```
# Pseudo-Python code for building ART-WD model

# Inputs:
# X_data : list of input vectors (with no missing attributes)
# m : number of attributes per vector
# P_r_func : function to calculate threshold

def build_ART_WD_model(X_data, m, P_r_func):

    # Step 1: Remove vectors with missing attributes (already assumed)

    # Step 2: Set number of input neurons
    input_neurons = list(range(m))  # i = 1..m

    # Step 3: Initialize recognition layer
    k = 1
    recognition_neurons = list(range(k))  # j = 1..k

    # Step 4: Initialize ascending weights (input -> recognition)
    W_up = [[1 for j in recognition_neurons] for i in input_neurons]  # w_ij = 1

    # Step 5: Initialize control layer
    S_j = 1
    control_neurons = [[0 for i in input_neurons] for l in range(S_j)]  # m x S_j

    # Step 6: Initialize descending weights (recognition -> control)
    W_down = [[[1 for i in input_neurons] for l in range(S_j)] for j in
recognition_neurons]  # w_ji^l = 1

    # Step 7: Calculate threshold for current class
```

```
P_r = P_r_func()  # user-defined function

# Step 8: Process input vectors
for X_star in X_data:
    U = normalize(X_star)  # optional normalization
    k_star = k

    # Step 9: Calculate outputs of recognition neurons
    Y = []
    for j in recognition_neurons[:k_star]:
        y_j = sqrt(sum((U[i] - W_up[i][j])**2 for i in range(m)))
        Y.append(y_j)

    # Step 10: Find winner neuron
    j_star = Y.index(min(Y))

    # Step 11: Calculate distance RN between U and winner neuron
    r_values = [rho(U, control_neurons[l]) for l in range(S_j)]
    RN = sum(r_values) / len(r_values)

    # Step 12: Resonance condition
    if RN < P_r:
        # Step 12a: Update ascending weights
        for i in range(m):
            W_up[i][j_star] = (S_j * W_up[i][j_star] + U[i]) / (S_j + 1)

        # Step 12b: Update descending weights
        W_down[j_star].append(U.copy())  # new control neuron

        # Step 12c: Create new class
```

```
            control_neurons.append(U.copy())
            S_j += 1

        # Step 13: No resonance
        else:
            if k_star > 1 and Y[j_star] < P_r:
                # Reset operation
                Y.pop(j_star)
                k_star -= 1
                j_star = Y.index(min(Y))  # repeat Step 10
            else:
                # Adaptation: create new recognition neuron
                recognition_neurons.append(k)

                # Step 13.2: Ascending weights
                for i in range(m):
                    W_up[i].append(U[i])

                # Step 13.3: Descending weights
                W_down.append([U.copy()])  # S = 1

                k += 1

        # Step 14: Continue with next input vector
        # if all vectors processed, model is built

    return {
        "input_neurons": input_neurons,
        "recognition_neurons": recognition_neurons,
        "ascending_weights": W_up,
```

```
        "descending_weights": W_down,
        "control_neurons": control_neurons,
        "class": k
    }

# Helper functions
def normalize(X):
    # Example normalization (optional)
    return [x / max(X) for x in X]

def rho(U1, U2):
    # Distance measure (e.g., Euclidean)
    return sqrt(sum((u1 - u2)**2 for u1, u2 in zip(U1, U2)))
```