Grundüberlegungen zur Ausbildung in Software-Engineering im Studiengang Informatik

Lutz Prechelt (prechelt@ira.uka.de)
Institut für Programmstrukturen und Datenorganisation
Universität Karlsruhe, Postfach 6980
D–76128 Karlsruhe, Germany
++49/721/608-4068, Fax: ++49/721/694092

(In ähnlicher Form erschienen in : Jörg Raasch, Thomas Bassler (Hrsg.) "Software Engineering im Unterricht der Hochschulen SEUH 92", Workshop des German Chapter of the ACM und der Gesellschaft für Informatik am 27. und 28. Februar 1992 in Stuttgart, B.G. Teubner Stuttgart, German Chapter of the ACM Berichte 37, 1992.)

These 1:
Wer programmieren kann ist potentiell ein gefährlicher Produzent von Software-Müll.

Begründung:
Da die Software-Krise noch längst nicht behoben ist, besteht immer die Neigung, daß wer programmieren kann auch dazu herangezogen wird. Wer kein SE gelernt hat, bringt es nur schwer fertig, dabei wohlstrukturiert zu arbeiten. Unstrukturiertes Arbeiten führt jedoch zumindest im Falle längerer Lebensdauer der Software (Wartung!) schon bei relativ kleinen Komponenten zu unbeherrschbaren Zuständen.

Folgerung:
Die Grundausbildung in SE muß für alle erfolgen, die eine Grundausbildung in Programmieren bekommen (also nicht nur in der Informatik!). Um der Bildung von schlechten Angewohnheiten von vornherein vorzubeugen, sollte sie auch zeitlich mit den ersten Programmierübungen einhergehen.

These 2:
SE-Wissen ist in allen Bereichen der Informatik von Nutzen.

Begründung:
Wo Software implementiert wird, ist die Richtigkeit der These offensichtlich. Beim theoretischen Arbeiten und im Hardwarebereich sind zumindest die Bereiche Entwurfstechniken und Wissen über die Effekte von Teamarbeit nützlich.

Folgerung:
Da SE für jedes Spezialgebiet eine notwendige oder zumindest nützliche Grundlage bildet, sollte die SE-Ausbildung zeitlich vor der Ausbildung in anderen Spezialgebieten liegen.

These 3:
Egal wie gut die theoretische SE-Ausbildung gemacht wird, verinnerlicht werden ihre Aussagen erst in der Praxis.

Begründung:
Es ist eine menschliche Eigenschaft, daß das einfach-drauflos-Arbeiten zunächst stets bequemer aussieht. Der daraus entstehende innere Schweinehund wird erst aufgrund von eigenen Erfahrungen über-

wunden. Ein zweiter Grund ist Selbstüberschätzung: “Ich weiß zwar, daß man das eigentlich nicht soll, aber IN DIESEM SPEZIELLEN FALL habe ich das schon im Griff.”

Folgerung:
Es muß ein starker praktischer Anteil in der SE-Ausbildung vorhanden sein, wenn nicht viel von ihr verpuffen soll.

These 4:
Praxis kostet Zeit.

Begründung:
Praxis kostet Zeit.

Folgerung:
Die Auswahl geeigneter Projekte für die praktische SE-Ausbildung ist sehr kritisch, um in vertretbarer Zeit möglichst viele Aspekte nahebringen zu können. Es ist vermutlich zweckmäßiger, eine kleine Menge sehr gut ausgewählter Projekte zu verwenden, als ständige Abwechslung anzustreben.

These 5:
Der größte Freund und Feind des SE heißt Wartung.

Begründung:
Einerseits wird der Nutzen von SE-Methodik oft erst bei längerfristiger Weiterentwicklung eines Systems voll sichtbar.
Andererseits werden die schönen akademischen Erkenntnisse über SE selbst dort, wo sie bei der Erstellung eines Systems mehr oder minder getreulich umgesetzt werden, bei der Wartung und Pflege der Systeme mit schöner Regelmäßigkeit über den Haufen geworfen. Dies ist eine nicht leicht erklärliche, aber durch viele Erfahrungen gestützte Beobachtung.

Folgerung:
Ein “Wartungserlebnis” sollte schon im Studium herbeigeführt werden.

Ist-Zustand an der Universität Karlsruhe und Beurteilung:

Grundstudium:

In Informatik I-IV finden Programmierübungen aber keine explizite SE-Ausbildung statt. Der Verzicht ist mit der großen auf einmal anfallenden Stoffmenge am Beginn des Studiums begründet und ist tolerierbar, weil die erstellten Programme stets sehr klein sind. Viele SE-Grundlagen werden aber natürlich hier gelegt, z.B. prädikatenlogische und algebraische Spezifikation, Top-Down Entwurf, abstrakte Datentypen, strukturiertes Programmieren, formale Programmverifikation etc.

Hauptstudium:

In einem Kanon von 8 Pflichtvorlesunge, die paarweise in vier 1+1 stündigen schriftlichen Prüfungen abgeprüft werden, gibt es eine Vorlesung “Softwaretechnik” diese soll Grundlagen des SE in folgenden Bereichen vermitteln: Softwarekrise, Anforderungsanalyse, Spezifikation, Grobentwurf, Kostenschätzung, Feinentwurf, Implementierung, Test, Verifikation, Validierung, Abnahme, Versionen- und Variantenverwaltung, Projektmanagement, Wartung, Werkzeuge, Effekte von Teamarbeit, Software-Wiederverwendung. Es werden formale, halbformale und informale Verfahren vorgestellt. Die Vorlesung umfaßt 3 Wochenstunden plus 1 Stunde Übung.

Ein reines SE-Praktikum gibt es derzeit nicht mehr. Es werden aber etwa 20 verschiedene andere Praktika angeboten, unter denen zumindest das Parallelrechner-, das Compilerbau- und das Betriebssystempraktikum teilweise SE-Inhalte berücksichtigen. Die Absolvierung mindestens eines Praktikums ist Pflicht.

Beurteilung:

Es ist in Karlsruhe möglich, sein Studium in Informatik mit einem Minimum an SE-Kenntnissen abzuschliessen: Da SE zum überwiegenden Teil ein sehr schlecht schriftlich prüfbarer Stoff ist, führt die Vorlesung hier nicht zwangsläufig zu einer ausreichenden Basisqualifikation. Praktische Programmiererfahrungen lassen sich im Extremfall (meist ausgenommen die Diplomarbeit) auf wenige hundert Zeilen im ganzen Studium begrenzen. Praxisnahe Projekterfahrungen lassen sich im normalen Studienverlauf kaum sammeln, geschweige denn, daß sie verbindlich wären. Dieser Zustand läßt bezweifeln, daß die Absolventinnen und Absolventen typischerweise als praxistauglich für SE-Aufgaben zu bezeichnen sind. Diese Einschätzung bestätigt sich meist in Verlauf und Ergebnis von Diplomarbeiten.