# CTEC 2902 Code Review Form

The following exercise accounts for 10% of the marks for the second half of CTEC2902. The deadline is the same as the final project deadline – 23:59:59 on the 3rd May.

## 1: Clean Code

Take one piece of code that you think is a *good example of clean code* from any of the code bases of the team projects in the CTEC 2902 organisation GitHub (note, you may need to switch from the main branches to development branches to find the code). This does NOT include the code for the labs.

The code should be as clean as possible, but also be code that does something complicated / significant. (It’s really easy to keep simple code that’s not doing very much clean). You can pick one or more methods, or a whole class, or even a couple of classes that interact with each other.

It can be your own code, or someone else’s. Keep the creator anonymous, though – it doesn’t matter who wrote it, just that you think it’s clean, and can explain why.

Paste the code into the box below (don’t worry if it scrolls onto the next page or three):

|  |
| --- |
| //get all reviews about this article  //Declares a var called ArticleReview which stores all of the reviews for any given  //article. This is connected from the review table to the article table, and returns  //all reviews for the article it is fetching data for via the viewbag.  var ArticleReview = (from reviews in db.Review  join Article in db.Article  on reviews.ArticleID equals article.ArticleID  where (reviews.ArticleID == article.ArticleID)  select reviews);  List <Review> ReviewList = ArticleReview.ToList();  ViewBag.ReviewList = ReviewList;  return View(article); |

Now explain in no more than one side of A4 (at the very most) why you think it’s a good piece of clean code. (You may need to refer to the *List of Code Smells* at the end of this document for more ideas about clean versus smelly code).

Why I like this code

This piece of code is my favourite out of all of them because of how the way it works and how cleanly written it is. It carries out an important function for the site it was made before and links many different properties and elements together.

When the user is on the article page and wishes to see all of the reviews for said article, they click on ‘View Reviews’, which is where this piece of code kicks in. From the selected article, this code fetches all of the reviews for this article and stores them in the variable ArticleReview, which is then passed to the viewbag and appears on the site page.

The method is not too large and makes it easy for anyone else to come along, read it and understand what it does and why it is there. The comments are clear and precise and describe what the code below it does. This is done in enough detail than non-programmers would be able to understand it but not so much that it is explaining the absolutely obvious needlessly.

The variables are aptly named as are the tables and table headers. It is clear what each one is and what they do in the database.

In other development frameworks and programming languages, such a thing would require a lot more code, methods and classes in order to incorporate into functionality. However, MVC makes this a relatively easy task.

## (Abbreviated) list of code smells

These example *code smells* come from Clean Code by Robert C. Martin (Chapter 17), which you were pointed at (along with a lot of free resources provided by the author) during the very first lecture. If you want an example of some classic *dirty code*, check out the method I wrote for the start of week 26’s lab.

### Comment smells

1. A comment contains inappropriate or misleading information.
2. A comment is out of date and no longer matches the code.
3. A comment is redundant: i.e. it just describes something obvious that the code is clearly doing.
4. Commented out code has been left in the codebase.

### Method smells

1. A method has a large number of parameters, so that it’s hard to be clear what they all do.
2. The method isn’t called by any other method (i.e. it’s a dead method that’s still in the code when it doesn’t do anything).
3. One of the parameters is a *Boolean flag* or *switch*, which is passed in to make the method behave in more than one way. (Instead of a switch like this, you should have two methods that only do one thing, and put the switching logic at the point you call them).
4. Method too large: as a rule of thumb, a method you have to scroll through for a page or three is too long. It probably does too much, and needs breaking down into helper methods.

### General code smells

1. Duplicate code: a large chunk of code has been copied and pasted, and runs twice. (Week 26’s lab has a great example of this).
2. Dead code: chunks of code that don’t even get executed – e.g. an if statement for a condition that can’t ever happen, or a switch for a case that will never occur.
3. Vertical separation: helper methods should be close to the point in the code where they are called. You shouldn’t have to scroll several pages up or down to find them.
4. Misplaced responsibility: functionality should go in the place it makes most sense to put it. E.g. a “calculateOrderTotal()” method should go in an Order class, not in the OrdersController (see week 26 again).
5. Don’t use ‘magic numbers’. If you need to calculate VAT, set a constant VAT variable and set it to .2, rather than multiplying totals by .2 (with no real explanation why). Even better, set the VAT\_RATE as a variable in the app / web.config file so you can change it without rebuilding the whole application.
6. Code defensively: if a method needs a property to be set before it will run, check the variable is set, and if it isn’t throw an IllegalOperationException.
7. And while we’re talking about Exceptions, you have wrapped code that could fail (e.g. calls to a database that might go offline) in try / catch blocks, haven’t you?

### Variable Naming Smells

1. Make names descriptive. So ‘VATRate’ rather than ‘vr’.
2. Use standard names and stick to them. A good example from MVC is the ‘Create, Edit, Details, Delete’ standard: don’t rename these ‘Make, Change, Read, Kill’.
3. Make names unambiguous. So if you have a function for renaming an image file, call it ‘renameImageFile(string pathToFile)’ rather than ‘rename(string path)’.
4. The longer the variable is in scope for, the clearer its name should be: so if you name a loop counter ‘i’, but it’s only used on three lines then never referred to again, that’s fine. But calling a variable with class scope ‘i’, and then referring to it constantly throughout 750 lines of code is NOT fine.
5. Don’t use Hungarian notation. E.g. don’t call classes ‘clsMovie’. (MVC will break, for a start).

### Finally:

1. You did put some unit tests in, didn’t you?