Ulrike Jaeger, Kurt Schneider (Hrsg.) (2009): Software Engineering im Unterricht der Hochschulen,
SEUH 11, Hannover, dpunkt.verlag, Heidelberg

Erfahrungen bei der Lehre des Software Engineering

Jochen Ludewig
Universität Stuttgart
ludewig@informatik.uni-stuttgart.de

Zusammenfassung

Dieser Beitrag stellt einige Erfahrungen zusammen, die ich in 25 Jahren Lehrpraxis im Software Engineering gesammelt habe. Die wichtigsten Punkte sind in zwölf Thesen formuliert. Ich hoffe, auf diese Weise jüngeren Kolleginnen und Kollegen Anregungen zu geben und der weiterhin notwendigen Diskussion über die Lehre des Software Engineering Nahrung zu liefern.

1 Hintergrund

Im Dezember 1983 erreichte mich die Anfrage, ob ich bereit sei, im Neu-Technikum Buchs, einer Fachhochschule in der Ostschweiz, eine Vorlesung über Software Engineering zu halten. Das habe ich gern getan. Zwei Jahre später war ich Assistenzprofessor für das Fachgebiet Software Engineering an der ETH Zürich, und seit 1988 sitze ich auf einem Lehrstuhl für Software Engineering an der Universität Stuttgart.

Damit blicke ich auf ein Vierteljahrhundert Lehre im Software Engineering (SE) zurück. Die interessanteste Zeit begann 1995, als in Stuttgart der neue Studiengang Softwaretechnik konzipiert, eingeführt und verbessert wurde [Ludewig 2008]. Nachfolgend trage ich einige positive und negative Erfahrungen zusammen. Vielleicht sind sie für jüngere Kolleginnen und Kollegen nützlich; mindestens sollten sie eine Diskussion provozieren.

Es liegt in der Natur einer Erfahrungssammlung, dass die Feststellungen auf Beobachtungen beruhen, die reflektiert, aber nicht wissenschaftlich, also durch eine sie stützende Literatur oder empirische Untersuchungen, begründet sind.

Auch erhebe ich nicht den Anspruch, völlig neue Einsichten zu präsentieren oder solche, die *nur* für unser Fachgebiet gelten. Die Aussagen scheinen mir wichtig und bislang nicht allgemein anerkannt; das ist Grund genug, sie vorzutragen.

2 Der Reiz eines jungen Fachs

Es ist üblich geworden, das Jahr 1968, genauer: die NATO-Konferenz in Garmisch, als Geburtsdatum des Software Engineering zu betrachten [Randell 2001]. Genau genommen fand aber in Garmisch nur die Zeugung statt; geboren wurde das SE einige Jahre später, als Boehms wegweisende Artikel erschienen [Boehm 1973, 1976]. Vorher war der Begriff kaum ernst genommen worden. Wer die Berichte von Brian Randell und anderen Pionieren des Gebiets auf der ICSE 2008 in Leipzig gehört hat, wird sich erinnern, dass es zunächst gar nicht danach aussah, als ob das SE Zukunft hätte. Die Nachfolgekonferenz in Rom 1969 war ein Flop und veranlasste viele der Beteiligten, die Sache nicht weiter zu verfolgen. Dass ein paar Jahre später trotzdem ein neues Forschungs- und Lehrgebiet mit Zeitschriften, Tagungen und Lehrstühlen entstand, lag vor allem am starken Bedarf der Praxis, aber auch am Wunsch einiger der Pioniere, einen Heimathafen für ihre Forschung zu haben, die sonst nirgends richtig hineinpasste. (Man schaue sich an, in welchen Zeitschriften und auf welchen Konferenzen die frühen, heute zum SE gerechneten Arbeiten erschienen sind!)

Ungeklärt blieb dabei, was SE eigentlich sein sollte. Diese Unklarheit war kein Mangel, sondern ein Glück. Denn so konnten sich mathematisch geprägte Forscher wie Bauer, empirisch arbeitende wie Boehm und konstruktiv denkende wie Parnas unter dem Schirm des SE versammeln.

Daraus folgt **These 1**:
SE vereint ganz unterschiedliche Auffassungen, Problemsichten und Lösungsansätze. Wir sollten nicht versuchen, eine klare Definition zu schaffen, die Teile der Gemeinschaft ausgrenzt. SE ist ein Vielvölkerstaat. Wir sollten auch die fruchtlose Diskussion einstellen, was SE *wirklich* ist. SE ist, was wir, seine *Professionals*, darunter verstehen. Dazu gehört natürlich auch, dass wir, trotz unterschiedlichen Positionen und Schwerpunkten, einander respektieren und akzeptieren.

Die Möglichkeit, die allermeisten Pioniere persönlich zu treffen und zu sprechen, zeichnet das SE vor fast allen anderen Teilgebieten der Informatik und noch weit deutlicher vor den übrigen Wissenschaften aus. Dies wurde nirgends besser sichtbar als 2002 in Bonn, wo sich auf Einladung von sd&m viele der Pioniere trafen [Broy/Denert 2002].

Der persönliche Kontakt mit den Großen des Fachs bewirkt vor allem, dass man in allen Entwicklungen die menschliche Dimension sieht. Die Erfindungen,

Ideen und Irrtümer sind nicht vom Himmel gefallen, sondern in den Köpfen denkender Personen entstanden. Das zu wissen, gibt mir selbst die Zuversicht, Beiträge leisten zu können und irren zu dürfen.

Dies führt zur **These 2**:
Wer aktiv im SE mitmischen will, sollte wissen, dass es überall menschelt, auch bei den Koryphäen. Darum sollte keine Lehre des SE darauf verzichten, die Protagonisten vorzustellen und ihre Leistungen und Irrtümer zu diskutieren. Am besten ist es natürlich, sie in den eigenen Hörsaal zu holen. Das gilt ebenso für die Urheber aktueller Ideen und Konzepte, z.B. Entwurfsmuster, agile Prozesse und Model Driven X. Sollte sich dabei zeigen, dass *der Kaiser gar keine Kleider anhat*, ist auch das sehr erhellend.

Die Möglichkeit, die Pioniere zu treffen, schwindet; seit dem Bonner Symposium sind wichtige Leute gestorben (Dijkstra, Dahl, Nygaard). Aber andere (wie Parnas, Brooks, Boehm, Jackson und auch Bauer) sind weiterhin aktiv und oft auch auf Konferenzen anzutreffen.

3 Praktika und Projekte

Praktika und Projekte sind für die SE-Lehre zentral. Sie sollten präzise auf die übrigen Lehrveranstaltungen abgestimmt sein (oder besser: die anderen auf sie). In Stuttgart gibt es die *Studienprojekte*, in denen acht bis zehn Studenten zwölf Monate lang an einer konkreten Aufgabe arbeiten, von der Erhebung der Anforderungen bis zur Ablieferung des Produkts.

Das halbe Dutzend solcher Projekte, die unsere Abteilung bislang angeboten hat, war für alle Beteiligten sehr lehrreich. Wir sind uns heute einigermaßen sicher, dass

- Studenten mit ausreichenden SE-Vorkenntnissen auch anspruchsvolle Probleme lösen können und in der Regel tatsächlich lösen,
- die Aufgabe unbedingt von außen vorgegeben sein muss, im besten Fall aus einem Bereich, der den Teilnehmern völlig fremd ist,
- die Rolle des Kunden mit der eines Betreuers definitiv inkompatibel ist,
- ein Vorgehensmodell auf der Basis eines »iterierten Wasserfalls« (7 Monate bis zur Ablieferung eines Kernprodukts, 5 Monate für die Verbesserung und Ergänzung) in den meisten Fällen gute Aussichten auf Erfolg bietet,
- die Projekte meist nach einem Vierteljahr in eine Krise geraten, weil sich zeigt, dass die Kooperationstechniken aus kleinen Projekten nicht mehr ausreichen,
- jedes Projekt durch die Teilnehmer, vor allem durch den Projektleiter, eine andere Kultur entwickelt, die sich im Selbstbewusstsein der Gruppe, in der Kooperation und in der Kommunikation niederschlägt,

- technologische Probleme keine große Rolle spielen, da die Studenten in der Regel auch mit neuer Technologie nach kurzer Zeit klarkommen,
- das Verhältnis zum Kunden dagegen schwierig bleibt, weil die Studenten unsicher sind, was sie fragen, vorschlagen oder selbst entscheiden sollen,
- vielen Studenten die Vermarktung ihrer Lösung abwegig erscheint,
- es keineswegs notwendig, sondern eher schädlich ist, als Veranstalter des Projekts schon vorher *eine* oder gar *die* Lösung der Aufgabe zu kennen.

These 3:
Projekte müssen sein, sukzessive vom Programmierkurs über Kleingruppen-Projekte bis hin zu solchen, in denen Projektmanagement und geregelte Kommunikation erforderlich sind. Die Studenten können dabei alle Rollen übernehmen außer der des Kunden. Auch Betreuer werden benötigt; sie schauen den Teilnehmern auf die Finger und geben ihnen gelegentlich, gefragt oder ungefragt, Ratschläge.

Unser aktuelles Projekt läuft anders als die früheren, weil wir diesmal im Gegentakt liegen, also nicht im Herbst, sondern im Frühjahr begonnen haben. Praktisch heißt das: Die Teilnehmer haben die zweite SE-Vorlesung mit dem Schwerpunkt Projektmanagement noch nicht gehört. Das erweist sich als Handicap. Vorlesung und Projektbeginn im selben Semester sind optimal. Es ist nicht klar, ob die technikzentrierte Sicht der Teilnehmer, die wir in diesem Projekt beobachten, ebenfalls mit dem Mangel an Managementwissen zu tun hat.

These 4:
Projekte und Vorlesungen zum Projektmanagement und zu den eingesetzten Techniken sollten simultan stattfinden. Die »Einheit von Theorie und Praxis«, im real existierenden Sozialismus gern beschworen: Hier ist sie möglich und nötig.

Wir haben außerdem zum ersten Mal ein Projekt, bei dem die Teilnehmer einem extremen Zeitdruck ausgesetzt waren. Es hat sich gezeigt, dass sie darauf nicht vorbereitet sind und in dieser Situation nicht immer sinnvoll entscheiden.

Eine in diesem Zusammenhang aufgeflammte Diskussion in unserer Abteilung befasste sich mit der Frage, wie realistisch Projekte sein können und sollten. Die spontane Antwort »So realistisch wie möglich« trägt nicht. Wir müssen uns darüber klar sein, dass wir die Realität nicht nachstellen können und auch nicht *wollen*. Praktisch kein Softwaresystem ist wirklich fertig, wenn es ausgeliefert wird. Für den Erfolg der Software ist gerade wichtig, wie sie nach der Auslieferung betreut und gewartet wird. Darum ist ein Projekt, an dessen Ende eine fertige, also korrekte und brauchbare, Software steht, offensichtlich unrealistisch.

Aber die Wartungsphase lässt sich beim besten Willen nicht in unsere Projekte integrieren. Wo also liegt das Optimum?

Und natürlich wollen wir all die – nach unserer festen Überzeugung – kontraproduktiven Randbedingungen *nicht* kopieren, die in der Praxis nicht selten vorkommen, beispielsweise unsinnige Vorgaben des Managements, hohe Fluktuation usw.

These 5:
Projekte in der Hochschule bleiben selbst im Idealfall verschieden von realen Projekten außerhalb der Hochschule. Es ist Sache der Lehrenden, die Ähnlichkeiten – so weit möglich und sinnvoll – herzustellen und die Unterschiede zu reflektieren und mit den Teilnehmern zu diskutieren.

Im Bachelor-Studienplan (ab 2009) gibt es, wenn unsere Vorschläge die Gremien passieren, die Studienprojekte unverändert, allerdings folgt nicht mehr wie bisher nach dem ersten in der Informatik-Fakultät noch ein zweites in einem Anwendungsfach.

4 Prüfungen und Noten

Software Engineering ist sehr einfach. Was wollen wir denn vermitteln? Es sind doch banale Weisheiten, beispielsweise die, dass man ein neues Artefakt nicht weitergeben sollte, ohne es zuvor geprüft zu haben. Das ist *schrecklich* einfach, weil jedes Kind die Aussage begreift; dagegen braucht das tiefe Verständnis und die Aufnahme in den eigenen Wertekanon Monate oder Jahre. Ähnliches kann man über Planung, Dokumentation, Strukturierung und andere Aspekte sagen. Ob jemand ein Integral berechnen kann, lässt sich leicht prüfen. Aber wie prüft man, ob ein Student begriffen hat, dass Planung wichtig ist? Wenn wir so etwas abfragen, bekommen wir Absolventen, die den SE-Katechismus auswendig gelernt haben, aber keine Software-Ingenieure.

Diese Situation führt dazu, dass wir entweder nicht prüfen, was wir lehren, oder nicht lehren, was wir lehren sollten. Tatsächlich habe ich von einem Kollegen als Begründung für seine Stoffauswahl gehört: »Schließlich muss das ja auch prüfbar sein!« Hier wedelt offensichtlich der Schwanz mit dem Hund.

These 6:
Wir brauchen neue Verfahren, um Studenten zu prüfen und ihre Leistungen zu bewerten. Denn Klausuren und mündliche Prüfungen, wie wir sie bislang praktizieren, sind ungeeignet, das tiefe Verständnis zu kontrollieren. Vielleicht können wir (in Anlehnung an Prüfungen in Maschinenkonstruktion) in der Prüfung komplexe Szenarien vorgeben, die mehr als eine schematische Reaktion erfordern.

Bei Projekten fällt es leicht, die Leistung der Gruppe insgesamt zu bewerten. Viel schwieriger ist die (durch die Prüfungsordnung gestellte) Aufgabe, individuelle Noten zu verteilen. Wir haben das Problem nicht gelöst, aber entschärft, indem wir jeweils zu Beginn das Verfahren erklären: Eine kollektive Note wird festgelegt und dann für Teilnehmer, die (positiv oder negativ) aufgefallen sind, durch ein angemessenes Delta korrigiert. Das ergibt die individuellen Noten.

5 Industrie und Software-Engineering-Lehre

Auf den ersten Blick scheint es geradezu ideal, den Studenten Praxiserfahrungen zu vermitteln, indem man ihnen Prüfungsarbeiten (insbesondere Diplomarbeiten) gibt, die ganz oder zu einem erheblichen Teil in der Industrie stattfinden. (»Industrie« steht hier für alle Organisationen, die gewinnorientiert arbeiten und nicht der Forschung oder Lehre dienen.)

Unsere Erfahrungen mit solchen Arbeiten sind überwiegend negativ. Vor allem drei Punkte haben immer wieder Schwierigkeiten gemacht:

1. Viele Studenten suchen gezielt nach Industrie-Arbeiten (vor allem Diplomarbeiten), weil sie die Bezahlung brauchen, die die Industrie meist in Aussicht stellt. Damit stehen außerfachliche Gesichtspunkte bei der Wahl der Arbeit im Vordergrund. Und aus Sicht der Studenten gibt es Diplomarbeiten erster und zweiter Klasse.
2. Die Industrie will vor allem verwertbare Resultate. In vielen Fällen ist dies ausführbarer Code. Kommt die Arbeit nicht so schnell voran wie ursprünglich geplant (eine sehr oft auftretende Situation!), so wird die Codierung von Industrieseite forciert, auch dann, wenn die Grundlagen dafür längst nicht geklärt sind.
3. Selbst wenn völlig klar ist, dass die Industrieseite keinen Einfluss auf die Benotung der Arbeit hat, wollen die Diplomanden es ihren Betreuern (und in der Regel Gönnern) recht machen. Sie treffen darum Entscheidungen, die nicht sachlich begründet, aber von den Partnern in der Industrie gewünscht sind. (Typische Formulierungen: »Das habe ich nicht näher untersucht, weil sich das dort ohnehin nicht durchsetzen lässt.«)

Diese Erfahrungen, die nicht auf meinen Lehrstuhl beschränkt sind, haben dazu geführt, dass die gesamte Stuttgarter Informatik inzwischen Arbeiten mit externen Partnern sehr kritisch gegenüber steht.

Wir haben daraus die Konsequenz gezogen, dass wir extern durchgeführte Arbeiten in der Regel ablehnen (also das Thema nicht ausgeben und die Arbeit entsprechend auch nicht prüfen). Das gilt insbesondere, wenn ein Student mit einem Thema hausieren kommt. Dagegen führen wir solche Arbeiten durch, wenn die Industrie uns anspricht, wir am Thema interessiert sind, die Bedin-

gungen der Arbeit vollständig vorgegeben sind und wir selbst eine Person suchen, die das Thema bearbeiten will und kann.

These 7:
Bei Prüfungsarbeiten, die außerhalb der Hochschule durchgeführt werden, ist das Risiko groß, dass Aspekte, die nichts mit dem Studium zu tun haben (sollten), einen negativen Einfluss auf den Verlauf und die Ergebnisse haben. Prüfer sollten ihrer Verantwortung gerecht werden, indem sie sicherstellen, dass solche Einflüsse ausgeschlossen bleiben. Das läuft auf eine sehr restriktive Ausgabepraxis hinaus.

Ganz anders ist die Lage, wo die Kooperation von vornherein vorgesehen ist. Unsere sogenannten Fachstudien (Untersuchung einer zu klärenden Frage, meist in der Industrie, durch eine Gruppe von drei Studenten) sind ein Schlager geworden. Die Industrie freut sich über die meist wirklich kompetente Beratung und über den Kontakt zu Studenten, die bald eine Stelle suchen werden. Die Studenten können das Gelernte anwenden und finden interessierte Zuhörer, die das Ergebnis wirklich brauchen. Obwohl es keine Note, sondern nur einen Schein gibt, haben wir noch nicht erlebt, dass dünne Bretter gebohrt werden.

6 Entscheiden lernen

Wissenschaft ist radikal. Das ist eine ihrer wichtigsten Eigenschaften. Ein Resultat ändert sich nicht dadurch, dass es erwünscht oder bedrohlich ist. Im SE fehlt es oft an Radikalität; immer wieder hört man in der Praxis Aussagen wie: »Ja, wir wissen, dass wir das anders machen sollten. Aber es lässt sich nicht so einfach durchsetzen.« Und Resultate, die klare Implikationen haben, sind keineswegs populär. Wir haben immer wieder gesehen, dass in Industriekooperationen Analysen willkommen, klare Worte aber unerwünscht sind.

Auch unsere Studenten schrecken – aus ganz anderen Gründen – vor klaren Aussagen zurück, denn sie setzen sich damit der Kritik aus. Aber wenn sie die Verhältnisse in der Praxis verändern, verbessern sollen, müssen sie genau solche Feststellungen treffen und vertreten können. Wir sollten uns also nicht mit lauwarmen Formulierungen abfinden, die alle Entscheidungen offenlassen. Die Studenten müssen schon bei uns lernen zu entscheiden.

These 8:
Es reicht nicht aus, das Analysieren der Probleme zu lehren und einzuüben. Wir müssen auch verlangen (und selbst vormachen), dass am Ende einer Analyse eine Wertung steht, die zu einer Entscheidung führt.

7 Deutsch

Da ich seit Jahren auch die Funktion habe, als »das Praktikantenamt« die Berichte der Studenten entgegenzunehmen und zu prüfen, sehe ich viele von Studenten verfasste Texte, meist auf Deutsch. Jedenfalls halten die Urheber das, was sie abliefern, für Deutsch. Ich sehe – bei beträchtlicher Streuung – eine stetige Abnahme der sprachlichen Kompetenz. Am schlimmsten ist die Interpunktion, dann kommen Rechtschreibung und Satzbau. Von Stil zu sprechen, erscheint fast als Witz. Ich erinnere mich noch an meinen Schock, als ich zum ersten Mal einen Satz der Art »Mir viel auf das ...« sah.

Solche Fehlleistungen beschränken sich nicht auf die Studenten; mangelhaftes Deutsch hören und sehen wir heute auch bei Politikern, Journalisten und Professoren. In vielen Briefen werde ich mit »sie« angeredet, und »wegen« mit Dativ ist ebenso wie »dank« mit Genitiv Standard (oder, wie es immer öfter heißt, »Standart«).

Der beliebte Hinweis auf Unsicherheiten, die durch die Rechtschreibreform entstanden sind, geht völlig an der Realität vorbei: Die Texte sind so falsch, dass sie keiner deutschen Rechtschreibung seit Luther zugeordnet werden können.

Ich weiß einfach nicht, wie ich damit umgehen soll. Es gibt offensichtlich zwei klare, einfache Standpunkte. Wir können entweder verlangen, dass die Regeln eingehalten werden. Oder wir können alles akzeptieren, was bei geeigneter Transkription irgendeinen Sinn ergibt. Die zweite Möglichkeit erscheint mir indiskutabel. Über Jahrhunderte gab es einen Konsens, dass der Schreiber Aufwand treibt, um dem Leser das Leben zu erleichtern. Ist das heute obsolet? Sprache ist ein Spiegel des Denkens. Akzeptieren wir, dass Gedanken nicht geordnet und geprüft, sondern unverdaut ausgewürgt werden?

Informatik ist eine Kommunikationswissenschaft. Wer die Syntax einer Programmiersprache lernen kann, sollte mit dem Deutschen keine größeren Probleme haben. Ich neige also dazu, die Einhaltung der Regeln zu fordern. Dass es dabei nicht auf ein paar Fehler ankommt, ist klar; die unterlaufen uns allen.

Damit lautet meine **These 9**:
Wir sollten uns bewusst um präzise, korrekte Sprache bemühen, sie auch einfordern und in der Bewertung von Texten, auch Dissertationen und Publikationen, die sprachliche Qualität entsprechend gewichten.

8 Alternde Professoren

In den Achtzigerjahren habe ich gern und viel programmiert. Später gab es dazu immer weniger Gelegenheiten, und ich war (und bin bis heute) umgeben von jungen Leuten, die sehr routiniert mit der aktuellen Technologie umgehen, ob es nun neue Programmiersprachen oder irgendwelche Umgebungen und Frameworks

sind. Ich glaube noch immer, die Konzepte zu kennen und zu verstehen, aber ich weiß auch, dass es große Vorteile hat, sich immer wieder selbst mit den Details der Softwareentwicklung zu befassen.

2012 wird meine Pensionierung dieses Problem für mich lösen. Aber ich kann mir nicht vorstellen, dass ich der Einzige bin, der es hat. Wir, die Hochschullehrer, sollten darüber nachdenken, ob es nicht auch für uns eine Art Life Cycle gibt, der unsere sich ändernden Stärken und Schwächen berücksichtigt. Es hätte viele Vorteile, wenn es für meinesgleichen ein Altenteil gäbe, auf dem ich die letzten Jahre mit Einführungs- und Überblicksvorlesungen verbringen könnte (die meine jüngeren Kollegen offensichtlich fürchten). Dazu müsste aber heute schon eine Nachfolgerin oder ein Nachfolger da sein, um die spezielleren Lehrveranstaltungen zu übernehmen.

Meine **These 10** ist damit:
Die Stärken und Schwächen der Professoren ändern sich; eine Fakultät sollte wie eine Familie funktionieren, in der sich die Rollen ebenfalls mit den Jahren ändern. Das Visionäre, das Gestalten der Zukunft ist Sache der Jungen, ebenso die Präsentation der jüngsten wissenschaftlichen Resultate in der Lehre. Die Älteren können sehr gut die Grundlagen und den fachlichen Überblick vermitteln; durch die Übernahme wichtiger Ämter (Studiendekan, Vorsitz im Prüfungsausschuss usw.), in denen die lange Erfahrung nützt, halten sie den Jüngeren den Rücken frei.

9 Ingenieure für Software

In Deutschland wohnt das SE wie in den meisten anderen Ländern bei der Informatik zur Untermiete. Damit sind die Rahmenbedingungen der Lehre durch die Informatik vorgegeben. Das bedeutet vor allem: Die Grundausbildung in Programmierung erfolgt bottom-up, von Micky-Maus-Programmen zur komplexen Software (wobei man dorthin in der Hochschule kaum kommt), das Grundstudium hat eine starke Prägung durch die Theorie (das Pumping-Lemma muss sein), und im Übrigen bekommen die Studenten eine breite Übersicht der Gebiete, nicht am Bedarf orientiert, sondern an der realen Mischung der Lehrstühle.

Ich behaupte nicht, dass das falsch ist. Wahrscheinlich ist es besser, zwei Vorlesungen eines begeisterten Datenbankers zu hören als eine gute über Datenbanken und eine wirklich *vorgelesene* über Software Engineering. Aber ich sehe auch nicht, dass dieses Vorgehen didaktisch legitimiert oder durch die berufliche Zukunft unserer Studenten gerechtfertigt ist. Eine offene Diskussion über die Inhalte und den Aufbau der Lehre ist, wenigstens in der deutschen Universität, wie ich sie kenne, nicht möglich. Die Entscheidungen werden von Leuten getroffen, die sich mehrheitlich als Lobbyisten ihrer Disziplinen verstehen. In der Dis-

kussion über die Umstellung auf Bachelor und Master (die von Politikern ohne jede fachliche Kompetenz erzwungen wurde) war das meistgenannte Argument meiner Kollegen: »Wir wollen die besten Leute in unserem Masterstudiengang haben!« Die Frage, wie denn Leute *ohne* Master ausgerüstet sein werden, wurde regelmäßig mit der Gegenfrage beantwortet: »Glaubt wirklich irgendjemand, dass das eine sinnvolle Qualifikation ist?«

In dieser Umgebung haben wir, die Software-Ingenieure, zwar die Möglichkeit, unsere Sicht der Dinge zu präsentieren, aber keine Chance, aus unseren Studenten Ingenieure zu machen. Chinese wird man nicht durch den Besuch eines Sprachkurses oder eines Diavortrags, sondern indem man in China unter Chinesen lebt.

Die Existenz eines Studiengangs Softwaretechnik in Stuttgart widerlegt diese Einschätzung nicht. Die Chance, ihn einzurichten, war 1995 durch eine Reihe glücklicher Umstände entstanden und ist nicht reproduzierbar. Wir haben damit wenigstens, um im Bild zu bleiben, ein Stück China, aber leider viel zu wenig Chinesen.

Das bedeutet (**These 11**):
Wer SE lehren will, darf sich nicht damit begnügen, die für ihn vorgesehene Nische zu gestalten. SE erhält seine Motivation und seine Notwendigkeit aus den Erfahrungen bei der Lösung realer, praktischer Probleme, es ist also – wie jedes Engineering – nicht an der Erkenntnis orientiert, sondern an der Problemlösung. Wir müssen dafür sorgen, dass unsere Studenten von Beginn an mit diesem Denkansatz vertraut sind. Das alte, heute leider vergessene Motto des Gymnasiums »Jede Stunde eine Deutschstunde« sollte auf unser Fach übertragen werden: »Jede Vorlesung eine Lehrstunde in SE!« Das klappt natürlich nur, wenn sich die Dozenten als Software-Ingenieure verstehen.

Konkrete Konsequenzen dieser Überlegung waren in Stuttgart der frühe Einsatz der SE-Vorlesungen und die Umstellung des Programmierkurses im 1. Semester auf ein Praktikum, in dem die Auseinandersetzung mit einem großen Programm (20 kLOC) geübt wird. Eine spezielle Einführungsvorlesung für Softwaretechniker hat sich leider nicht realisieren lassen. Allerdings hatten SE-Veranstaltungen positive Rückwirkungen auf die Lehre im Informatik-Studiengang. So führten Erfahrungen im zusätzlichen Programmierkurs dazu, dass ihn nun auch die Informatiker bekommen. Im Bachelor-Studiengang Informatik werden eine SE-Einführung und ein Projekt obligatorisch.

10 Institutionalisierung

Gerade Ingenieure neigen zu der Annahme, dass sich die bessere Lösung durchsetzt. Dabei ist die Welt voll von schlechten Lösungen, die die guten aus dem Rennen geworfen haben. Ich selbst war Jahrzehnte lang überzeugt, dass sich die offensichtlichen Vorteile einer sinnvollen SE-Ausbildung in einem Zuwachs an Lehrstühlen und Mitteln niederschlagen würden.

Das war ein Irrtum. Anders als die künstliche Intelligenz, die in den Achtzigerjahren so viel Lärm gemacht hat, dass sie noch heute vom Echo leben kann, ist das SE ein Aschenbrödel geblieben. Anders als im Märchen hat die Sache aber kein glückliches Ende, Aschenbrödel sitzt noch immer an der Feuerstelle, während ihre Stiefschwestern mit dem Prinzen (dem Herrn aus dem Ministerium) flirten.

Wenn es uns nicht gelingt, ein Lehr- und Forschungsgebiet SE zu etablieren, das strukturell nicht von der Informatik abhängt, müssen wir uns damit abfinden, dass unsere Arbeitsbedingungen und Gestaltungsmöglichkeiten von Leuten vorgegeben werden, die sich durch das Software Engineering vor allem bedroht fühlen. Natürlich ist diese Unabhängigkeit fern und alles andere als leicht erreichbar. Aber die Geschichte hat immer wieder gezeigt, dass ferne Ziele nicht utopisch bleiben, wenn sie nur mit großer Hartnäckigkeit verfolgt werden.

These 12:
Nur in einem *eigenständigen* Fachbereich, der mit der Informatik eng zusammenarbeitet und Module der Lehre austauscht, ihr aber nicht untergeordnet ist, wird sich das SE entfalten. Die Informatik stellt den Software-Ingenieuren die theoretischen Grundlagen im weitesten Sinne bereit.

11 Fazit

Unsere Lehre des Software Engineering ist, gemessen an der Nachfrage der Studenten und dem Echo aus der Industrie, recht erfolgreich. Aber wir schöpfen die Möglichkeiten bei weitem nicht aus. Verbesserungen sind auf allen Ebenen möglich: bei den Rahmenbedingungen, Studienplänen und Lehrveranstaltungen, bei der Beurteilung und Notengebung und an vielen anderen Stellen. Dass wir nicht alle Probleme lösen können, sollte uns nicht davon abhalten, wenigstens die einfacheren anzugehen.

Und wir sollten daran arbeiten, dass das Software Engineering nicht länger das falsche Etikett eines Spezialgebiets der Informatik trägt, sondern sich als Technik neben anderen Techniken versteht.

Danksagung

Es versteht sich von selbst, dass in einer solchen Sammlung von Erfahrungen viele Gespräche mit Studenten, Mitarbeitern und Kollegen verarbeitet sind. Sie alle haben Anteil an diesem Text.

Drei anonyme Gutachter haben sich die Mühe gemacht, mein Manuskript zu lesen und detailliert zu kommentieren. Dafür danke ich ihnen sehr herzlich.

Frau Kuhle hat die (vermeintliche) Endfassung geprüft, wie immer sehr effektiv.

Literatur

[Boehm 1973] B.W. Boehm: Software and its impact: a quantitative assessment. DATAMATION, 19, No.5, 48-59.

[Boehm 1976] B.W. Boehm: Software Engineering. IEEE Trans.s on Computers, C-25, pp.1226-1241.

[Broy, Denert 2002] M. Broy, E. Denert (eds.): Software Pionieers: Contributions to Software Engineering. Springer Verlag, Berlin. 2002

[Ludewig 2008] J. Ludewig: Software Engineering at Full Scale: A Unique Curriculum. Chapter XIV in H.J.C. Ellis, S.A. Demurjian, J.F. Naveda (eds.): Software Engineering: Effective Teaching and Learning Approaches and Practices. IGI Global, Hershey, PA, USA. 2008

[Randell 2001] B. Randell: Webseite mit Quellen zu den NATO-Konferenzen 1968 und 1969. *http://homepages.cs.ncl.ac.uk/brian.randell/NATO/*