Do-Review-Redo

A CRITIQUE-BASED ALTERNATIVE TO HOMEWORK, EXAMS AND GRADES

Outline

Critique-Based Continuous Assessment

Examples

Pedagogical Connections

Observations

- Support Tools
- Materials
- Student responses
- Scaling

The Model

Critique-Driven Courses

Course focuses on challenge problems, not lectures and exams

Students research, design and submit solutions.

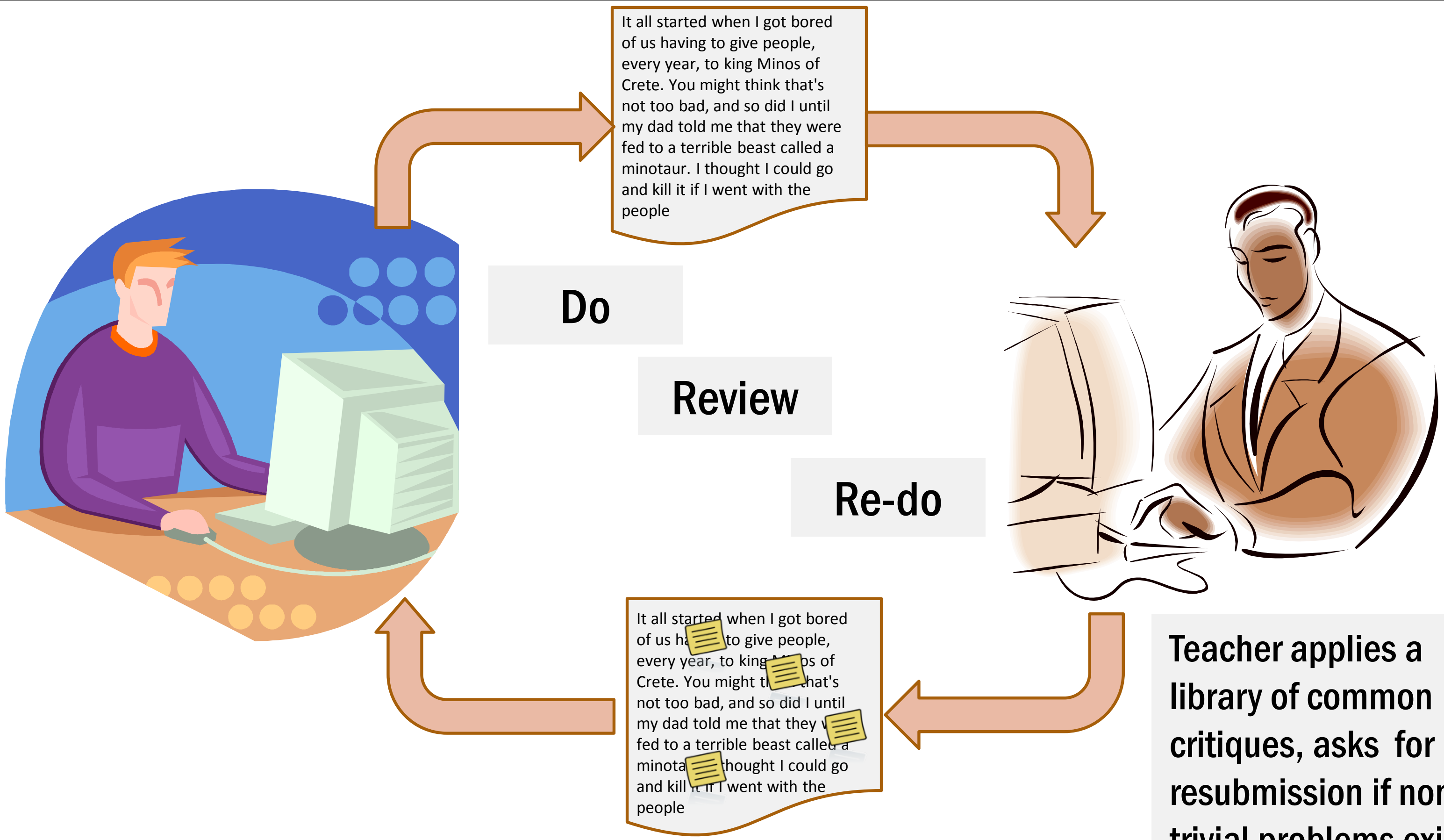
Mentors review solutions, note flaws, point to relevant learning materials.

Students re-do and resubmit.

Students move to next challenge only when no serious critiques remain.

Assessment based on what gets accomplished, quality of final submissions.

Critiquing



It all started when I got bored of us having to give people, every year, to king Minos of Crete. You might think that's not too bad, and so did I until my dad told me that they were fed to a terrible beast called a minotaur. I thought I could go and kill it if I went with the people

Do

Review

Re-do

It all started when I got bored of us ha to give people, every year, to king os of Crete. You might t hat's not too bad, and so did I until my dad told me that they w fed to a terrible beast called a minota hought I could go and kill went with the people

Teacher applies a library of common critiques, asks for resubmission if non-trivial problems exist

A Student …

Begins with a challenge problem

- I like to offer a pool of challenges

- Constructs solution, using resources as needed
- Submits solution for review
- Receives critiqued solution
- Fixes and resubmits.
- Repeat until no more critiques

Repeat until end of course

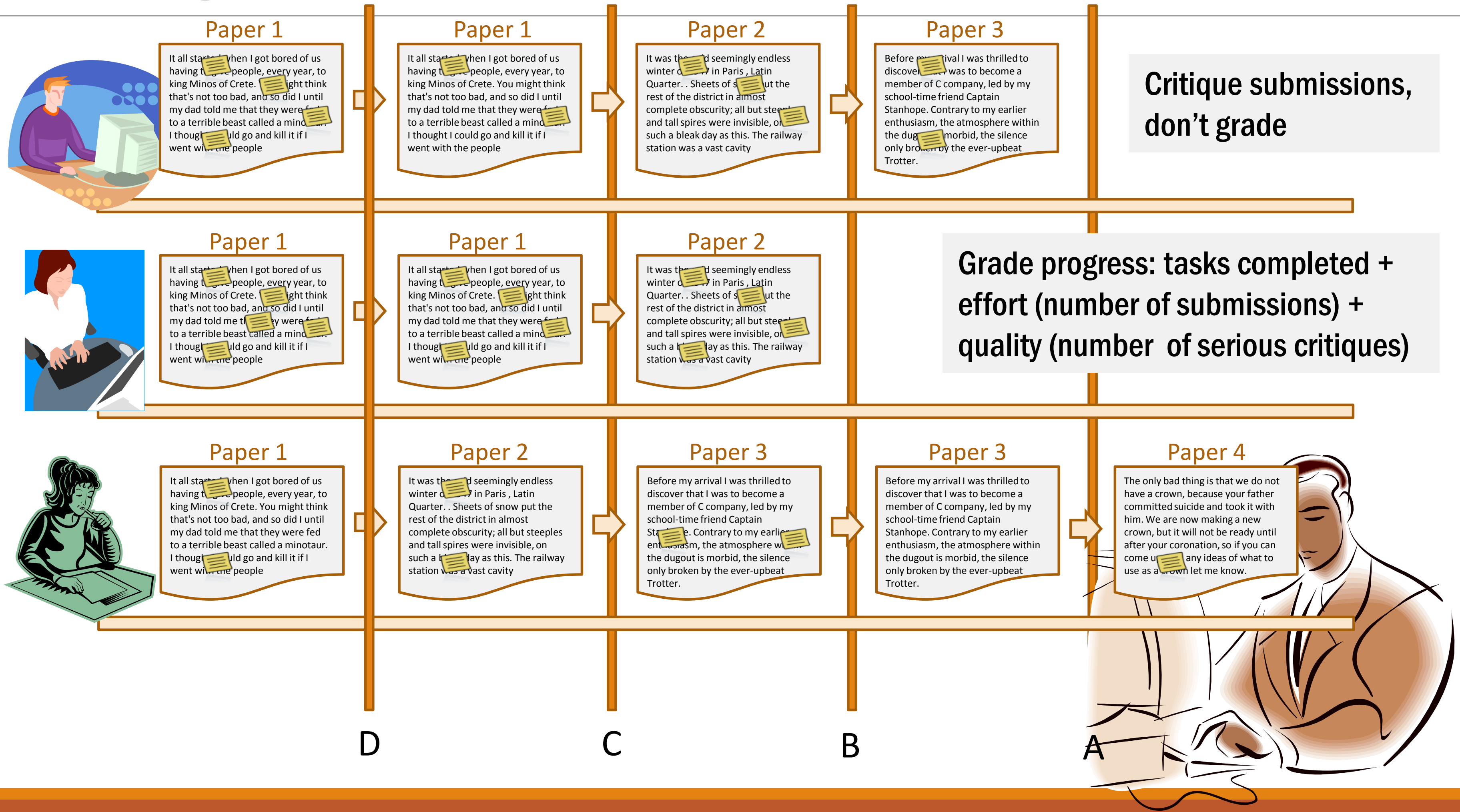
A Mentor …

Annotates submissions with critiques and returns to student

Assesses student based on submission and critique history

- Number of tasks done
- Range of challenges engaged
- Quality of submissions, e.g., lack of repeated important critiques

Grading



Critique submissions, don't grade

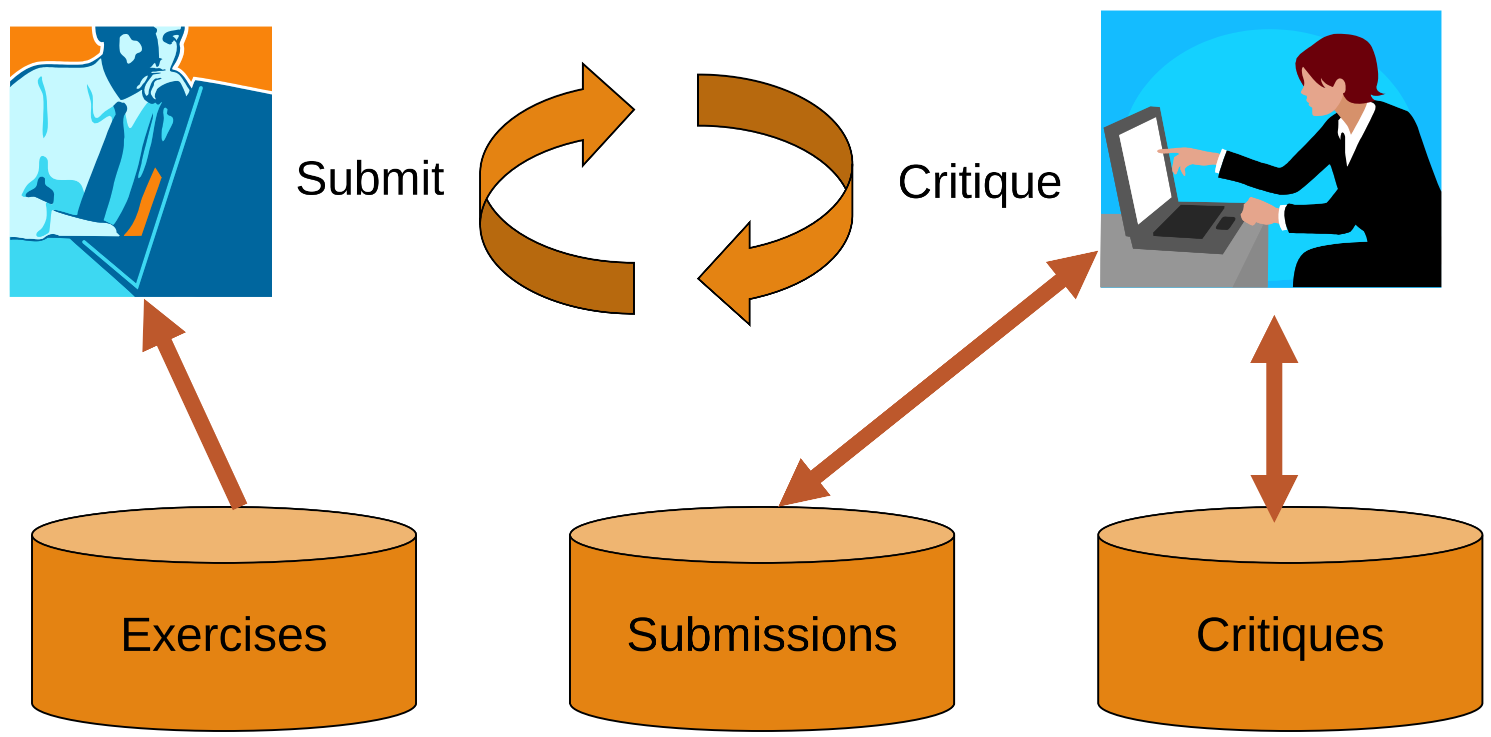
Grade progress: tasks completed + effort (number of submissions) + quality (number of serious critiques)

Critique-based Assessment

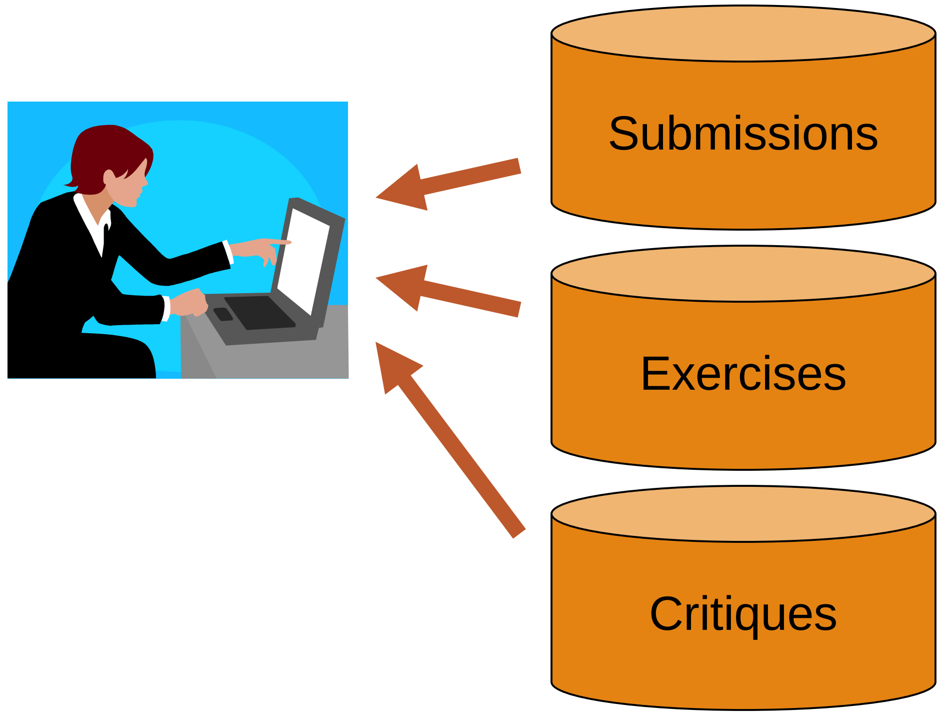
Combination of

- Challenges completed, their difficulty and diversity
- Effort displayed
- Quality of later initial submissions (absence of critiques)
- Critique history
  - Which critiques repeatedly appear, which don’t
  - Content and seriousness of repeated critiques

Critique Process



Assessment



Assessment based on:

- Exercise history
  - Content
  - Difficulty
  - Quality of first drafts
- Critique history
  - Content
  - Seriousness
  - Recurrence

History of Development and Application

1997? – EECS 325 (AI Programming)
- Emailed submissions, freeware Windows clip management tool for comments

2001 – EA-1 (Matlab, linear algebra)
- Dean Birge (McCormick)

2002 – EECS 325
- Browser-based critiquer replaces Windows clip tool

2002 – Intro Java, Business ESL
- Cognitive Arts online courses for Columbia University
- Proprietary web-based critiquer with submission database

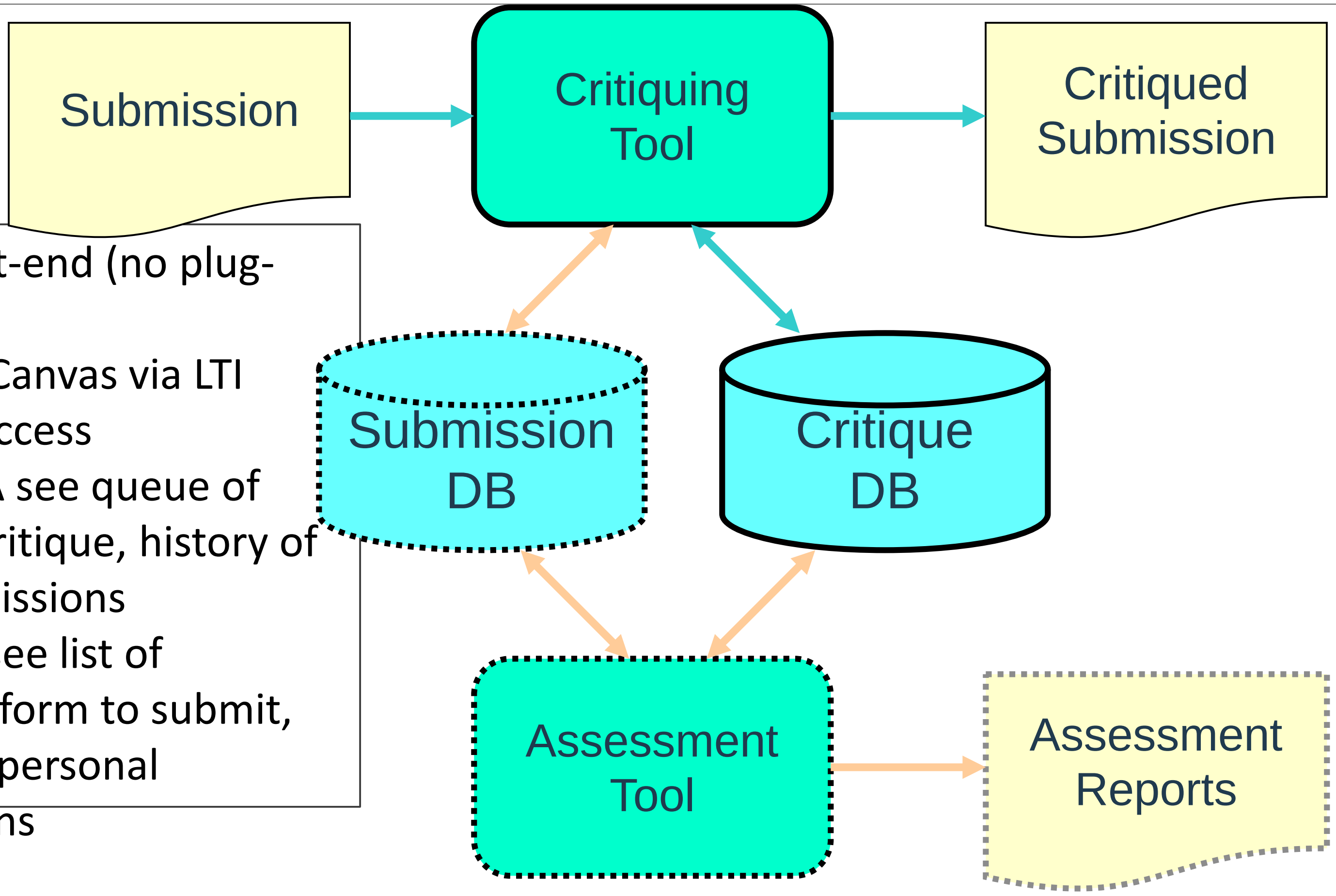
2006 – EECS 325, EECS 110 (intro programming)
- Submissions database, student interface, assessment interface added

2012 – Intro Web Development, and Software Engineering
- Socratic Arts / XTOL online MS for Touro University

2013 – MPD 405 (Software Project Management), EECS 394 (Agile Software Development)
- Case study critiquing

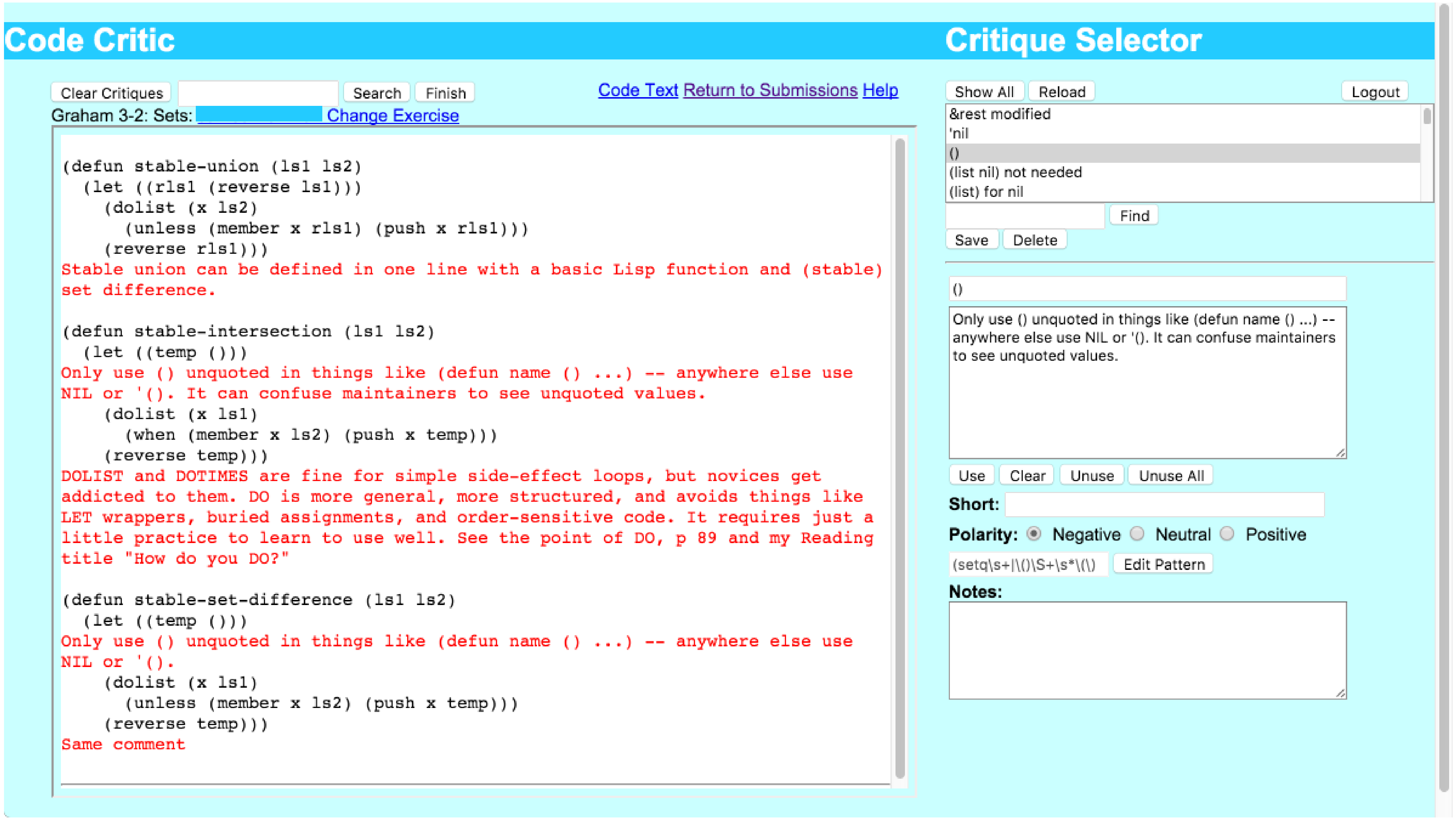
2013, 2015 – Intro Java
- Cascadia College

Critiquing Tool



Browser front-end (no plug-ins)
Access from Canvas via LTI
Role-aware access

- Faculty, TA see queue of items to critique, history of past submissions
- Students see list of exercises, form to submit, history of personal submissions



Graham 3-2: Sets: [redacted] Change Exercise

```
(defun stable-union (ls1 ls2)
  (let ((rls1 (reverse ls1)))
    (dolist (x ls2)
      (unless (member x rls1) (push x rls1)))
    (reverse rls1)))
```

Stable union can be defined in one line with a basic Lisp function and (stable) set difference.

```
(defun stable-intersection (ls1 ls2)
  (let ((temp ()))
```

Only use () unquoted in things like (defun name () ...) -- anywhere else use NIL or '(). It can confuse maintainers to see unquoted values.

```
    (dolist (x ls1)
      (when (member x ls2) (push x temp)))
    (reverse temp)))
```

DOLIST and DOTIMES are fine for simple side-effect loops, but novices get addicted to them. DO is more general, more structured, and avoids things like LET wrappers, buried assignments, and order-sensitive code. It requires just a little practice to learn to use well. See the point of DO, p 89 and my Reading title "How do you DO?"

```
(defun stable-set-difference (ls1 ls2)
  (let ((temp ()))
```

Only use () unquoted in things like (defun name () ...) -- anywhere else use NIL or '().

```
    (dolist (x ls1)
      (unless (member x ls2) (push x temp)))
    (reverse temp)))
```

Same comment

&rest modified
'nil
()
(list nil) not needed
(list) for nil

()

Only use () unquoted in things like (defun name () ...) -- anywhere else use NIL or '(). It can confuse maintainers to see unquoted values.

Short:

Polarity: [✓ Negative] Neutral Positive

(setq\s+|\(()\S+\s*\(\)

Notes:

Lessons Learned

Prompting matters

	Winter 2006	Winter 2007	Winter 2008	Fall 2014	2007 vs 2006	2008 vs 2006
System Arch.	Email	Web site	Web + email			
Class size	32	30	26	76		
# Submissions	1245	929	1371	2912		
# Critiques	?	2218	2906	8038		
Avg # Subs / Student	39	31	53	38	20% ⬇	35% ⬆
Avg # Exs / Student	21	15	28	20	26% ⬇	37% ⬆

Critiquing informs pedagogy

From an email to a TA in 1999:

```
Important things I learned from the critiquing process:
◦ The unpredictability of novice mistakes
◦ The commonness of some mistakes
◦ The number of micro-skills implied by these mistakes
```

Critiquing informs pedagogy

Example: in C++, to change the sign of a number, e.g., -3 to 3, or 4 to -4

- Correct, expected: `– x`
- Many students, not surprising: `0 – x`
- Very common, unexpected: `x – 2*x`
- I’ve yet to find another CS professor who is aware that this occurs
- No code testing would uncover this. It works, it’s just silly.

Making Critiquing Feasible

Use structured submissions (forms can help)

Use standard problems (use many if copying is an issue)

Require automated learner-side critiquing tools (lint, Lisp critic, …, spell / grammar / readability checkers, ..)

Sample – don't critique everything, just the diagnostic parts

Refine and standardize critiques

Short is good

Shorter more focussed submission are easier to critique, easier to have a dialog on

Students only resubmit those parts needing work, so later submissions get shorter and shorter

Tool provides link to version trail, if needed

Focus on details

Broad thematic critiques are hard to apply, often debatable, and become frustrating and ineffective when reapplied to resubmissions

- "Be more modular." "Use clear names."

Highly specific critiques are easy to apply, more objective, more easily fixed. Big themes emerge from them.

- "Refactor code more than 6 lines or so into subfunctions."
- "Refactor repeated code into common utility functions."
- "Check-xxx" is an unhelpful name. Doesn't say what happens after checking occurs"
- "This name is too generic. What kind of data does it contain?"
- [on a function name like "max-recursive"] "A function name should only need to say what it does, not how it does it."

Separate critiquing from helping

Move questions and objections to email or other channels

Reserve critique channel for “I think this is done”

- Supports use in assessment review
- Encourages repeated student self-evaluation and commitment

Critiques as transferrable pedagogy

EECS 325 Fall 2013, I was on leave

We hired an advanced PhD student from another school to teach the course.

He voluntarily used the Code Critic throughout the course.

He used a printout of all my critiques for the year before to get a baseline.

He got better CTECs than I do.

Critiquing with TAs

When EECS 325 reached 100 students in Fall 2014, I used a two-tier critiquing approach:

- I critiqued initial submissions from each student for each exercise.
- TAs handled all follow-up submissions
- TAs referred problematic submissions to me

Challenges

Deadlines vs Progress

Do-Review-Redo enables, encourages learner-centered progress.

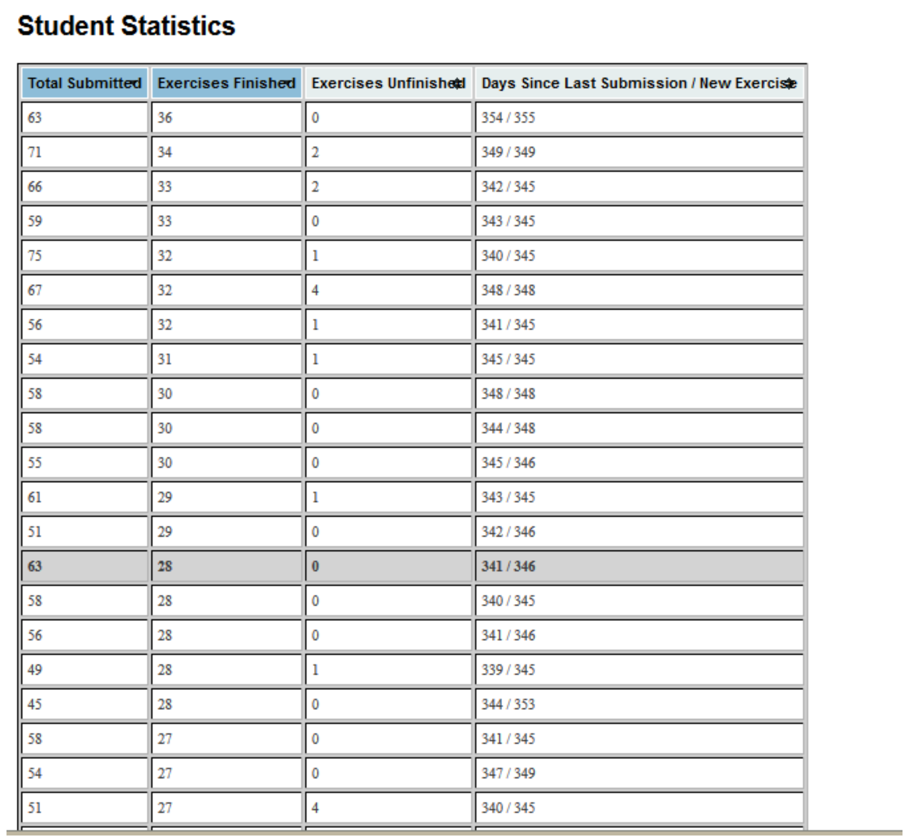
BUT

Other classes have due dates.

Due dates dominate. “Urgent vs important”

Every year, a few students come to my office in the lastweek of the quarter, asking if it’s too late to start submitting exercises.

Student status report



Student Statistics

Total Submitted	Exercises Finished	Exercises Unfinished	Days Since Last Submission / New Exercise
63	36	0	354 / 355
71	34	2	349 / 349
66	33	2	342 / 345
59	33	0	343 / 345
75	32	1	340 / 345
67	32	4	348 / 348
56	32	1	341 / 345
54	31	1	345 / 345
58	30	0	348 / 348
58	30	0	344 / 348
55	30	0	345 / 346
61	29	1	343 / 345
51	29	0	342 / 346
63	**28**	**0**	**341 / 346**
58	28	0	340 / 345
56	28	0	341 / 346
49	28	1	339 / 345
45	28	0	344 / 353
58	27	0	341 / 345
54	27	0	347 / 349
51	27	4	340 / 345

- Anonymous
- Not a grade but a relative indication
- Personal position highlighted
- Always up to date

Being critiqued is no fun

One student for another course on modeling emotion in simulated characters used EECS 325 as his storyline:

- **Frustration** Why doesn’t this #$@%@ code work!
- **Joy** Yay! It passes all the tests!
- **Anticipation** waiting to hear from professor
- **Depression** code comes back loaded with critiques

Class attendance plummets

Learning and progress are tracked individually.

Most of the real learning occurs during

- coding and problem solving
- review and resubmission

Lectures cover particularly tricky or broad topics but are clearly optional, too soon for some, too late for others

A Tale of Two Courses

EECS 325: Intro AI Programming

EECS 394: Agile Software Development

Both

Software development

Learn by doing

Teach by critiquing

Lectures? If I must

How The Courses Work

EECS 325: Intro AI Programming

Individual submit solutions to dozens of Lisp and AI coding challenges

I **critique** their code and they re-work and resubmit until the code is free of serious issues

EECS 394: Agile Software Development

Teams iteratively develop 2 mobile web apps, one for themselves, one for a client

I meet weekly with and **critique** each team's product and team development processes

Individual submit coaching advice to several case studies

I **critique** the coaching advice

EECS 325 Programming Sample Critiques

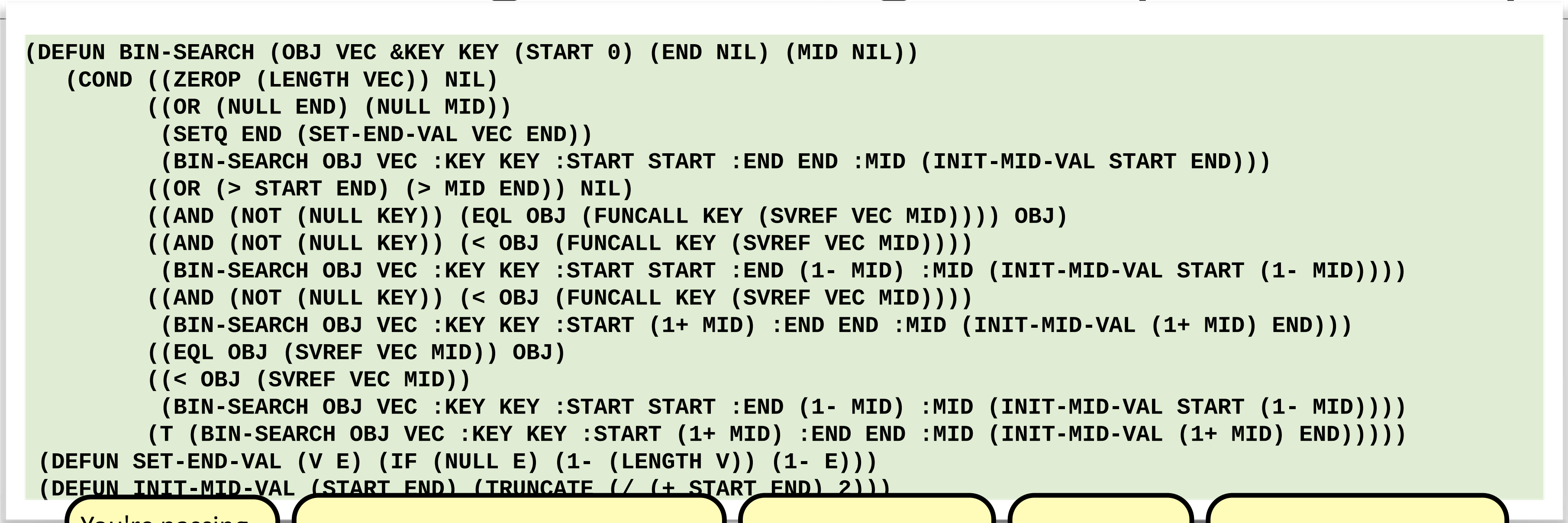
```
(defun horner (x &rest coeff)
  (reduce #'(lambda (a &optional b)
              0
              (if (null b)
                  a
                (+ b (* a x))))
          coeff))
```

Variable names should say what a variable contains. coeff does not contain just one coefficient.

What do you think that 0 does?

Can you avoid doing an IF every iteration?

EECS 325 Programming Sample Critiques



```
(DEFUN BIN-SEARCH (OBJ VEC &KEY KEY (START 0) (END NIL) (MID NIL))
   (COND ((ZEROP (LENGTH VEC)) NIL)
         ((OR (NULL END) (NULL MID))
          (SETQ END (SET-END-VAL VEC END))
          (BIN-SEARCH OBJ VEC :KEY KEY :START START :END END :MID (INIT-MID-VAL START END)))
         ((OR (> START END) (> MID END)) NIL)
         ((AND (NOT (NULL KEY)) (EQL OBJ (FUNCALL KEY (SVREF VEC MID)))) OBJ)
         ((AND (NOT (NULL KEY)) (< OBJ (FUNCALL KEY (SVREF VEC MID))))
          (BIN-SEARCH OBJ VEC :KEY KEY :START START :END (1- MID) :MID (INIT-MID-VAL START (1- MID))))
         ((AND (NOT (NULL KEY)) (< OBJ (FUNCALL KEY (SVREF VEC MID))))
          (BIN-SEARCH OBJ VEC :KEY KEY :START (1+ MID) :END END :MID (INIT-MID-VAL (1+ MID) END)))
         ((EQL OBJ (SVREF VEC MID)) OBJ)
         ((< OBJ (SVREF VEC MID))
          (BIN-SEARCH OBJ VEC :KEY KEY :START START :END (1- MID) :MID (INIT-MID-VAL START (1- MID))))
         (T (BIN-SEARCH OBJ VEC :KEY KEY :START (1+ MID) :END END :MID (INIT-MID-VAL (1+ MID) END)))))
 (DEFUN SET-END-VAL (V E) (IF (NULL E) (1- (LENGTH V)) (1- E)))
 (DEFUN INIT-MID-VAL (START END) (TRUNCATE (/ (+ START END) 2)))
```

You're passing an argument you don't need.

See the table on page 64 for standard keyword defaults. Note especially the default for KEY.

The "usual default" for :end is NOT length - 1.

Try to avoid repeating tests.

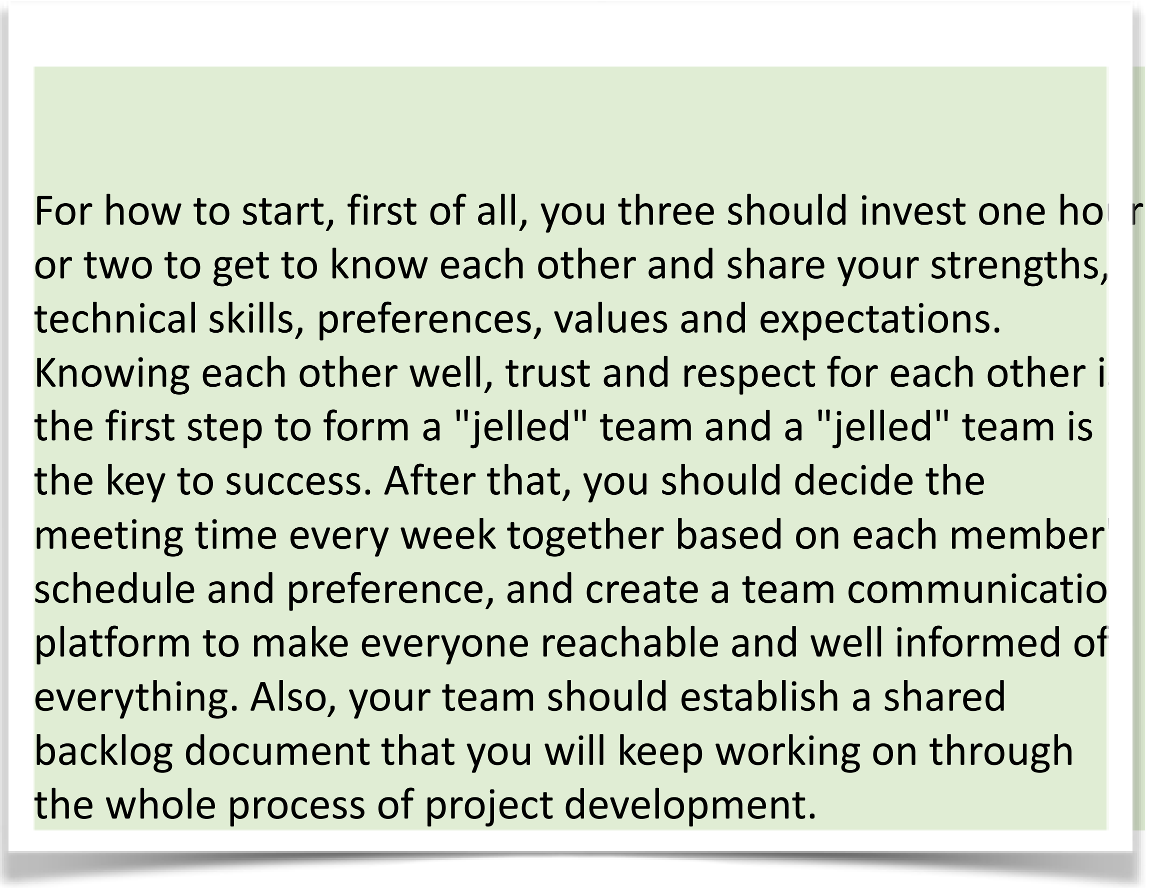
There's no need to divide before calling FLOOR, CEILING etc.

The function passed in should only need to be called at most once per element. It might be expensive.

This is way more complicated than necessary. Binary search is a very simple algorithm.

You don't need the subfunctions. A rule of thumb is: define a function if its name is clearer than the code it replaces. That doesn't seem to apply here.

EECS 394 Agile Development Sample Critiques

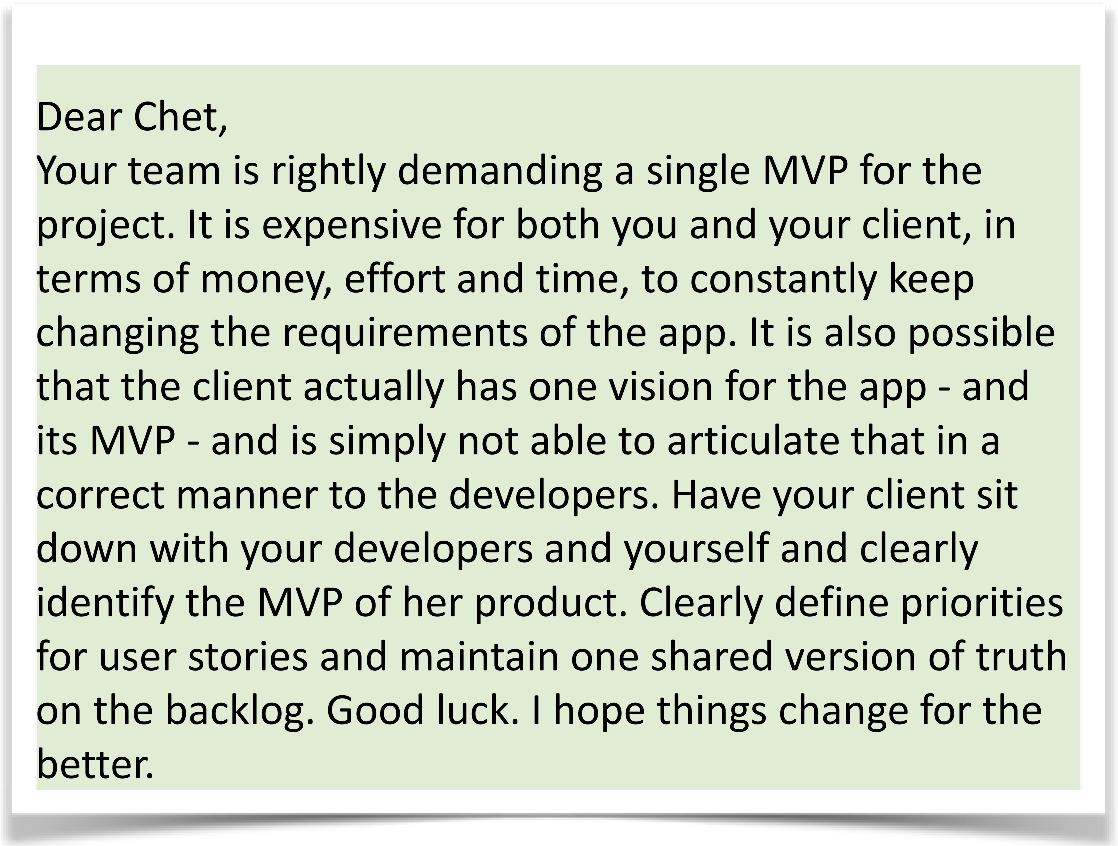


For how to start, first of all, you three should invest one ho r or two to get to know each other and share your strengths, technical skills, preferences, values and expectations. Knowing each other well, trust and respect for each other i the first step to form a "jelled" team and a "jelled" team is the key to success. After that, you should decide the meeting time every week together based on each member' schedule and preference, and create a team communicatio platform to make everyone reachable and well informed of everything. Also, your team should establish a shared backlog document that you will keep working on through the whole process of project development.

What problems is this advice trying to solve? Before someone will listen to advice, they have to believe there's a problem.

This is a laundry list of things to do, not tailored advice.

EECS 394 Agile Development Sample Critiques



Dear Chet,
Your team is rightly demanding a single MVP for the project. It is expensive for both you and your client, in terms of money, effort and time, to constantly keep changing the requirements of the app. It is also possible that the client actually has one vision for the app - and its MVP - and is simply not able to articulate that in a correct manner to the developers. Have your client sit down with your developers and yourself and clearly identify the MVP of her product. Clearly define priorities for user stories and maintain one shared version of truth on the backlog. Good luck. I hope things change for the better.

What agile principle supports this advice?

What do you see that suggests this is the case?

This is aspirational not operational. You give a goal but not how to achieve it.

What works, what's a struggle

EECS 325: Intro AI Programming	EECS 394: Agile Software Development
Why this has worked fine	Why this remains a struggle
Much of the learning comes just from the effort involved in writing working code	Most of the learning comes from pointing out that everything teams think they know is wrong
Critiquing much of the code is (relatively) easy to automate, using classic pattern matching techniques	This requires analyzing free-form text and group conversations to detect and causally explain team development issues, then persuading the teams to try alternative behaviors.

Pedagogical Connections

CRITIQUING AND THE LEARNING SCIENCES

Themes

competency and mastery

continuous situated assessment

test-driven learning

grades vs critiques

critiques vs rubrics

Critiques and Learning

Critiques aren’t grades.

Critiques say what’s wrong and why.

Critiques tie principles to practice.

Critiques are just-in-time links to lessons.

Critiques support many right answers.

Critiques support detailed assessment.

Student Advantages

In-depth, personalized, private feedback ("No one ever looked at my code before!")

Effort focused on weakest areas

Stronger students get advanced feedback

Rubrics vs Critiques

Rubrics	Critiques
Performance descriptions are combinations of contradictory ambiguously defined issues	Critiques are specific, separate, and consistent
Reviewer must make repeated borderline judgment calls, combined with a simple weighted sum	Reviewer decide if submission needs re-work
Final grade based on weighted average of the subjective submission scores	Final grade based on visible objective metrics: number of tasks done, submissions sent, and history of critiques
Criteria, performance descriptions, and scoring must be fully defined and published in advance	Criteria and progress metrics must be defined and published in advance but specific critiques can be added and refined over time
New instructors must learn how to interpret criteria such as “a strong sense of both authorship and audience”	New instructors can review in-context examples of critiques given for multiple submissions for each task