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.
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.
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

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.

Critique submissions, don’t grade

Before my arrival I was thrilled to discover that I was to become a member of C Company, led by my school-time friend Captain Stanhope. Contrary to my earlier enthusiasm, the atmosphere within the camp was one of despair. The silence only broke by the ever-upbeat Trotter.

The only bad thing is that we do not have a crown, because your father committed suicide and took it with him. We are now making a new crown, but it will not be ready until after your coronation, so if you can come up with any ideas of what to use as a crown let me know.

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

Exercises → Submit → Critique → Submissions → Critiques
Assessment

Assessment based on:

◦ Exercise history
  ◦ Content
  ◦ Difficulty
  ◦ Quality of first drafts

◦ Critique history
  ◦ Content
  ◦ Seriousness
  ◦ Recurrence
## History of Development and Application

<table>
<thead>
<tr>
<th>Year</th>
<th>Course/Program</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1997?</td>
<td>EECS 325 (AI Programming)</td>
<td>Emailed submissions, freeware Windows clip management tool for comments</td>
</tr>
<tr>
<td>2001</td>
<td>EA-1 (Matlab, linear algebra)</td>
<td>Dean Birge (McCormick)</td>
</tr>
<tr>
<td>2002</td>
<td>EECS 325</td>
<td>Browser-based critiquer replaces Windows clip tool</td>
</tr>
<tr>
<td>2002</td>
<td>Intro Java, Business ESL</td>
<td>Cognitive Arts online courses for Columbia University</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Proprietary web-based critiquer with submission database</td>
</tr>
<tr>
<td>2006</td>
<td>EECS 325, EECS 110 (intro programming)</td>
<td>Submissions database, student interface, assessment interface added</td>
</tr>
<tr>
<td>2012</td>
<td>Intro Web Development, and Software Engineering</td>
<td>Socratic Arts / XTOL online MS for Touro University</td>
</tr>
<tr>
<td>2013</td>
<td>MPD 405 (Software Project Management), EECS 394 (Agile Software Development)</td>
<td>Case study critiquing</td>
</tr>
<tr>
<td>2013, 2015</td>
<td>Intro Java</td>
<td>Cascadia College</td>
</tr>
</tbody>
</table>
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
(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 ()))
    (do (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 ()))
    (do (when (member x ls2) (push x temp)))
    (reverse temp)))

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

Notes:

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>System Arch.</strong></td>
<td>Email</td>
<td>Web site</td>
<td>Web + email</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Class size</strong></td>
<td>32</td>
<td>30</td>
<td>26</td>
<td>76</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># Submissions</strong></td>
<td>1245</td>
<td>929</td>
<td>1371</td>
<td>2912</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong># Critiques</strong></td>
<td>?</td>
<td>2218</td>
<td>2906</td>
<td>8038</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Avg # Subs / Student</strong></td>
<td>39</td>
<td>31</td>
<td>53</td>
<td>38</td>
<td>20% ↓</td>
<td>35% ↑</td>
</tr>
<tr>
<td><strong>Avg # Exs / Student</strong></td>
<td>21</td>
<td>15</td>
<td>28</td>
<td>20</td>
<td>26% ↓</td>
<td>37% ↑</td>
</tr>
</tbody>
</table>
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: \(x - 2x\)
- 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 last week of the quarter, asking if it’s too late to start submitting exercises.
Student status report

<table>
<thead>
<tr>
<th>Student Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total Submitted</strong></td>
</tr>
<tr>
<td>63</td>
</tr>
<tr>
<td>74</td>
</tr>
<tr>
<td>59</td>
</tr>
<tr>
<td>75</td>
</tr>
<tr>
<td>67</td>
</tr>
<tr>
<td>56</td>
</tr>
<tr>
<td>54</td>
</tr>
<tr>
<td>58</td>
</tr>
<tr>
<td>58</td>
</tr>
<tr>
<td>53</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>51</td>
</tr>
<tr>
<td>63</td>
</tr>
<tr>
<td>58</td>
</tr>
<tr>
<td>64</td>
</tr>
<tr>
<td>49</td>
</tr>
<tr>
<td>45</td>
</tr>
<tr>
<td>58</td>
</tr>
<tr>
<td>54</td>
</tr>
<tr>
<td>51</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>EECS 325: Intro AI Programming</th>
<th>EECS 394: Agile Software Development</th>
</tr>
</thead>
</table>

Both

- **Software development**
  - Learn by doing
  - Teach by critiquing
  - Lectures? If I must
<table>
<thead>
<tr>
<th>EECS 325: Intro AI Programming</th>
<th>EECS 394: Agile Software Development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individual submit solutions to dozens of Lisp and AI coding challenges</td>
<td>Teams iteratively develop 2 mobile web apps, one for themselves, one for a client</td>
</tr>
<tr>
<td><strong>critique</strong> their code and they re-work and resubmit until the code is free of serious issues</td>
<td>I meet weekly with and <strong>critique</strong> each team's product and team development processes</td>
</tr>
<tr>
<td></td>
<td>Individual submit coaching advice to several case studies</td>
</tr>
<tr>
<td></td>
<td>I <strong>critique</strong> the coaching advice</td>
</tr>
</tbody>
</table>
(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?
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 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.

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.
EECS 394 Agile Development Sample Critiques

For how to start, first of all, you three should invest one hour 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 is 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's schedule and preference, and create a team communication 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.
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

<table>
<thead>
<tr>
<th>EECS 325: Intro AI Programming</th>
<th>EECS 394: Agile Software Development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Why this has worked fine</td>
<td>Why this remains a struggle</td>
</tr>
<tr>
<td>Much of the learning comes just from the effort involved in writing working code</td>
<td>Most of the learning comes from pointing out that everything teams think they know is wrong</td>
</tr>
<tr>
<td>Critiquing much of the code is (relatively) easy to automate, using classic pattern matching techniques</td>
<td>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.</td>
</tr>
</tbody>
</table>
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

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