Mandatory exercise: Hand in number 3 – Developing in a workflow

This mandatory exercise reviews your skills in planning and implementing new features for a Microwave Oven System, which has already been developed and tested. The design and explanation of the Microwave Oven System can be seen in the handouts for Lecture 09.1.

NOTE: This exercise is *mandatory*. This means that it is a prerequisite for your admission to the exam of this course that you have handed this exercise in and have a grade of PASSED (see details below).

You shall hand in in groups – no solo work. The guidelines for group work are stated in the note "*Teams, lab exercises and hand-ins in I4SWT.pdf*" available on the course Brightspace site under *Gruppetilmelding*.

---

1 The exercise

You shall hand in your solution to the exercise introduced in Lecture 10.1 and 10.2: adding new features using a structured work-flow, using git, GitHub, Jenkins and other tools.

Some details on the hand-in:

1. Your hand in is uploaded to Brightspace, using the Assignment made available for this hand in, by delivering exactly one (1) PDF document containing:
   - Your team number
   - A table containing the student number and name of each participant in the exercise.
   - Direct URLs for all the Jenkins build jobs executing your tests for each new feature, and the Jenkisn job for the main branch, eg. http://ci3.ase.au.dk:8080/job/SWTE21_xx_MW_Feature1
   - A URL to the GitHub repository, you are using as the shared remote repository for your team, e.g. http://github.com/TeamSWTxxx/MicrowaveHandin3/
   - A new class diagram describing the final testable design after you have added all features
   - Relevant new sequence diagrams describing each new feature
   - A new STM diagram for the `UserInterface` class, describing the final result after you have added all features
   - Any new STM diagrams, if you have chosen to use an STM in other classes to implement the new features
   - A short description of **what** has been changed and **what** has been added in relation to the diagrams (you **don't** have to explain **why**)
   - A short description of your decisions for the individual features where you had a choice
   - If you have made personal features, a full specification of those

2. The GitHub repository shall contain
   - The original code, with all changes you found necessary as a result of adding the new features. This will be changes you have made to the original implementation classes, **and** necessary changes to the original unit tests, **and** new unit tests to test the new features, **and** changes to the main program to make a demonstration of the new features
   - The main branch with the finished result after adding all features
   - All feature branches used for working on the new features
   - All history and other information as a result of following the workflow using GitHuB for commits, pull requests, pushes, merges, etc. In other words, **do not clean up**, we will use it to grade your hand in

3. The Jenkins server shall contain the Jenkins Build and Unit Test with Coverage jobs – one for each featurebranch you have been working on **and** one for the main branch with the integrated new features. Do not clean up.

4. Your hand-in shall be made in groups as stipulated in Section 2 in the aforementioned note. No solo work!
5. Your hand-in shall be on time. A late hand-in is not accepted and will be evaluated as "**FAILED**" (see below)

No ZIP files! If you need separate files with diagrams, upload them together with the PDF file under item 1) above as extra files in JPG, PNG or Visio format. The diagrams must be easy to read, even if they are big.

2 Evaluation of your hand-in

2.1 Evaluation criteria

The evaluation of your exercise depends on fulfilling the total set of criteria. The total set of criteria is described in a Rubric, visible at the Assignment Hand-In link on Brightspace.

2.2 Evaluation grades

The evaluation of this exercise will have 1 of 3 possible grades:

"**2 - PASSED**"	Your hand-in is of sufficient quality to pass the criteria for approval
"**1 - FAILED – RESUBMIT**"	Your hand-in is of insufficient quality to pass the criteria for approval. You are granted one resubmittal. The requirements and deadline for this resubmittal will be conveyed along with the grade. The resubmittal will be graded either "PASSED" or "FAILED" (no 2nd resubmittal)
"**0 - FAILED**"	Your hand-in is of insufficient quality to pass the criteria for approval. You are not granted resubmittal.

Note that your hand-in must be graded PASSED (in either first or second attempt) for the members of the group to be admitted to the exam in the course.

3 The Microwave Oven system

The design and explanation of the Microwave Oven System can be seen in the handouts for Lecture 09.1.

There are several ways you can arrange to get the code for the original Microwave Oven System, described in the above mentioned handouts.

Don't just clone from TeamSWTFrabj, as you **cannot** use **my** repository for your feature work!

It is very important that you have your common and all local repos (on your work PCs) set up correctly before you start adding the features for this exercise. Make some simple tests from each of the team's working PC and a Jenkins job running the unit tests with coverage based on the main branch. You will need this Jenkins job anyway. You will also need to have a web-hook from GitHub to Jenkins.

4 New features

This section describes some new features for the Microwave Oven as it is in its current state. Some of these you **must** implement, some of them you are free to add if you have time.

For these new features you must work in parallel following a structured work flow, described during Lecture 10.1 and 10.2, and further specified in the next section. That is the main purpose of this mandatory exercise.

The new features **must** be unit tested. All necessary changes to the original unit tests **must also be made**. And the main program **must be changed** to demonstrate the new features when relevant.

4.1 Mandatory new features

The following features must be added to the Microwave Oven.

4.1.1 Add a buzzer/beeper

Support for a hardware sound buzzer must be added, and it must be used at the end of the cooking period, sending out 3 short burst of buzzing sounds.

You are free to add further specifications – e.g. also using the buzzer in other situations during the operation and usage of the Microwave Oven, or making the buzzer have several sound effects.

You are free to specify the interface, but there probably must be a new boundary class interfacing to the hardware, in a similar way to the other hardware, e.g. the Light.

4.1.2 Make the power tube power configurable

The Microwave as described and implemented has a power tube power of 700 W hard coded into it. This means that it is only usable directly for such a hardware configuration.

You must make changes to the design, such that this value is configurable to other values – 500 W, 800 W, 1000 W, etc. – from the `main` function, when setting up and connecting all the modules before the system starts running.

Storing this configuration, e.g. in some attached ROM chip or file, is not part of this exercise.

So (after this) the value will be hard coded in the main function, but the rest of the code will not need to be changed, when using the software with another power tube.

This will demand changes in several classes. If you can redesign how the maximum power is used, to get better handling of this, this is an extra plus.

4.1.3 Change the time while cooking

You must add the functionality to extend – or perhaps shorten – the cooking time when the oven is actually cooking.

A simple way to do that is letting a press on the time button during cooking add an amount of seconds to the remaining time, and the cooking period will then be prolonged for that amount. The display should of course show the new remaining cooking time.

If this function can make the time go to 0 (zero), then the oven should stop immediately.

You are free to define the user interface, etc. for this function, but it will probably demand changes in several modules. If you can make good design choices, e.g. considering the single responsibility principle, that is an extra plus and will probably make it easier to implement and test.

E.g. you are free to add an extra button that detracts from the time – so that there is an add time and a subtract time in steps you define.

4.2 Optional new features

The following features can be added to your solution.

They cannot replace any of the mandatory new features – see the previous section – unless it has been approved by the teachers.

4.2.1 Add a second button

Add support for a fourth button which can be used to add seconds instead of whole minutes, to make it possible to specify the cooking time more precisely.

Each press should add a amount of seconds to the cooking time, during the state when the cooking time is input, and it should be reflected on the display. It can do that in steps you decide, and when it wraps around to zero, it can increment the minute number or not, according to your own decision.

4.2.2 Let the time steps be progressively larger

Another way to enable a more precise time setting would be to let the pressing of the time button progress from 5 s, to 10 s, to 30 s to 1 min, etc, such that short times can be input. The jumps in the cooking time for each press of the time button will be progressively larger according to a pattern you decide.

4.2.3 Your own feature.

You are also free to think of a new feature to be added. It doesn't have to be approved by us, as long as you do the mandatory features, but it is a good idea to use us for feedback concerning the size and difficulty of adding your own idea.

5 Work process

The main purpose of this exercise is to practice and learn about ways to work in parallel, similar to a real world project.

You will be introduced to the general principles in the lectures 10.1 and 10.2, but for this hand in you must follow the instructions described in the following sub-sections.

5.1 Working on a feature

To learn the most, you should distribute the features to several sub-teams, who will work in parallel on different features.

For a new feature, create a feature branch.

You must integrate from main often, using a git rebase, to reduce the amount of problems, when you will integrate the new feature into main. That way will solve conflicts early.

Use commit often when working on a feature, but before integrating into main, you must make “history rewrite”, reorganizing your commits into a suitable (smaller) number of commits, perhaps even one.

You must have a Jenkins Build and Test with Coverage job for the feature branch.

Remember – always give your commits a meaningful description so that the history of changes as seen in the commit log will give a good explanation of what has been done.

For “history rewrite” to work with the least problems, it is important that you work on a feature in **one** local repository. You can work more than one person on a feature, but it must be pair programming on the same PC/local repo. If your network and your PC is powerful enough, you could use the Live Share extension for Visual Studio for working virtually on one PC.

5.2 Integrating into the main branch

When a feature is finished, you start the integration into main process.

First make a last merge from main.

When any problems from that are fixed, issue a pull-request.

This pull-request must be reviewed by somebody else from your team in a line-by-line review.

If the first pull-request is not accepted because issues are discovered, continue work on the feature and restart the integration, including a new pull-request.

When a pull-request is accepted, use the special merge pull-request (essentially a no-fast-forward option, --no-ff). This will make your commits in the feature branch and the final merge commit visible on the main branch.

6 Tips and tricks

As we all – including the teachers – collect some experience with this work process and our setup, Tips and Tricks will be published on Brightspace. Keep an eye on those!