Professionalität und Verantwortung
und
Einführung in den Kurs

Lutz Prechelt
Institut für Informatik, Freie Universität Berlin

Was bedeutet "Profession"?

- von lateinisch *professere*, "bekennen"

- Ein Dienstleistungsberuf (von "Berufung"),
  der auf einer ausführlichen Hochschulausbildung beruht und intensiven Gebrauch von vertieftem Wissen macht.
  - Ausübung ist zudem evtl. besonderer Lizensierung unterworfen

- Leider haben die einschlägigen Wörter im praktischen Gebrauch auch viele andere Bedeutungen
  - Englisch: profession, professional, professionality
  - Deutsch: (Beruf), professionell/Profi, Professionalität

Typische Professionen

- Medizin/Pharmazie
- Juristerei
- Ingenieurwesen

- Je nach Standpunkt auch viele weitere
  - die Grenzen sind fließend

Ungefähre Abgrenzung (*sehr* vereinfacht)

- nach Ausbildung (A), Wissen (W), Können (K), Verantwortung (V)

- **Arbeiter_in:**
  - A: kurz, praktisch
  - W: wenig
  - K: allgemeine menschliche Fertigkeiten, Körperkraft
  - V: nur Ausführungsqualität

- **Handwerker_in:**
  - A: mittellang, überwiegend praktisch
  - W: über Materialien und Werkzeuge
  - K: Planung, manuelles Geschick
  - V: Ergebnisqualität

- ***professional*:**
  - A: lang, v.a. theoretisch
  - W: breit und tief, über Grundlagenfakten, Zusammenhänge, Technologie, Methoden
  - K: Analyse von Situationen/Strukturen, Synthese von Lösungen, Urteilskraft
  - V: **weit reichend**

Anforderungen an professionals

- Beherrschung der Grundlagen ihres Fachs
  - Dafür ist das Studium gedacht!
- Tiefes Wissen in einem erkennbar abgegrenzten Teilbereich
  - Das erwirbt man überwiegend erst im Beruf
  - Evtl. formalisiert (z.B. Facharztausbildung)
- Kontinuierliche Fortbildung
  - Denn das Wissen aller Professionen entwickelt sich schnell fort
  - Das ist ein Anlass für die Existenz von Fachgesellschaften
    - bei uns z.B. GI, ACM, IEEE CS
- Sorgfalt und verantwortliches Handeln
  - Denn die Klienten können sich kaum selbst schützen
  - und Software hat oft hohes Gefährdungspotential

Das ist keine Kleinigkeit!

- Legen Sie Wert darauf, dass dieser Sicherheitsingenieur diese vier Eigenschaften mitbringt?
  - Software ist an sehr vielen Stellen ebenfalls heikel

guim.co.uk

Besonderheit der Informatik

- Alle anderen Ingenieur_innen konzipieren überwiegend nur, aber setzen kaum um
  - dafür gibt es passende Handwerker_innen
- Die Informatik hat es (noch?) wenig geschafft, Handwerksteile genügend gut abzugrenzen
  - deshalb sind wir überwiegend auch unsere eigenen Handwerker_innen
    - Ausnahmen z.B. beim Kunsthandwerk (web design etc.)
  - *"software craftsmanship"*
- Wir schließen in diesem Kurs handwerkliche Aspekte mit ein
  - Es geht also sowohl um Wissen
  - als auch um Geschick: Üben, üben, üben

Ethische Leitlinien

- Fachgesellschaften formulieren Verantwortungsregeln oft aus
  - maßvoll oder z.T. unrealistisch

- http://www.gi.de/wir-ueber-uns/unsere-grundsaetze/ethische-leitlinien.html
  - maßvoll und bodenständig; ziemlich gelungen
- http://www.acm.org/about/code-of-ethics
  - detailliert und wortreich,
    z.T. leicht abgehoben (ignoriert manche Abwägungen)
- http://www.computer.org/portal/web/getcertified/resources/code_of_ethics
  - sehr detailliert (dadurch recht konkret), aber knapp formuliert

Beispiele:

Ethische Leitlinien: Zitate

- GI, Präambel:
  - "Kompetenz in der Ausübung des Berufs ist zwar selbst noch kein moralisches Handeln, doch ist die bewusste Hinnahme fehlender Fähigkeiten verantwortungslos."
  - Art. 1 Fachkompetenz
  - Art. 2 Sachkomp. und kommunikative Komp.
  - Art. 4 Urteilsfähigkeit
- ACM, 2.1:
  - "Excellence is perhaps the most important obligation of a professional."
- IEEE CS:
  - 3.01: "Strive for high quality, acceptable cost and a reasonable schedule, ensuring significant tradeoffs are clear to and accepted by the employer and the client"
  - 3.04: "Ensure that they are qualified for any project on which they work […] by an appropriate combination of education and training, and experience."
  - 3.15: "Treat all forms of software maintenance with the same professionalism as new development."

Professionalität ist keine Fähigkeit, sondern eine Verhaltensweise

- Die bedrohten Anforder-ungen an professionelle Arbeitsergebnisse sind vielfältig

- Aber die häufigsten sind
  1. Geeignete Funktionalität
     - d.h. zweckmäßig
     - das schließt sehr viel sozio-technisches mit ein
  2. Zuverlässigkeit der SW
  3. Sicherheit der SW
  4. Dauerhafte Fortentwickel-barkeit der SW

- Und die wichtigste Bedrohung ist meistens:
  5. Zeitmangel

- Wichtige Fähigkeiten sind deshalb:
  - Teamarbeit
    - wg. 1, 2, 3, 4, 5
  - Domäne verstehen
    - wg. 1, 2, 3
  - Guter SW-Entwurf
    - wg. 2, 3, 4, 5
  - Disziplin
    - wg. 2, 3, 4, 5
  - Verhandeln
    - wg. 5

!

Verhandeln

- Entwickler_innen dürfen unsinnigen Zeitvorgaben nicht nachgeben
  - Beispiel T (lokal)
    - Projektleiter Mike und Entwicklerin Paula
- sondern müssen Umfänge verhandeln
  - Beispiel L (lokal)
    - wieder Mike und Paula
- ebenso das höhere Management
  - und auch, wenn viel auf dem Spiel steht:
  - Beispiel M (lokal)
    - Entwicklungsleiter Don und Firmenchef Chuck
- Wenn eine der Stellen versagt, folgt Unbill
  - Beispiel B, Szene 1 (lokal)
    - Projektleiter Mike (oje!) und Entwicklerin Paula (OK)
      - Das ist Betteln, nicht Verhandeln
  - Beispiel B, Szene 2 (lokal)
    - Projektleiter Mike (ojeoje!) und Entwicklungsleiter Don
      - Das ist Verleugnen.
    - Wer ist hier "Teamplayer"?

(Szenen aus "Clean Coder", Kapitel 2)

Selbstverpflichtung (commitment)

- Am Ende einer erfolgreichen Verhandlung steht die Abgabe einer Selbstverpflichtung
  - "There is no trying"
- Diese hat drei Elemente:
  1. Sagen *"Ja, ich werde X bis zum Zeitpunkt Y erledigen"*
  2. Es auch tatsächlich meinen
  3. Und es dann auch wirklich tun
- Warnsignale: Wörter wie
  - *versuchen, sollte, müsste, würde, wahrscheinlich u.ä., lass uns*
- Eine Selbstverpflichtung kann zurückgenommen werden, wenn Unvorhersehbares dazwischen kommt
  - (Aber es muss wirklich unvorhersehbar sein
  - nicht nur unvorhergesehen
  - oder absichtlich ignoriert.)
- Wenn sie zurückgenommen werden muss, muss sie sofort zurückgenommen werden.

Verhandeln (2)

- Mindeststandards müssen unter allen normalen Umständen eingehalten werden
  - Beispiel H (Medizin) (lokal)
  - Professionell?

- Anmerkung:
  - Ausnahmen, also unnormale Umstände, sind per Definition selten!

Ziele dieses Kurses

- Professionalität im Ganzen reicht weit
  - z.B. zu Personalführung, Kollegialität, Umgang mit Auftraggebern, Organisationsentwicklg., etc.
- Wir beschränken uns hier weitgehend auf eine **persönliche Ebene**
- und dort vor allem auf Praktiken, die **eng am Code orientiert** sind.

- Die Praktiken bewegen sich auf zwei Ebenen:
  1. Wie sieht guter Code aus? (Softwareentwurf)
  2. Wie bekommt man ihn immer besser hin? (Methoden, Disziplin, persönliche Entwicklung)

Hauptquelle

- Der Kurs orientiert sich an clean-code-developer.de (**CCD**)
  - Webseiten von Stephan Lieser und Ralf Westphal
- Deren Ansatz ist ein Stufensystem, in dem man sich mit allmählich komplexer werdenden Praktiken auseinandersetzt
  - immer 3 Wochen und dann nächste Stufe
  - und nach Stufe 6 wieder von vorn!
- Auf jeder Stufe gibt es
  - Entwurfsprinzipien ("Prinzipien"):
    → Wie sieht guter Code aus?
  - Methoden ("Praktiken"):
    → Wie bekommt man ihn immer besser hin?

Stufen 0, 1, 2: schwarz, rot, orange

Clean-Code-Developer Wertesystem

- Die Prinzipien und Praktiken zielen auf die Einhaltung der folgenden 4 Werte:
- Evolvierbarkeit
  - Je erfolgreicher eine SW, desto mehr Änderungen.
  - Die fallen allmählich immer schwerer.
  - Man muss ständig investieren, um SW änderbar zu halten.
- Korrektheit:
  - Wichtig für Zuverlässigkeit
    - auch nichtfunktional
  - Unzuverlässigkeit bedeutet Risiko
- Produktionseffizienz
  - Ingenieurmäßiges Arbeiten ist immer an Sparsamkeit orientiert.
  - Wichtig für Evolvierbarkeit und Korrektheit.
- Reflexion
  - CCD zielt auf persönliche Weiterentwicklung.
  - Ohne Reflexion findet die nur sehr langsam statt.

Weitere Quellen

- CCD wiederum orientiert sich am Buch Robert C. Martin: "*Clean Code: A Handbook of Agile Software Craftsmanship*" *("Clean Code: Refactoring, Patterns, Testen und Techniken für sauberen Code")*
  - für viele SW-Entwickler eine "Bibel"
  - Robert Martin wird oft Bob Martin oder "Uncle Bob" genannt

- Wir verwenden außerdem zahlreiche andere Quellen
  - Bücher, wiss. Artikel, Web-Artikel, C2.com-Wikiseiten, Code
    - siehe die Liste auf der Webseite für einige davon
    - viele mehr erscheinen mit Links auf den Folien
      - ein paar der Links betreffen lokale Dateien (nicht öffentlich)
      - die meisten sind aber öffentlich im Netz

Clean-Code-Developer Lernkonzept

- CCD enthält über 40 Elemente (Prinzipien und Praktiken)
  - viel zu viel, um sie sich in kurzer Zeit wirklich anzueignen (=verstehen + akzeptieren + einüben + angewöhnen)
- Deshalb das Stufensystem:
  - Nehme die nächsten Elemente erst ins Visier, wenn die aktuellen eine Weile "sitzen" (21 Tage ohne Vernachlässigung)

- Die Elemente sind sinnvoll auf die Stufen verteilt
  - so dass oft Grundlagen in einer früheren Stufe gelegt werden
- CCD empfiehlt, die Stufen immer wieder zu durchwandern
  - wir machen das hier im Kurs zwei mal
    - (allerdings jede viel kürzer als 21 Tage)
    - weil man beim zweiten Mal einen neuen Blick auf die Themen hat
    - und auch genug Details zu besprechen übrig sind.
  - Siehe den Stoffplan

Clean-Code-Developer: Bewertung

1. CCD enthält viele wertvolle Ideen
   - Viele sind nur sehr oberflächlich beschrieben.
   - Hier im Kurs etwas genauer.

2. Manches ist etwas überzogen
   - jedenfalls überziehen es viele andere Leute in ihrer Interpretation

3. Vieles Wichtige fehlt
   - und wird das auch in unserem Kurs tun.
   - Insbesondere vieles, das sich auf Teamebene abspielt

Urteilskraft ist gefragt!

- Robert Martin ist oft recht dogmatisch.
- Martin Fowler ist viel differenzierter im Urteil. Sein ähnlich denkender Kollege Brett Schuchert schreibt in Fowlers Blog:
  - *"I think design principles*
    - *Should be "violated" sometimes*
    - *Are often at odds with each other*
    - *Often mix together for something even better than when used in isolation*
    - *Often overlap with other ones*
  - ***There are no free lunches**, all abstractions have a cost."*
    - http://martinfowler.com/articles/dipInTheWild.html

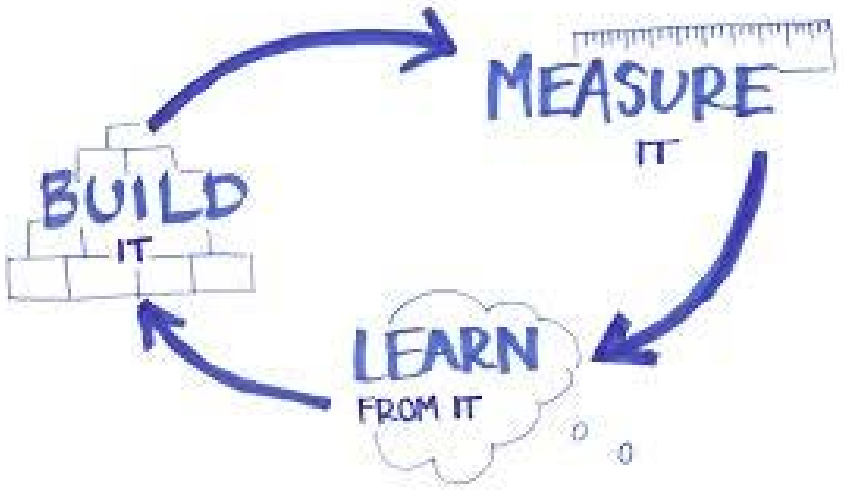
Recht hat er. Bitte nie vergessen!

Urteilskraft ist gefragt! (2)

- Auch die Gegner dieser Ideen sind oft sehr dogmatisch
  - und argumentieren gern mit idiotischen Auslegungen/Beispielen
  - Ron Jeffries nennt das "Idea looking comparatively good next to a straw man"
    - "straw man": Missrepräsentation eines gegnerischen Arguments
  - Bitte immer selbst kritisch mitdenken
    - Erster Schritt: Begriffe klar kriegen
  - **Maß halten!**

Urteilskraft ist gefragt! (3)

- Drittens muss der (ggf.) Nutzen eines Prinzips im konkreten Fall gegen den Aufwand abgewogen werden
- Konzepte:
  - Technical Debt
  - Experimentation



inspiringcapital.ly

Domänenunabhängigkeit??

- Im Prinzip sind die Prinzipien und Praktiken unabhängig von der Anwendungsdomäne

- Tatsächlich orientiert sich viel von unserer Diskussion (Beispiele!) aber an web-basierten Informationssystemen
  - und sind auf andere Domänen nur eingeschränkt zu übertragen

Technologieabhängigkeit

- Wir reden möglichst technologieunabhängig
- Wo nötig, ziehen wir meist zwei Technologiewelten heran:
  - Java — sehr verbreitet, sehr geschwätzig
    - "high ceremony"
  - Python — verbreitet, schlank und lesbar, pragmatisch
    - "low ceremony"
    - Für kleine Vorhaben oft produktiver, für große evtl. schwieriger: http://programmers.stackexchange.com/questions/54630/python-productivity-vs-java-productivity

- andere kommen hauptsächlich (durch Sie!) in den Übungen vor:
  - z.B. C++, C#, Javascript, PHP, Ruby, ...

Dynamische Sprachen: Python

- Entscheidender Unterschied zu Java:
  Keine statische Typisierung
  - sehr wohl aber "starke" Typisierung
- Beispiel: Sortieren
  - Java: braucht Comparable<T>.compareTo(T obj2)
    oder Comparator<T>.compare(T obj1, T obj2)
    - beide liefern -1, 0, 1 für <, ==, >
    - Äpfel mit Birnen vergleichen geht nur mittels Oberklasse Obst
  - Python: eingebaute Operation sorted(iterable[, key][, reverse])
    - Erzeugt neue Liste aus beliebigem Iterable
    - Vergleich erfolgt per <
    - Bei Bedarf zu überschreiben per überschreiben von __lt__()
    - Wenn das nicht reicht, def key(obj)
    - und das kann notfalls auch mit heterogenen Typen umgehen! z.B.:
      - def key(obj): return obj if isinstance(obj, str) else obj.firstname

High vs. low ceremony

- Fowler and Scott:
  - "Projects vary in how much ceremony they have.
  - **High-ceremony** projects have a lot of formal paper deliverables, formal meetings, formal sign-offs.
  - **Low-ceremony** projects might have an inception phase that consists of an hour's chat with the project's sponsor and a plan that sits on a spreadsheet.
  - Naturally, the bigger the project, the more ceremony you need. The fundamentals of the phases still occur, but in very different ways."
    - siehe http://gerrardconsulting.com/?q=node/480

- Das gleiche geht bei Sprachen und deren Entwicklungsstilen:
  - Manches Förmliche ist gleich gar nicht erst vorhanden
    - Insbes. Typdeklarationen und also statische Typfestlegungen
  - Anderes wird zumindest anders gehandhabt
    - z.B. mehr pragmatischer Einsatz eingebauter Datenstrukturen

Danke!

und nun:

Praktischer Teil:

- Jede_r Teilnehmer_in braucht zur Teilnahme ein Projekt:
  - große, überwiegend fremde Codebasis schon vorhanden
  - mehrere Projektmitglieder
  - Fortentwicklung oder Pflege erfolgt jede Woche
- Jede Woche üben Sie ein paar der jeweiligen CCD-Elemente in diesem Projekt ein
  - und manche stellen das dann hier vor
- Jetzt: Die Hälfte von Ihnen stellt in je 8 Minuten ihr Projekt kurz vor
  - siehe nächste Folie
  - Die andere Hälfte nächste Woche

"Mein Projekt"

- In je 8 Minuten zu beschreiben:
  1. Zweck, Funktionalität, Benutzer des Produkts
  2. Alter, Entwicklungsstand
  3. Team, eigene Rolle
  4. Technologie
  5. Entwicklungsprozess
  6. Stärken und Schwächen

- Stärken u. Schwächen (subjektiv):
  - Entwurf
    - Technologiemix
  - Codestruktur, Defektdichte
  - Team
    - Verantwortungsgefühl
    - Investititionsbereitschaft
  - Entwicklungsprozesse
    - z.B. Handlichkeit, Werkzeugeinsatz, Benutzernähe
  - Automatisierungsgrad
    - insbesondere für Test
    - und Freigabe (Release)
  - u.ä.