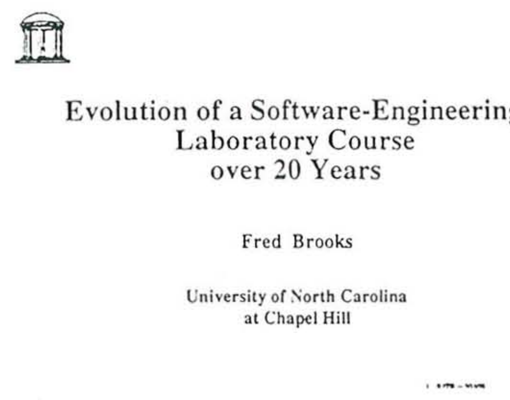
EVOLUTION OF A SOFTWARE ENGINEERING LABORATORY COURSE

F P Brooks

Rapporteur: Jonathan Halliday

EVOLUTION OF A SOFTWARE ENGINEERING LABORATORY COURSE

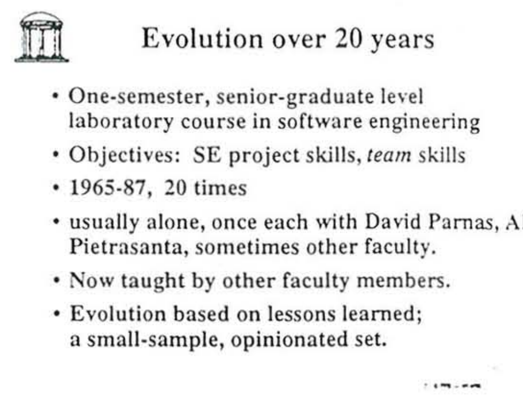
Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill 27599-3175 USA



Evolution of a Software-Engineering Laboratory Course over 20 Years

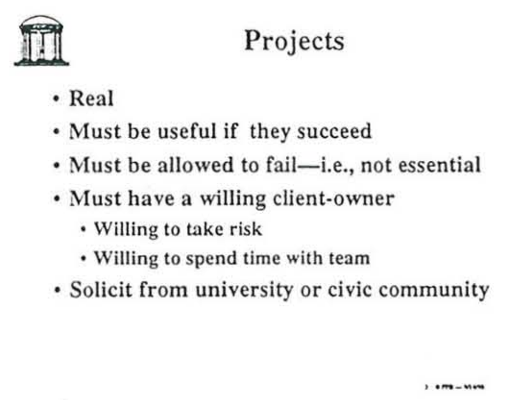
Fred Brooks

University of North Carolina at Chapel Hill



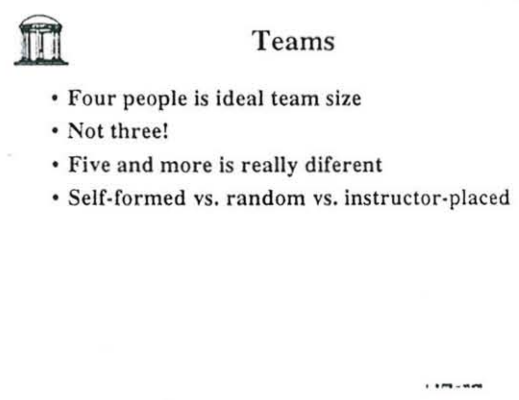
Evolution over 20 years

- One-semester, senior-graduate level laboratory course in software engineering
- Objectives: SE project skills, *team* skills
- 1965-87, 20 times
- usually alone, once each with David Parnas, Al Pietrasanta, sometimes other faculty.
- Now taught by other faculty members.
- Evolution based on lessons learned; a small-sample, opinionated set.



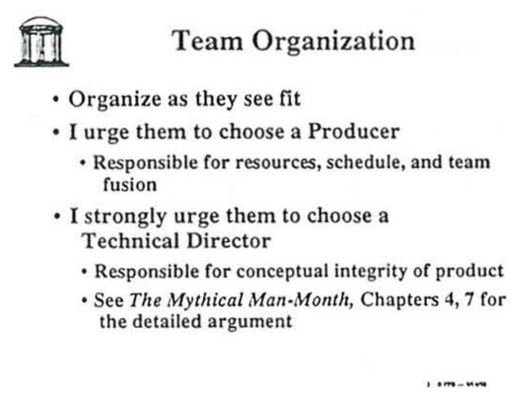
Projects

- Real
- Must be useful if they succeed
- Must be allowed to fail—i.e., not essential
- Must have a willing client-owner
  - Willing to take risk
  - Willing to spend time with team
- Solicit from university or civic community



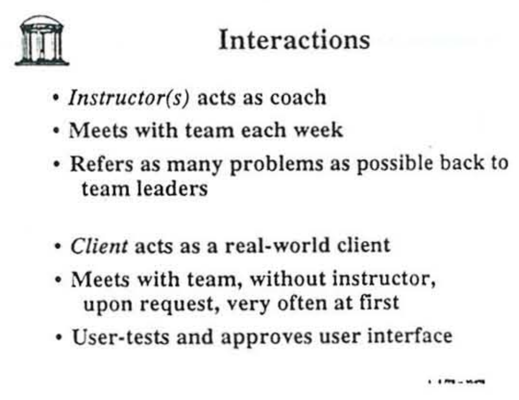
Teams

- Four people is ideal team size
- Not three!
- Five and more is really diferent
- Self-formed vs. random vs. instructor-placed



Team Organization

- Organize as they see fit
- I urge them to choose a Producer
  - Responsible for resources, schedule, and team fusion
- I strongly urge them to choose a Technical Director
  - Responsible for conceptual integrity of product
  - See *The Mythical Man-Month,* Chapters 4, 7 for the detailed argument



Interactions

- *Instructor(s)* acts as coach
- Meets with team each week
- Refers as many problems as possible back to team leaders

- *Client* acts as a real-world client
- Meets with team, without instructor, upon request, very often at first
- User-tests and approves user interface

EVOLUTION OF A SOFTWARE ENGINEERING LABORATORY COURSE

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill 27599-3175 USA

Lectures

- One hour/week
- *The Mythical Man-Month* (1975), grew out of the original lectures
- Later, more carefully synchronized with their project experience
- Today I would include different material, especially from DeMarco & Lister, *Peopleware*
- Present instructor emphasizes formal methods

Schedule

- Week 2—Project description and schedule
- Week 4—Produce a draft user manual and requirements document
- Week 6—Prototype graphical user interface
  - Must be tested with users
  - Magic for team fusion and morale
  - Accelerates progress
- Week 8—Design document
- Week 14—Demo and final product

Design Reviews

- Mid-term, just before (or after) design document
- Done by industry-experienced students who were taking lectures, waiving project
- Could be done by teams for each other
  - I haven't tried that
- Really helped the designs, *and* the teaching of methodology

Final Product

- Program
- User documentation
- Maintenance documentation
- Demonstration for client, classmates, friends and department public
- Demo documentation, foils, etc.
- Time logs by week and 9 kinds of activity, to help them calibrate themselves
- "Objectives, achievements, team self-evaluation, lessons-learned" document

Sample Time Log

Comp 145
Spring 1986

TIME LOG

NAME ____________
Project: CMOS router

Week Ending	Reqts. incl. User Manual & Meetings	Learning Tools	Internal Design	Coding	Lines	Comp. Test	System Test	Docs	Demo.	Totals
1/25	3	5 1/4								8 1/4
2/1	2	6 1/4								8 1/4
2/8	2	1 1/4	3							6 1/4
2/15	8.75		4							12.75
2/22	1.75		1	4	300	7				13.75
3/1	1.5					9				10.5
3/8	.75					.5		1.5		2.75
3/15								8		8
3/22	1.75	1	11					9		22.75
3/29	1.75		7	5	400	2				15.75
4/5	2			5	500	11				18
4/12						18				18
4/19	1.5			10	1100	9		4		24.5
4/26	2						3	12	7	24
Totals	26.75	13.75	26	24	2300	56.5	3	34.5	7	191.5

Include keying time under coding, test-case generation with component test or system test as appropriate, Don't count comments.

Show and Tell

- Public demo at end of course, week before final product is turned in
- Presentation of need, design, product working, lessons learned
- Every team member must present something
- An important skill we too rarely teach

EVOLUTION OF A SOFTWARE ENGINEERING LABORATORY COURSE

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill 27599-3175 USA

Grading

- Must include a team component
  - "There are no losers on winning teams; there are no winners on losing teams."
  - "The hole isn't in my end of the boat!"
- Must include an individual component
- Instructor grade
- Teammates' grades
  - Each team member gets a grade budget to assign
  - Teammate grading really works!

Sample Team Grading T= /10

- Achievement 3.0
  - Ambition 1.0
  - Accomplishment 2.0
- Usability 3.0
  - User Manual-1 0.5
  - User Manual-2 1.5
  - Ergonomics 1.0
- System Quality 3.0
  - Code qual 0.6
  - TechManual-1 0.4
  - System Design/Tech Manual-2 1.0
  - Testing 0.6
  - Client satisfaction 0.4
- Presentation and demonstration 1.0

Individual Grading

- Code
  a. Quantity 10 b. Quality 10
- Documentation
  c. Quantity 10 d. Quality 10
- Other
  d. Technical mastery 10 e. Effort, teamwork 10
- t. Teammates' average evaluation =t/1.0
- Individual grade I= (a+b+c+d+e+f)/6 * t
- Total grade = I+ T ; perfect grade = 20

What Didn't Work?

- Big teams
- Mixing graduate students and undergrads on same team
- Two projects in one semester
- Vague project definitions
- Weak or inattentive clients
- Too much attention to process, as opposed to product
- Slow instructor reaction to infighting

What Worked Really Well?

- Self-selected teams
  - Quick fusion, high morale, esprit de corps
- Teams that formed around own project idea
  - Highest yield of great projects
  - Drove themselves unmercifully
- Projects using body of tools already in place
  - Randy Pausch reports same result for his 1997-98 graphics project course at CMU
- Teams with eager clients

A Parnas Experiment

- Divide class into A and B groups
- Simple task—a reader-writer
- Each person in A builds a read module
- Each person in B builds a write module
- A-B pairs drawn at random and tested for proper functioning
- Dramatic lesson in information-hiding, building to a precisely specified interface
- Cost 3-4 weeks of a 14-week term

EVOLUTION OF A SOFTWARE ENGINEERING LABORATORY COURSE

Frederick P. Brooks, Jr.
University of North Carolina at Chapel Hill 27599-3175 USA

If a Year-Long Course?

- Could be *much* better
- Much more instruction and practice in technical, as opposed to project management, methodology
  - Especially teach good style within the object-oriented paradigm
  - More theory and practice on code control and file management
  - Richer prototyping cycle
- Would definitely include the Parnas mini-project on interchangeable parts

Course vs. On-the-Job—3x?

- Planned short project, selected for right scope
- Enriched by able colleagues
- Concentrated—little busywork per lesson
- Broad experience—each one does code, docs
- Guided—"How-to" lectures at right time
- Coached and commented—as you go
- Critiqued at end—especially documents, code
- Shared—learn from other teams
- Reflected upon—by team; "lessons" document

Results

- Alumni reaction—"the best course I had at UNC"
  - "Bragging and complaining"
- Some projects produced finished products that were put into production.
- Out of 5-8 projects/year, one project, on average, failed.
- Many useful systems have been prototyped by course projects.
- Many PhD dissertations have grown from course projects.

DISCUSSION

Rapporteur: Jonathan Halliday

Lecture Two

The speaker was asked to elaborate on the method by which student project groups were allowed to self-form. Professor Brooks responded that a menu of projects, usually around twice the number that would actually run, was posted for students. Individuals or groups could then indicate their first, second and third choices. If a group indicated they wished to work together this request would be honoured, although they would not necessarily get their first choice of project. Brooks noted that this method led to strong teams, but had the disadvantage of not allowing all of the weaker students to work with more able colleagues.

Professor Brooks stated that the construction of a prototype user interface part way through the project caused a great increase in team fusion and morale. A member of the audience asked him what he felt was the cause of this. Professor Brooks replied that giving the software tangible form created a kind of magic - the students could see it, feel it, realised that it was no longer just abstract code, that it would work. The comment was made that creating a new machine was always daunting and uncertain, but once you had succeeded in creating the basic function you had confidence that the rest was also possible. It was also observed that morale was better if the students used actual production tools for creating the prototype, rather than special prototyping tools. This was attributed to a greater confidence being gained in the use of the tools necessary to create the final version of the software.

In relation to the project marking scheme outlined by the speaker, it was observed that estimating one of the elements, code quantity, was very difficult. Code reuse, copying, teamwork and other factors all made estimation complex. Brooks replied that he was aware of these problems and that the marking of this area was frequently based on highly subjective judgements by the instructor. Another member of the audience supported the use of team members to grade one another, and additionally noted that in his experience using teams to grade each other also led to good, consistent marks. Finally on this topic, Professor Brooks was asked why client satisfaction was awarded such a low rating in the marking scheme. He replied that the client often could not see many of the key elements of the project, including important teamwork and software engineering aspects.

In the discussion following the presentation, the following points were made:

It was observed that, in running similar courses, members of the audience had experienced the group project taking up a considerable amount of student time. This often exceeded the timetabled allowance, to the detriment of other ongoing courses. The speaker agreed with this observation, noting that in his experience dedicated groups could often greatly exceed their scheduled time allowance, which was around one third of the timetabled hours in the semester.

A member of the audience commented that, in his experience, similarly structured projects worked well in industry training situations as well as in academia.

Some further questions were raised in regard to the marking of projects. Firstly, were any statistics available for average marks, best mark ever, etc? Were any trends apparent? Professor Brooks replied that he had no such information readily available. Secondly, the difficulty of comparative grading of different projects was raised. Professor Brooks responded that, once again, a high degree of subjective judgement was necessary on the part of the project supervisor. Finally on this topic, the difficulty of instructors being required to mark code written in unfamiliar languages was discussed. Here again it was felt that subjective judgement was the only available means of grading work.

Professor Randell, in outlining the structure of group projects employed at Newcastle University, commented that in his experience the best teams often had members from a wide range of backgrounds and experience.

The discussion ended with observations on the problems of students learning inductively, whilst teaching was conducted in a mostly deductive manner. It was observed that because of this it was unsurprising that (inductive) lab courses, although expensive in terms of time and money, should benefit the students so greatly. Another speaker, Professor Crampton Smith, observed that in her field the teaching was almost entirely inductive. It was felt that computing science could benefit from more teaching of this nature.