Ptolemy Software Practice

John Reekie

UC Berkeley

With a lot of help from:

Christopher Hylands

Edward A. Lee

Ptolemy II

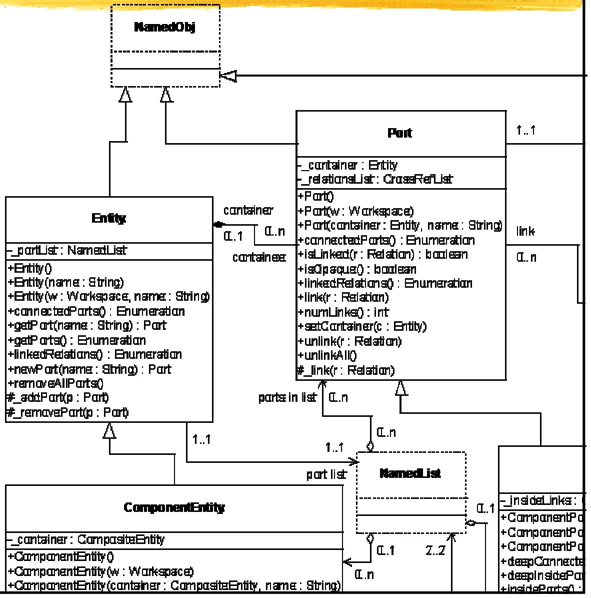
Diva

Motivation

- Increasingly, software is a *publication medium* for academic research
- But increasingly, process-oriented methodologies (e.g. SEI CMM) are seen as irrelevant to academic *practice*
- We need better techniques, and better communication between developers:
  - Can "best practice" techniques improve "research" software?
  - How do we maintain creativity and excitement?
  - What is the cost of improved quality in a research environment?
  - How do we introduce and *maintain* new practices?

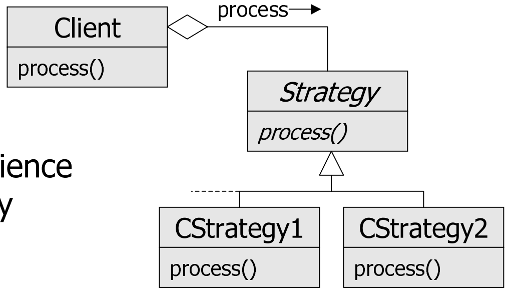
UML (Unified Modeling Language)

- We started with OMT, and migrated to UML
  - We now use static structure diagrams consistently
  - We need to work on interaction diagrams next
- This diagrammatic notation is concise and effective



Design patterns

- A high-level vocabulary for describing recurring patterns:
  - Strategy
  - Composite
  - Factory
  - Template method
- A way of factoring experience into concrete terminology
- We studied the most important patterns from Gamma *et al*



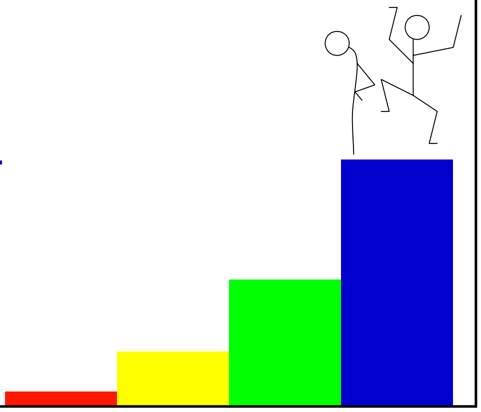
Formal reviews

- An invaluable technique for increasing communication, visibility, and quality
  - "Formal" means that a well-defined process is followed
  - The emphasis is on defect detection, not on who is "right"
- Process of adoption
  - Study group with readings from McConnell and NASA SEL
  - Study group performing a mock code review
  - Study group performing a mock design review
  - Incorporation into our code rating system
  - Practice!
  - Continual refinement

> *All technical reviews are based on the idea that developers are blind to some of the trouble spots in their work...*
>
> Steve McConnell

Code rating

- A simple *framework* for
  - quality improvement by peer review
  - change control by improved visibility
- Four confidence levels
  - Red. No confidence at all.
  - Yellow. Passed design review. Soundness of the APIs.
  - Green. Passed code review. Quality of implementation.
  - Blue. Passed final review. Backwards-compatibility.
- What is this about really?
  - *Confidence* in quality
  - *Commitment* to stability



The code rating “FAQ”

- I need to change green code
  - Change is part of software, so:
  - Change it, and rereview it.
- This adds a lot of extra work
  - It does if you’re not going to do reviews anyway
  - You don’t have to review everything. Use your judgment!
- This is a waterfall model
  - The rating applies to individual classes, not the whole system!
  - Makes the “normal” progress of code visible.

How we do a review

- Top level
  - The author announces that the package is ready for review
  - The moderator organizes and moderates the review
  - The author responds to the issues raised in the review, redesigning or reworking as necessary
  - The author announces the new rating.
- In the review
  - The *moderator* runs the meeting and keeps the discussion on track; and acts as *reader* (in our process).
  - The *reviewers* raise issues and defects
  - The *author* answers questions
  - The *scribe* notes raised issues and defects
  - *Nobody* attempts to find solutions!

Roles define and clarify responsibility

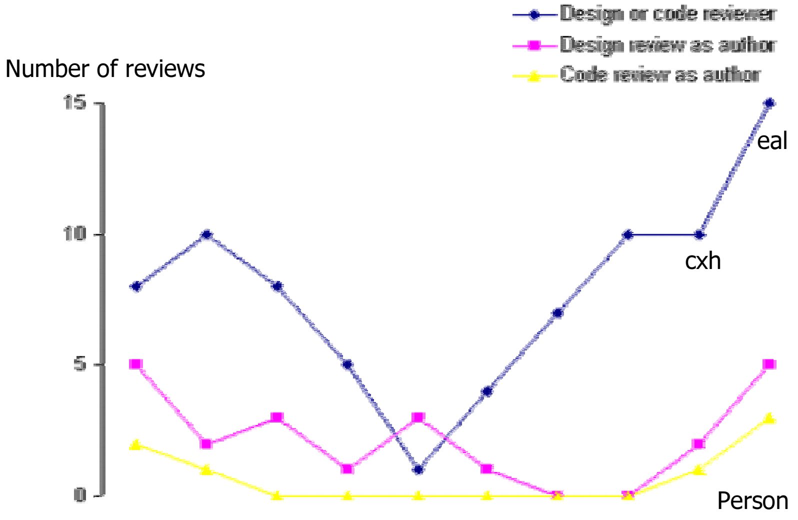
Our extensions and clarifications

- Things that make life easier
  - Preallocate two time slots per week for reviews
  - Choose a scribe beforehand. Use a laptop.
  - Require that the review material is *stable*
  - Don’t try to review too much
  - If the meeting runs out of time, *stop*, and schedule a second one
- Things that make reviews more useful
  - Use UML and Javadoc for all design reviews
  - Create detailed “how-to” check-lists for moderator and author
  - Create a Web page for each review
  - Link review materials from the Web page

Is it effective?

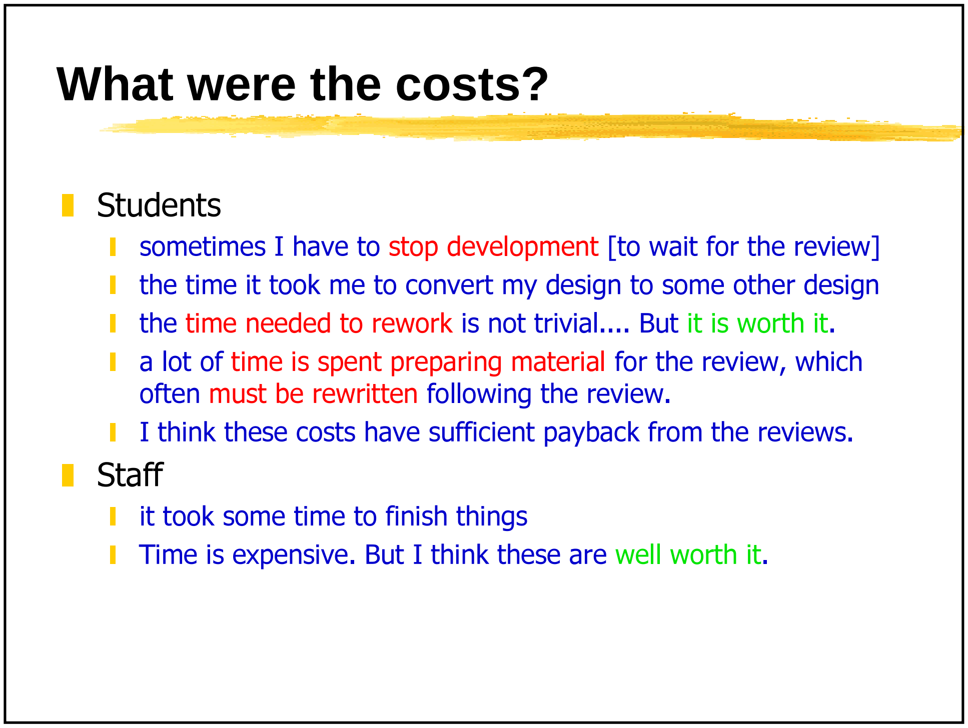
- JohnR conducted a survey
  - Ptolemy II and Diva people
  - Responses were allowed to be anonymous
  - All those involved in reviews (and still at UCB) responded
- Issues the survey was intended to examine
  - How many reviews are we actually doing?
  - What do the reviews gain?
  - What do the reviews cost?
  - What do we need to fix?

How many reviews?



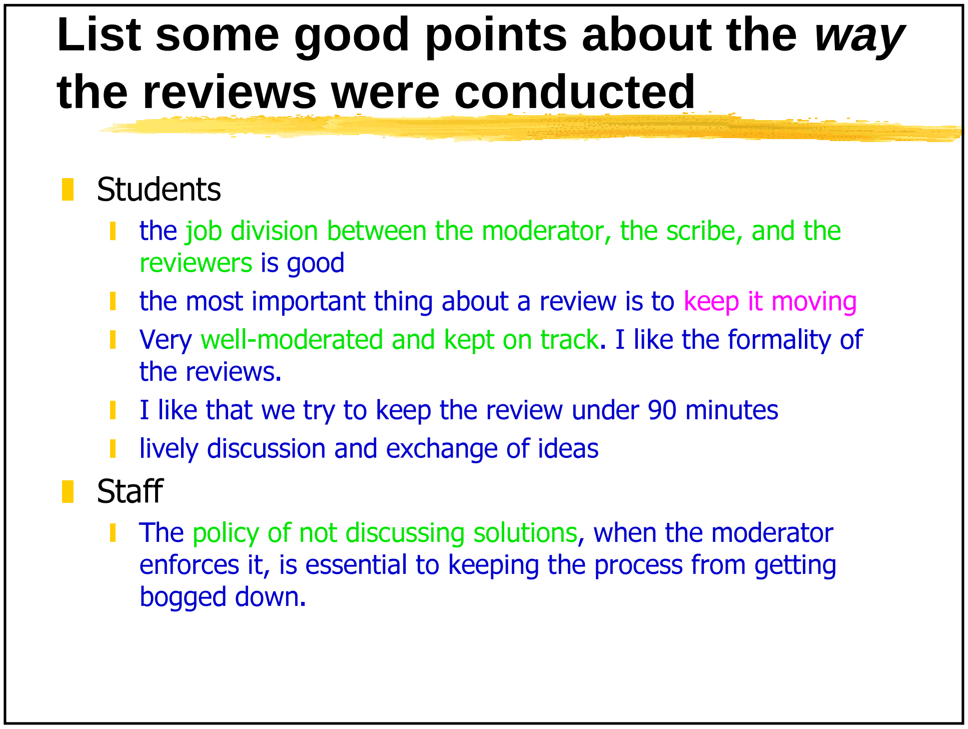
What were the review benefits?

- Students
  - better design and more confidence.
  - good feedback about documentation and naming issues
  - revealed quite a few flaws
  - an affirmation that your architecture is sound
  - encourage other people in the group to reuse code
  - forcing function to get documentation in order
  - my coding style changed
- Staff
  - exposed quite a few design flaws
  - caught lots of minor errors, and quite a few insidious errors



What were the costs?

- Students
  - sometimes I have to stop development [to wait for the review]
  - the time it took me to convert my design to some other design
  - the time needed to rework is not trivial.... But it is worth it.
  - a lot of time is spent preparing material for the review, which often must be rewritten following the review.
  - I think these costs have sufficient payback from the reviews.
- Staff
  - it took some time to finish things
  - Time is expensive. But I think these are well worth it.



List some good points about the *way* the reviews were conducted

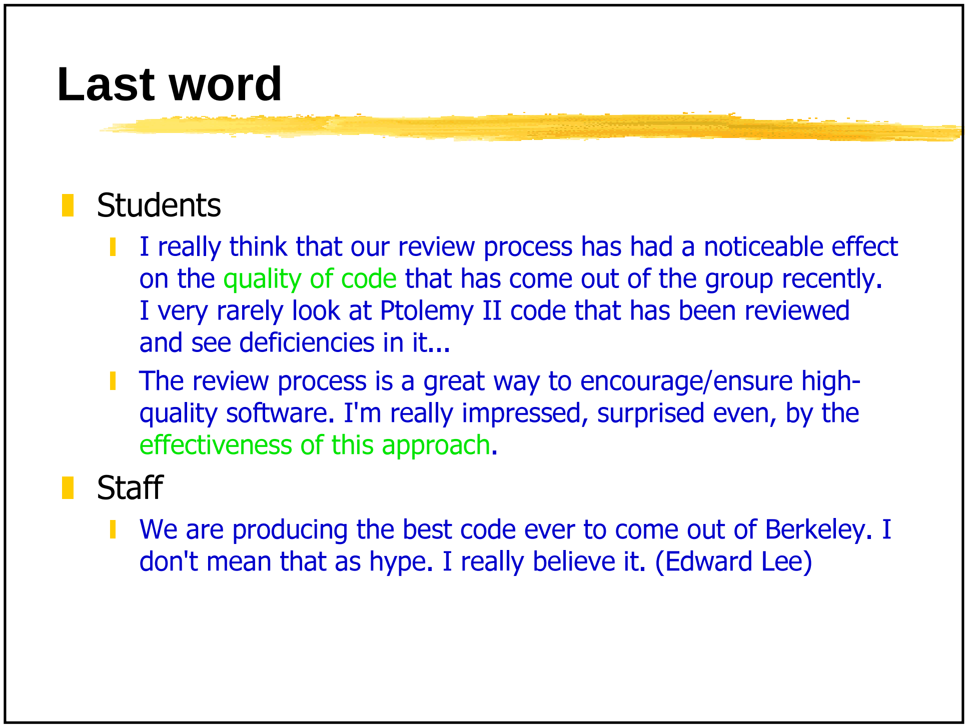
- Students
  - the job division between the moderator, the scribe, and the reviewers is good
  - the most important thing about a review is to keep it moving
  - Very well-moderated and kept on track. I like the formality of the reviews.
  - I like that we try to keep the review under 90 minutes
  - lively discussion and exchange of ideas
- Staff
  - The policy of not discussing solutions, when the moderator enforces it, is essential to keeping the process from getting bogged down.

List some *bad* points about the *way* the reviews were conducted

- Students
  - the reviewers weren't familiar with the code… suggestions were limited to some typos and gratuitous changes
  - reviewers that have read the code, but don't understand the architecture
  - the author didn't really address most of the points raised
  - reviewers are nit-picking over the most trivial little details
  - the approach of "say something positive first and then criticize" is not applied
- Staff
  - We are always rushed for time. Some people are not prepared.
  - Sometimes the reviewers and moderator focus on trivialities...

Other concerns

- Students
  - Reviews where a strong personality modereadered (a new word!) tended to stay on track better.
  - It would be nice to look at the test suite at some point, say after design review and before code review.
  - the notes that most scribes take are very brief and hard to follow outside of the context of the review
  - I would suggest that more than one person take charge of a domain.
- Staff
  - The review process breaks down when we are in a crunch for a demo.



Last word

- Students
  - I really think that our review process has had a noticeable effect on the quality of code that has come out of the group recently. I very rarely look at Ptolemy II code that has been reviewed and see deficiencies in it...
  - The review process is a great way to encourage/ensure high-quality software. I'm really impressed, surprised even, by the effectiveness of this approach.
- Staff
  - We are producing the best code ever to come out of Berkeley. I don't mean that as hype. I really believe it. (Edward Lee)