SE 3K04 - L01 G7
Assignment 1 - DCM

Carlos Capili
Raeed Hassan
Shaqeeb Momen
Aaron Pinto
Udeep Shah

Contents

1 Requirements

1.1 Welcome Screen

All required aspects of the welcome screen are implemented in the current software revision:

- Register a new user
- Login as an existing user
- Maximum of 10 users stored locally

All requirements will remain in future software revisions.

1.2 User Interface Essential Aspects

All required essential aspects of the user interface have been implemented in the current software revision:

- User interface is capable of utilizing and managing windows for display of text and graphics
- User interface is capable of processing user positioning and input buttons
- User interface is capable of displaying all programmable parameters for review and modification
- User interface is capable of visually indicating when the DCM and the device are communicating
- User interface is capable of visually indicating when a different PACEMAKER device is approached than was previously interrogated

All requirements will remain in future software revisions.

1.3 DCM Utility Functions

1.3.1 About

All required aspects of the About function are implemented in the current software revision:

- Application model number
- Application software revision
- DCM serial number
- Institution name

All requirements will remain in future software revisions.

1.3.2 Set Clock

The Set Clock function is not implemented, but has been allocated a window and a button on the DCM main window. This function will likely be removed in a future revision of the software depending on the feasibility of implementing the feature.

1.3.3 New Patient

The New Patient function is implemented in the current software revision. The function will remain in future software revisions.

1.3.4 Quit

The Quit function is implemented in the current software revision. The function will remain in future software revisions.

1.4 Pacing Mode Interfaces

Interfaces for pacing modes and pacing mode selection for the following pacing modes are implemented in the current software revision:

- AOO
- AAI
- VOO
- VVI

Additional pacing modes will be added to the requirements and implemented in future software revisions including:

- DOO
- AOOR
- VOOR
- AAIR
- VVIR
- DOOR

1.5 Printed Reports

All common requirements for printed reports are implemented in the current software revision:

- Header Information
  - Application model and version number

- Device model and serial number
- DCM serial number
- Date and time of report printing
- Report name

All common requirements for printed reports will remain in future software revisions.

1.5.1 Bradycardia Parameters

The feature is implemented in the current software revision. The feature will remain in future software revisions.

1.5.2 Temporary Parameters

The feature has been allocated a button in the reports window in the current software revision. This feature will likely be removed in a future revision of the software as it is unlikely that a temporary pacing mode is introduced.

1.6 Programmable Parameters

Provisions for storing programmable parameter data have been implemented in the current software revision for the following programmable paramters:

- Lower Rate Limit
- Upper Rate Limit
- Atrial Amplitude
- Atrial Pulse Width
- Atrial Refractory Period (ARP)
- Ventricular Amplitude
- Ventricular Pulse Width
- Ventricular Refractory Period (VRP)

These programmable parameters will remain in future software revisions. Additional programmable parameters will be implemented in future software revisions which will depend on future requirements.

1.7 Real-time Electrograms

The following requirements of real-time electrograms and the displaying of electrograms have been implemented in the current software revision:

- Real-time internal electrograms shall be made available from atrial and ventricular sense/pace leads
- Electrogram viewing
  - The user shall have the option of viewing electrograms on the screen
  - The user shall have the option of selecting which electrograms are viewed
- The system is capable of displaying real-time traces in a scrollable fashion

These features will remain in future software revisions.

Requirements that will be implemented in future software revisions include:

- The DCM shall be capable of printing real time telemetered data
  - will be implemented after serial communication with pacemaker for telemetry has been implemented

Requirements that involve the surface ECG will not be implemented in future requirements due to the lack of a surface ECG on the pacemaker.

1.8 Future Requirements

Future requirements that will be implemented include:

- Serial communication between DCM and Pacemaker
  - Sending programmable parameters to Pacemaker
  - Receiving telemetry data from Pacemaker
- Input validation for safety critical Pacemaker systems operated by DCM
- Additional pacing modes

2 Software

2.1 General Design Decisions

The DCM and all its modules were programmed in Python 3. The decision to use Python was largely due to the programmers' familiarity with the Python language and Python libraries that could be used to handle different DCM systems. The release of Python decided was Python 3.7.9; an earlier release was selected to ensure compatibility with potential libraries.

The GUI was designed using PyQt, a Python library for Qt, a widget-toolkit for designing and creating graphical user interfaces. It was initially decided that Python's standard Tkinter package and it's associated tools would be used, however it was later decided that a tool that would allow us to create our user interface graphically instead of programmatically would ease our workflow. The PyQt5 library was ultimately decided upon due to the ability to automatically generate the code for the graphical user interface from design files created using Qt Designer, a tool to graphically design GUI elements for Qt.

Serial communcation between the DCM and Pacemaker and device recognition is handled by the pyserial library due to the programmers' familiarity with the library.

Graphs in the DCM are handled by the PyQtGraph library. The library was chosen due to it's straightforward integration with the PyQt library and it's ability to easily create interactive plots.

2.2 GUI

2.2.1 Welcome Screen

The welcome window is the window the user is greeted by when the DCM program is launched, allowing the user to register users or login into the DCM. The Welcome Screen is a QStackedWidget, a Qt class that contains multiple widgets but only displays one at a time. The QStackedWidget class was chosen to allow all modules of the welcome screen to be accessible in a single window, while still allowing individual modules to behave independently. Additional modules can easily be added as new widgets without affecting implemented modules.

2.2.1.1 Welcome

The welcome screen is the first widget in the welcome window that is displayed when the window is launched. The widget gives the user the option to register a new user or login as an existing user through push buttons displayed in the widget. The welcome widget is a QWidget with internal functions that cannot be accessed by the other widgets. The widget's only functions are to direct the user to another widget when a push button is pressed. The register button will lead the user to the register widget and the login button will lead the user to the login widget. A tabular expression detailing the expected behaviour of the widget's functions are shown in Table 1.

Push Button	Function
welcome_ui.reg_btn	welcome_gui.setCurrentIndex(1)
welcome_ui.log_btn	welcome_gui.setCurrentIndex(2)

Table 1: Welcome Screen GUI Connections

2.2.1.2 Register

The register screen can only be navigated to from the welcome screen. The register screens gives the user the ability to register a new user using a username and password, or return to the welcome screen. The register widget is a QWidget with internal functions that cannot be accessed by the other widgets. The widget's functions are accepting user inputs in the username and password fields, authorizing the user inputs, or directing the user to another widget. The register button will call the authentication handler's `register` funnction with the text in the username and password fields as inputs. The back button will return the user to the welcome screen. A tabular expression detailing the expected behaviour of the widget's functions are shown in Table 2.

Push Button	Function
welcome_ui.reg_back_btn	welcome_gui.setCurrentIndex(0)
welcome_ui.reg_submit_btn	auth.register

Table 2: Register GUI Connections

2.2.1.3 Login

The login screen can only be navigated to from the welcome screen. The login screens gives the user the ability to login as an existing user using a username and password, or return to the welcome screen. The login widget is a QWidget with internal functions that cannot be accessed by the other widgets. The widget's functions are accepting user inputs in the username and password fields, authorizing the user inputs, or directing the user to another widget. The login button will call the authentication handler's `login` funnction with the text in the username and password fields as inputs. The back button will return the user to the welcome screen. A tabular expression detailing the expected behaviour of the widget's functions are shown in Table 3.

Push Button	Function
welcome_ui.log_back_btn	welcome_gui.setCurrentIndex(0)
welcome_ui.log_submit_btn	auth.login

Table 3: Login GUI Connections

2.2.2 DCM Main Window

The main window is the window which contains all the DCM functions. The main window has three main functions: presenting important information about the pacemaker (DCM's connection status with the pacemaker, the pacemaker's pacing mode, graphical telemetry data from the pacemaker), providing the user the ability to change pacing modes, and allowing the user to access the DCM's functions. The main window is a QMainWindow, a Qt class intended for main window management. The class was chosen due to it's ability to add status bars, which will be used to display information about the DCM's connection with the pacemaker. The main window consists of three main sections: real-time electrograms, pacing mode interfaces, and DCM functions. While there are some interactions between these sections (particularly the pacing mode interfaces and the modules in the DCM functions), they are largely contained in their own modules that can function independently from each other. Future modules can be added to any section without affecting implemented modules (with the exception of modules that do not have any defined behaviour for additional pacing modes).

The real-time electrograms show real-time telemetry data from the pace and sense leads on the pacemaker. Checkboxes for pace/sense leads will show and hide the pace/sense plots. The expected behaviour of the checkboxes are shown in Table 4.

The pacing mode interfaces allow the user to select the operating pacing mode of the pacemaker. Pacing mode selection is implemented using the QRadioGroup class, containing instances of the QRadioButton class for each pacing mode. The QRadioGroup class only allows one radio button to be switched on at a time, ensuring only a single pacing mode is selected at all times. Functions in other modules that rely on the pacing mode can rely on the functions of the QRadioGroup class, ensuring that the individual radio buttons cannot be altered.

The DCM's functions are operated through push buttons on the main window. The push buttons will either call a function from a handler (new patient), or direct the user to another window. Relevant handler functions are called in these new windows depending on user selected options presented in these windows. The expected behaviour of the push buttons to access DCM functions are shown in Table 4.

Additional pacing modes are expected to be added in future revisions of the software, and behaviour for these pacing modes have to be accounted for in modules that require the pacing mode (or implement behaviour to handle unknown pacing modes).

UI Element		Function
dcm_ui.about_btn		about_gui.exec_
dcm_ui.parameters_btn		params_gui.exec_
dcm_ui.reports_btn		reports_gui.exec_
dcm_ui.set_clock_btn		set_clock_gui.exec_
dcm_ui.new_patient_btn		conn.register_device
dcm_ui.quit_btn		dcm_gui.close
dcm_ui.pace_box	isChecked	graphs.pace_show
	!isChecked	graphs.pace_hide
dcm_ui.sense_box	isChecked	graphs.sense_show
	!isChecked	graphs.sense_hide

Table 4: DCM Main Window GUI Connections

2.2.2.1 Electrograms

The graphs use the PlotWidget class from PyQtGraph, a QGraphicsWidget with additional functions for plotting. The data in the plots is randomly generated and the plotted when the DCM main window is opened. The random data is currently a placeholder until the ability to receive telemetry data from the pacemaker is implemented. All functions of the plots are handled by graphs handler, including showing and hiding the graphs with the lead checkboxes.

2.2.2.2 Reports

The reports window provides the user the option to view one of the following printed reports: the electrogram report, the bradycardia parameters report, and the temporary parameters report. The reports window is a QDialog window that does not interact with other modules in the DCM. The window contains three push buttons that call functions in the reports handler with the pacing mode of the DCM as an input. The expected behaviour of the push buttons is shown in Table 5. The reports are shown in a popup window that is generated by the reports handler. The temporary parameters button will likely be removed in future software revisions, depending on if a temporary pacing mode is implemented.

Push Button	Function
reports_ui.egram_btn	reports.generate_egram(get_pace_mode_params())
reports_ui.brady_btn	reports.generate_brady(get_pace_mode_params())
reports_ui.temp_btn	reports.generate_temp(get_pace_mode_params())

Table 5: DCM Reports GUI Connections

2.2.2.3 Parameters

The parameters window displays all the DCM programmable parameters to the user and allows the user to modify them and save their changes. The parameters window contains a QTableWidget, a widget providing an item-based table view. The horizontal headers contains a list of the programmable parameters that have been implemented, while the vertical headers contain the current values, the maximum and minimum values for the parameter, and the increment of the parameter. The values in the cells of the table are pre-programmed, with the maximum, minimum and increment values being unselectable (the user is unable to modify the cells' values).

The current parameter values are programmed into the table, however they are immediately replaced by any changes that have been saved to the parameters file. The current value cells are selectable, allowing the user to modify their values, however these values are not saved to the parameters file until the "Confirm changes" button has been pressed. The "Reset to defaults" button will reset the parameter files values to the default values programmed into the QTableWidget cells. These functions are all handled by the parameters handler, with the push buttons calling functions within the handler.

Modules that rely on the programmable parameters will retrieve them from the parameters file through the parameters handler, the parameters window does not directly interact with any modules other than the parameters handler.

Input validation will be implemented in future software revisions to ensure the user inputs are within the permissible ranges due to the safety critical nature of the system. The parameters window will also emphasize programmable parameters that are relevant to the current pacing mode. The implementation has not been decided yet, but it is likely either irrelevant parameters will be hidden or irrelevant parameters will be disabled (greyed out in GUI). These features will primarily be implemented through the parameters handler.

2.2.2.4 About

The about window displays the application model number, application software revision, DCM serial number and institution name. The about window contains a QTableWidget. The headers and cells of the table are set to the appropriate values and made unselectable, making the user unable to modify the cells' values. The widget only shows pre-programmed values and does not interact with any other modules in the DCM.

2.2.2.5 New Patient

The New Patient push button calls the `register_device` function in the connection handler. The connection handler will generate a popup message depending on the results of the function.

2.2.2.6 Quit

The Quit push button closes the DCM GUI and exits the program.

2.2.2.7 Animated Status Bar

The animated status bar is a status bar for displaying the connection status between the DCM and the pacemaker with some integrated animations that depend on the connection status. The animated status bar is an AnimatedStatusBar, a class that extends the QStatusBar class from Qt with additional functions that interact with the connection handler. The AnimatedStatusBar's `handle_conn_anim` function is called whenever the state changes in the connection handler (the AnimatedStatusBar monitors a signal for state changes), and the function will display a programmed message and animation depending on the connection state.

The status bar operates independently from other modules in the DCM, and its behaviour is driven by events from the connection handler.

2.3 Handlers

The handlers are the backend of the program, where all the actions from the GUI are implemented. Each handler is independent from each other, but they can interact through passing in function arguments. For example with the reports and parameters handlers, the reports are generated from the input passed into each method which comes from the parameters handler.

2.3.1 Authentication

The authentication handler handles the authentication process for the user. The three main public functions are the constructor, `login`, and `register`.

The constructor takes in a Python callable named `on_success`, which gets called when the login or registration process is successful. It also initializes the credential store, which stores the user credentials while the program is running.

The `login` method takes in a username and password as strings, which are used to authenticate the user. It essentially compares the username and password to the existing entries in the credential store, and if they match it calls the `on_success` function.

The `register` method also takes in a username and password as strings, which are used to create a new user. The function checks if the username exists or if there are already 10 users, and if none of those are true it updates the credential store, writes it to a file and calls the `on_success` function.

The state variables for the auth module are the credential store, which is a dictionary of strings where the keys are the usernames and the values are the passwords, and the `on_success` function. The private function of the module is the `_show_alert(msg: str)` function which takes in a message as a string and displays that on an error alert box that appears on top of the current window.

2.3.2 Connection

The connections handler handles the pacemaker connection for the DCM and extends the PyQt5 `QThread` class to allow for multithreading. There is also the `_SerialHandler` class which is a private class that handles the serial communication with the pacemaker, but it is just a placeholder and not fully implemented, because that is for Assignment 2. It is still commented, but will not be talked about in this document, for the aforementioned reasons.

In the connections handler, the 5 public functions are the constructor, `run`, `stop`, `register_device`, and `send_data_to_pacemaker`.

The constructor only initializes the state variables to default values, and starts the serial handler thread.

The `run` method is called when the thread starts and overrides the method in `QThread`. It sets its running variable to True, emits an initial state change signal and then loops until the program is closed, calling the `_update_state` method every 10ms approximately.

The `stop` method is used to stop the thread and stop the serial conn thread. The `register_device` method is called when the New Patient button is pressed and it registers the pacemaker if the current state is CONNECTED, otherwise it displays an alert depending on other conditions. For example, it will display "Please plug in a pacemaker!" when the New Patient button is pressed and there are no pacemakers plugged into the PC.

The `send_data_to_pacemaker` method takes in a dictionary of strings, named params and is not fully implemented because of no serial communication, so its just a placeholder.

The state variables for the connection module are running, which is just a bool that controls the while loop in the `thread`, `device`, `devices`, `old_devices`, `first_serial_num`, `current_state`, `prev_state`, `wanted_state`, `serial`, and `connect_status_change`.

`device` is a pyserial `ListPortInfo` variable that stores the info of the connected pacemaker, `devices` is a list of `ListPortInfo` devices, which contains all the connected COM port devices, `old_devices` is the same as `devices` but one cycle behind and used for figuring out which devices were added and removed between cycles, using the difference between the two lists. `first_serial_num` contains the serial number of the first pacemaker plugged into the PC, for auto-registering purposes. `current_state` holds the current `PacemakerState` state, `prev_state` is the same as `current_state` but one cycle behind and is used for detecting state transitions in the new state, and `wanted_state` is the same as well but one cycle ahead of `current_state`. `serial` is the `_SerialHandler` instance. The `connect_status_change` PyQt5 `pyqtSignal` is a thread-safe signal that is used to inform the animated status bar when the connection status has changed.

The 4 private functions in the module are `_update_state`, `_handle_removed_device([ListPortInfo])`, `_show_alert(msg)`, and `_filter_devices([ListPortInfo])`.

`_handle_removed_device` which takes in a list of ListPortInfo objects named removed, checks if a removed device matches the connected pacemaker serial number, and if it does then it sets he wanted state to `NOT_CONNECTED`, emits the `connect_status_change` sig-

nal and resets `device` to its initial value. `_show_alert` takes in a message as a string and displays that on an information alert box that appears on top of the current window. `_filter_devices` takes in a list of `ListPortInfo` objects and filters them by Vendor ID and Product ID so that we only connect to pacemaker devices, and then it returns that filtered list.

`_update_state` is a method that contains a state machine for the pacemaker connection state. It was implemented like this because it offers us many benefits such as cleaner, easier to read code, ensuring that a pacemaker gets registered only once, handling multiple pacemakers being plugged into the same computer, and handling the New Patient button presses in a much simpler way.

Every 10ms cycle, the method gets the filtered list of connected COM port devices, and finds the devices that were added or removed between cycles, using the difference between `devices` and `old_devices`. Then it updates the `current_state` if its not aligned with the `wanted_state`.

If we're not connected to a pacemaker, we figure out if any new devices were added, and then set the `wanted_state` accordingly, based on the state table below. If we are connected to a pacemaker, regardless of whether ot not it is registered, we call `_handle_removed_device` to check if any devices were removed, and then transition state accordingly (refer to table below).

On state transitions, we emit the `connect_status_change` signal with the corresponding current or wanted state, depending on if the emit happens before or after the `current_state` is updated.

At the end of the method, we update the variables that store previous cycle information, namely `prev_state` and `old_devices`. As you can see below in Table 6, the state transitions are straightforward.

		PREVIOUS STATE		
Event	Condition	NOT_CONNECTED	CONNECTED	REGISTERED
_update_state	len(added) >0 and ((self.first_serial_num == "") or (self.first_serial_num == self.device.serial_num))	REGISTERED	-	-
	len(added) >0 and !((self.first_serial_num == "") or (self.first_serial_num == self.device.serial_num))	CONNECTED	-	-
register_device	-	-	REGISTERED	-
_handle_removed_device	len(removed) >0 and (self.device.serial_number == removed[0].serial_number)	-	NOT_CONNECTED	NOT_CONNECTED

Table 6: Reports Handler Stateflow

2.3.3 Graphs

The graphs handler handles the drawing and updating of the graphs in the DCM window and extends the PyQt5 QThread class to allow for future multithreading.

The 7 public functions are the constructor, `pace/sense_plot`, `pace/sense_show`, and `pace/sense_hide`.

The constructor takes in two PyQtGraph PlotWidgets named `atri/vent_plot`, for the atrial and ventricular plots, which are used to generate the plot data items. It also generates some random sample data for right now, so that we can show something on the graphs.

The `pace/sense_plot` methods set the data of the plot data items to their corresponding sample data set, and then show the data on the graphs as well. The `pace/sense_show` methods show the corresponding plot data item on the graphs, while the `pace/sense_hide` methods hide them.

The state variables for the graph module are `atri/vent_data_pace` and `atri/vent_data_sense`, which store the pace and sense lead data as 1D arrays, and `atri/vent_pace_plot` and `atri/vent_sense_plot` which are the PlotDataItems for the graphs.

There are no private functions in the module.

2.3.4 Parameters

The parameters handler handles storing and updating the parameters for the DCM.

The 4 public functions are the constructor, `confirm`, `reset`, and `filter_params`.

The constructor takes in a PyQt5 `TableWidget` named table, which is used to retrieve and update values from the GUI, so they are only hard-coded once. It also creates a dictionary of parameters per pacing mode, and it initializes the `default_params`, the units and the `params_store` dictionaries. If the `params_store` file already exists, it will load the params from there and then update the GUI to display those values instead.

The confirm method is used to update the `params_store` dict from the params GUI table and write the new values to file, creating the file if it doesn't exist. The reset method is used to reset the param values to the defaults hard-coded into the params GUI table. It prompts the user if they are sure they want to reset all the values, and then optionally loads the GUI defaults and updates the file and params GUI table to reflect those changes.

The `filter_params` method takes in a string named `pacing_mode` which is used to get a dictionary that contains only the parameters for the specified pacing mode. It returns a dictionary with the param names as keys, and the param value with units as values.

The state variables for the params module are `table`, which is just the params GUI table, `params_per_mode`, `default_params_store`, `units`, and `params_store`.

`Params_per_mode` is a dictionary that maps the pacing mode as a string, specified as the keys, to the programmable parameters for that mode as a list of strings, specified as the value. This mapping was found in Table 6 of the PACEMAKER System Specification document.

The `default_params_store` and units variables are simply dictionaries of strings with the parameter names as keys and the parameter values and units, as values respectively. The

`params_store` is the exact same data structure as `default_params_store`, but it stores the most recent values of the params GUI table.

The two private functions in the module are `_update_params_file`, which just writes the `params_store` to file, creating a new one if it doesn't exist, and `_update_params_gui` which just updates the params GUI table with the values from the `params_store`.

Input validation will be implemented in future software revisions to ensure the user inputs are within the permissible ranges due to safety critical nature of the system. The parameters window will also emphasize programmable parameters that are relevant to the current pacing mode. The implementation has not been decided yet, but it is likely either irrelevant parameters will be hidden or irrelevant parameters will be disabled (greyed out in GUI).

2.3.5 Reports

The reports handler handles generating and displaying all the reports for the DCM window.

The 4 public functions are the constructor, `generate_egram`, `generate_brady`, and `generate_temp`.

The constructor takes in a PyQt5 `TableWidget` named table, which is used to generate the header for each report.

The `generate_egram` method is not implemented yet because we don't have actual electrogram data, and `generate_temp` will most likely get removed.The `generate_brady` method takes in a dictionary of strings named params, which are all the parameters for the current pacing mode. It then joins the header string and all the parameters as strings on new lines, before displaying the report in a custom message box.

The state variable for the report module is just the header string, which contains each "About" property on a new line.

The two private functions in the module are `_format_params(params: Dict[str, str])` and `_show_report(report: str)`. `_format_params` just takes in a dictionary of parameters as strings, and then returns a string with each param aligned to the colon and each parameter name/value on a new line. `_show_report` takes in a report as a string and shows a customized message box with the specified text on top of the current window.

3 Testing

3.1 GUI

All testing for the GUI was done manually. Tests for all modules confirmed that their functions were behaving as intended.

3.1.1 Welcome Screen

The push buttons were tested to check if they led to the appropriate GUI elements and called the correct functions.

In the Welcome window, the register and login buttons were pressed, and they lead to the register and login screens respectively.

In the register screen, the back button led the user back to the welcome screen. The text fields allowed user input up to 15 characters with the password field being masked with bullet points. The submit button saved the new user in the credentials file and sent the user to the DCM main window when there were less than 10 users registered. The submit button displayed a popup message telling the user that the maximum of 10 registered users had been reached when there were 10 users registered. The submit button displayed a popup message telling the user to login instead if the username already exists in the credentials file.

In the login screen, the back button led the user back to the welcome screen. The text fields allowed user input up to 15 characters with the password field being masked with bullet points. The submit button sent the user to the DCM main window when the user's credentials were present in the credentials file. The submit button displayed a popup message instructing the user to register the account if the user did not exist in the credentials file. The submit button displayed a popup message telling the user that the password is incorrect if the username existed in the credentials file but the passwords did not match.

3.1.2 DCM Main Window

The push buttons, check boxes, graphs and radio buttons were tested to see if they led to the appropriate GUI elements and called the correct functions.

The pace electrograms were no longer displayed when the pace leads checkbox was unchecked. The pace electrograms were displayed again when the pace leads checkbox was checked. The sense electrograms were no longer displayed when the sense leads checkbox was unchecked. The sense electrograms were displayed again when the sense leads checkbox was checked. The electrogram graphs were scrollable.

The Reports window was opened when the reports button was pressed. The Parameters window was opened when the parameters button was pressed. The About window was opened when the about button was pressed. The Set Clock window was opened when the set clock window was pressed. The new patient button could not be tested in its current implementation as the testers only had access to one pacemaker. The program was exitted when the quit button was pressed.

3.1.3 Reports

The push buttons were tested to check if they called the correct functions.

The "Electrogram" and "Temporary Parameters" buttons did not display anything. The "Bradycardia Parameters" button displayed a popup window showing the necessary header information. The pacing mode selected in the DCM main window and the relevant programmable parameters for the pacing mode were displayed following the header information.

Other functions of the main window could not be accessed while the Reports window was active.

3.1.4 Parameters

The push buttons and the data table were tested to check if they called the correct functions and displayed the intended behaviour.

The only values displayed that could be changed were the ones listed under the "Values" column. All other text could not be modified. Changing the values in the column did not immediately update the parameters file.

The parameters file was updated when the "Confirm changes" button was pressed with modified values in the table.

The reset to defaults button displayed a popup window to confirm if you want to reset all values. The "No" option returned the user to the Parameters window with no changes to the paramaters file. The "Yes" option returned the user to the Parameters window with the values in the parameters file and the values showed in the table being set to their default values.

Other functions of the main window could not be accessed while the Parameters window was active.

3.1.5 About

None of the cells in the table could be selected or modified.

Other functions of the main window could not be accessed while the About window was active.

3.1.6 New Patient

The New Patient functions could not be tested in its current implementation due the testers only having access to one pacemaker.

3.2 Handlers

The handlers were largely tested alongside the GUI as most handler functions are called from within the GUI. Tests for all modules confirmed that their functions were behaving as intended.

3.2.1 Authentication

The authentication handler and its functions were tested in the register and login widgets in Section 3.1.1.

3.2.2 Connection

The status bar showed the message "Not connected to pacemaker" when the DCM main window was reached with no pacemaker connected. The status bar showed "Registered pacemaker 7" when a pacemaker was connected. When the pacemaker was disconnected, the status bar showed "Not connected to pacemaker 7". There was no testing done on functions that relied on an unregistered pacemaker to be connected, as the testing could only be done with a single pacemaker.

3.2.3 Graphs

The graphs handler and its functions were tested with the DCM main window electrograms in Section 3.1.2

3.2.4 Reports

The reports handler and its functions were tested with the reports window in section 3.1.3.

3.2.5 Parameters

The parameters handler and its functions were tested in the parameters window in Section 3.1.4.