Develop an App to Efficiently Create Peddimat Programs with C#.NET for Peddinghaus Machines

Anthony Wang[1]

Received September 28, 2025
Accepted January 14, 2026
Electronic access January 31, 2026

Inefficiencies and manual errors in generating Peddimat programs within the legacy software (v1.08.07) limit productivity in fabrication workflows. This study presents an Excel-based VSTO Add-In, developed with Visual Studio 2022, Microsoft Excel 365, and AutoCAD 2026, for the first time to create Peddimat programs and generate AutoCAD Dwg or Dxf files for verification and nesting programs for Peddimat plate processors. Key contributions include interpretation of Peddimat program structure, a fluent builder interface with recursive design patterns for object construction, dependency injection for VSTO Add-In architecture, and custom .NET Framework class libraries supporting dynamic formulas, unit conversion, and clip management of plates. A specialized Excel template enables rapid insertion of hole rows and types using predefined patterns, streamlining data preparation for both Peddimat programs and AutoCAD Dwg files. Performance evaluation shows a 20% reduction in program creation time, along with a lower error rate and improved verification efficiency for components with many holes. These results demonstrate that the proposed system enhances usability, reduces errors, and provides a practical, scalable framework for integrating engineering automation tools into Excel-based environments.

Keywords: Peddimat Programs, .NET Framework Class Library, User-Defined Functions (UDFs), VSTO Add-in, Excel Add-ins, AutoCAD Dwg, C#.NET

Introduction

Peddinghaus Corporation is a global leader in machine-tool technology for structural steel and plate fabrication, offering legacy software used to process components such as plates, angles, channels, beams, and tubes. While this software remains functional and well-integrated with Peddinghaus hardware, it exhibits inefficiencies in the generation and verification of Peddimat programs, particularly for large-scale drilling operations. This issue is inherent to legacy software [1].

The legacy List window displays only Abs-X-Dim, Inc-X-Dim, Rev-X-Dim, Gage (Abs-Y-Dim), S (tool type), T (tool number) and Options for holes, slots and copes. Because Inc-Y-Dim values aren't shown, programmers must manually determine and enter Abs-Y-Dim values for the Gage field instead of using the incremental Y-dimensions provided in the shop drawings when the vertical spacings vary. Similarly, the absence of a Rev-Y-Dim column requires programmers to manually calculate reverse Y-dimensions when verifying programs against drawings. Most manual errors are caused by miscalculations in the Rev-Y-Dim. The legacy List window provides a Matrix (Rectangle) pattern that allows programmers to specify the number of rows, columns, and pitch to generate hole data lines. The first step is to create the initial hole data line since the pattern will be applied according to this reference. In addition, the row and column pitches are constant within the pattern; if shop drawings require variable horizontal or vertical spacing, programmers must manually modify the hole data lines after the pattern is applied. Because the List window provides no positional references between hole data lines, changes made to one hole position affect only that individual line. As a result, programmers must adjust hole positions and rely heavily on the select, copy, and paste functions. For enormous quantities of holes, the Matrix pattern generates numerous hole data lines, significantly increasing manual operations. The Matrix (Rectangle) pattern can be cached by entering its corresponding pattern number; however, in the legacy system (v1.08.07), reloading it via shortcut keys is not possible. Moreover, the caching functions provide little efficiency gain, particularly when vertical or horizontal spacings vary. Another shortcoming is that when programmers modify the Abs-X-Dim, Inc-X-Dim, or Rev-X-Dim dimension, the cursor automatically moves to the Gage column, even when the Gage value is already correct. Although the List window provides the cells, programmers still must constantly use calculators to determine the dimensions and manually enter them into the cells. These limitations introduce substantial manual operations, computation and increase the likelihood of pro-

[1] *Parkland High School, 2700 N Cedar Crest Blvd, Allentown, PA 18104* anthonywang1215@gmail.com

gramming errors, which affects small and medium-sized fabricators that continue to rely on traditional AutoCAD-based workflows.

Another major limitation of the legacy software is its inability to generate AutoCAD Dwg files. Since modern plate-processing nesting programs require AutoCAD Dxf or DSTV files[2–9], programmers must rely on separate systems to produce Dwg outputs, further fragmenting the workflow.

This paper addresses these challenges by proposing, for the first time[10], an integrated workflow that enhances both the creation and verification of Peddimat programs while enabling the generation of corresponding AutoCAD Dwg files from the same dataset. By incorporating spreadsheet templates with patterns, formula-driven inputs, and AutoCAD, the proposed solution aims to increase operational efficiency and reduce errors in structural steel fabrication.

Approaches

Microsoft Excel provides a flexible and extensible platform for computational automation through native Excel Add-ins and COM Add-ins[11]. Excel Add-ins enable the creation of custom formulas (UDFs) that automatically update with cell value changes, supporting real-time, data-driven calculations. COM Add-ins developed using Visual Studio Tools for Office (VSTO) further extend this functionality by integrating advanced user interface components, such as Ribbon controls, directly within Excel. This integrated environment offers a user-friendly experience, reduces the learning curve, and eliminates the need to switch between applications.

Based on these advantages, an Excel VSTO Add-in is chosen over a standalone desktop application, leveraging Excel's widespread adoption as a standard engineering tool. Excel's native spreadsheet interface is used for input management, formula computation, and visualization, while the VSTO layer manages automation tasks, including data processing, program generation, and file export. This design minimizes development overhead, ensures system compatibility, and enables direct linkage between calculated data and generated Peddimat program outputs.

Using this approach, users can define hole patterns, apply formula-driven dimensions, and adjust IncX and IncY parameters to automatically compute X and Y coordinates for fabrication programs. The same dataset is then used to generate both Peddimat programs and AutoCAD Dwg files, which can be programmatically created and converted to Dxf format without manually opening AutoCAD. This enables integration with Peddinghaus plate processors and allows direct visual comparison between generated drawings and shop drawings for verification. Overall, this architecture reduces manual effort, minimizes errors, and enhances productivity and product quality in structural steel fabrication.

Peddimat File Structure

With legacy Peddimat software, sample program gradually is modified, saving it under different names, and then compare it with the original using NotePad++ and AISC shapes database to understand the meanings of each line in the program. Peddimat program is the proprietary legacy programming format used by Peddinghaus CNC machines to define drilling, and cutting operations. Each Peddimat program is a structured ASCII text file consisting of header data, machining block, and termination character. The syntax follows a strict positional and numerical layout that controls both toolpath and geometry execution. There are four lines for header data in Peddimat program. The definition of the first three lines are shown in Table 1.

The definition of the fourth line is shown in Table 2.

The machining block includes hole data lines, mark data lines and cope lines. Table 3 shows partial of machining block.

This research only focuses on plate Peddimat programs creation and AutoCAD Dwg files generation[12,13]. Future work should include all AISC shapes.

Implementation

Design Architecture

A VSTO Add-in project follows a layered architecture designed to separate user interface elements from application logic and Office automation. PlantUML (v1.2025.10) and Visual Studio Code 2022 are used to create design architecture by generating puml files[14,15]. Figure 1 illustrates the application entry point and user interface layer of the Peddimat Excel add-in. This add-in serves as the VSTO add-in entry point, configuring injection through Microsoft.Extensions.DependencyInjection during startup to register all services, builders, and repository implementations as transient based on their lifecycle requirements[16,17]. The Ribbon class provides the Excel ribbon interface with custom buttons that trigger the add-in's core operations for generating Peddimat programs and AutoCAD Dwg files. The plate-Copes Windows form offers a dialog interface for user interactions specific to cope operations on structural steel plates. This architecture establishes a clean separation between UI concerns and business logic while enabling testability through constructor-based dependency injections.

Figure 2 represents the core business logic layer of the Peddimat system, implementing seven service interfaces that encapsulate domain-specific operations for Peddimat program and CAD integration. The services transform Excel worksheet data into structured formats: GeometryAndToolSize converts structural member dimensions and tool specifications, HeaderData assembles program metadata, and MachiningBlock gen-

Table 1 First Four Lines for Header Data

Position	Value	Definition	Comments
First line	B12 beam	piece mark and comments	maximum 8 characters for piece mark
Second line	W12X50	shape	
Third line	B	type	B for beams, C for channels, L for angles and P for plates
Fourth line	1 3111.5 95.25 2063.75 158.75 0 1 2 15240 254 238.125 269.875 206.375 238.125 269.875 238.125 238.125 269.875 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0		

Table 2 The Definition of the Fourth Line of Header Data (All Measurements are in Millimetres $\times$ 10)

Position	Value	Definition	Position	Value	Definition
1	1	quantity	11	238.125	bottom flange tool 1 size
2	3111.5	web height	12	269.875	top flange tool 1 size
3	95.25	web thickness	13	206.375	web tool 2 size
4	2063.75	flange width	14	238.125	bottom flange tool 2 size
5	158.75	flange thickness	15	269.875	top flange tool 2 size
6	0	unknown	16	238.125	web tool 3 size
7	1	left miter angle	17	238.125	bottom flange tool 3 size
8	2	right miter angle	18	269.875	top flange tool 3 size
9	15240	length	19	18@0	reserved for future uses
10	254	web tool 1 size			

erates CNC hole positions and cope sizes with proper scaling. Pattern automates the creation of hole patterns in Excel, while QuickOperation provides worksheet manipulation utilities. CreateDrawing orchestrates AutoCAD Dwg files generation with geometric primitives, and SaveAsDxf manages Dxf file export. Each service follows dependency injection patterns, accepting repository and converter dependencies through constructors to maintain loose coupling and testability. This diagram shows all service interfaces and implementations and business logics for geometry, headers, drilling, burning, drawing, and patterns.

Figure 3 demonstrates the data orchestration and infrastructure layer of the Peddimat system. The Builder implements the Builder pattern to construct Peddimat program files and AutoCAD Dwg files through a fluent interface. The Repository provides centralized data access to Excel worksheet cells, abstracting the underlying data source from spreadsheet template. Three external .NET libraries extend the system: PeddimatCopesLib handles cope with geometry calculations for plates, FracDecConverter manages bidirectional conversions between fractional inches and decimal values, and CustomFormula provides Excel UDF functions for coordinate transformations and dimensional conversions used in the worksheet interface. This diagram shows builder pattern PeddimatProgramBuilder, DrawingBuilder and data access abstraction repository and libraries PeddimatCopesLib, FracDecConverter, and CustomFormula.

All source code in the design architecture was developed using Visual Studio 2022 C#.NET, Microsoft 365 Excel, and AutoCAD 2026, running on the Windows 11 operating system [18–21].

Computational Logic

Peddimat and AutoCAD Coordinate Systems

In the Peddimat coordinate system, a hole located at (X,Y) corresponds to $(X,W-Y)$ in the AutoCAD coordinate system, where W is the width of the piece. The Excel spreadsheet template uses the Peddimat coordinate system for defining X and Y coordinates. As shown in Figure 4, the origin $(0,0)$ in Peddimat is at the upper-left corner whereas in Figure 5, the AutoCAD coordinate system places the origin at the lower-left corner.

Pseudocode for X and Y Coordinates Calculation with Custom Formulas (UDFs)

The Formulas class, located in the CustomFormula namespace, contains all custom formulas that can be used in an Excel spreadsheet after building the project and integrating the resulting Excel Add-in. The FeetInchToDec and DecToFeetInch classes are responsible for converting between fractional and decimal values for English units within the Formulas class. The formulas below are used to determine the X and Y coordinates.

Table 3 The Definition of Machining Block (All Measurements are in Millimetres × 1000)

Value	Description	Definition
56	number lines of machine block plus 1	number lines of machine block + 1
25400 53975.1010	hole data Line	(25400 53975) is X coordinate and Y coordinate; ".1010" is Web Tool 1, and 53975.1010 is formatted with left padding to 12 characters.
16400 2031.0020	mark data line	(16400, 2031) are the X and Y coordinates; ".0020" represents the mark "0", ".0020–.0090" map to digits "0–9", and ".0100–.0360" map to letters "A–Z". The value 2031.0020 is left-padded to 12 characters.
1462590 448500.6000 p10261	cope lines	(1,462,590, 448,500) are the X and Y coordinates; ".6000" represents the cope named P10261.
SUB	special character	End of the program

Algorithm 1 XCor Function

function XCor(len, prevXcor, incX, startX, prevTool, tool)
declare xTemp
if incX is null or empty **then**
    incX ← "0"
end if
remove leading and trailing spaces from len
remove leading and trailing spaces from incX
if (prevXcor equals "X" **or** prevTool equals "TOOL") **and** startX is not "R" **then**
    if incX contains "," **or** incX **contains** "@" **then**
        xTemp ← Cor(incX)
    else
        xTemp ← ConvertFeetInchesToDecimal(incX)
    end if
else if first character of prevTool is not equal to first character of tool **or** startX equals "S" **then**
    if incX contains "," **or** incX **contains** "@" **then**
        xTemp ← Cor(incX)
    else
        xTemp ← ConvertFeetInchesToDecimal(incX)
    end if
else if startX equals "R" **then**
    xTemp ← ConvertFeetInchesToDecimal(len) minus ConvertFeetInchesToDecimal(incX)
else
    if incX contains "," **or** incX **contains** "@" **then**
        xTemp ← Cor(incX)
        xTemp ← xTemp + ConvertFeetInchesToDecimal(prevXcor)
    else
        xTemp ← ConvertFeetInchesToDecimal(prevXcor) plus ConvertFeetInchesToDecimal(incX)
    end if
end if
return xTemp rounded to 6 decimal places

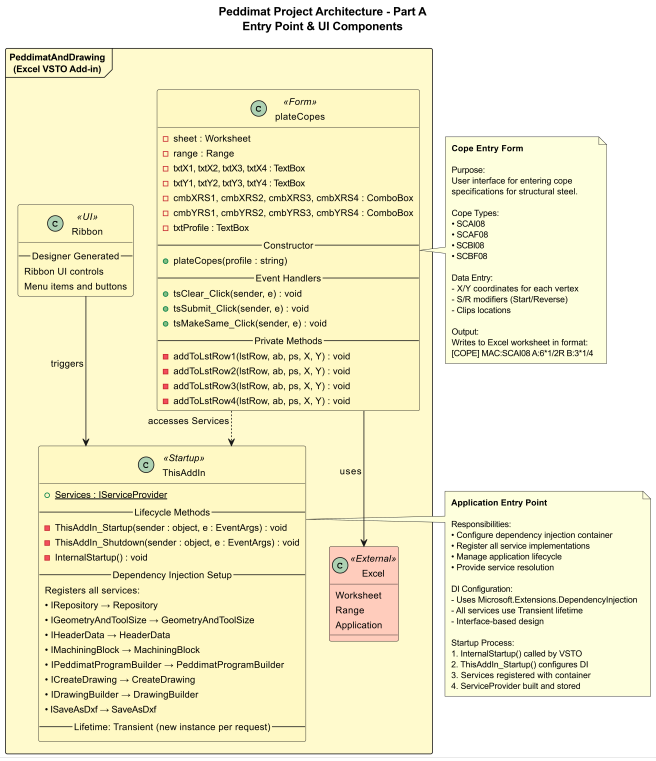

Fig. 1 Architecture of Entry Point and UI Components

Where `len` represents the length of the part, `width` represents the width of the part, `prevXcor` and `prevYcor` represent the previous X and Y coordinates respectively, `incX` and `incY` represent the spacing in the X and Y directions, and `tool` represents the tool type.

Holes Patterns

Algorithm 2 Cor Function

function XCor(inc)
split inc by “,” into incParts (ignore empty values)
temp ← 0
for each part **in** incParts **do**
    if part contains “@” **then**
        multiplier ← integer before “@”
        value ← substring after “@”
        temp ← temp + (multiplier × ConvertFeetInchesToDecimal(value))
    else
        temp ← temp + ConvertFeetInchesToDecimal(part)
    end if
end for
return temp rounded to 6 decimal places

Algorithm 3 YCor Function

function YCor(width, prevYcor, incY, startY, prevTool, tool)
declare yTemp ← 0
if incY is null or empty **then**
    incY ← “0”
end if
remove leading and trailing spaces from incY
if (prevYcor equals “Y” **or** prevTool equals “TOOL”) **and** startY is not “R” **then**
    if incY contains “,” **or** incY **contains** “@” **then**
        yTemp ← Cor(incY)
    else
        yTemp ← ConvertFeetInchesToDecimal(incY)
    end if
else if first character of prevTool is not equal to first character of tool **or** startY equals “S” **then**
    if incY contains “,” **or** incY **contains** “@” **then**
        yTemp ← Cor(incY)
    else
        yTemp ← ConvertFeetInchesToDecimal(incY)
    end if
else if startY equals “R” **then**
    yTemp ← ConvertFeetInchesToDecimal(width) minus ConvertFeetInchesToDecimal(incY)
else
    if incY contains “,” **or** incY **contains** “@” **then**
        yTemp ← Cor(incY)
        yTemp ← yTemp + ConvertFeetInchesToDecimal(prevYcor)
    else
        yTemp ← ConvertFeetInchesToDecimal(prevYcor) plus ConvertFeetInchesToDecimal(incY)
    end if
end if
return yTemp rounded to 6 decimal places

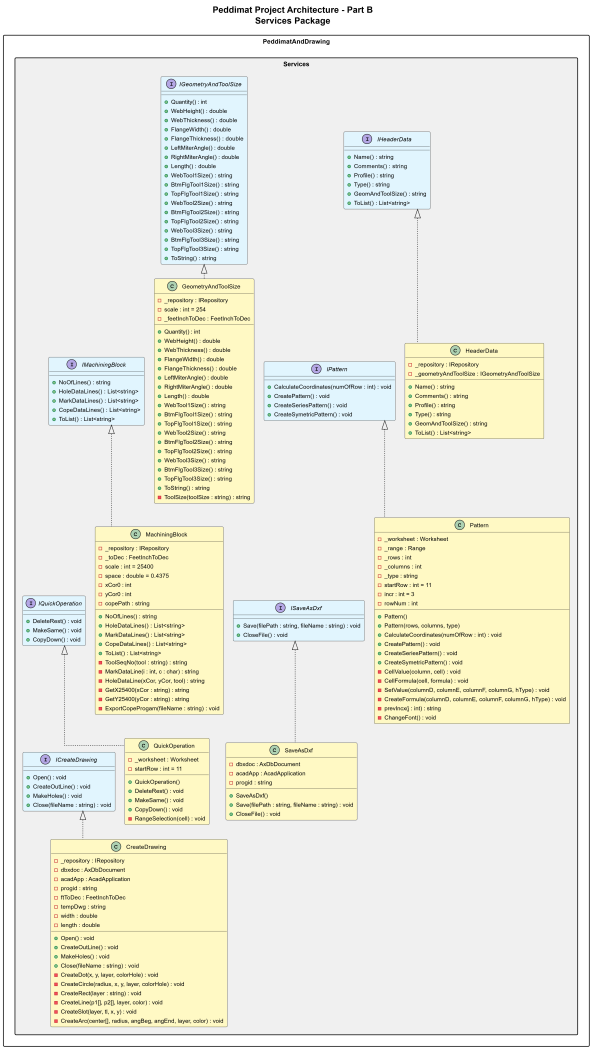

Fig. 2 Core business logic layer

The Pattern class (`PeddimatAndDrawing\Services\Pattern.cs`) implements `IPattern` and automates populating an Excel worksheet via `Microsoft.Office.Interop.Excel`: it stores pattern parameters (`_rows`, `_columns`, `_type`) and a Worksheet reference, then generates a rectangular pattern by iterating columns and rows in `CreatePattern()`, writing cell values and R1C1 formulas with `CellValue`/`CellFormula`, computing derived coordinates in `CalculateCoordinates()`, and finally applying font formatting in `ChangeFont()`. Functions `SetValue`, `CreateFormula`, and `prevIncx` centralize cell assignments and relative-reference formula creation, while `Helper.LastRow() + 1` defines the insertion start. The class relies on project utilities (e.g., `Helper`) and

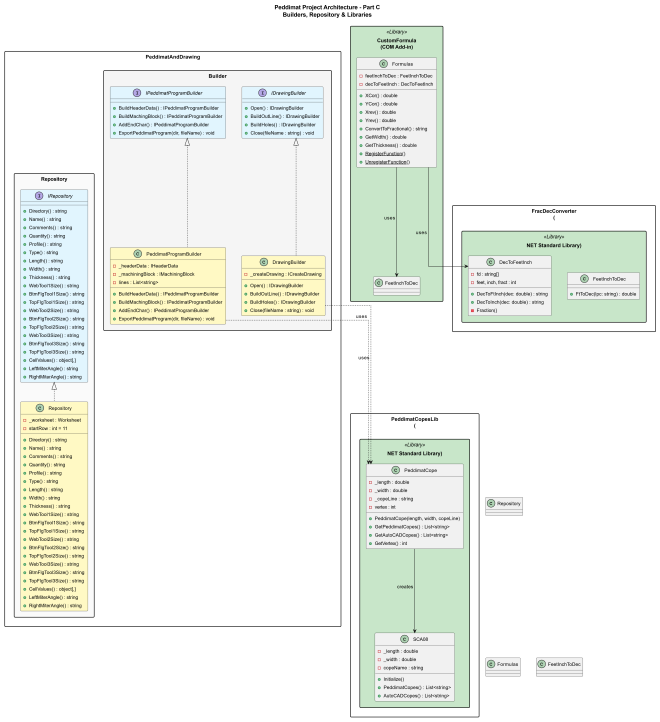

Fig. 3 Data Orchestration and Infrastructure Layer

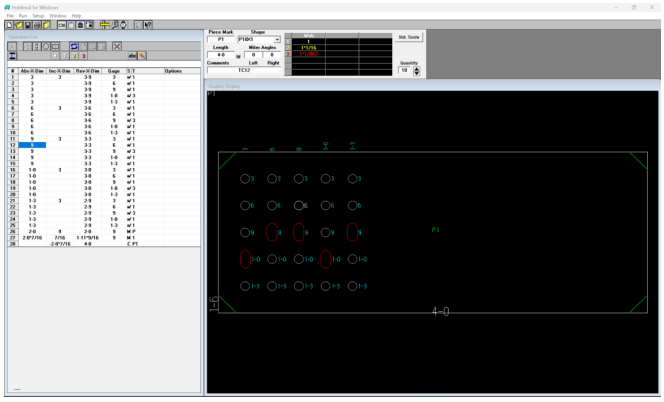

Fig. 4 Peddimat Coordinate System

custom Excel formulas (e.g., `XCor`, `YCor`) and assumes a VSTO add-in context via `Globals.ThisAddIn`. See the following sample code:

```
public void CalculateCoordinates(int numOfRow
    ) {
    CellFormula("A" + numOfRow, "=
        ConvertToFractional(XCor(WL,R[-1]C,RC
        [3],RC[4],R[-1]C[8],RC[8]))");
    CellFormula("B" + numOfRow, "=
        ConvertToFractional(YCor(WH,R[-1]C,RC
        [4],RC[5],R[-1]C[7],RC[7]))");
    CellFormula("C" + numOfRow, "=
        ConvertToFractional(Xrev(WL,RC[-2],RC
        [1],RC[2]))");
    CellFormula("H" + numOfRow, "=
        ConvertToFractional(Yrev(WH,RC[-6],RC
        [-2],RC[-1]))");
}
public void CreatePattern() {
    rowNum = Helper.LastRow();
    for (int m = 1; m <= _columns; m++) {
        for (int n = 1; n <= _rows; n++) {
            rowNum++;
            if (m == 1){
                if (n == 1)
                    SetValue(incr, "S", incr,
                        "S", _type);
                else if (n == 2)
                    SetValue(0, "", incr, "",
                        _type);
                else
                    CreateFormula("", "",
                        prevIncx(1), "", _type
                        );
            }
            else  {
                if (n == 1 && m == 2)
                    CreateFormula(incr.
                        ToString(), "",
                        prevIncx(_rows),
                        prevIncx(_rows),
                        prevIncx(_rows));
                else if (n == 1)
                    CreateFormula(prevIncx(
                        _rows), "", prevIncx(
                        _rows), prevIncx(_rows
                        ), prevIncx(_rows));
                else
                    CreateFormula(prevIncx(
                        _rows), "", prevIncx(
                        _rows), "", _type);
            }
            CalculateCoordinates(rowNum);
        }
```

Fig. 5 AutoCAD Coordinate System

```
    }
    ChangeFont();
}
private void CellValue(string column, string
    cell) {
    _range = _worksheet.get_Range(cell, Type.
        Missing);
    _range.Value2 = column;
    _range.HorizontalAlignment = -4108;
}
private void CellFormula(string cell, string
    formula) {
    _range = _worksheet.get_Range(cell, Type.
        Missing);
    _range.Activate();
    _range.FormulaR1C1 = formula;
    _range.HorizontalAlignment = -4108;
}
private void SetValue(double columnD, string
    columnE, double columnF, string columnG,
    string hType) {
    CellValue(columnD.ToString(), "D" +
        rowNum);
    CellValue(columnE, "E" + rowNum);
    _range = _worksheet.get_Range("F" +
        rowNum, Type.Missing);
    if (columnF == 0)
        _range.Value2 = "";
    else
        _range.Value2 = columnF.ToString();
    _range.HorizontalAlignment = -4108;
    CellValue(columnG, "G" + rowNum);
    CellValue(hType, "I" + rowNum);
}
private void CreateFormula(string columnD,
    string columnE, string columnF, string
    columnG, string hType) {
    if (string.IsNullOrEmpty(columnD))
        columnD = "0";
    CellFormula("D" + rowNum, columnD);
    CellFormula("E" + rowNum, columnE);
    CellFormula("F" + rowNum, columnF);
    CellFormula("G" + rowNum, columnG);
    CellValue(hType, "I" + rowNum);
}
private string prevIncx(int j) {
    return "=R[-" + j + "]C";
}
```

Plate Clips

Corner plate clips are widely used in the construction and bridge building industries. In the spreadsheet template, four standard copes are provided for generating both Peddimat programs and AutoCAD Dwg files. To eliminate the need for manually creating clips in AutoCAD or drawing them line by line in Peddimat, a C# class library PeddimatCopesLib has been developed to automate these tasks. Users simply enter the required values in the form, submit the data to the spreadsheet, and then quickly generate the corresponding Peddimat program and AutoCAD Dwg file. This approach provides an efficient way to produce copes for Peddimat and AutoCAD. Building and maintaining a cope library significantly improves the workflow for creating Peddimat programs and AutoCAD Dwg files, especially those used in the nesting system of the Peddinghaus plate processors. See Figure 6.

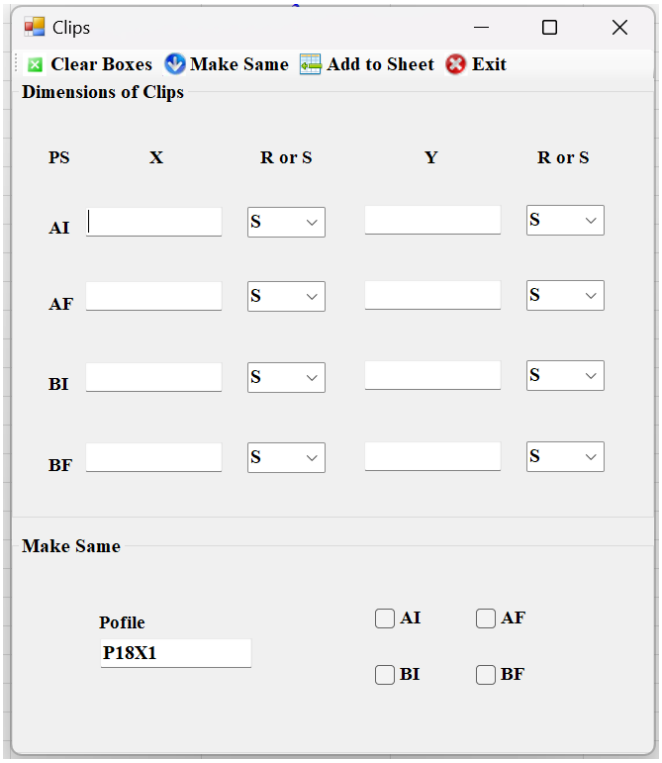

Fig. 6 Plate Clips

In the form, AI and AF represent the top and bottom clip dimensions on the left hand side, while BI and BF represent the top and bottom clip dimensions on the right hand side. X denotes the horizontal dimension and Y denotes the vertical dimension of the plate clips. Clip dimensions can also be entered in their reversed orientation by selecting "R" from the drop down menu. See Figure 7 for the plate clips for the peddimat program.

Where W represents the start point, and the line following W contains dimensions related to the previous line. All measurements are in millimeters $\times$ 10.

User Interface and Workflow

Ribbon Menu Items on Microsoft Excel Spreadsheet Template

W 0 508 0
508 -508 0
W 12192 508 0
-508 -508 0
W 0 4064 0
508 508 0
W 12192 4064 0
-508 508 0

Fig. 7 Cope File for Plate Clips for Peddimat Program

A Microsoft Excel spreadsheet with the Ribbon menu visible at the top, showing the tab Add-ins. Within the Add-ins tab related to CAD or fabrication workflows, including options like Create, Make, Add Clips, Operation, Peddimat, Dxf Save as, and Dxf Open. See Figure 8.

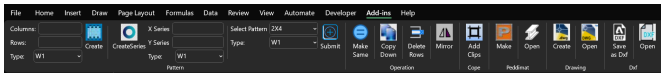

Fig. 8 Ribbon Menu Items on Microsoft Excel Spreadsheet Template

Microsoft Excel Spreadsheet Template

Below the Ribbon, the Excel sheet contains fabrication part data. Headers include Directory, Name, Quantity, Shape or Profile, Length, Width, Thickness, and miter angles, followed by rows listing coordinate-based drilling or burning instructions. Many rows contain values under headings such as X, Y, RevX, IncX, StartX, StartY, RevY, and Tool, with custom formulas attached to cells under those heading. Some cells display fractional dimension formats, and the formula bar shows a conversion formula used to transform coordinate values into fractional measurements. Near the bottom, several "[COPE]" entries appear along with cope names like MAC:SCAI08 and MAC:SCAF08.

Input all hole-related data using predefined patterns on ribbon text boxes and combo boxes. The formulas for X, Y, RevX, RevY and Tool are automatically generated in their respective cells. See Figure 9.

Program Creation

The application generated both a new Pedimat file and an AutoCAD Dwg file. In the Ribbon menu, enter 8 in the Column text box, 5 in the Row text box, and select 'W1' from the Type dropdown. Click the "Create" button to generate 40 rows of hole data in the Excel spreadsheet. Adjust the IncX and IncY

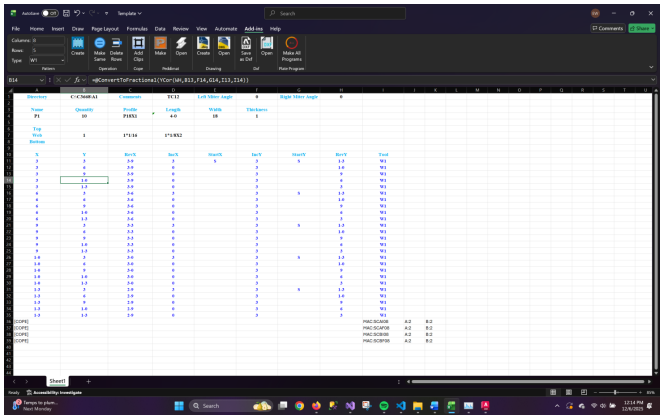

Fig. 9 Excel Spreadsheet Template

values as needed. Then, select the desired range and click the "Make Same" button to apply a repeating pattern for IncX (see Figure 10). Continue modifying the IncX and IncY values until the X and Y hole coordinates match those shown in the shop drawings.

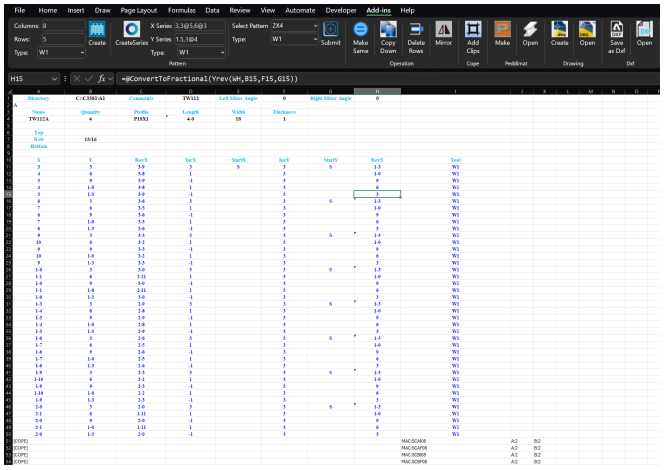

Fig. 10 Hole Information and Set up IncX and IncY Values

Click the "Make" button to generate the Peddimat program. To view the program, click the "Open" button. Figure 11 displays the Peddimat program that was produced.

Click "Create" button to create Dwg file and click "Open" button. The resulting Dwg file is shown in Figure 12.

Dimensions of peddimat file and AutoCAD Dwg file are perfect match by comparing their dimensions from Figures 11 and 12.

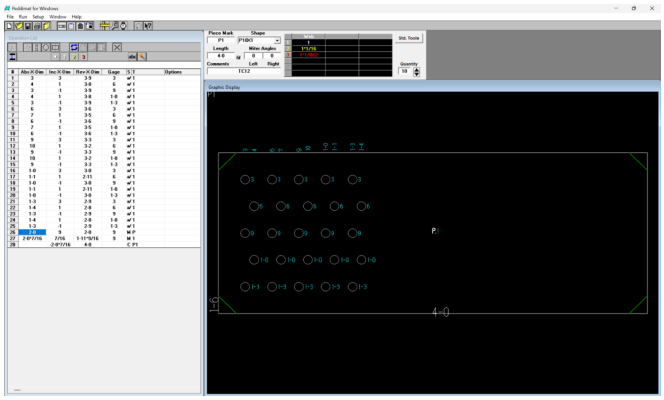

Fig. 11 View Peddimat Program

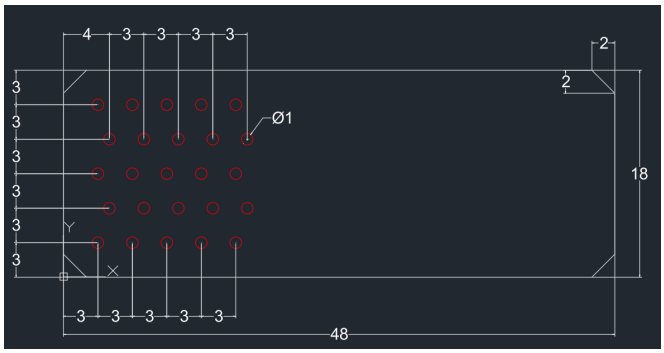

Fig. 12 View AutoCAD Dwg

Discussion

Comparison with Peddimat Software

Two quantitative metrics were used:

- Program Creation Time (min) – Time from data input start to create Peddimat programs for each shop drawing.
- Time Reduced (%) – Time reduced percentage per shop drawing.

A senior programmer from a local steel fabrication company created Peddimat programs on the same Windows 11 computer using the following drawings, as shown in Figures 13 to 16.

Test results show in Table 4.

Efficiency Analysis

Cell references

The developed app uses custom cell formulas that reference one another, so any change to a single value automatically updates all related cells. As a result, the X, Y, and reverse X/Y coordinates are recalculated simultaneously (see Figure 17 for explanation).

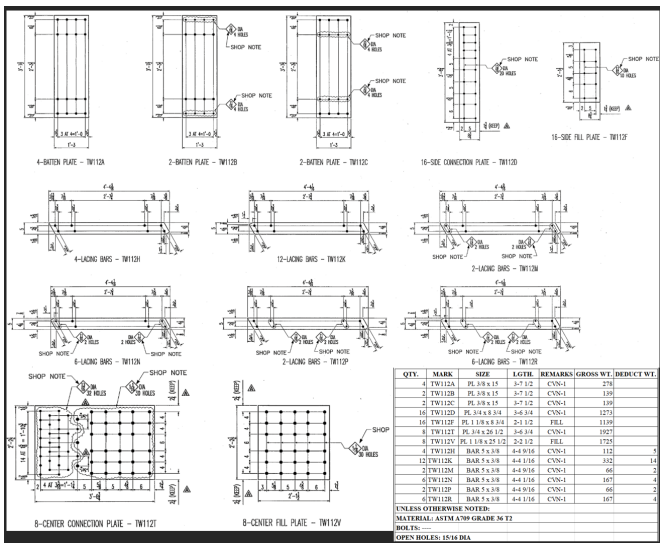

Fig. 13 Dwg No. TW112

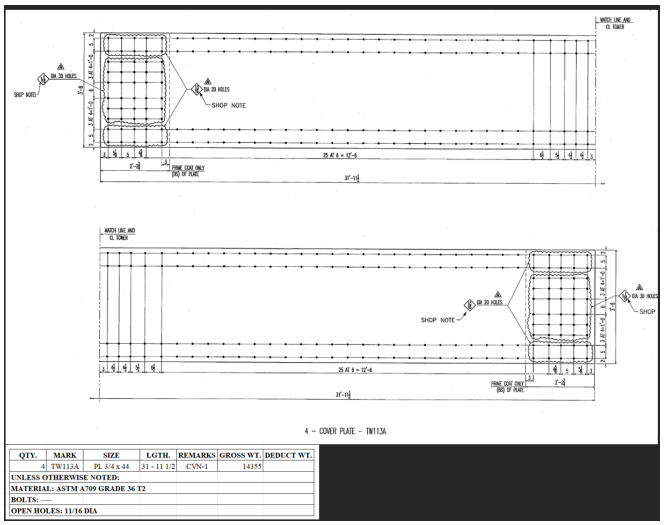

Fig. 14 Dwg No. TW113

Figure 17 illustrates that the colored lines demonstrate how adjusting the value in the first cell causes the values in the following cells to update automatically. The Tool column references are like those in the IncY column.

Series Pattern

Concatenate multiple hole data lines into a single line using a series pattern, eliminating repetitive select, copy, and paste operations. See Figure 18.

Analysis

Table 4 Comparison between Peddimat Software and the Developed App

Drawings	Time with Peddimat Software (min)	Time with the Developed App (min)	Time Reduced (%)
TW112	19	17	10.52%
TW113	3.4	2.7	20.58%
TW121	7.7	6.0	22.08%
TW128	4.2	3.3	21.43%

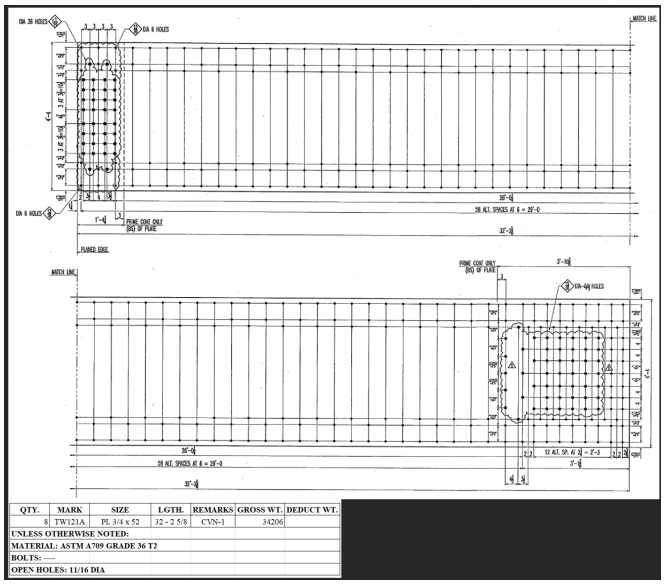

Fig. 15 Dwg No. TW121

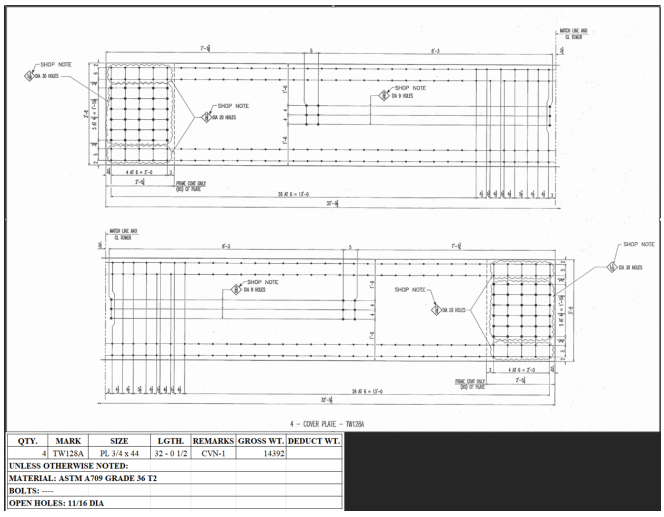

Fig. 16 Dwg No. TW128

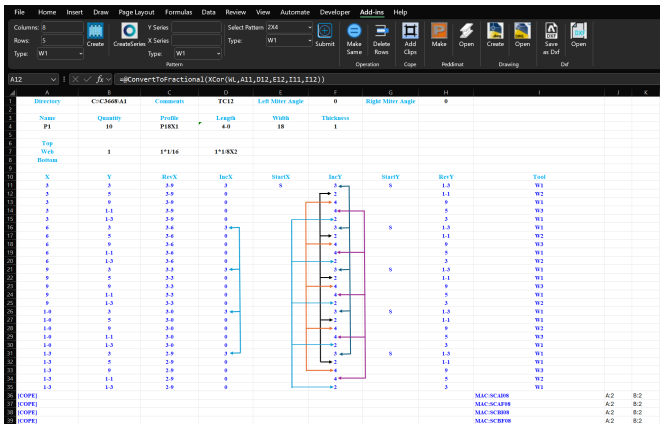

Fig. 17 Explanation of Spreadsheet Cell References

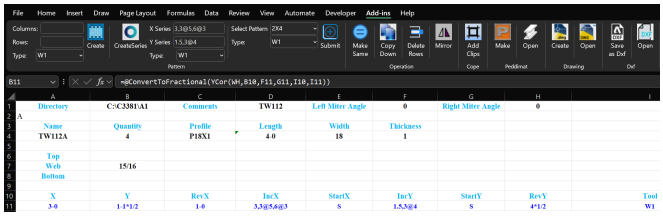

Fig. 18 Series Pattern with One Line Hole Data

In drawing TW112, the same hole patterns (Matrix and Mirror) can be applied to piece marks TW112A through TW112C using both the Peddimat legacy software and the developed app, resulting in no time difference between the two systems. However, for TW112D and TW112F, series patterns (IncX: 1.75, 6, 4.625, 5, 5.125, 5, 4@3.3125; IncY: 2, 5) can be applied to accommodate varying horizontal spacings without line by line adjustments, making the developed app faster than the Peddimat legacy software. The programmer can also select patterns from a dropdown menu and only needs to update nine values for TW112D in the spreadsheet with reference advantages. In contrast, when modifying Abs X Dim or Inc X Dim in the Peddimat legacy software, the cursor advances hole by hole to Gage, which slows the process. The developed app avoids this step, maintaining faster performance. For piece

marks TW112H through TW112R, programming utilizes series patterns and the "copy down" operation. Since the developed app eliminates the need for hole by hole adjustments, it continues to outperform the Peddimat legacy software. Additionally, when applying the Matrix pattern in Peddimat legacy software for TW112D and TW112F, both vertical and horizontal spacings must be changed manually, whereas the developed app handles these adjustments more efficiently, further increasing its speed advantage.

In drawing TW113, the varying vertical and horizontal spacings require programmers using Peddimat legacy software to perform additional manual steps—such as modifying, selecting, copying, and pasting—after applying matrix patterns. By leveraging cell reference advantages, the developed app remains faster.

In drawings TW121 and TW128, programmers can use the series patterns to define a single hole data line instead of multiple lines, unlike Peddimat legacy software. Overall, this approach significantly reduces manual operations, including frequent cursor movement.

Additionally, the spreadsheet template includes a "RevY" column, allowing programmers to verify reverse Y dimensions against shop drawings without performing lengthy manual calculations. Programmers can also configure reverse X and reverse Y coordinates for programming purposes, further reducing manual computation. Within the predefined patterns, Microsoft Excel can perform the necessary calculations and provides correct spacing increments in the cells with proper calculation, making this approach more convenient than using a calculator and the Peddimat legacy software.

Conclusions

The Peddimat program structure was successfully interpreted for the first time using Notepad++, the legacy Peddimat software (V1.08.07), and the AISC shapes database. An Excel VSTO Add-In project was developed for the first time using Visual Studio 2022, Microsoft Excel 365, and AutoCAD 2026 on Windows 11 to create both Peddimat programs and AutoCAD Dwg files.

A fluent builder interface employing a recursive design pattern was applied during development to simplify object construction, enforce consistency, and improve code readability. This approach allows complex objects to be composed step-by-step through chained method calls, while the recursive structure ensures type-safe transitions between builder stages.

Dependency Injection (DI) techniques were also implemented in the VSTO Add-In project. By injecting dependencies rather than instantiating them directly, the add-in achieves loose coupling, enhanced maintainability, and improved testability. Additionally, PlantUML combined with Visual Studio Code was used to generate design architecture diagrams.

Several .NET Framework class libraries were developed to support modular functionality:

- **CustomFormula**: Implements custom formulas (UDFs) that users can apply directly within Excel spreadsheets.
- **FracDecConverter**: Provides shared unit conversion functionalities for all projects, reducing redundant code.
- **PeddimatCopesLib**: Enables efficient addition of clips to Peddimat programs and facilitates the creation of AutoCAD Dwg files.

An Excel spreadsheet template was developed to enable users to input hole data and specify hole types using predefined or dynamic patterns. The template offers a user-friendly interface, and the resulting data can be used to generate both Peddimat programs and AutoCAD DWG files.

A comparison of efficiency between the legacy Peddimat software and the developed app demonstrates that creating Peddimat programs with the developed app is more faster. Around 20% reduction based on test cases in program creation time Furthermore, verifying Peddimat programs with the developed app is considerably easier compared to using the legacy software, and AutoCAD Dwg files can be quickly converted to Dxf files for other Peddinghaus plate processors. Peddimat files are specific to the Peddinguase machine. Future work should focus on how to create DSTV files for most of fabricator machines.

References

1 M. Srinivas, G. Ramakrishna, K. R. Rao and E. S. Babu, *Analysis of legacy system in software application development: a comparative survey*, 2016, http://doi.org/10.11591/ijece.v6i1.pp292-297.

2 S. Zhang and J. Bai, *Research on CNC programming and machining process based on CAD/CAM technology*, 2024, https://doi.org/10.2478/amns-2024-0516.

3 N. Thakur and E. A. Gupta, *Automation of design process using Etabs, Microsoft Excel and AutoCAD*, 2017.

4 W. Xiao, L. Zheng, J. Huan and P. Lei, *A complete CAD/CAM/CNC solution for STEP-compliant manufacturing*, 2015, https://doi.org/10.1016/j.rcim.2014.06.003.

5 X. Xu and Q. He, *Striving for a total integration of CAD, CAPP, CAM and CNC*, 2004, https://doi.org/10.1016/j.rcim.2003.08.003.

6 S. Omirou, M. Charalambides, A. Lontos, M. Menicou and G. Demosthenous, *A dedicated CAD/CAM module for high precision CNC 528 milling of complex threads with fixed or variable pitch and radius*, 2025, https://doi.org/10.1007/s00170-024-13970-5.

7 M. Simonič, I. Palčič and S. Klančnik, *Advancing intelligent toolpath generation: a systematic review of CAD–CAM integration in industry 4.0 and 5.0*, 2025, https://doi.org/10.5545/sv-jme.2025.1370.

8 L. Kirner, V. Jung, J. Oraskari and S. Brell-Cokcan, *Enhancing robotic steel prefabrication with semantic digital twins driven by established industry standards*, 2024, https://doi.org/10.1016/j.autcon.2024.105699.

9 X. Zhao and Y. Chung, *Development of a robotic structural steel cutting system*, 2019, https://doi.org/10.1088/1757-899X/538/1/012042.

10 T. B. Elserwy, T. Aly and B. E. El-Demerdash, *End-user software engineering approach: improve spreadsheets capabilities using Python-based user-defined functions*, 2025, https://doi.org/10.11591/ijeecs.v38.i2.pp1024-1032.

11 E. Carter and E. Lippert, *Visual Studio tools for Office 2007: VSTO for Excel, Word, and Outlook*, 2007.

12 T. Chen, Y. Ren, G. Xie, F. Liu and K. Luo, *Comparative study on secondary development methods of AutoCAD: performance comparison and application of C#, VB, and ObjectARX*, 2025, https://doi.org/10.54097/adpthn55.

13 X. Chen, *Research on the application of CAD technology in mechanical drawing*, 2023.

14 M. Seidl, M. Scholz, C. Huemer and G. Kappel, *UML @ classroom: an introduction to object-oriented modeling*, 2015, https://doi.org/10.1007/978-3-319-12742-2.

15 PlantUML, *PlantUML language reference guide*, 2025.

16 M. Chadwick, *Design patterns in .NET 6: reusable approaches in C# and F# for object-oriented software design*, 2022, https://doi.org/10.1007/978-1-4842-8245-8.

17 S. V. Deursen and M. Seemann, *Dependency injection principles, practices, and patterns*, 2019.

18 P. Michaels, *Software architecture by example: using C# and .NET*, 2022, https://doi.org/10.1007/978-1-4842-7990-8.

19 A. Troelsen and P. Japikse, *Pro C# 10 with .NET 6: foundational principles and practices in programming*, 2022, https://doi.org/10.1007/978-1-4842-7869-7.

20 B. Kramer, *ObjectARX primer (Autodesk's programmer series)*, 2000.

21 J. A. Leach and S. Lockhart, *AutoCAD 2026 instructor: a student guide for in-depth coverage of AutoCAD's commands and features*, 2025.