuer zusammen, um einen Eindruck von der Vorgehensweise und den anstehenden Problemen zu geben.

### Montag

Wir begannen den Montag mit einem Kurzvortrag (ca. 1 Stunde) zum Thema XP. Während des Vortrages waren alle 32 Studenten anwesend. Nach dem Vortrag wurden die Studenten auf zwei Gruppen zu je 16 Leuten aufgeteilt. Dabei haben wir jeweils jeden zweiten auf der Anmeldungsliste einer Gruppe zugeordnet, um möglichst häufig „bewährte“ Studier-Paare in unterschiedliche Gruppen zu platzieren. Es hat sich gezeigt, dass der Vortrag und die anschließende Betreuung ausreichte, um die XP-Techniken deutlich zu machen.

Die Betreuer agierten in den beiden Gruppen unterschiedlich -– ein Betreuer schreibt die Anforderungen an das Spiel auf Story-Cards und machte klar, dass niemand sonst entscheidet, welche Features in das Programm sollen und auf welche verzichtet wird. Der andere Betreuer gab nur allgemeine Vorstellungen über das Spiel vor und überließ die Detaillierung und den weiteren Prozess der Gruppendiskussion.

In der ersten Gruppen gingen die Studierenden recht zielgerichtet ans Werk. Sie erarbeiteten an der Tafel gemeinsam einen Entwurf. Der Entwurf hatte am Ende 5 Klassen, die im Kern der Anwendung auch bis zum Ende der Praktikums erhalten blieben: Dungeon, Raum, Geldsack, Spieler, Controller (für Ein-/Ausgabe).

Der Betreuer der ersten Gruppe schrieb systematisch Anforderungen und Aufgaben auf Story Cards und Engineering-Card. Die Paare orientierten sich an diesen Karten und fingen an, arbeitsteilig entsprechende Klasse plus Testklasse zu schreiben.

Da der Betreuer der zweiten Gruppe keine Story Cards geschrieben hatte, nahm die Gruppendiskussion über die Systemanforderungen und den ersten Entwurf viel Zeit in Anspruch und verlief recht chaotisch. Dennoch kristallisierten sich Teilaufgaben heraus, die dann arbeitsteilig von den einzelnen Paaren bearbeitet wurden.

Die erste Gruppe arbeitete nach kurzer Zeit mit einem Integrationsrechner, während die zweite Gruppe noch nicht dessen Sinn sah.

Die Feedback-Runden am Ende des ersten Tags brachten erwartungsgemäß unterschiedliche Ergebnisse. Übereinstimmend positiv wurde das Pair-Programming bewertet. Auch fiel den Studierenden auf, dass sie sehr viel miteinander reden mussten, um die Schnittstellen der Klassen abzustimmen. Die zweite Gruppe war aber mit der Selbstorganisation und dem Diskussionsprozess unzufrieden. Sie beschlossen, sich am nächsten Tag auf die erste lauffähige Version des Spiels zu konzentrieren und disziplinierter vorzugehen.

### Dienstag

Die erste Gruppe entwickelte auf der Basis der existierenden Story- und Engineering-Cards weiter. Gegen Mittag lief die erste spielfähige Version des Programms auf dem Integrationsrechner. Die Integration der Klassen war reibungslos, wozu die Testklassen ganz wesentlich beigetragen haben.

Die zweite Gruppe begann mit einer Gruppendiskussion, die jetzt deutlich zielgerichteter lief. Alle waren sehr interessiert, rasch eine lauffähige Version zu entwickeln. Als Organisationsprinzip wurden täglich zwei Gruppendiskussion vereinbart, die von da ab recht konzentriert und strukturiert waren. Allerdings gelang es den Gruppen am zweiten Tag nicht, eine lauffähige Version zu integrieren. Dazu trugen auch die fehlenden Testklassen bei.

Das Pair-Programming mit wechselnden Paaren war in der ersten Gruppen fest etabliert, während sich in der zweiten Gruppe oft drei oder vier Entwickler um einen Rechner scharten.

In beiden Gruppen bestand die Tendenz, Technologie auf Vorrat in die Entwürfe einzubauen. Die Betreuer erläuterten noch einmal das KISS- und das YAGNI-Prinzip. Allerdings wurden die Rollen der Betreuer als „Kunden“ und als „OO-Experten“ nur wenig in Anspruch genommen. Einzelne Technologievorträge wurden aber gezielt angefordert.

In den Feedback-Runden wurde klar, dass die Studierenden der ersten Gruppe den Überblick über das Projekt verloren hatten. Jedes Paar wusste zwar, was es als nächstes zu tun hatte. Vielen war aber unklar, was die anderen Paare taten. Die Studierenden beschlossen daher, im weiteren immer mit einem aktuellen Entwurf an der Tafel zu arbeiten. Ähnliche Probleme hatte die zweite Gruppe nicht, da sie sich über

Horst Lichter, Martin Glinz (Hrsg.) (2001): Software Engineering im Unterricht der Hochschulen
SEUH 7 – Zürich, dpunkt.verlag, Heidelberg.



die Gruppendiskussion und die Tafel bereits ausreichend koordiniert hatten. Die zweite Gruppe stellte fest, dass sie am zweiten Tag deutlich produktiver gearbeitet hatte.

### Mittwoch

Gruppe 1 entwickelte ein gemeinsames Tafelbild mit den einzelnen Komponenten, so dass danach jeder wieder einen Überblick über das Gesamtsystem hatte. Dazu wurde an der Tafel eine Liste mit allen Rechnern gemacht, in die jeweils eingetragen wurde, welches Paar gerade an welcher Aufgabe arbeitete. Diese Liste wurde im weiteren als sehr sinnvoll bewertet.

In der zweiten Gruppe lief die koordinierende Gruppendiskussion besser, da jetzt ein Moderator bestimmt wurde. Es gelang erst gegen Mittag, eine ersten lauffähige Version des Spiels zu integrieren. Wieder erwiesen sich die fehlenden Testklassen als Problem.

Vormittags wurde vom Betreuer der Gruppe 1 ein Code-Review an Kernklassen des Systems durchgeführt. Es zeigte sich, dass die Studierenden nicht selbst in der Lage waren, den Code anderer zu bewerten. Sie konnten jedoch Nutzen aus den Hinweisen des Betreuers ziehen. Sie haben dann selbst entschieden, welche Refactorings sich vom Aufwand her im GSP lohnen und welche nicht.

Beide Gruppen hatten Schwierigkeiten, die Anforderungen an den Multi-User-Betrieb des Spiels zu realisieren. Die entsprechenden Vorträge über RMI wurden nur von wenigen Studierenden verstanden.

In der Feedback-Runde zeigte sich die Gruppe 1 weitgehend zufrieden. Die zweite Gruppe war mit dem Projektfortschritt unzufrieden und führte dies auf die noch nicht reibungsfreien Diskussionen und Koordinationsaufwände zurück. Außerdem bemängelten sie eigene technische Wissenslücken.

### Donnerstag

Gruppe 1 arbeitete parallel an zwei Versionen, der Einzelplatzversion mit weiteren Ausbaustufen und der Mehrbenutzerversion, die von einer RMI-Task-Force erstellt wurde. Die zweite Gruppe konzentrierte sich im wesentlichen auf die Einzelplatzverion.

In beiden Gruppen ist die Arbeit mit den Story- und Engineering-Cards mittlerweile etabliert. Auch die zweite Gruppe hatte ihre Bedeutung erkannt.

Der Integrationsrechner wurde nun ebenfalls von beiden Gruppen benutzt. Die zweite Gruppe merkte immer schmerzlicher, dass Testklassen fehlten. Da deshalb nicht regelmäßig integriert wurde, verlor eine Paar seine Änderungen, was zu einem emotional heftigen Lerneffekt führte.

Die Studierenden der ersten Gruppe erklärten während der Feedback-Runde, dass sie selbst überrascht seien wie weit sie schon gekommen waren. Die zweite Gruppe betonte den hohen Lerneffekt und die Tatsache, dass einige softwaretechnische Prin-

zipien und die Techniken des XP durch eigene Erfahrung als sinnvoll erkannt wurden.

### Freitag

Am Vormittag konnte die erste Gruppe alle noch offenen Programmieraufgaben abarbeiten. Während die Präsentation von einigen vorbereitet wurde, feilten andere an zusätzlichen Features oder versuchten, die Multi-User-Version doch noch ans Laufen zu bringen. Insgesamt war die Stimmung eher euphorisch.

Die zweite Gruppe war vormittags hektisch mit Abschlussarbeiten beschäftigt. Die Koordination zwischen diesen Arbeiten und der Vorbereitung der Präsentation war schlecht, was der insgesamt guten Stimmung keinen Abbruch tat.

Die gemeinsame Präsentation der beiden Gruppenergebnisse war für bei Gruppen der Höhepunkt der Praktikumswoche. Präsentiert wurden die drei Teile: das laufende Programm, die Architektur, der Entwicklungsprozess.

Bei der Präsentation zeigte sich, dass beide Gruppen das Entwicklungsziel erreicht haben: Sie können ein lauffähiges System vorführen. Das Ergebnis der ersten Gruppe war dabei etwas ausgereifter.

Bei der Präsentation wurde deutlich, dass eine starke Konkurrenzsituation zwischen den beiden Gruppen entstanden war. Die Diskussion über Ergebnis und Prozess wurde zeitweise sehr emotional zwischen den Studierenden geführt. Für die Betreuer wurde deutlich, dass in beiden Gruppen eine gemeinsame zielorientierte Projektkultur entstanden ist.

## 4 Bewertung der XP-Elements

### Testklassen

Nur Gruppe 1 hat ausgiebig Gebrauch von Testklassen gemacht. Dies hat sich sehr positiv auf die Integrationen und die wenigen Refactorings ausgewirkt.

### Continuous Integration

Der explizite Integrationsrechners hat sich positiv ausgewirkt, da die Studierenden einen expliziten Schnitt zwischen „Entwickeln“ und „Integrieren“ machen mussten.

### KISS + YAGNI

Auf diese beiden Prinzipien musste immer wieder hingewiesen werden. Insbesondere das Wissen um noch kommende Anforderungen verleitete die Studierenden dazu, „Technologie auf Vorrat“ zu konstruieren.

Aufgrund der mangelnden Erfahrung gab es zwischen Studierenden und Betreuern nicht immer sofort einen Konsens darüber, was „einfach“ bedeutet. So hatten Teile der Studierenden in beiden Gruppen zunächst die Idee, Räume, Geldsäcke und

Spieler in assoziativen Arrays zu repräsentieren. Hier haben die Betreuer deutlich darauf hingewiesen, dass diese Lösungen „zu einfach“ seinen. Am Ende des GSP hatte sich aber in Ansätzen eine gemeinsame Vorstellung darüber etabliert, was „einfach“ bedeutet.

### Story-Cards + Engineering-Cards

Die explizite und sehr strikt gespielte Kundenrolle in Gruppe 1 hat dieser Gruppe einen klaren Rahmen und eine deutliche Zielvorstellung vermittelt.

Gruppe 2 hat ihre Story-Cards im wesentlichen selbst geschrieben und viel Zeit mit der Diskussion der zu schreibenden Story-Cards verbraucht. Außerdem war in Gruppe 2 die Tendenz erkennbar, Anforderungen zu definieren, welche sie Studierenden gerne umsetzen würden.

Die Koordination über Story- und Engineering-Cards lief in beiden Gruppen sehr gut.

### Pair-Programming

Das Pair-Programming mit wechselnden Paaren hat in beiden Gruppen mit wenigen Ausnahmen sehr gut funktioniert. Es hat sich deutlich gezeigt, dass Pair-Programming ein mächtiges Mittel ist, um Wissen zwischen den Studierenden zu transferieren. So konnten sie sich häufig selber helfen.

### Refactoring

In Gruppe 1 war das Code-Review die Ausgangsbasis für die durchgeführten Refactorings. Refactoring hat im GSP jedoch eher eine untergeordnete Rolle gespielt. Dies lag nach Einschätzung der Betreuer daran, dass das Projekt insgesamt eher klein war. Daher gab es keine wirkliche Notwendigkeit zum Refactoring. Außerdem fehlte den Studierenden die Programmiererfahrung, um wirklich effektiv Refactoring umsetzen zu können.

## 5 Gesamtbewertung

Die Auswertung einer Umfrage erbrachte, dass die Studierenden das GSP außerordentlich positiv bewerteten. Es wurde vielfach sogar gefordert, dass das GSP auf mindestens zwei Wochen auszudehnen.

Es zeigte sich, dass relativ große Projekte (mit 16 Personen) funktionieren können, wenn konsequent Pair-Programming mit wechselnden Paaren und Collective Code Ownership angewendet wird. Nach Einschätzung der Betreuer wären vier Gruppen à 8 Leuten wahrscheinlich effektiver gewesen, weil die Koordination viel einfacher gewesen wäre. Allerdings hatten die Studierenden in den großen Gruppen die Gelegenheit, eine Projektgröße im Grenzbereich mit den zugehörigen Problemen zu erleben.

Nach Einschätzung der Betreuer haben die Studierenden weniger Technologien (Swing, Threads, Exceptions, RMI) gelernt, als dies im traditionellen Praktikum der Fall war. Allerdings haben sie deutlich mehr in den Bereichen XP, Softwaretechnik, Architektur, Projektkultur und Entwicklungsprozess gelernt. Die Studierenden hatten letztlich aber einen größeren Lerneffekt, der sie in die Lage versetzen sollte, sich die noch fehlenden Technologien selbst anzueignen.

Das potenziell größte Problem dieser Form der Lehrveranstaltung dürfte die Qualifikation der Betreuer sein. Diese müssen sich nicht nur sehr gut mit Java und der verwendeten Programmierumgebung auskennen, sondern selbst einen breiten Erfahrungshintergrund zu XP und Entwicklungsprojekten haben. Sie müssen im Notfall sogar Projektleitungskompetenzen mitbringen, um ein kippendes Projekt wieder zu stabilisieren. Ein gescheitertes Projekt bietet sicher viel Stoff zum Lernen. Allerdings wäre dafür dann keine Zeit mehr. Daher sollte das Projekt erfolgreich sein.

Dazu kommt, dass die Betreuer in der Lage sein müssen, im Prinzip beliebige Vorträge situativ zu halten, ohne sich vorher noch darauf vorbereiten zu müssen. Die funktioniert nur, wenn die Betreuer die potenziell in Frage kommenden Vorträge bereits mehrfach gehalten haben.

## Danksagung

Wir möchten an dieser Stelle den Studierenden danken, die durch ihre engagierte Teilnahme an dem Praktikum ganz wesentlich zu diesem Text beigetragen haben.

### Literatur

[Bec00] K. Beck: eXtreme Programming explained: Embrace Change. Reading, Massachusetts, Addison-Wesley. 2000.

[Fow99] M. Fowler: Refactoring: Improving the Design of Existing Code. Reading, Massachusetts, Addison-Wesley, 1999.

[Gam96] E. Gamma, R. Helm, R. Johnson, J. Vlissides: Entwurf-smuster – Elemente wiederverwendbarer objektorientierter Software. Übersetzung von D. Riehle, Bonn: Addison Wesley, 1996.

[Lip00] M. Lippert, S. Roock, H. Wolf, H. Züllighoven: JWAM and XP - Using XP for framework development. Proceedings of the XP2000 conference. Cagliari, Sardinia, Italy, 2000.

[Mer00] Peter Merel: eXtreme Hour. http://c2.com/cgi/wiki?ExtremeHour. 2000.