Link to Repo:
https://github.com/CJDAmico/CommsOnWheels
Link to JIRA:
https://comms-on-wheels.atlassian.net/jira/software/projects/SCRUM/boards/1/backlog

Set up Steps:

- Download Qt Creator 14.0.2 (Check if a new version exists)
- Clone the repository
- In Open Project, select CMakeLists.txt

Code Documentation:

- **Backend:**
    - **dbcdata.h and DbcDataModel class:**
        - Defines key data structures such as Network, Node, Message, Signal, and their relationships.
        - Provides the DbcDataModel class, which contains the logic for importing and managing DBC (Database Container) and JSON files.
        - Key public methods:
            - setFileName(const QString&) and fileName(): Manage the associated file's name.
            - importDBC(const QString&): Imports and parses a DBC file.
            - loadJson(const QString&): Loads and parses a JSON file.
            - Accessors for parsed entities (networks(), nodes(), messages()).
    - **importDBC() [dbcdata.h]:**
        - Parses DBC files line by line, using regular expressions to identify and extract information about nodes, messages, signals, and attributes.

    - Constructs and organizes these elements into structured classes (Node, Message, etc.), linking them appropriately.
    - Handles default values and attribute assignments during parsing.

  - **parseJson() [dbcdata.h]:**
    - Reads and processes JSON files.
    - Extracts data about networks (buses), messages, signals, and nodes, mapping them into class structures.
    - Sorts entities (e.g., networks, messages) alphabetically for easier management and consistency.

- <u>**Frontend:**</u>
  - **findOrCreateItem() [dbctree.cpp]:**
    - Helper function for populateTree(). If populateTree() is called on an already populated tree, this function helps find existing Tree Widget items or creates them if they don't exist. Returns the newly created or found QTreeWidgetItem.
  - **populateTree() [dbctree.cpp]:**
    - Goes through the list of DbcDataModels inputted and generates a QTreeWidget with the correct data hierarchy and connections.
    - QTreeWidgetItem::setData() is used to store information about what that item is.
      - data(0, Qt::UserRole) is the type of item (Message, Signal, etc.)
      - data(0, Qt::UserRole + 1) is the associated model (network) name
      - data(0, Qt::UserRole + 2) is the unique PGN for messages
  - **onTreeItemClicked() [mainwindow.cpp]:**
    - Called whenever a QTreeWidgetItem is clicked. Will extract the data from the QTreeWidgetItem and call the corresponding handle*Item() function based on the item type (Network, Node, Message, or Signal).

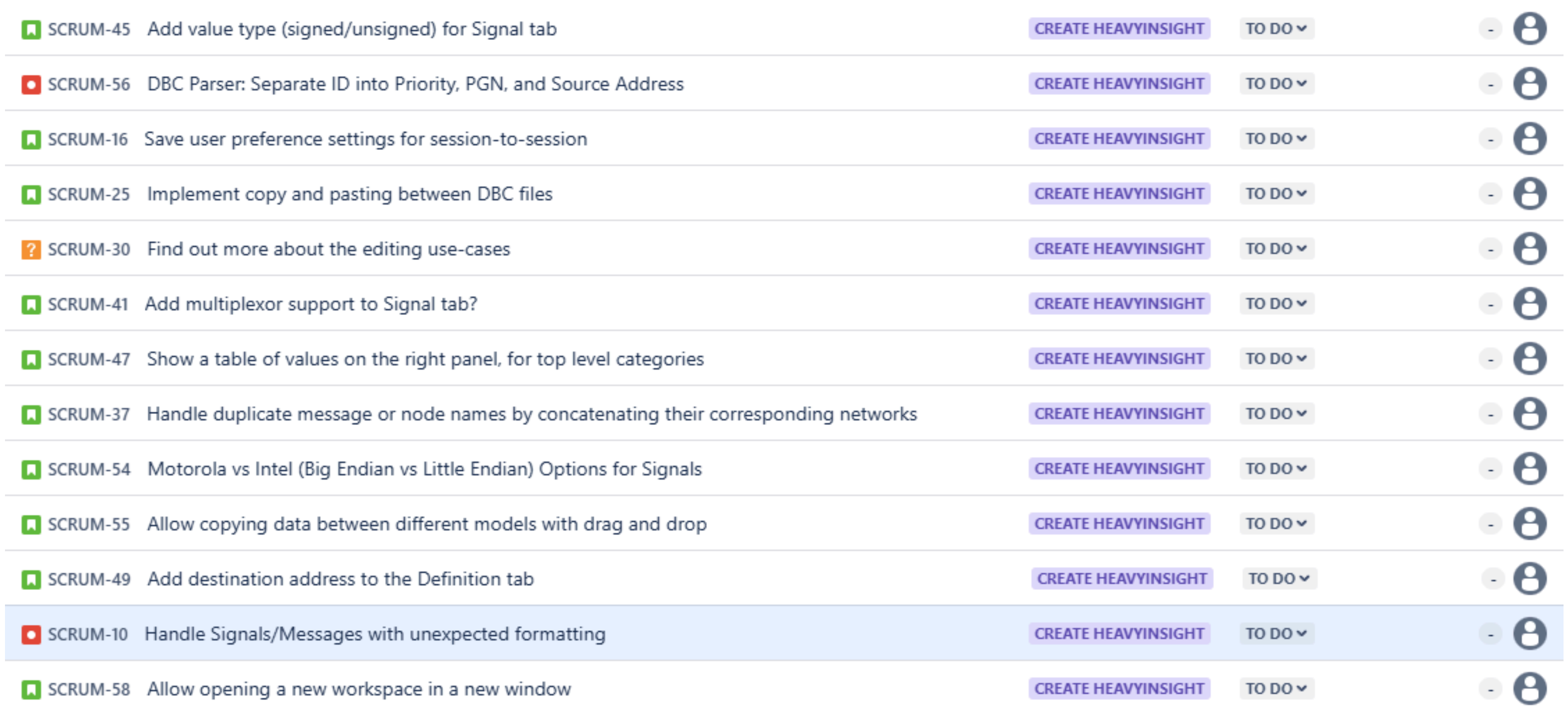
        - Adjusts the left and right panel split dimensions every time the right panel is activated/populated.
    - **handle*Item() functions [mainwindow.cpp]:**
        - * represents the type of item: Network, Node, Message, or Signal.
        - Finds the corresponding item through the given modelName and any other unique info that might've been stored with QTreeWidgetItem::setData().
        - Extracts all the info that will be displayed for the item's tab and sets the corresponding UI widgets with that information.
        - Adds any additional tabs that need to be seen to the rightPanel:
            - Ex: rightPanel→addTab(layoutTab, "Layout");
        - rightPanel→setCurrentWidget(tab) will set the right panel to show that tab
    - **setupRightPanel() [mainwindow.cpp]:**
        - setupRightPanel() is only called once to initialize all the possible right panel tabs, their widgets, and the UI elements.
        - We don't need to call this multiple times because we will only change the data and select which tabs are hidden or displayed for the right panel based on which item is clicked.
    - **clearRightPanel() [mainwindow.cpp]:**
        - clearRightPanel() is called inside onTreeItemClicked() every time a different item is selected.
        - Since we need to reset the data and tabs for different types of items, clearRightPanel() will do the clean-up to prepare for adding new UI connections and data for the right panel.

<u>What works so far</u>:

- **File → Open From Json..** opens a workspace from a save file (JSON)
    - File → Recent Files shows 5 most recently used save files for quicker opening.
- **File → Import DBC…** allows importing data from a DBC file into the workspace.
- **File → Recent Imports** shows 5 most recently used DBC files for quicker importing.

- **File → New** creates a new workspace to start from scratch.
- Users can click on any item on the tree, which will open a right panel to view information about all item types (Network, Node, Message, Signals) and edit their information.
- Users can use the search bar to filter for any item name
- **Edit → Expand All** will expand all tree items' children
- **Edit → Collapse All** will collapse all tree items' children
- Undo & Redo:
    - **Ctrl-Z**: Undo most recent changes while editing a text box
    - **Ctrl-Shift-Z**: Redo most recent changes while editing a text box
    - Clicking on another item will reset and prevent this
- **File → Save** uses a timestamp as the default name and saves the workspace into the default "saves/" folder inside HeavyInsight. (JSON).
- **File → Save As…** allows the user to choose a name and path to save the workspace into (JSON).

Future Roadmap:



SCRUM-45	Add value type (signed/unsigned) for Signal tab	CREATE HEAVYINSIGHT	TO DO
SCRUM-56	DBC Parser: Separate ID into Priority, PGN, and Source Address	CREATE HEAVYINSIGHT	TO DO
SCRUM-16	Save user preference settings for session-to-session	CREATE HEAVYINSIGHT	TO DO
SCRUM-25	Implement copy and pasting between DBC files	CREATE HEAVYINSIGHT	TO DO
SCRUM-30	Find out more about the editing use-cases	CREATE HEAVYINSIGHT	TO DO
SCRUM-41	Add multiplexor support to Signal tab?	CREATE HEAVYINSIGHT	TO DO
SCRUM-47	Show a table of values on the right panel, for top level categories	CREATE HEAVYINSIGHT	TO DO
SCRUM-37	Handle duplicate message or node names by concatenating their corresponding networks	CREATE HEAVYINSIGHT	TO DO
SCRUM-54	Motorola vs Intel (Big Endian vs Little Endian) Options for Signals	CREATE HEAVYINSIGHT	TO DO
SCRUM-55	Allow copying data between different models with drag and drop	CREATE HEAVYINSIGHT	TO DO
SCRUM-49	Add destination address to the Definition tab	CREATE HEAVYINSIGHT	TO DO
SCRUM-10	Handle Signals/Messages with unexpected formatting	CREATE HEAVYINSIGHT	TO DO
SCRUM-58	Allow opening a new workspace in a new window	CREATE HEAVYINSIGHT	TO DO

Firstly, parsing DBC files will need to be shored up. This application needs to be able to handle entries with unexpected formatting as well as entries with duplicate names. From there, in order to achieve a minimum viable product (i.e. something that's actually usable), it is important to begin work on exporting changes made in HeavyInsight to a DBC file. Once that is implemented, HeavyInsight will essentially have the same capabilities as CANdb++. From there, it will be important to implement the changes that address the pain points with CANdb++. Namely, functionality related to managing multiple DBC files at once such as copy and pasting between them. Along the way, it will be important to communicate with potential end users so that any potential edge cases are handled and information/functionality is presented in an intuitive way.