# Abschlussbewertung: Softwareprojekt 1 & 2

Projekt im Rahmen der Software-Projekt Vorlesung an der TFH-Berlin

---

| | | |
|---|---|---|
| Semester | : | WS 2004/2005 und SS 2005 |
| Gruppe | : | Klear ( http://www.Klear.org ) |
| Studenten | : | Omar El-Dakhloul, Manuel Habermann, Patric Sherif, Marco Kraus |
| Dozent | : | Hr. Prof. Knabe |

---

**Kurzbeschreibung des Projektes:**

Klear ist eine OpenSource-Anwendung für digitales Fernsehen (DVB im Speziellen) unter Linux. Die Kernfunktionen sollen neben normalem Fernsehen auch die Nutzung der digitalen Funktionen sein. Darunter zählt unter anderem auch die direkte Aufzeichnung von Sendungen in verschiedenen digitalen Formaten am Computer, das zeitgesteuerte Aufzeichnen, Videofilter, Audiosteuerungen, die Integration in die Betriebssystemumgebung und vieles mehr. Neben den sichtbaren Komponenten muss eine DVB-Anwendung auch die Hardware ansprechen, Kanaleinstellungen und Tuner steuern (verschiedene Steuerungen für DVB-T/S/C) und eine persistente Konfiguration beinhalten.

**Hauptarbeitsfelder der Gruppenmitglieder:**

*Manuel Habermann:* Live/Scheduled-Recording, Systray-Integration, KlearXine-Engine

*Patric Sherif:* Live/Scheduled-Recording, Systray-Integration, KlearXine-Engine

*Omar El-Dakhloul:* Dokumentation, Klassendiagramme. Code: GUI-Design

*Marco Kraus:* Projektkoordination und Releasemanager. Code: alles was nicht explizit als Fremdcode gekennzeichnet oder anderen Entwicklern zugeordnet ist.

**verwendete Technologien:**

1) * Linux als Betriebssystem (Debian, Knoppix und SuSE)
   * gcc 3.3.x als C++ Compiler
   * cvs zur Versionskontrolle

2) qmake als Build-Tool
   Pro: Besseres System als das übliche Automake
   Pro: Kompatibel zu anderen Make-Systemen
   Contra: Kein Autoconf-System dabei
   Contra: Nicht bei allen User bei der Standardinstallation verfügbar

3) KDevelop3 als Entwicklungsumgebung
   Pro: Freie Software und kostenfrei zu beziehen
   Pro: gut strukturierte und übersichtliche GUI mit IDEA
   Pro: sehr flexibel (selbst Editor ist austauschbar)
   Contra: Sehr schwache (bis keine) Unterstützung für Refactoring
   Contra: etwas undurchsichtige Unterstützung für Versionkontrollsysteme

4) QT / QT-Designer von Trolltech als GUI-Framework und C++-Toolkit
   Pro: Freie Software und kostenfrei zu beziehen
   Pro: auf viele Plattformen (Windows/MacOSX, Solaris, ...) verfügbar
   Contra: Keine Signal/Slots – Kommunikation über einzelne Threads hinaus

5) DVB Kernel-API für die DVB-Hardware und Tuner-Ansteuerung
   Pro: offene Schnittstelle
   Pro: gute Dokumentation
   Contra: nur bestimmte Chipsätze unterstützt (ca. 70% aller DVB Empfänger)
   Contra: Fehleranfällig bei zeitkritischen Ereignissen

6) Xine als Wiedergabe-API für die Video-Engines
   Pro: gut durchdachte Architektur
   Pro: Standard-API mit vielen Beispielimplementierungen im Netz
   Contra: komplex in der Programmierung
   Contra: Geschwindigkeit beim Starten der Videowiedergabe

7) MPlayer für die alternative Video-Engine
   Pro: schneller Player
   Contra: Keine API verfügbar
   Contra: keine direkte OSD Unterstützung

**Erfahrungen und Probleme:**

Wir wählten zu Beginn des Semesters *Extreme Programming* als Vorgehensmodell, da wir uns noch nicht über die technischen Details unsere Projekts im Klaren waren. Die Entwicklung sollte daher auch komplett neue Erfahrungen in Hard- und Software mitbringen. Wir starteten mit insgesamt fünf Entwicklern. Hiervon hatte aber nur ein Student praktische Erfahrung mit QT und C++ unter Linux. Am Ende war dies aber kein grosser Nachteil. Die Einarbeitungszeit in QT war recht kurz und die Probleme mit C++ konnten in Teamarbeit immer recht schnell gelöst werden.

Durch das “Extreme Programming” entstanden in den ersten Entwicklungsmonaten mehrere tausend Zeilen an Code. Zusätzlich wurden aber auch viel Code wieder verworfen, da die Tests zeigten, dass die Technologien ungeeignet waren (beispielsweise die direkte Verwendung von ffmpeg zur Videotranskodierung). Die Nutzungs von XP als Vorgehensmodell verursachte aber auch, dass im Laufe des Projekts generelle Änderungen (zum Beispiel im Styleguide, Codingguide oder der Architektur) weitere Änderung an sehr vielen Stellen im Code mit sich brachte. Vor allem im SWP2 beschäftigten wir uns viel mit Refactoring, weshalb wir deutlich weniger neuen Funktionen und Versionen erarbeiten konnten. Auch vernachlässigten wir zu Beginn des Projektes ein wenig die Qualitätssicherung (Exceptions, Unittests). Dies mussten wir durch Mehrarbeit im zweiten

Projektteil dann aufarbeiten. Zudem verließ uns ein Entwickler, der primär für die Dokumentation und Klassendiagramme zuständig war nach SWP1, so dass wir auch diese Aufgaben intern noch verteilen mussten.

Auch stellten wir Klear als OpenSource-Software früh und oft der Öffentlichkeit vor. Das Feedback aus aller Welt konnten wir wieder in unser Projekt einfliessen lassen. So haben wir auf Grund der großen Nachfrage die Unterstützung für DVB-S (Satellitenfernsehen) in Klear eingebaut. In unserem ursprünglichen Plan war dies nicht vorgesehen. Durch die Vielzahl an Testern, die wir so erreichten konnten wir auch Probleme lokalisieren, die bei uns auf den Entwicklungssystemen nicht aufgetreten sind. Viele Tester und viel Feedback sind daher ein deutliches Plus in der Entwicklung.

Durch das Vorgehensmodell entstanden uns weiter Vor-, aber auch Nachteile in der Lehrveranstaltung. Da nur zwei Gruppen XP wählten, war die Lehrveranstaltung natürlich sehr auf die klassischen Entwicklungsmodelle ausgerichtet. Die für uns nützlichen Unittest- und Mocktest-Vorlesungen kamen zu spät. Gewünschte Code-Richtlinien und Spezifikationen wurden manchmal erst im SWP2 bekannt gegeben, wenn die anderen Gruppen in die Implementierungsphase eintraten. Da hatten wir aber bereits mehrere tausend Zeilen Code erzeugt und mussten diese nachbearbeiten. Wir trafen uns jede Woche zur Rücksprache mit Herr Prof. Knabe und konnten aber so die Entwicklungsrichtung zusammen bestimmen und Fehler frühzeitig beheben. Auch konnten wir etwas dem Druck der Meilensteine entgehen, was für uns ein etwas entspannteres Arbeiten bedeutete.

Zusammenfassend ist daher zu sagen, dass die Anwendung im SWP1 also die grobe Struktur und vor allem den nötigen Funktionsumfang und im SWP2 die verbesserte Architektur, einen deutlichen Qualitätsanstieg und die Stabilität bekommen hat. Neben dem enormen Wissenszuwachs in den verwendeten Technologien war vor allem die Teamarbeit und die Teamorganisation eine wichtige Erfahrung. Auch “extreme programming” war eine wichtige Erfahrung. Fehler, die man in klassischen Vorgehensmodellen vermeidet, treten hier doch wieder auf. Sie lassen sich jedoch durch viel praktische Arbeit und Erfahrung minimieren. Diese muss man jedoch erst Sammeln. Den ersten Schritt haben wir somit getan.

**Daten und Fakten:** [ Bis einschließlich Klear Version 0.1 vom 16.06.2005 ]

- Vier Projektmitglieder
- Ca. 25 000 LOC inklusive GUI Komponenten und Unittests
- Neun öffentliche Releases
- Ca. 5 000 Downloads der Preview-Versionen
- Ca. 50 User auf der Mailingliste
- Binärpakete u.a. für Debian und Mandriva verfügbar
- Über 100 eMails an Feedback aus der ganzen Welt