LESSONS LEARNED FROM A UNIVERSITARY LARGE-SCALE REVERSE ENGINEERING PROJECT

Klaus Bothe
Dept. of Computer Science
Humboldt University Berlin

INFO-TECH 2000
Vrnjacka Banja
20 - 23 June 2000

Subject of the talk

real-life large-scale reverse engineering project:

- control software (physical experiments)

- educational software engineering project at HU

technical aspects of reverse engineering

project organization in an educational environment

.

Practice of software engineers:

- **maintenance tasks: 70 - 90 %**
- **development of new software: 10 - 30 %**

Universities have to provide knowledge and projects based on legacy software

University experience reports: Maintenance and Reverse Engineering Projects

- Carnegie Mellon University, Pittsburgh (Bruegge 94): RE projects based on software previously developed by students at predecessor courses
- Lund University (Sweden) (Wolin 97): students projects with large-scale programs
- Universities of Arizona and Minnesota (Pierce 97):
  - Pascal-S compiler of about 4000 LOC
  - undergraduate project
  - develop requirements and design documents
  - extend functionality and adapt a given test suite.
- University of Twente (Sikkel 99):
  - real-life hospital information system
  - analysed by the students in one week
  - preparation phase: 6 weeks
  - requirements and design had to be considered

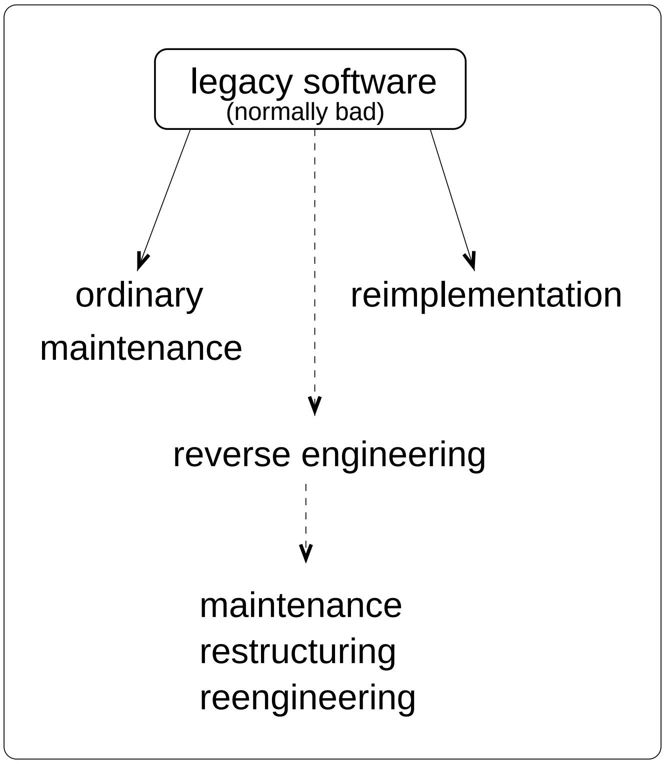
- Some more reports: (annual) Conferences
  - CSEET: Conference on Software Engineering Education & Training
  - SEEP: Software Engineering: Education & Practice
  - SEUH: Software Engineering im Unterricht an den Hochschulen

Limitations (of these projects):

(compared to real-life software)

- requirements stem from lecturer rather from real client
- the project task is academic rather than real-life
- the application field: computer science itself
- only some phases are included
- only a prototype was developed rather than a production version
- (partly) rather moderate size

Reverse Engineering: the 3rd way ...



Definitions: reverse engineering, reengineering ...

(IEEE Software 1990)

Reverse engineering:

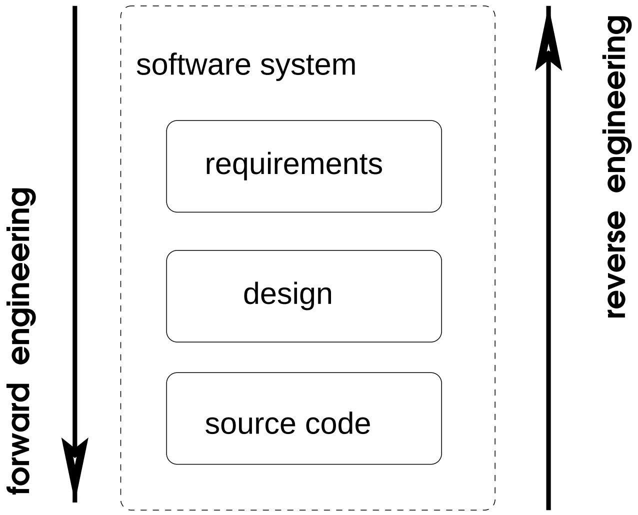
the process of analyzing a subject system with two goals:

(1) to identify the system's components ...
(2) to create representations of the system in another form or at a higher level of abstraction

Reengineering:

the examination of a subject system to reconstitute it in a new form

Forward engineering vs. reverse engineering



characteristics of team-oriented project courses

factors	attributes
application domain	computer science - others: economy, natural sciences ...
problem source	academic (artificial) - real-life
client / customer	lecturer - non-commercial - commercial
sponsor (money ...)	no - yes
goal project status	prototype - production version
project priority	low - high: without deadlines / with deadlines
project size	low - medium - large scale
phases (and documents)	requirements - design - code - test - maintenance (RE)
original project status	new project: forward eng. - existing project: reverse eng.

Our Project Characteristics (1)

Our position: The closer we approach real-life attributes, the better the students' motivation, the learning effect, and the overall success of the project.

- *application domain:*
  - outside computer science
  - software-controlled physical experimental devices (X-ray topographic cameras)
    ⇒ domain-specific knowledge
- *real-life* requirements:
  from researchers of the department of physics

Project history: the beginning

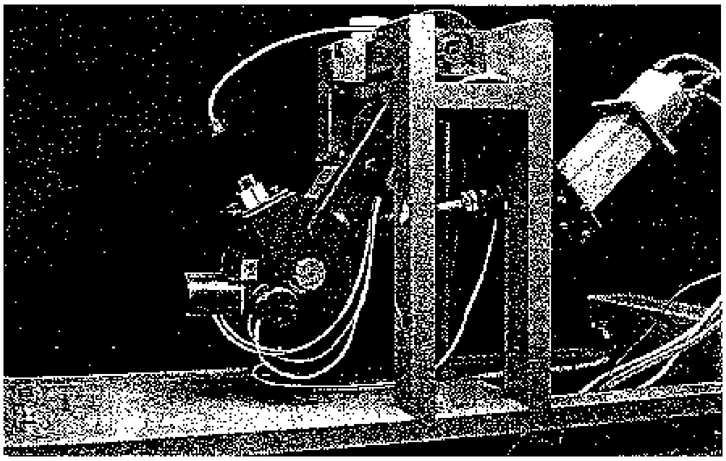
Email from previously unknown professor of the Department of Physics of HU

" ... I refer to you with the request for advice. We are an experimentally working group in the field of X-ray defraction on layer systems. In our laboratory 10 PCs are controlling X-ray topographic cameras. Some years ago, a software engineer has been developed a C++ program for Windows 3.11. Unfortunatly, the program is not very readable. My question: Could you give us advice how to gain a sound software base and how to maintain it with a reasonable effort?"

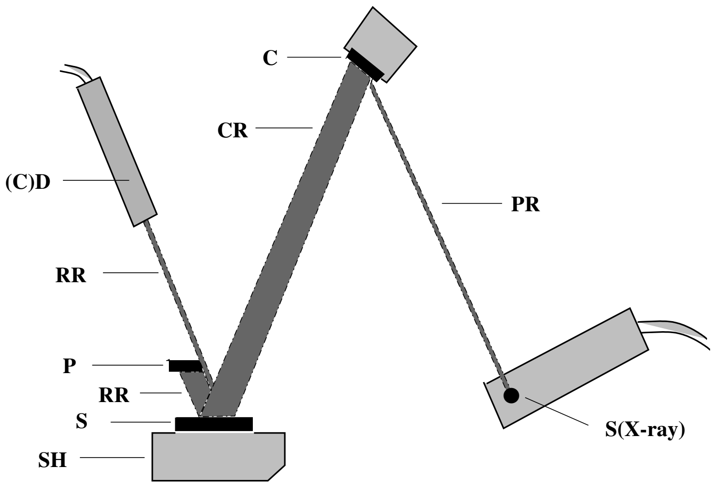
Question catalogue:

1. Is the software engineer still available?
2. How well is the program documented or commented?
3. Are there any further documents: user manual, requirements specification ... ?
4. What is the size of the program in LOC?
5. Which C++-system is the basis of your development?
6. Is there any special design method you applied (e. g. oo design, structured design)?
7. What are your problems: errors, stability, portation, extentions ...?
8. What do you think is the right way to proceed: restructure the system or rewrite it?
9. When do you need the results?
10. Who should maintain the software in the future?

Photo of X-Ray Topographic Camera



Scheme of X-Ray Topographic Camera



S (X-ray): source of X-ray; PR: primary ray; C: collimator; CR: collimated ray; SH: sample holder; S: sample; RR: reflected ray; P: photo plate; (C)D: (control-)detector

Our Project Characteristics (2)

- *existing* program:
  typical *reverse engineering situation*:
  - original software developers no more available
  - no additional documents
    (requirements specification, design ...).
- program *size*: 27.000 LOC
- project as a whole:
  - *all phases* and their respective documents
  - ultimate goal:
    production version
- project *priority* (for the physicists): *high*
- *deadlines:* cannot be guaranteed by students' projects:
  ⇒ not a too big problem: academic partner rather than a sponsor.

Experience report

(based on 3 1/2 semesters project course work)

- basic state of the legacy software
- project goals
- applied reverse engineering strategy: tools vs. manual methods
- organizing multi-semester projects with variable group members
- project reuse and evolving project maturity
- decomposition of tasks
- ordinary project management applies to reverse engineering projects: meetings, protocols, guide lines, version control, project repository
- superiority of reverse engineering over forward engineering students' projects
- motivating factors
- advantages of non-commercial clients

Step 1: basic state of the legacy software

(static manual analysis: code review ...)

deficiencies:

- ill-structured software architecture
  (no clear decomposition of the 27 KLOC)
  → burdens the decomposition of work amongst the students
- mixture of imperative C-style code and object-oriented C++-code
- bad layout
- dead code (e. g. intended extentions)
- violation of principles of software ergonomy
  (dialog windows)
- non-availability of additional documents and poorly commented
  → all information has to be drawn from program sources
- instability of the system in particular situations
  (partly non-reproducable)

Step 1: basic state ... (cont.)

positive aspects:

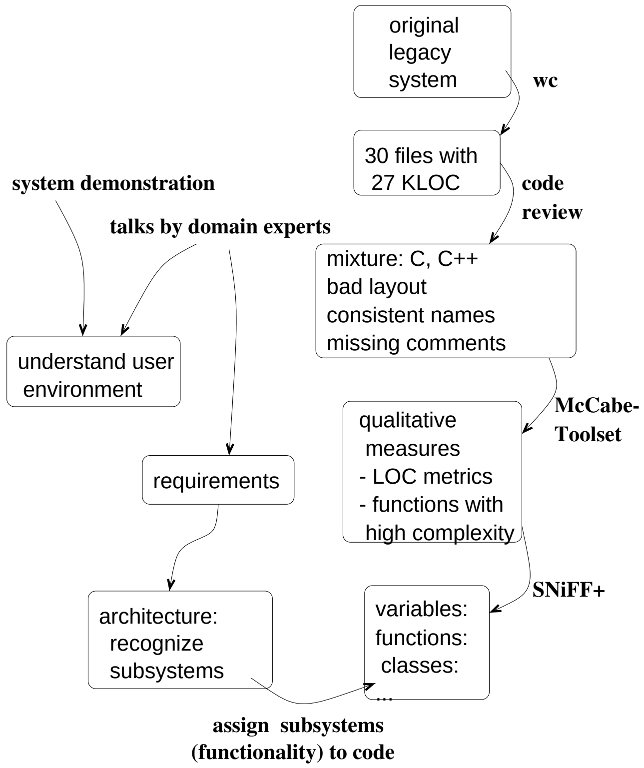
- unified programming style
- consistent (private) naming conventions
- sound imperative interfaces (functions)
- some useful class hierarchies

Project goals:

- *Reverse engineering* of the system:
  - development of missing documents
    (requirements specification, user manual, design, test cases ...)
  - architecture recovery
  - program understanding (commenting the sources)
- Fulfillment of *client requests*
  (*reengineering, restructuring*)
  - stability of the system
  - extentions: new devices ...
  - portation to an actual operating system
- *Educational goals:* team work, project management, real-life project, production version, application outside computer science, contacts with real clients ...

→ A poor reverse engineering project would not have been successful since reverse engineering activities alone never would have satisfied the partner (absolutely necessary as the domain expert)

Applied reverse engineering strategy: tools vs. manual methods



Reverse Engineering Tools: Which are their properties?

SNiFF+

McCabe Toolset

ParadigmPlus

Rational Rose

Together/Professional

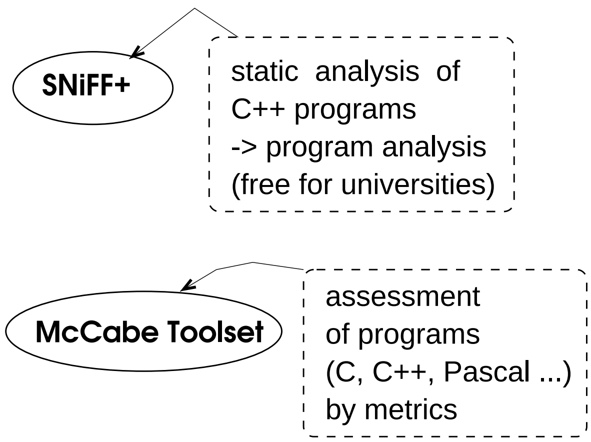
Bauhaus

URCA

Kapsul-Tool

Renaissance System

Reverse Engineering Tools: quite different functionalities (1)



ParadigmPlus

Rational Rose

Together/Professional

Bauhaus

URCA

Kapsul-Tool

Renaissance System

Reverse Engineering Tools: quite different functionalities (2)

SNiFF+

McCabe Toolset

ParadigmPlus

Rational Rose

Together/Professional

architecture recovery (UML diagrams),

prerequisite: object-oriented designed program

Bauhaus

URCA

Kapsul-Tool

Renaissance System

Reverse Engineering Tools: quite different functionalities (3)

SNiFF+

McCabe Toolset

ParadigmPlus

Rational Rose

Together/Professional

-> how to find architectural components?

URCA (Bojic u. a. Belgrade)

architecture recovery, esp. for imperative programs

Bauhaus (Kosche, Stuttgart)

Kapsul-Tool (Kloesch,Gall, Vienna)

-> how to find objects and classes ?

Reverse Engineering Tools: Which did we apply?

SNiFF+ ✓

McCabe Toolset ✓

ParadigmPlus ✓

Rational Rose

Together/Professional

Bauhaus

not applicable: does not accept C++

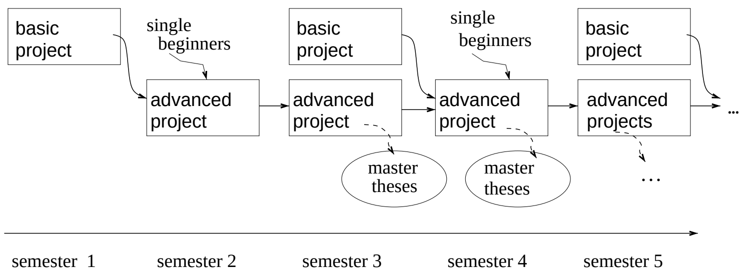
not applicable:
- only 255 functions
- Borland - not Visual C++

URCA

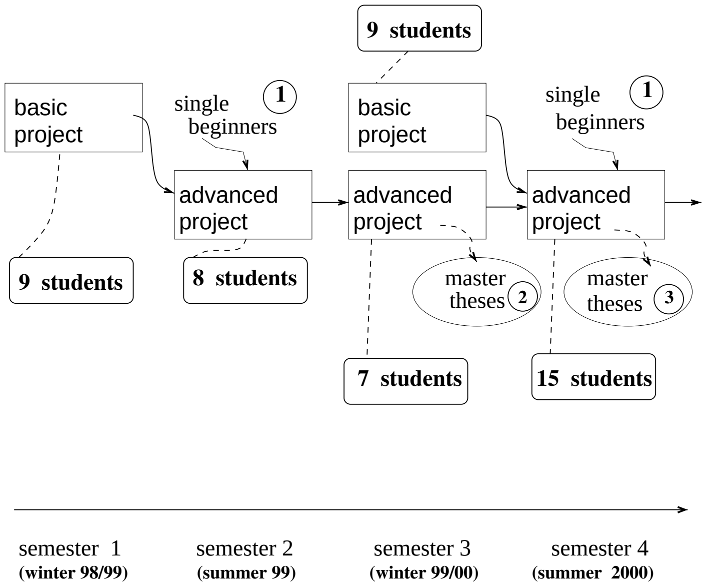
Kapsul-Tool

Renaissance System

Organizing multi-semester projects with variable group members



Project courses : statistics



Large-scale projects: extend beyond one or two semesters

- basic project model:
  - open end
  - open number of participants
- three levels of experience: introductory, advanced, expert
- heterogenous group members:
  - participants in different curriculum phases: 5th - 11th semester
  - knowlege rather different
  - different current educational goals

Project reuse and evolving project maturity:

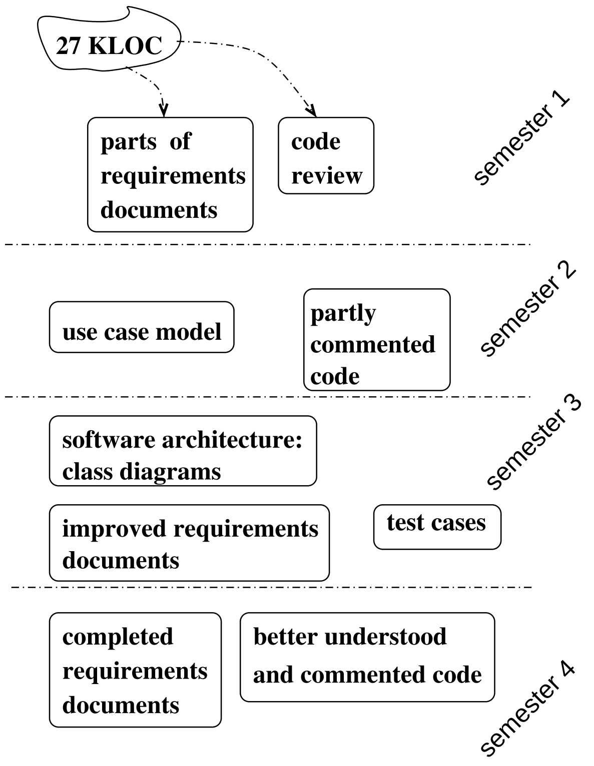
The supervision of universitary project courses is expensive for lecturers - large-scale real-life projects increase the effort dramatically ...

- *"Projects of the latter category (unknown application domain) require large investment by the instructor to set up the infrastructure for the course. It would be extremely useful if this structure could be reused ...* (B. Bruegge, SEUH '94, p. 67).
- Pascal-S compiler maintenance project course (Univ. Arizona and Minnesota, Pierce 97): reused and evolved over the years 1988 -1997

Efforts for the instructor:

- organizing the contact with the clients
- learn about the application domain: e. g. to assess requirements documents
- assess the legacy program: convenient in project course?
- project management: control the students' activities

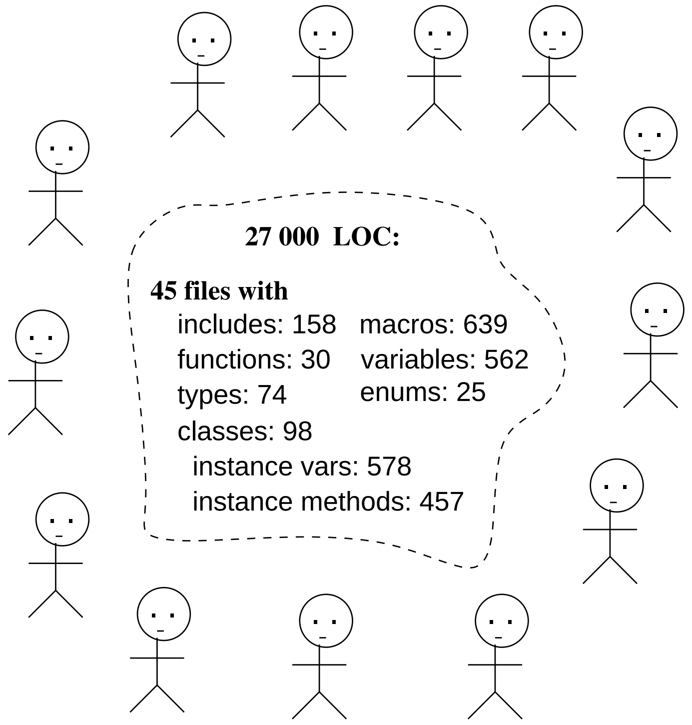
Our project reuse: over years + evolving maturity



-> The project evolves from semester to semester.

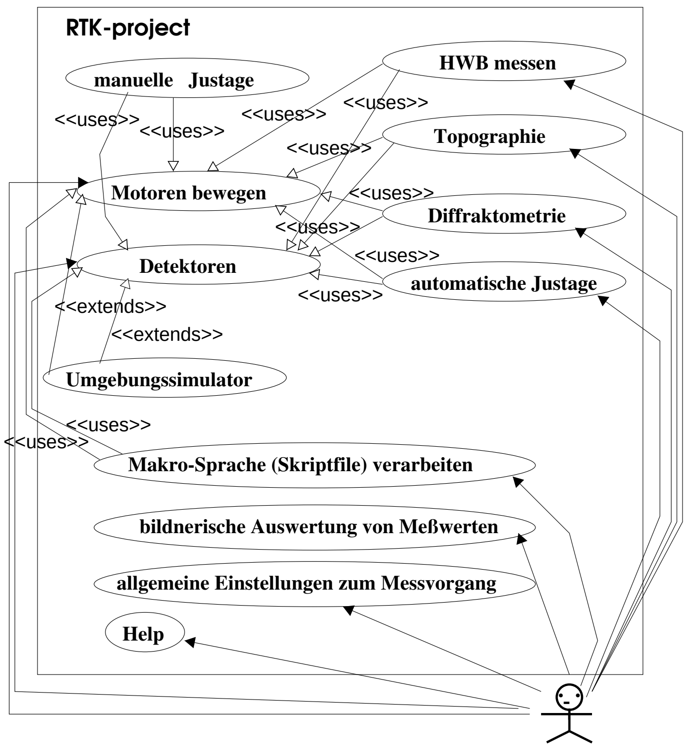
-> Each subsequent class builds upon the work of the previous class.

Decomposition of tasks: original state



Decomposition of tasks: determine subproblems by use cases

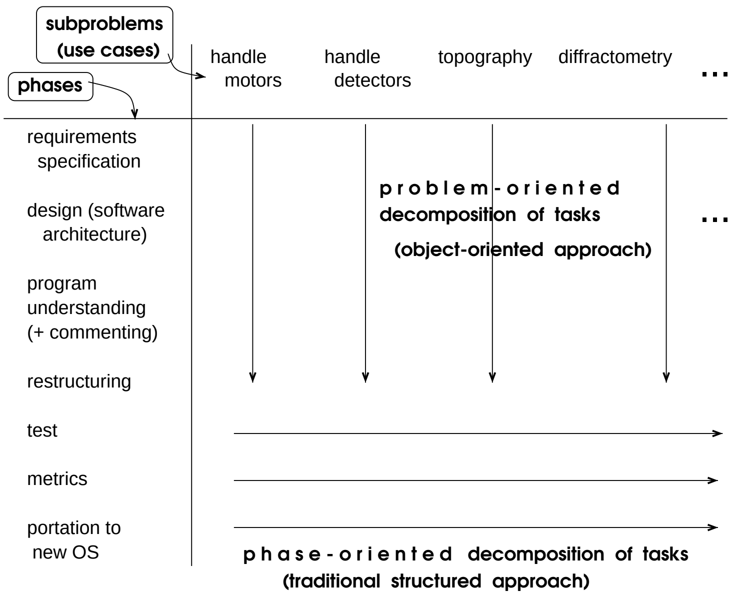
Use Case Diagram (in German language)



Decomposition of tasks: employ specialists for certain phases

- requirements engineer
- software architect
- programmer
- test engineer
- software quality engineer
- …

Decomposition of tasks: phase-oriented vs. problem-oriented



Ordinary project management applies to reverse engineering projects:

- meetings,
- protocols,
- guide lines,
- version control,
- project repository

Superiority of reverse engineering over forward engineering students' projects

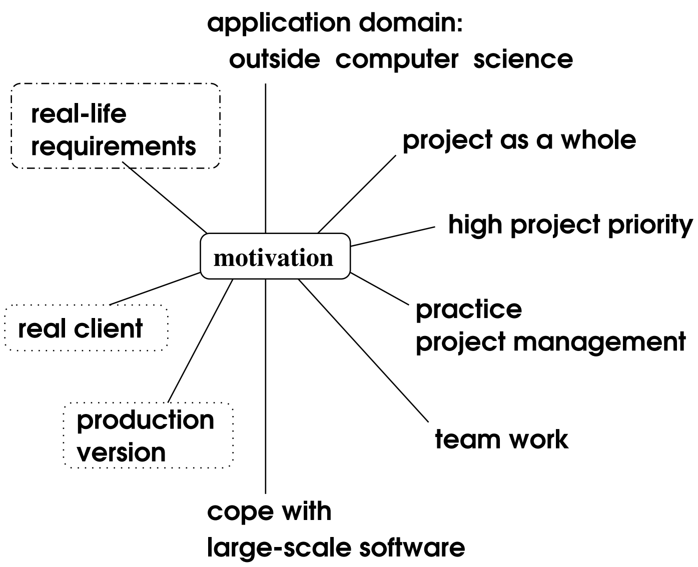
- Reverse enginnering projects: students are immediately confronted with large-scale software

  (FE projects: it takes time until large software has been developed)
- Building large-scale software *from scratch* takes much effort und much time: In a universitary educational environment these projects generally end with disastrous results or remain incomplete
- Legacy systems represent much men power performed by others:

  - implementation strategies and techniques
  - testing effort

  $\rightarrow$ RE projects can reduce the coding and testing pressure
- legacy software: learn about a software solution to a application domain

Motivating factors



Advantages of non-commercial clients

- no support (money)
- → no demands, only cooperation
- → no deadlines
- less problems with legal issues (copy source code)

Summary

- Large-scale reverse engineering projects are more likely to succeed than forward engineering projects in an educational environment
- Success factors:
  - multi-semester project
  - three levels of project participation: introductory, advanced, master theses
  - extensive reverse engineering phase (without modifications of the code, for understanding the code, the structure, and the requirements of the software)
  - motivation: real-life requirements, real client, interesting application domain, learning about project management ...