# ITC205 Assignment 2 Guidelines.

There are three aspects to the assignment: teamwork, version control, and static code review. Here are some guidelines to help keep you on track and focused on what is most important to the assessment.

Perhaps the most important overall principle is this :

*things must not only be done, they must be* ***seen*** *to be done.*

What this means is: write things down and provide evidence for the assessment.

# Teamwork

1. Get started quickly. Time lost at the beginning of the project never comes back.
2. Don’t underestimate the importance of the team charter. It’s not a wishy-washy set of motherhood statements. It’s a contract between team members setting out expectations and commitments in terms of work completion, quality, time commitments, and communication commitments. One of its key features has to be its ‘failure to conform’ procedures – commonly known as its ‘strike policy’. This sets out how the team will deal with team members who don’t meet the charter commitments and is absolutely necessary to keep things civil and objective if things don’t work out. It’s the rules of the game, and its there to avoid nasty personalized conflicts.
3. Adopt a formal process to record team meetings minutes – attendance, topics discussed, decisions taken. Record action items assigned to team members.
4. Upload your team meeting minutes to a directory in your version control repository. Do this at the end of your team meeting, not in a couple of days when they’ve bee ‘polished’. The most important thing is that things are recorded and uploaded in a timely manner, not how beautifully they are presented.
5. Be prepared to do some work every week. In a team environment, other people will be depending on access to the work you are supposed to have done. Its not OK to let things slide.
6. Record who is assigned what task and be objective about whether the task is completed. ‘Done’ means ‘no further work required’. It means other team members can take the result of the task an incorporate it in their own tasks without issue. Maybe adopt some gradations of incomplete, but ‘done’ or ‘complete’ is binary. If something is not ‘done’ record the reasons why. It is extremely important that meeting minutes and iteration reports reflect accurately the actual state of your project – otherwise they are useless in post-mortems and as input into improving processes. Also, people forget.
7. Communication is key. Respond to messages from coworkers as fast as possible, ideally within a couple of hours. Taking a day to respond can really slow things down.
8. Reach out if you need help. Don’t leave it to the last minute. No-one likes the surprise of ‘Oh, I didn’t get it done, because …’ at the team meeting where the completed task is expected to be delivered. If you aren’t going to meet a deadline, let people know as soon as you think it might be an issue. Reach out for help as soon as you need it. Use the ‘2-hour rule' – if you have been bashing your head against a brick wall for 2 hours without progress – stop, and get help.

Overall – it is a big mistake to regard ‘team processes’ as burdensome and unnecessary. They might be in a small project undertaken by a group of close friends who have worked together in the past. However, for anything complex, involving relative strangers, for stuff that at least some of the team want a good result in, then making commitments explicit, recording things, and having processes to deal with failures is essential.

Be very wary of team members who don’t want to make commitments explicit and recorded. Frankly, it’s a red flag. If you do get one, make sure you insist on specifying outcome expectations, and get very explicit, definite, and strict on ‘failure to perform’ procedures. Avoid any sort of ‘get out of jail free’ clauses.

# Version Control

1. Establish a ‘feature branch’ for each task you are assigned. Don’t make the tasks too fine grained, but ‘fix class A’ is not a bad granularity. Don’t delete feature branches for the purpose of the assessment (you would in actual practice). However, DON’T keep working in a feature branch once you’ve merged it with master – always create a new one for further work and make sure you change into it before doing more coding.
2. Use many small commits. Generally, ‘atomic commits’ that represent a step in completing your task is a good level. Remember the thing here is that your commits are evidence of your work habits – a few commits spread out over a week is regarded more highly than an avalanche of commits just before a team meeting – or even worse a single commit right at the last minute.
3. Do your coding work in the repository directory – i.e. modify the files in the repository. DON’T make a copy of the repository files in another directory and then copy your modified files over the repository files once you think they are ready. That ALWAYS causes trouble.
4. Ensure that your feature branch is ALWAYS operational. I.E. use the ‘refactor’ mechanism in your IDE to rename variables and methods rather than doing it all by hand. If you do it by hand, go through and eliminate errors every time you change something – do NOT let errors build up, it becomes very difficult to sort out. Make small incremental changes and fix things to keep the code base operational.
5. DO NOT work in the master branch of your local repository. ALWAYS work in a feature branch. Easiest is to create branches on the server and then update your local repository from the server. However, if you do create a branch locally and need it on the server as evidence you’ve done some work (for the assessment) then you’ll have to do some googling or ask the lecturer to get the local branch up on the server.
6. DO NOT merge your feature branch into master unilaterally. All merges to master are to be done in the context of a static code review performed as part of a ‘pull request’. This means – don’t merge to master in your local repository. Leave master alone. Only update master by pulling a new version down from the server on completion of a reviewed pull request.
7. Organize merging your work on a weekly or fortnightly basis through a mechanism of coordinated pull requests.
8. You need to process pull requests in sequence. This because the results of a previous merge have to be incorporated and deconflicted BEFORE the next pull request can be properly reviewed and processed.
9. Take it slow and steady. Don’t go wild. Don’t take shortcuts. And whatever you do, and no matter how much some team members know, or think they know, use the standard feature branch process and stick to the merge protocol outlined below.
10. Make sure master still works after every merge. This is critical, and ‘breaking master’ will cost you in the assessment.

Overall, the single most important piece of advise is this :

*“merge from, before merging to”*

What this means is that PRIOR to issuing a pull request, you should update your local repository with the LATEST VERSION of master on the server, and then merge that latest version INTO YOUR FEATURE BRANCH. There may be conflicts, and your feature branch may stop working. It is important to sort that out BEFORE issuing a pull request to merge your branch into master on the server.

Remember a pull request is meant to be a WORKING update to master – it HAS to be based on the latest version of master. If your team issues multiple simultaneous pull requests, then all but the very first one have to update their local copy of master and merge/deconflict in their local branch, and then update the branch on the server AFTER the preceding pull request. It can get very confusing, and if a pull request is reviewed and approved without incorporating the latest version of master, then it could break master.

By far the simplest approach is to LINEARISE issuing pull requests. Coordinate with your team and in sequence:

For each feature branch to be merged:

1. update local master from the server,
2. merge local master into your feature branch,
3. deconflict/debug your modified feature branch so that every thing works,
4. THEN issue the pull request.
5. Repeat until all defects resolved
   1. Review the code and identify defects (uniquely identify defects)
   2. update your branch to eliminate any defects found (commit per defect),
   3. ensure your updated branch is still operational, upload changes to the server
   4. review fixes
6. approve the pull request and update master on the server.

The key concepts I’m looking for in the assessment are ‘methodical’, ‘coordinated’, ‘collaborative’, ‘organized’, and of course ‘competent’. Not to mention an ability to take a hint, and follow instructions.

# Static Code Review

1. Do not be mislead by the relative informality of the online environment for conducting the static review. The task is a FORMAL code review, so if defects are not identified uniquely, and there isn’t a commit identified with each defect to resolve it, or there is NO EVIDENCE that such a process was followed, then marks will suffer.
2. Be aware – penalties for remaining defects will be applied to the REVIEWER of the code – final code quality is the REVIEWER’s responsibility.
3. Similarly, it is the MEDIATOR’s responsibility if master is broken following a merge.
4. Make sure you create and retain a record of your review process. Most importantly, make sure you create a commit per defect resolution and identify the fix with the defect it addresses.
5. Allow plenty of time to conduct reviews. It can take a day to review code, a day to resolve issues, and another day to check the resolutions. Then it can take another day to merge the updated master with the next feature branch to be merged before the review cycle starts again for the next pull request.

Overall – the **formal** static review is the point where most teams struggle. The key requirement is documented **formality** of process – get yourselves organized, know what order you are going to conduct the pull request/merges, KEEP A RECORD of defects found, and IDENTIFY the defects the resolution commits are associated with.

OK – I hope that gives some guidance as to what is expected, what to pay attention to, and a little bit as to how to go about the assignment. All the best with the assignment.