

GDC

March 21-25, 2022
San Francisco, CA

Abolish Points: Using Specifications Grading in Game Courses

Sanjay Madhav
Associate Professor of Practice
University of Southern California

#GDC22



Hello, and welcome!

Before I start, please remember to silence your phones. Also, after the talk please fill out your session evaluations.

My name is Sanjay Madhav and I'm an Associate Professor of Practice at the University of Southern California. Today I'm going to talk about every educator's least favorite topic – grading. I've spent the last couple of years revamping the way I do grading in my game courses, and I'm going to cover how you can do this, too!

My examples today are all from my video game programming course, but the principles could easily apply to any discipline.

Why Specifications Grading?

- Is **results-focused**
- Injects **iteration** into the grading process
- Gets students to think about the **quality** of their work

Before I dive into the details, here are what I think are the best qualities of specifications grading

- It's focused on making sure students get results and complete their work
- It requires students to iterate on their work if they want to get an A
- It gets students thinking about concerns beyond simply meeting the requirements

GDC

March 21-25, 2022
San Francisco, CA

The Problem With Points

#GDC22



We used to grade assignments using point rubrics, but there were a lot of issues we noticed over time

Complexity and grading time

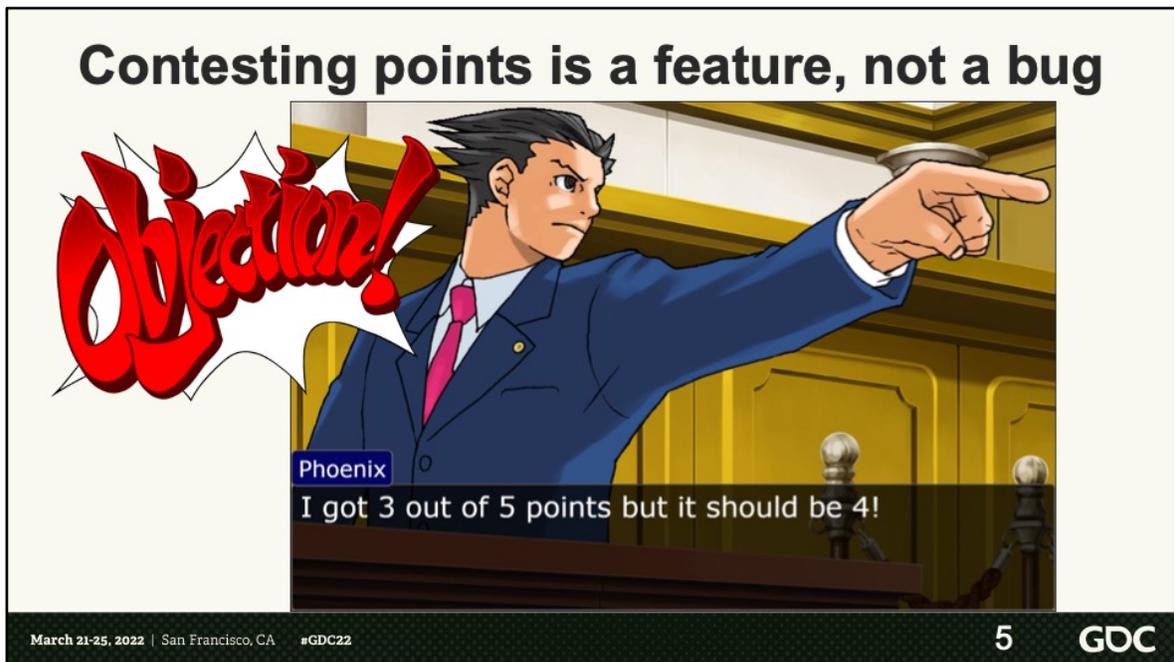
Part	code check	Possible	Earned	Note
Part 1 (Level, Player, and Jumping)				
4 Background	no	1		
5 LoadLevel() - Loads blocks for chars 'A' to 'I'	yes	1		
6 Player Class and Constructor	yes	1		
7 PlayerMove Class and Constructor	yes	1		
8 LoadLevel() - Spawn Player at 'P'	yes	1		
9 Player moves with left/right arrow keys at correct speed	no	2		
10 Player y-speed gravity calculates and correct	yes	3		
11 Player collides and applies offsets with Blocks	yes	3		
12 Block top collision behaves correctly (check for mYSpeed > 0)	yes	1		
13 Input logic detects leading edge correctly	yes	3		
14 Player can jump as expected	no	2		
15 Player can't double or triple jump (only allow jump when mYAir is false)	no	1		
16 Player can't walk off edge and jump in air (detect not colliding with any blocks)	no	1		
17 Player doesn't float when hitting bottom of a block	no	1		
18 Game is loading Level.txt	no	1		
Part 2 (Scrolling and Enemies)				
20 SpriteComponent::Draw converts to Screen Space before drawing	yes	6		
22 Camera position updates to follow Player	yes	2		
23 Player is horizontally centered on screen (subtract half screen width)	yes	1		
24 Player.x can never be less than Camera.x (can't walk off screen to left)	no	1		
25 Camera.x never decreases (always moving to the right)	no	1		
26 Camera.x can't go lower than 0 (can't see to left of start point)	no	1		
27 Goomba Class and Constructor	yes	1		
28 GoombaMove Class and Constructor	yes	1		
29 Goombas add/remove from Game vector in Constructor/Destructor	yes	1		
30 Spawner Class and Constructor	yes	1		
31 Spawner spawns Goombas if x-distance is less than 600	yes	2		
32 Spawners created when a 'Y' exists in level file	yes	1		
33 Goombas move correctly	no	5		
34 Mario can stomp Goombas from the top	no	1		
35 Mario can stomp Goombas from the sides	no	1		
36 Stomped Goombas change textures and stop moving	no	1		
37 Stomped Goombas disappear after 0.25 seconds	no	2		
38 Mario does a half-jump after stomping a Goomba	no	1		



Here's an example of an old rubric. The assignment is to program the first level of Super Mario Bros.

This rubric may seem extreme, but one of the reasons it was so granular was to ensure fairness since we had multiple graders. It also served to head off grading complaints from students.

Clearly, breaking down a detailed rubric like this just means that the graders must spend a lot of time verifying each individual item to tabulate the score.



Unfortunately, it's a feature of points-based grading systems that students will argue for points.

Even with very detailed rubrics, you will consistently have students who want 1 or 2 points here or there. And the problem is exacerbated if the rubric are less granular and/or you have several graders.

Some students learn to fight over every little thing because they know they'll probably get one or two points back. It's frustrating when the majority of your post-assignment interaction with students is about wanting more points.



Points also lead to student min/maxing points like it's an RPG. It seems okay to skip certain parts of the course materials which hinders student's learning. Students are incentivized to care about achieving specific point thresholds more than anything else.

And this is also not very "real world" – if your boss wants you to add a new level and you only add half a level, they're going to want you to finish that level eventually.

Students didn't care about code quality

```
537 frontier = currentNode->Adjacent;
538 for (int i = 0; i < frontier.size();i++)
539
540 //1.1 here are what kind of nodes in frontier that wouldn't be counted
541 if (frontier[i] == beforeNode) continue; // 1.1 It's turning back
542 //1.1 As above
543 if (std::find(explored.begin(), explored.end(), frontier[i]) != explored.end()) // if already explored
544 {
545     PathNode* leftT = #Ghost->GetGame()->TunnelLeft;
546     PathNode* rightT = #Ghost->GetGame()->TunnelRight;
547     Vector2 eDis1;
548     eDis1.x = nTargetNode->GetPosition().x - frontier[i]->GetPosition().x;
549     eDis1.y = nTargetNode->GetPosition().y - frontier[i]->GetPosition().y;
550     float h1 = eDis1.Length();
551     Vector2 eDis2;
552     eDis2.x = nTargetNode->GetPosition().x - leftT->GetPosition().x;
553     eDis2.y = nTargetNode->GetPosition().y - leftT->GetPosition().y;
554     Vector2 eDis2b;
555     eDis2b.x = rightT->GetPosition().x - frontier[i]->GetPosition().x;
556     eDis2b.y = rightT->GetPosition().y - frontier[i]->GetPosition().y;
557     float h2 = eDis2.Length() + eDis2b.Length();
558     Vector2 eDis3;
559     eDis3.x = nTargetNode->GetPosition().x - rightT->GetPosition().x;
560     eDis3.y = nTargetNode->GetPosition().y - rightT->GetPosition().y;
561     Vector2 eDis3b;
562     eDis3b.x = leftT->GetPosition().x - frontier[i]->GetPosition().x;
563     eDis3b.y = leftT->GetPosition().y - frontier[i]->GetPosition().y;
564     float h3 = eDis3.Length() + eDis3b.Length();
565     float h = std::min(h1,std::min(h2,h3));
566     Vector2 Dis;
567     Dis.x = frontier[i]->GetPosition().x - currentNode->GetPosition().x;
568     Dis.y = frontier[i]->GetPosition().y - currentNode->GetPosition().y;
569     float dis = Dis.Length();
570     float g = dis + nodescore[currentNode];
571     float f = g + h;
572     if (f >= nodescore[frontier[i]])
573     {
574         continue;
575     }
576     nodescore[frontier[i]] = g;
577     nodescore[frontier[i]] = f;
578     nodesparent[frontier[i]] = currentNode;
```

March 21-25, 2022 | San Francisco, CA #GDC22

7

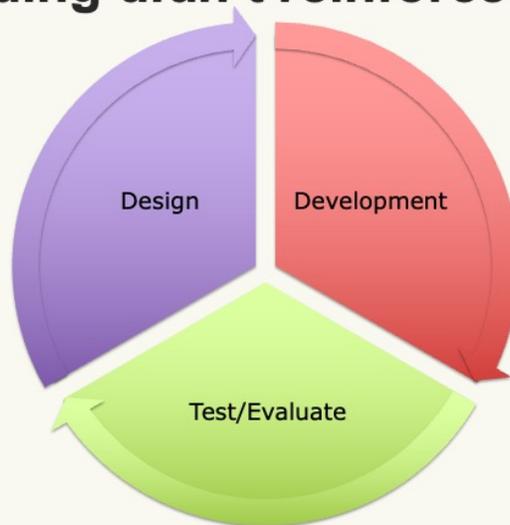
GDC

Because the points rubric was solely based on functionality, the students could get away with poor code quality. If you don't know anything about coding – this example is a pathological case of copying/pasting code which is a bad practice.

Although you *could* assign points to code quality, it's very subjective and becomes a big source of complaints about point deductions.

Developers recruiting graduates also consistently say that poor code quality is an easy way to never get a callback. I felt like it was a disservice to students to say that “this is A-quality work simply because it functions correctly.”

The grading didn't reinforce iteration



March 21-25, 2022 | San Francisco, CA #GDC22

8

GDC

Game development is a very iterative process. But we weren't able to achieve reinforce that with points. Students get their points and never would touch the project again. That's not to say it's impossible to add iteration with points, but with the limited amount of grading hours we had, we were spending too much time tabulating points in the first place.

GDC

March 21-25, 2022
San Francisco, CA

A Step Back: Thinking About Learning Outcomes

#GDC22



I started thinking critically about course learning outcomes and if it would be possible to restructure the grading to support the outcomes.

What should students get out of the class?

1. Be able to write C++ code to solve common game programming problems as well as game-specific problems.
2. Be able to write code that is maintainable and **NOT** a throwaway mess.
3. Be prepared for game programming interviews.
4. Practice being a dependable and responsible adult.

These are our high-level goals:

- Write code to solve game problems
- Write quality code
- Be able to intelligently answer game programming questions in the style of what a typical interview may have
- And be responsible

How should we assess these objectives?

1. Be able to write C++ code to solve common game programming problems as well as game-specific problems. *(Assignments)*
2. Be able to write code that is maintainable and **NOT** a throwaway mess. *(Code Review)*
3. Be prepared for game programming interviews. *(Exams)*
4. Practice being a dependable and responsible adult. *(Due dates, requiring complete work, in-class labs)*

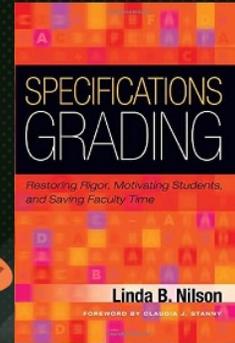
Although our old system did assess #1 and #3, we didn't really assess #2 and only assessed half of #4

GDC

March 21-25, 2022
San Francisco, CA

Specifications Grading

#GDC22



After researching different options, I eventually stumbled across specifications grading, which was proposed by Nilson in 2014. We've been using specifications grading for a couple of years now, so I'm going to dive into how we've implemented it. I should note that what I'm going to describe isn't exactly like the system in the book, but it has a lot of similarities.

Defining the specifications

1. The Level1.txt level file loads as expected. Mario can run and jump through the level, and collision with the blocks/pipes works as expected.
2. Camera advances as Mario runs through the level. The camera should not go backwards, and Mario should not be able to run off camera.
3. Goombas spawn and move as expected. They should fall when not standing on anything, and when they hit a wall or another Goomba, they switch directions.
4. Mario can stomp on Goombas, and Mario dies if he runs into a Goomba and isn't stomping it.
5. The animations for Mario (idle, run, jump, dead) and the Goomba (walk, dead) play as appropriate.
6. The main music plays and loops. The sound effects for jump, stomp, and bump play as expected. When Mario dies (either killed by a Goomba or falls in a pit), the main music stops, and the death music plays. When Mario reaches the end of the level, the main music stops, and the victory music plays.

In order to use specifications grading, first you define the requirements (or specifications) of the assignment. In this assignment, I'm expecting the student to program some very specific Mario behavior, so these specs are extremely detailed.

But there's no requirement specs must be this detailed – they can be as broad or specific as it makes sense for your course. If you have a more creative assignment like say a level design course, you can go with much more general or broader specs. Think about what would you consider “B” quality work, and that should be clearly defined by the specifications.

Initial Grade – Are specifications met?

F – The student failed to satisfy at least half of the specs

C – The student satisfied at least half of the specs, but not all

B – The student satisfied all specs



First, the grader verifies that the student met the specifications. This generally will just require playing the game for a couple of minutes and making sure each feature works as expected. The goal is to be able to decide within a couple of minutes whether a student's submission satisfies all the specifications or not.

This means that if students just choose not to implement one of the specs (like music), then they are bucketed into a C.

I'd note here that this is a little different from Nilson's approach which grades credit/no credit. I would personally love to do all grading credit/no credit, but in my experience the students do need the extrinsic motivation of getting an A.

Initial Grade – Is it a B or an A?

A - This work meets or exceeds the specification of the assignment. There are no non-trivial errors. There are few or no code quality issues. ***This work could be used as a classroom example.***

B - This work meets the specification of the assignment. If there are errors, they are minor. There are likely code quality issues that should be resolved.

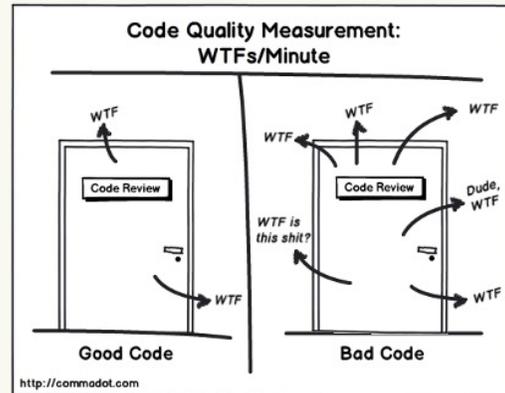
If the grader confirms that the student meets all specs, the next step is to decide whether the assignment is an A or a B.

This is how we describe the difference between an A and a B to students. I think “classroom example” is a great way to describe an A, which should be excellence. B means you met all the requirements but there’s still items to improve.

As part of the A requirements, we evaluate code quality with code reviews.

What do we look for in the code review?

- Compiler warnings
- Copy-pasting code instead of making a function
- Poorly named variables/functions
- Inconsistent indentation
- Code that is difficult to follow (lots of conditionals in a spaghetti-like fashion)
- Using magic numbers inline rather than declaring constants
- Writing out equations for things like distance in code rather than using the library functions

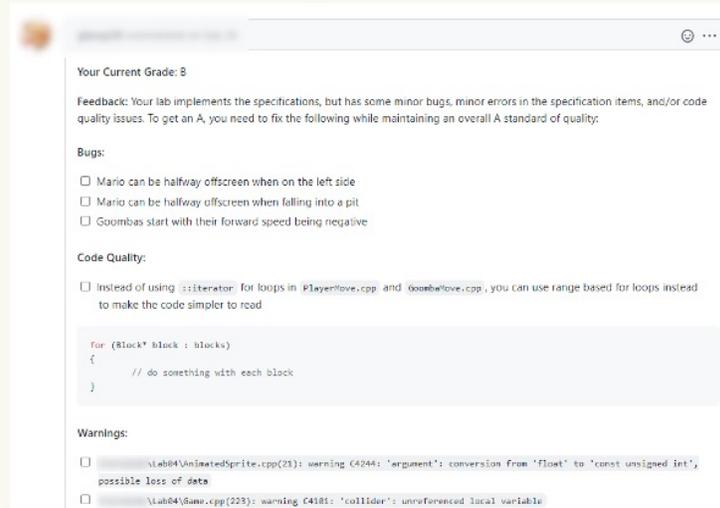


We see the code review as a way for the TAs (who are all former students) to impart their wisdom on the current students. It's a learning experience for both students and TAs to be on either end of the code review.

This is not an exhaustive list, but it's a good starting point and what we tell students not to do.

Here's a comic we show the students when talking about code quality. There is some subjectivity when evaluating code quality, just like evaluating any creative product like design or art.

Giving feedback



Your Current Grade: B

Feedback: Your lab implements the specifications, but has some minor bugs, minor errors in the specification items, and/or code quality issues. To get an A, you need to fix the following while maintaining an overall A standard of quality:

Bugs:

- Mario can be halfway offscreen when on the left side
- Mario can be halfway offscreen when falling into a pit
- Goombas start with their forward speed being negative

Code Quality:

- Instead of using `::iterator` for loops in `PlayerLove.cpp` and `GoombaLove.cpp`, you can use range based for loops instead to make the code simpler to read

```
for (Block* block : blocks)
{
    // do something with each block
}
```

Warnings:

- `\\lab04\\AnimatedSprite.cpp(21): warning C4244: 'argument': conversion from 'float' to 'const unsigned int', possible loss of data`
- `\\lab04\\Game.cpp(223): warning C4181: 'collider': unreferenced local variable`

Since we have the students submit everything on GitHub, we use GitHub issues to give their initial grading feedback, and it's also how the students request their regrade if they're going for it. Here's an example of a grade report for a B.

(As an implementation detail, I added some tooling to make it easy for graders to create these reports)

Regrades to promote iteration

B → A

C → B

F → C



Students have one regrade opportunity where they can increase their grade at most one letter grade. They must submit the regrade within 4 days of their initial grade. For F->C and C->B it's quick to confirm.

For B->A the grader will do a sanity check to make sure the game still functions, and then confirm that each issue brought up was fixed. It's guaranteed to be a closed list – meaning if the student fixes everything which was originally noted (without breaking their game), we won't come back with problems we didn't notice on the original grade report.

What assessments do we use this on?

- Assignments (use specifications grading)
- In-class labs (graded credit/no credit)
- Exams (graded with points)

Ultimately, we have three main graded components of the class. Right now, we're only using specifications grading for the "assignments" component.

The in-class labs are graded credit/no credit because we're simply confirming that the student was there and made a genuine effort on the lab. We don't grade it for correctness.

The exams are still graded with points because we've not devised a good alternative. For our course, exams make sense as they test some specific skills that we wouldn't be able to in the assignments. We think of this as "interview prep" for the student. If your class doesn't have exams, then you might be completely free from points.

Final Course Grades

- To get an **A** in the class, you need have:
 - Credit (**CR**) on at least 9/12 in-class labs
 - **B** or higher (after regrade) on all 12 assignments
 - **A** (after regrade) on at least 8/12 assignments
 - Average exam grade of at least 85%*

* *The median exam grade is usually ~85*

Ultimately, we're required to assign a final course grade. To get an A they do need to get an A on most of the assignments (though it's okay if they get a B on a few). They also can't just skip features they don't feel like doing, because that would give a C on that assignment.

Each criterion is evaluated separately and ties into our different learning outcomes.

GDC

March 21-25, 2022
San Francisco, CA

Outcomes

#GDC22



Let's look a little bit at how things have turned out

What do instructors think?

- We expect more of the students
- We know students read feedback and act on it



March 21-25, 2022 | San Francisco, CA #GDC22

22

GDC

Students can't just min/max their way to an A anymore

Students are writing better code overall (though it's not perfect)

Students are more familiar with code review and iteration, which is more real world, and we know they are implementing the feedback we give them, and our interactions with the students about the grading are almost always about improving their work as opposed to arguing over points.

What about grades/grading time?

- Course GPA is almost identical before/after
- Grading takes ***slightly*** longer than before:
 - Each grader grades ~10-12 students a week
 - About 20 minutes/student including all regrading



March 21-25, 2022 | San Francisco, CA #GDC22

GDC

The course GPA is almost the same – however it’s worth noting that since each criterion is evaluated independently, students who do very poorly on the exams can’t “make up” for it by getting straight As on the assignments.

We spend a little more time on grading, but the key thing is most of the time spent on grading is now on giving meaningful feedback as opposed to tabulating points.

GDC

March 21-25, 2022
San Francisco, CA

Implementation Tips

#GDC22



Now I'm going to go over some things I've learned along the way on how to successfully launch the system.

Budget enough time to explain it



March 21-25, 2022 | San Francisco, CA #GDC22

25 GDC

This is what your students might look like when you first try to explain the system. Some of you look like this right now, too.

You should plan on budgeting at least one hour in the first week to go over the grading system. And you can expect that you will have to reinforce the concepts repeatedly.

Until you have a pipeline of previous students, your graders will also have a hard time understanding the expectations of the grading. You will need to spend time going over what they should be looking for especially your dividing line between a B and an A.

As an instructor, it also takes getting used to. You can expect to iterate on the system for a while.

You don't need to change everything!

E, M, R, Z?



A, B, C, F?



One thing we've learned is to try to keep as many things as familiar as possible.

For example, our first stab at specifications grading was to use E, M, R, and Z for our grades. This just added another layer of unnecessary complexity, so we just renamed them to the much more familiar A, B, C, and F.

Keep in mind, you don't have to change everything at once.

Thanks

- Matt Whiting
- Clark Kromenaker

- All the undergrad TAs who do the grading!

Before I go to questions, I want to thank the other two faculty have been important in refactoring the grading system system – Matt Whiting and Clark Kromenaker. And I want to thank all our undergrad TAs over the past couple of years who've helped make the grading system a reality.

GDC

March 21-25, 2022
San Francisco, CA

Questions



#GDC22

GDC

March 21-25, 2022
San Francisco, CA

Extra Slides

#GDC22



GDC

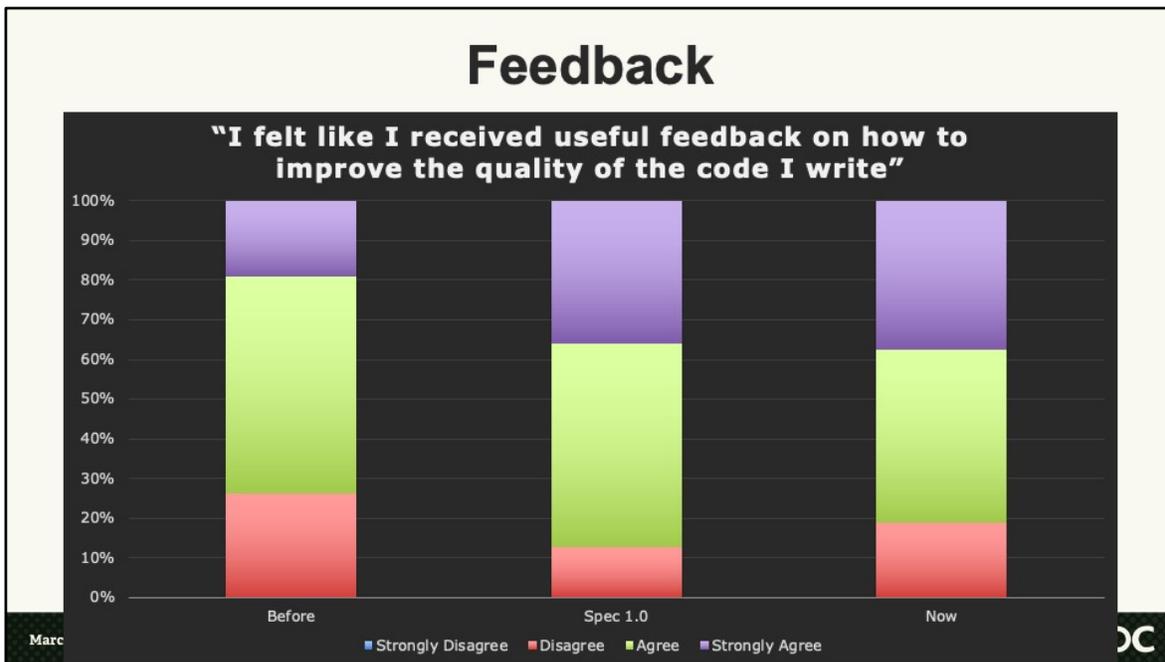
March 21-25, 2022
San Francisco, CA

What do students think?

#GDC22

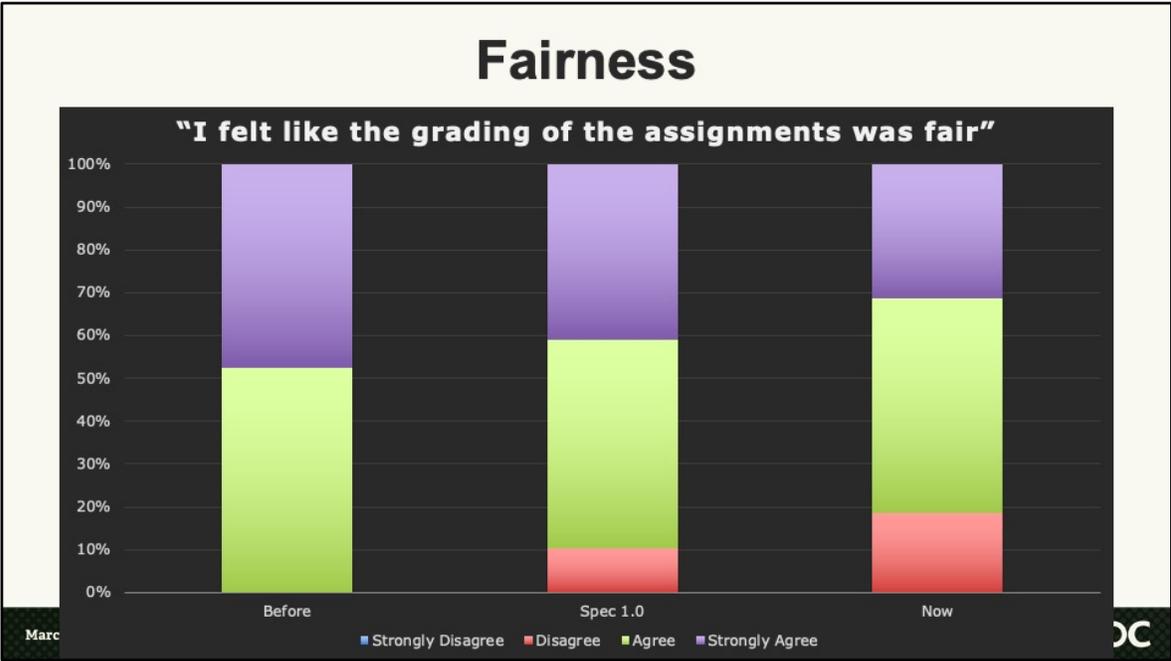


I'm not entirely sure how useful a data point it is to see what the students think, as sometimes they will be opposed to things which are beneficial to them. But for completeness we also have some survey results



“Before” was the last semester we used points, and “Specs 1.0” was the first semester we used spec grading, and now was the most recent semester (fall 2021). The first two surveys were about the same number of responses (around 40 students), but unfortunately, we only had ~15 responses for “now”.

There definitely are more students who “strongly agree” though the number of students who disagree isn’t quite 0.



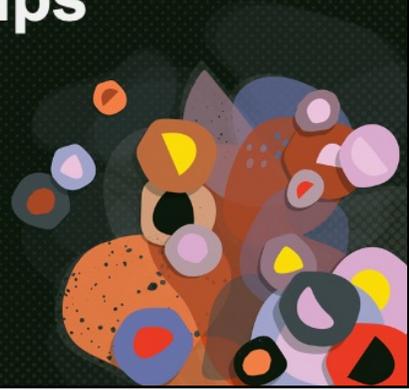
This was an interesting result, as we have 10-20% of the students who don't feel like the assignment grading is as fair. I think in part this is just because numbers and math inherently seem fairer. I think also some students don't like that you can't "make up" poor performance in one aspect by doing especially well on another aspect, which you could with just flat points.

GDC

March 21-25, 2022
San Francisco, CA

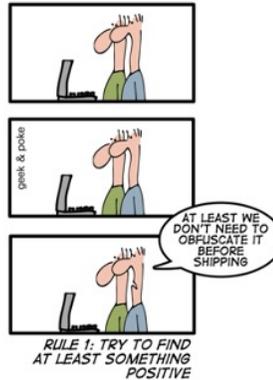
More implementation tips

#GDC22



Adapting to code review takes time

HOW TO MAKE A GOOD CODE REVIEW



By <https://geek-and-poke.com/> licensed under CC-BY

Since most classes (at least in the computer science world), don't implement code reviews, some of the students will be unfamiliar with the concept of not being able to just write the code and forget about it. It will take some time for students to understand how code review works. But over time the students become more familiar with it and increasingly get an "A" grade on the first submission as the semester progresses.

Visualizing the criteria

	1	2	3	4	5	6	7	8	9	10	11	12
A	B	B	B	B	A	A	A	A	A	A	A	A
A-	C	B	B	B	B	B	A	A	A	A	A	A
B+	C	B	B	B	B	B	B	B	A	A	A	A
B	C	C	B	B	B	B	B	B	B	A	A	A
B-	F	C	C	B	B	B	B	B	B	B	B	B
C+	F	F	C	C	B	B	B	B	B	B	B	B
C	F	F	C	C	C	B	B	B	B	B	B	B
C-	F	F	F	C	C	C	B	B	B	B	B	B
D	F	F	F	F	C	C	C	B	B	B	B	B

One issue you may run into is mid-semester, students may have difficulty understanding where they currently stand in the course. We've found making a table like this helps – it basically shows the minimum required for the assignments grade to achieve each grade. This also helps students “target” what grade they're looking for. For example if they're just targeting a C+ this means they can choose to skip 2 assignments (and get Fs on them).