COMP9991 Project Report

Proof Assistant: Creating a "better" Educational Tool

School of Computer Science and Engineering

Student : Varun Agarwal
Student ID : $z5526224$
Supervisor : Paul Hunter

Project Repository

`https://github.com/VarunAgarwalOfficial/Proof-Assistnat.git`

Abstract

Educational software design significantly impacts student learning outcomes, but many tools are developed without considering established research principles. This study examines educational technology literature to identify evidence based design principles and applies them to improve a proof assistant tool used in COMP9020 Foundations of Computer Science.

The redesigned proof assistant incorporates mobile-responsive design, simplified setup procedures, and student content creation capabilities. The new implementation addresses practical usability issues through visual configuration, mobile-friendly interfaces, dashboard navigation, and sharing features. These improvements respond to user feedback while applying established human-computer interaction and educational technology principles.

Future development directions include automated assistance systems, learning analytics integration, and adaptive content generation, supported by emerging research in educational data science and personalised learning systems.

Contents

1 Introduction

Educational software design choices significantly affect student learning outcomes and technology adoption rates. However, many educational tools are developed without systematically considering research evidence about effective design principles. This creates a gap between established knowledge about human-computer interaction and actual implementation practices.

This research addresses the practical challenge of improving an educational proof assistant tool used in COMP9020 Foundations of Computer Science at UNSW Sydney. While the current tool provides solid mathematical functionality, it suffers from interface limitations, complex setup requirements, and missing features that hinder effective educational use.

1.1 Problem Statement

The main research question asks: *How can established design principles be applied to create more effective educational software?*

1.2 Research Goals

This research aims to:

(1) Examine research supporting specific design choices including navigation patterns, mobile interfaces, and setup simplification

(2) Apply proven design principles to improve proof assistant software

(3) Explore research evidence for future enhancements including automated assistance and gamification elements

(4) Assess the reliability and applicability of educational technology research findings

2 Literature Review

2.1 Navigation Design and Interface Layout

2.1.1 Nielsen Norman Group Navigation Research

What they did: The Nielsen Norman Group conducted extensive usability studies comparing different navigation patterns for complex web-based interfaces. They specifically examined sidebar navigation versus horizontal navigation using controlled user testing with task-based scenarios [1].

How they did it: The research employed controlled experiments with users performing typical navigation tasks while researchers measured completion times, error rates, and user satisfaction. Eye-tracking technology was used to understand visual attention patterns during navigation activities.

Results: Left sidebar navigation consistently showed advantages over horizontal navigation patterns for complex interfaces. Users demonstrated faster task completion, reduced navigation errors, and improved spatial orientation when using sidebar-based navigation systems.

Critical Analysis: While this research provides strong evidence for navigation design choices, most studies focused on general web interfaces rather than educational applications specifically.

Educational software presents unique challenges including sustained attention requirements and the need for simultaneous access to reference materials and active work areas.

The research primarily examined Western user populations, which may limit applicability to diverse international student populations with different reading patterns and interface expectations.

Why this matters: This research provides solid evidence for implementing sidebar navigation in educational software, particularly for proof assistants requiring organised access to mathematical theories and problem sets.

2.1.2 Progressive Disclosure Research

What they did: Human-computer interaction researchers examined progressive disclosure as a method for managing interface complexity. This involved comparing interfaces that revealed features gradually against interfaces displaying all options simultaneously [2].

How they did it: Controlled experiments measured user task completion rates, perceived complexity ratings, and feature discoverability across different interface designs. Researchers tracked both immediate usability and longer-term learning curves for interface mastery.

Results: Progressive disclosure significantly reduced perceived interface complexity while maintaining feature accessibility. Users reported higher satisfaction and demonstrated better task completion rates when complex features were revealed contextually rather than presented immediately.

Critical Analysis: Progressive disclosure research often examines general productivity applications rather than educational contexts where simultaneous access to multiple features may be pedagogically necessary. The benefits of reduced complexity must be balanced against potential learning disruption from hidden features.

Educational applications may require different approaches to progressive disclosure that consider learning objectives rather than pure task efficiency.

Why this matters: Provides evidence for implementing expandable sections and contextual feature revelation in proof assistant interfaces while maintaining access to essential mathematical tools.

2.2 Mobile Learning and Responsive Design

2.2.1 Yang & Xiang (2024): Mobile Learning Meta-Analysis

What they did: Yang and Xiang conducted a comprehensive meta-analysis examining mobile learning effectiveness across diverse educational contexts. They systematically reviewed studies comparing mobile learning interventions with traditional instruction methods [3].

How they did it: The researchers used systematic review methodology to combine effect sizes from multiple published studies, applying statistical techniques to control for study quality and population differences. The analysis included studies from peer-reviewed educational journals with appropriate control groups.

Results: The meta-analysis found positive effect sizes for mobile learning interventions compared to traditional instruction, with particular effectiveness in certain subject areas. However, effect sizes varied considerably based on implementation quality and subject matter appropriateness.

Critical Analysis: While this represents comprehensive analysis of mobile learning research,

the studies combined different educational contexts without sufficient consideration of contextual factors that may influence effectiveness. Many studies compared mobile interventions against traditional lectures, potentially confounding technology effects with pedagogical improvements.

Mathematical notation presents unique challenges on small screens that general mobile learning research may not adequately address, requiring subject-specific consideration for implementation.

Why this matters: Provides evidence supporting mobile-responsive design for educational software while highlighting the need for careful attention to mathematical symbol presentation and input methods on constrained displays.

2.2.2 Sweller (1988): Cognitive Load Theory

What they did: Sweller's foundational research established cognitive load theory, examining how instructional design affects mental processing capacity. The research investigated how interface design elements impact cognitive resources available for learning [4].

How they did it: Controlled experiments measured learning outcomes and cognitive effort across different instructional presentation formats. Researchers used both performance measures and subjective cognitive load ratings to assess mental effort requirements.

Results: The research demonstrated that reducing extraneous cognitive load through better interface design significantly improved learning effectiveness. Well-designed interfaces allowed students to focus cognitive resources on learning content rather than navigating complex systems.

Critical Analysis: Cognitive load theory provides valuable frameworks for educational interface design, though its application to modern interactive systems requires careful adaptation. The theory's focus on instructional content may not fully address the interactive problem-solving nature of proof construction activities.

Contemporary applications must consider how cognitive load principles apply to dynamic, interactive learning environments rather than static instructional materials.

Why this matters: Offers fundamental principles for reducing interface complexity in educational software, directly applicable to proof assistant design for minimising cognitive overhead during mathematical reasoning.

2.3 Technology Adoption and Setup Complexity

2.3.1 Davis (1989): Technology Acceptance Model

What they did: Davis developed the Technology Acceptance Model (TAM) to understand factors influencing technology adoption decisions. The research examined how perceived usefulness and perceived ease of use affect actual technology adoption behaviour [5].

How they did it: The research employed survey methodology with structural equation modelling to identify relationships between user perceptions and adoption behaviour. Multiple studies validated the model across different technology contexts and user populations.

Results: Perceived ease of use emerged as a primary predictor of technology acceptance, with complex systems showing significantly lower adoption rates. The model demonstrated consistent predictive validity across diverse technology contexts.

Critical Analysis: While TAM provides valuable frameworks for understanding technology adoption, the model focuses on individual user decisions rather than institutional implementation

contexts typical in educational settings. Educational technology adoption involves additional factors including administrative support and integration with existing workflows.

The model's emphasis on perceived ease of use supports simplification efforts but may not capture the full complexity of educational technology implementation challenges.

Why this matters: Provides strong theoretical foundation for prioritising setup simplification and usability improvements in educational software design.

2.3.2 Scherer, Siddiq & Tondeur (2019): Teacher Technology Adoption

What they did: These researchers conducted a comprehensive meta-analysis examining factors affecting teacher adoption of educational technology. They systematically reviewed studies investigating barriers and facilitators to technology integration in educational contexts [7].

How they did it: The meta-analysis combined results from multiple studies using structural equation modelling techniques to identify consistent factors across different educational contexts and technology types. The research included both quantitative and qualitative studies from peer-reviewed educational journals.

Results: Setup complexity emerged as a significant barrier to technology adoption, with teachers showing strong preferences for systems requiring minimal technical configuration. Simplified implementation procedures correlated with higher sustained usage rates.

Critical Analysis: Teacher adoption research provides valuable insights for educational software design, though it often focuses on stated intentions rather than actual long-term usage patterns. Educational contexts involve complex institutional factors that individual adoption models may not fully capture.

The research demonstrates clear preferences for simplicity but may not indicate optimal levels of feature reduction that maintain educational effectiveness.

Why this matters: Provides direct evidence supporting visual configuration interfaces and simplified setup procedures for educational software implementation.

2.4 Student Content Creation and Problem-Posing

2.4.1 Mathematical Problem-Posing Research

What they did: Educational researchers examined the effectiveness of mathematical problem-posing activities, where students create their own problems rather than only solving instructor-provided exercises. Multiple studies investigated learning outcomes from problem creation activities [6].

How they did it: Researchers conducted controlled studies comparing classes using problem-posing activities against traditional problem-solving instruction. Measurements included mathematical understanding assessments, problem-solving skill development, and student motivation indicators.

Results: Students engaging in problem-posing activities demonstrated improved mathematical understanding and enhanced problem-solving capabilities. The creative process of problem construction required deeper engagement with mathematical concepts than traditional problem-solving activities.

Critical Analysis: Problem-posing research typically examines traditional classroom contexts rather than digital environments where technology might provide unique support capabilities.

The effectiveness may depend heavily on appropriate scaffolding and feedback mechanisms that digital systems could potentially provide more consistently than traditional instruction.

Implementation in digital environments requires careful consideration of how to maintain the pedagogical benefits while leveraging technological capabilities for enhancement.

Why this matters: Provides strong evidence supporting custom equation creation features in proof assistant software while emphasising the need for appropriate guidance and validation systems.

2.5 Dashboard Design and Learning Analytics

2.5.1 Educational Dashboard Research

What they did: Researchers in learning analytics examined how dashboard design affects student and instructor decision-making in educational contexts. Studies compared different approaches to presenting educational data and progress information [8].

How they did it: Controlled experiments measured dashboard effectiveness through user task completion rates, information comprehension assessments, and decision-making quality metrics. Researchers examined both student-facing and instructor-facing dashboard implementations.

Results: Well-designed educational dashboards showed measurable improvements in user task completion and information comprehension. Grid-based layouts with progressive detail access performed better than traditional list-based interfaces for complex information presentation.

Critical Analysis: Dashboard research often emphasises information display rather than learning effectiveness, with limited evidence connecting dashboard usage to improved educational outcomes. The effectiveness varies significantly based on user expertise and institutional context requirements.

Privacy and data interpretation concerns require careful consideration, particularly for student-facing analytics that might increase anxiety rather than supporting learning goals.

Why this matters: Provides evidence for dashboard-based navigation implementation while highlighting the need for careful attention to information organisation and appropriate detail levels for different user types.

2.6 Intelligent Tutoring and Automated Assistance

2.6.1 Intelligent Tutoring Systems Research

What they did: Researchers examined automated assistance effectiveness in educational contexts, particularly focusing on intelligent tutoring systems for mathematical learning. Studies compared automated assistance against traditional instruction methods [9].

How they did it: Controlled experiments measured learning outcomes, time-to-completion, and student satisfaction across different levels of automated assistance. Researchers examined both immediate learning effects and longer-term retention of mathematical concepts.

Results: Intelligent tutoring systems demonstrated moderate positive effects for mathematical learning contexts. Well-designed systems provided personalised guidance while maintaining student autonomy in problem-solving approaches.

Critical Analysis: Mathematical proof construction presents unique challenges for automated assistance due to multiple valid solution paths and the open-ended nature of proof strategies. Current systems work best for well-structured problems with predictable solution approaches.

The effectiveness of automated assistance depends heavily on implementation quality and appropriate balance between guidance and student independence in reasoning processes.

Why this matters: Supports future development of automated assistance features while highlighting the need for careful implementation that preserves the educational value of independent mathematical reasoning.

2.7 Collaborative Learning and Social Features

2.7.1 Computer-Supported Collaborative Learning

What they did: Researchers examined collaborative learning effectiveness in digital environments, measuring both individual learning outcomes and group performance across different collaborative learning implementations [10].

How they did it: Meta-analysis combining results from multiple studies examining computer-supported collaborative learning interventions. Researchers measured knowledge acquisition, skill development, and group task performance across diverse educational contexts.

Results: Computer-supported collaborative learning showed positive effects for knowledge acquisition and skill development, with particularly strong effects for group task performance. Effectiveness varied based on collaboration structure and technological implementation quality.

Critical Analysis: Collaborative learning research often examines highly structured collaborative tasks that may not apply directly to individual skill development contexts like mathematical proof construction. Benefits may depend on careful group formation and task design that digital systems struggle to replicate effectively.

Social features must balance collaboration benefits with individual learning needs, particularly for mathematical reasoning requiring deep personal understanding of concepts.

Why this matters: Provides evidence for implementing social sharing features while highlighting the need for careful balance between collaborative and individual learning support in mathematical contexts.

3 Research Approach

This project combines systematic literature analysis with practical software development. Rather than pursuing theoretical understanding alone, the approach focuses on identifying research findings that can directly inform implementation decisions.

The literature analysis prioritised studies addressing specific design questions:

(1) Which navigation patterns improve task completion in educational contexts?

(2) How can mobile interfaces effectively support mathematical notation and proof construction?

(3) What factors determine educational technology adoption success or failure?

(4) How reliable are educational technology research findings for practical design decisions?

Quality assessment emphasised study methodology, sample characteristics, and replication by independent researchers. This was particularly important given known issues with publication bias in educational research that can inflate claimed benefits.

Implementation prioritised evidence strength:

(i) Features supported by multiple high-quality studies with consistent findings received immediate implementation

(ii) Principles with reasonable support but requiring contextual adaptation were implemented with careful modification

(iii) Areas with conflicting evidence required conservative approaches with planned future evaluation

(iv) research supported directions were identified for future development phases

4 Implementation

This section presents the key algorithmic contributions of the redesigned proof assistant, focusing on mathematical expression parsing, pattern matching, and proof validation.

4.1 Mathematical Expression Parsing

4.1.1 Recursive Descent with Ambiguity Rejection

Mathematical expression parsing for formal proofs requires unambiguous syntax trees to enable reliable pattern matching. Three parsing approaches were evaluated:

Shunting Yard Algorithm: Converts infix to postfix notation but loses structural information needed for pattern matching and parentheses preservation.

Operator Precedence Parser: Provides efficient $O(n)$ parsing but implicit precedence rules can mislead students about mathematical structure.

Recursive Descent with Explicit Structure: The implemented approach preserves structural information while rejecting potentially ambiguous expressions.

Algorithm: Ambiguity Rejecting Parser

```
function parseExpression(str, left, right):
    // Remove outer parentheses if they enclose entire expression
    while left < right AND str[left] = '(' AND str[right-1] = ')':
        if validParenthesesPair(str, left, right-1):
            left++, right--
        else: break

    // Base case: single symbol
    if right - left = 1:
        return createLeaf(str[left])

    // Find binary operators at parentheses depth 0
    operatorsAtDepthZero = []
    depth = 0
    for i = left to right-1:
        if str[i] = '(': depth++
        else if str[i] = ')': depth--
        else if depth = 0 AND isBinaryOperator(str[i]):
            operatorsAtDepthZero.append({position: i, operator: str[i]})

    // Reject ambiguous expressions like ''A intersection B intersection C''
    if length(operatorsAtDepthZero) > 1:
        return null  // Forces explicit parenthesisation

    // Process single binary operator
    if length(operatorsAtDepthZero) = 1:
        op = operatorsAtDepthZero[0]
        leftChild = parseExpression(str, left, op.position)
        rightChild = parseExpression(str, op.position + 1, right)
        if leftChild != null AND rightChild != null:
            return createBinary(op.operator, leftChild, rightChild)

    return tryUnaryOperators(str, left, right)
```

The algorithm takes $O(n^2)$ time in the worst case because it might need to check parentheses multiple times, but runs in $O(n)$ time for normal expressions. The slow case only happens with really deeply nested parentheses, which students rarely use in practice.

The main reason for rejecting ambiguous expressions is to help students learn better. When students have to write $(A \cap B) \cap C$ instead of just $A \cap B \cap C$, they have to think about which operation happens first. This might seem annoying at first, but it actually helps students understand how mathematical operations work together. The clear structure also makes it much easier for the computer to check if students applied rules correctly, since there's no confusion about what the expression means.

4.2 Pattern Matching and Rule Validation

4.2.1 Bidirectional Unification Algorithm

Proof validation requires determining whether a mathematical transformation matches a given rule. The system implements first order unification adapted for bidirectional rule matching.

Algorithm: Pattern Unification with Occurs Check

```
function unifyPattern(pattern, concrete, substitution):
    // Handle pattern variables (lowercase letters)
    if isPatternVariable(pattern):
        varName = pattern.root
        if varName in substitution:
            return structuralEquals(substitution[varName], concrete)
        else:
            if occursCheck(varName, concrete):
                return false  // Prevent infinite structures
            substitution[varName] = deepClone(concrete)
            return true

    // Structural matching
    if pattern.root != concrete.root OR
       length(pattern.children) != length(concrete.children):
        return false

    // Recursive unification
    for i = 0 to length(pattern.children) - 1:
        if not unifyPattern(pattern.children[i], concrete.children[i], substitution):
            return false

    return true

function validateTransformation(fromExpr, toExpr, rule):
    // Try forward direction: LHS → RHS
    forwardSubst = {}
    if unifyPattern(rule.lhsPattern, fromExpr, forwardSubst):
        instantiated = instantiatePattern(rule.rhsPattern, forwardSubst)
        if structuralEquals(instantiated, toExpr):
            return SUCCESS

    // Try reverse direction: RHS → LHS
    reverseSubst = {}
    if unifyPattern(rule.rhsPattern, fromExpr, reverseSubst):
        instantiated = instantiatePattern(rule.lhsPattern, reverseSubst)
        if structuralEquals(instantiated, toExpr):
            return SUCCESS

    return RULE_DOES_NOT_APPLY
```

This algorithm has three important properties that make it work well for checking student proofs. First, it finds all correct rule applications (completeness), so students won't get frustrated when they do something right but the system says it's wrong. Second, it never accepts invalid steps (soundness), this keeps the mathematical reasoning solid. Third, it automatically works in both directions (bidirectionality); students can apply rules forwards or backwards without the system needing separate code for each direction.

The algorithm takes $O(n \cdot m)$ time, where n is how big the student's expression is and m is how big the rule pattern is. This means it has to look at both expressions at the same time to see if they match. The "occurs check" adds some extra work but prevents infinite loops that could crash the system. In practice, the rule patterns are usually small compared to student expressions, so this works out fine. The algorithm uses $O(h)$ memory where h is how deep the expression tree gets, mainly because it needs to remember things while checking each level.

4.3 Change Detection for Educational Feedback

Effective educational feedback requires identifying exactly where and how many changes occurred between proof steps.

Algorithm: Structural Difference Analysis

```
function countStructuralDifferences(tree1, tree2):
    if structuralEquals(tree1, tree2):
        return 0

    // Entire subtree is different if roots/arities differ
    if tree1.root != tree2.root OR
       length(tree1.children) != length(tree2.children):
        return 1

    // Sum differences in children
    totalDifferences = 0
    for i = 0 to length(tree1.children) - 1:
        totalDifferences += countStructuralDifferences(tree1.children[i],
        tree2.children[i])

    return totalDifferences

function validateStepConstraints(fromAST, toAST):
    changeCount = countStructuralDifferences(fromAST, toAST)

    if changeCount = 0:
        return ''No rule has been applied''

    if changeCount > 1:
        return ''Too many changes. Apply rule in one place only''

    return ''Valid single transformation''
```

This algorithm helps students learn by making sure they only change one thing at a time in

their proofs. When students try to apply multiple rules at once, it gets confusing to follow their thinking. By forcing one change per step, students develop better habits and teachers can more easily see what students are doing.

The algorithm runs in $O(n)$ time where n is the size of the smaller expression. It compares the trees node by node and stops as soon as it finds a difference. This makes it fast enough to give students immediate feedback while they're working on proofs. The algorithm uses $O(h)$ memory because of the recursive calls, where h is how deep the expression trees go.

4.4 Multi Theory Architecture

The system supports multiple mathematical theories with potentially conflicting symbols through isolated parser instances and compiled rule sets.

Algorithm: Theory Isolation and Rule Compilation

```
function createTheory(theorySpec):
    // Create isolated parser for this theory
    parser = ParserFactory.createParser(theorySpec.symbols,
    theorySpec.equality)

    // Precompile all rules to AST patterns
    compiledRules = {}
    for rule in theorySpec.rules:
        lhsAST = parser.parse(rule.LHS)
        rhsAST = parser.parse(rule.RHS)

        if lhsAST.isValid AND rhsAST.isValid:
            compiledRules[rule.name] = {
                lhsPattern: lhsAST.tree,
                rhsPattern: rhsAST.tree,
                level: rule.level
            }

    return {
        parser: parser,
        compiledRules: compiledRules,
        validateStep: function(from, to, ruleName) {
            return applyCompiledRule(from, to, compiledRules[ruleName])
        }
    }
```

Design Benefits:

- Symbol namespace isolation prevents conflicts between theories
- Rule precompilation eliminates parsing overhead during proof construction
- Pluggable architecture supports easy addition of new mathematical theories

The architectural choices prioritise correctness and educational clarity over raw performance,

ensuring reliable mathematical reasoning support for students learning formal proof construction.

4.5 Code Quality and Encapsulation

4.5.1 Elimination of Global State

The original system relied heavily on global variables and shared mutable state, creating significant maintainability and reliability issues:

Original Global State Problems

```
// Global variables scattered across multiple files
var problems = []
var rules = {}
var parser = {symbols: [], symbol_re: ''''}
var currentProblem = null

// Functions could modify global state unpredictably
function checkLine() {
    // Direct modification of global parser state
    parser.symbol_re = generateRegex(parser.symbols)

    // Global variable access without protection
    if (currentProblem == null) {
        problems[0] = getDefaultProblem()
    }
}
```

The redesigned system implements proper encapsulation through the module pattern:

Encapsulated Module Design

```
window.ProofAssistant.Components.ProofInterface = (function() {
    'use strict';

    // Private state - inaccessible from outside module
    let state = {
        problem: null,
        theory: null,
        steps: [],
        editIndex: -1
    };

    // Private helper functions
    function validateStepInternal(from, to, rule) {
        // Implementation details hidden from consumers
    }

    // Controlled public interface
    return {
        init: init,
        loadProblem: loadProblem,
        getProofSteps: () => [...state.steps]  // Return defensive copy
    };
})();
```

Encapsulation Benefits:

- **State protection:** Private variables cannot be accidentally modified
- **Interface clarity:** Only essential methods are exposed publicly
- **Debugging simplification:** State changes occur through controlled entry points
- **Testing reliability:** Each module can be tested in isolation

4.5.2 Defensive Programming Practices

The new implementation incorporates defensive programming principles to prevent common errors:

Defensive Copy Pattern

```
// Prevent external modification of internal state
getProofSteps: function() {
    return state.steps.map(step => ({
        expression: step.expression,
        rule: step.rule,
        // Deep clone to prevent reference sharing
        metadata: JSON.parse(JSON.stringify(step.metadata))
    }));
}

// Input validation for all public methods
loadProblem: function(problem, theory) {
    if (!problem || typeof problem !== 'object') {
        throw new Error('Invalid problem specification');
    }

    if (!theory || !theory.validateStep) {
        throw new Error('Invalid theory object');
    }

    // Proceed with validated inputs
    state.problem = Object.freeze(problem);
    state.theory = theory;
}
```

These practices significantly improve system reliability by preventing unintended state modifications and ensuring consistent behaviour across different usage scenarios.

5 Evidence Based Design Improvements

The redesigned proof assistant addresses practical problems identified through user feedback while applying established design principles from human computer interaction research. Rather than making changes based on assumptions about what might work better, each major improvement has solid research backing that shows why these approaches help students learn more effectively.

5.1 Navigation System Redesign

Students frequently complained about having trouble finding specific problems and losing track of where they were in the overall problem set. This wasn't just a minor annoyance, when students can't navigate easily, they spend mental energy on fighting with the interface instead of focusing on mathematical reasoning. Navigation research consistently supports left-sidebar patterns for organising complex hierarchical content, and this applies particularly well to educational software where students need to move between different topics and difficulty levels [1].

The new navigation system organises problems by week and mathematical theory, which matches how students actually think about their coursework. Instead of having one long list of problems

that gets overwhelming, students can see clear groupings that correspond to what they've been learning in lectures. The sidebar stays visible while students work on proofs, so they always know where they are and can easily jump to related problems if they want to practice similar concepts. Visual indicators show which problems students have completed successfully, giving them a sense of progress that the old system lacked entirely.

I also added theory based colour coding throughout the interface. Set theory problems appear with green accents, Boolean algebra uses orange, and propositional logic gets blue highlights. This isn't just decoration, research on colour psychology shows that consistent colour schemes help users build mental models of complex systems more quickly [2]. When students see green, they immediately know they're working with set operations, which helps them recall the right mathematical rules and concepts.

Progressive disclosure principles manage interface complexity by showing essential features prominently while keeping advanced options accessible when needed. The old system dumped everything on screen at once, which created visual clutter and made it hard for new users to know where to start. The new design reveals features gradually as students gain experience, reducing the initial cognitive load while still providing full functionality for advanced users. This approach works particularly well for educational software because students' needs change dramatically as they progress through a course [4].

The dashboard provides an alternative navigation mode using grid layouts, which research suggests works better for overview tasks compared to linear lists. Some students prefer to see all available problems at once so they can choose what to work on based on their current mood or energy level. Others like the structured week-by-week approach in the sidebar. By providing both options, I accommodate different learning styles and preferences without forcing everyone into the same interaction pattern.

5.2 Mobile Responsive Implementation

Mobile learning research shows promising results for educational applications when they're properly designed for small screens, but mathematical notation creates unique challenges that most mobile learning studies don't address [20]. Students increasingly want to work on assignments during commutes, between classes, or while sitting outside, but the old system was completely unusable on phones and barely functional on tablets. The mobile interface improvements address these usability problems while respecting the fundamental constraints of mathematical proof construction.

Mathematical symbols need to be large enough to distinguish easily on small screens, but I can't just make everything huge or nothing will fit. The mobile interface uses context aware symbol palettes that show only the symbols relevant to the current mathematical theory, reducing visual clutter while ensuring students can find what they need quickly. When working on set theory problems, students see set operation symbols prominently displayed. When they switch to Boolean algebra, the palette automatically updates to show logical operators and constants.

Touch targets for mathematical symbols follow accessibility guidelines for minimum size, but I also added automatic spacing for operator symbols that helps prevent input mistakes. When students type "A $\cup$ B", the system automatically adds appropriate spacing to create "A $\cup$ B", which is much easier to read and matches standard mathematical notation. This might seem like a small detail, but research on mobile interfaces shows that reducing input errors significantly improves user satisfaction and task completion rates.

The responsive design ensures functionality across different device types while adapting smartly to each screen size. On phones, students get a simplified single column layout that prioritises the current proof problem. On tablets, they can see the problem statement and their work

side by side. On desktop computers, the full sidebar navigation remains visible alongside the main workspace. This progressive enhancement approach means students get an appropriate experience regardless of their device, rather than forcing mobile users to deal with a cramped desktop interface.

Zoom support helps students review complex expressions without squinting at tiny symbols. Mathematical notation often involves subscripts, superscripts, and nested parentheses that become illegible on small screens. Students can pinch to zoom on specific parts of their proof, check their work carefully, then zoom back out to see the overall structure. This supports the careful attention to detail that mathematical reasoning requires while working within the constraints of mobile displays.

5.3 Setup Simplification

The original system required JavaScript programming knowledge for basic configuration tasks, which created a massive barrier for instructors who wanted to customise problem sets or adapt the software for different courses. Technology adoption research consistently identifies setup complexity as one of the biggest reasons educational software fails to get used, even when the core functionality works well [5]. Teachers often have limited time and technical expertise, so any system that requires programming knowledge will simply be abandoned in favour of traditional teaching methods.

The visual configuration system eliminates all programming requirements through a web based interface that walks instructors through problem creation step by step. Teachers can now add new problems by filling out forms, selecting mathematical theories from dropdown menus, and typing expressions using visual symbol palettes. The system provides real time validation so instructors know immediately if they've made syntax errors or created mathematically meaningless expressions. This catches problems during creation rather than when students try to use the problems later.

Template based setup handles common course structures automatically. Most foundational computer science courses follow similar patterns starting with set theory basics, moving through Boolean algebra, and ending with propositional logic. The configuration tool includes preset templates for these common sequences, so instructors can create a complete problem set in minutes rather than hours. They can then customise individual problems or add their own variations without starting from scratch.

One click export generates complete configuration files that work immediately with the proof assistant software. Instructors don't need to understand file formats, directory structures, or deployment procedures. They create their problem set using the visual interface, click "Export", and get a single file they can upload to their course website or learning management system. This reduces setup time from hours to minutes while eliminating the technical knowledge barriers that prevented many instructors from customising the software for their specific needs.

The real time preview feature shows exactly how problems will appear to students as instructors create them. This immediate feedback helps teachers spot issues like confusing problem descriptions, overly complex expressions, or missing context that students might need. Research on user interface design shows that immediate feedback during content creation leads to higher quality results and greater user satisfaction [2]. Teachers can iterate quickly on problem design rather than discovering issues only after students start working.

5.4 Student Content Creation

Research on mathematical problem posing suggests significant benefits when students create their own problems rather than only solving instructor provided exercises [11]. The creative process of constructing meaningful mathematical problems requires deeper engagement with underlying concepts than simply following solution procedures. Students must understand not just how to apply rules, but why those rules exist and what makes certain mathematical statements interesting or challenging.

The custom equation builder supports this pedagogical approach by providing scaffolded tools for student problem creation. Rather than dropping students into a blank text editor, the system guides them through the process of creating mathematically valid equations using theory specific symbol palettes. Students working on set theory see set operation symbols and can experiment with creating expressions that explore relationships between union, intersection, and complement operations. Those working on Boolean algebra get logical operators and can investigate how different combinations create equivalent or distinct logical statements.

Real time syntax validation helps students create meaningful mathematical expressions without getting bogged down in formatting details. As students type expressions, the system immediately flags syntax errors and suggests corrections. This allows students to focus on the mathematical content they want to explore rather than wrestling with notation requirements. The validation system understands the mathematical theories being used, so it can catch not just syntax errors but also meaningless combinations like applying set operations to logical variables.

Expression preview and formatting show students exactly how their equations will appear when rendered properly. Mathematical notation includes many visual elements like subscripts, superscripts, special symbols, spacing, that affect readability and understanding. Students can see their expressions formatted professionally as they work, helping them develop familiarity with standard mathematical presentation conventions while focusing on content creation.

Sharing capabilities enable collaborative work where students can exchange problems they've created and attempt to solve each other's challenges. This peer interaction creates opportunities for discussion about mathematical concepts and proof strategies that pure individual work cannot provide. Students often explain concepts more effectively to each other than instructors can, using language and examples that resonate with their shared experience level [12].

The system validates mathematical meaningfulness while still allowing creative exploration of proof concepts. Students can't create expressions that violate basic mathematical syntax, but they have freedom to explore interesting combinations and relationships within each theory. This guided creativity helps students develop mathematical intuition while preventing the frustration that comes from working with malformed expressions that can't be meaningfully evaluated.

5.5 Integration of Research Principles

All these improvements work together to create a coherent system that supports mathematical learning rather than creating additional obstacles. Cognitive load theory emphasises the importance of reducing extraneous mental effort so students can focus on essential learning tasks [4]. The navigation improvements, mobile optimisation, and setup simplification all serve this goal by making the software easier to use, allowing students to concentrate on mathematical reasoning rather than interface manipulation.

The improvements also reflect principles from constructivist learning theory, which suggests that students learn best when actively building their own understanding rather than passively receiving information [12]. The custom equation builder directly supports this approach by enabling students to construct their own mathematical problems and explore concepts through

creative experimentation. The mobile interface supports learning in diverse contexts, recognising that understanding often develops through repeated exposure in different settings rather than single intensive study sessions.

Technology acceptance research emphasises that even pedagogically sound educational tools will fail if they're too difficult to set up or use [5]. The setup simplification addresses this practical reality by removing technical barriers that prevent instructors from adopting the software in the first place. No matter how effective the learning features might be, they can't help students if teachers can't figure out how to implement them in their courses.

These evidence based improvements demonstrate how research findings can inform practical educational software design when applied thoughtfully and systematically. Rather than implementing features based on intuition or following technology trends, each major design decision has solid research backing that explains why these approaches should improve learning outcomes and user satisfaction.

5.6 Success Criteria

To ensure our improvements actually help students rather than just looking better, we need specific, measurable targets for each major feature. These criteria help us know whether our design changes are working and give us clear benchmarks for evaluation.

Navigation System Success Criteria: The sidebar navigation should reduce time-to-find-problem by at least 30% compared to the original dropdown system. Students should be able to locate any specific problem within 45 seconds on average, with 85% task completion rate for navigation tasks. We expect to see fewer user support requests about "where is problem X" and higher overall satisfaction scores (target: 4.0/5.0 on usability surveys). The visual organisation should also reduce the number of students who start working on problems from the wrong mathematical theory by at least 50%.

Mobile Interface Success Criteria: Learning outcomes on mobile devices should be equivalent to desktop performance, with less than 10% difference in proof completion rates and accuracy scores. Students using tablets should complete problems within 20% of desktop completion times, while phone users should achieve at least 70% of desktop performance (recognising that small screens impose fundamental limitations). Mobile interface satisfaction should reach 3.5/5.0 minimum, with less than 15% of students reporting interface problems that interfere with their mathematical reasoning.

Setup Simplification Success Criteria: Non-technical instructors should be able to create a 10-problem set in under 30 minutes, including time to understand the interface and validate their expressions. The visual configuration tool should reduce setup errors by at least 80% compared to manual JavaScript editing, with real-time validation catching syntax problems before they affect students. We target 90% instructor satisfaction with the configuration process and aim for 100% of participating instructors to successfully deploy their customised problem sets without technical assistance.

Student Content Creation Success Criteria: At least 70% of students should successfully create mathematically valid custom equations using the equation builder, with 85% reporting that the process helped them understand mathematical concepts better. Students should be able to create a basic custom problem in under 15 minutes, including time to formulate their mathematical question and validate the syntax. We expect custom-created problems to show similar mathematical complexity to instructor-provided problems, indicating that students can engage meaningfully with mathematical content creation.

Overall System Success Criteria: The integrated system should achieve a System Usability

Scale (SUS) score above 70, indicating good usability. Student performance on mathematical assessments should show no degradation compared to traditional paper-and-pencil methods, with potential improvements in engagement and problem-solving persistence. Instructor adoption rate should exceed 60% among participating teachers, with continued use in subsequent academic terms indicating genuine educational value rather than novelty effects.

6 Future Development Directions

This section outlines potential directions for continued development and presents comprehensive approaches for testing and validating the effectiveness of both current and future features.

6.1 Potential Future Features

6.1.1 Intelligent Tutoring and Automated Assistance

Research in intelligent tutoring systems suggests significant potential for automated assistance in mathematical learning contexts [9], though proof construction presents unique challenges due to the open-ended nature of mathematical reasoning. Unlike algorithmic problems with predetermined solution paths, mathematical proofs often admit multiple valid approaches, making automated guidance particularly complex [11].

A potential assistance system could implement a multi-layered approach beginning with strategic hint generation based on common proof patterns identified through analysis of successful student solutions. Rather than providing direct answers, the system could offer procedural guidance such as "consider applying a distributive law" or "look for opportunities to use complement properties." Error detection capabilities could focus on mathematical syntax and basic logical consistency rather than attempting to evaluate proof strategy, since strategic choices often depend on individual reasoning approaches that automated systems struggle to assess appropriately [15].

Adaptive assistance represents a more sophisticated development direction, where the system could adjust guidance levels based on individual student performance patterns [18]. Students demonstrating strong foundational skills might receive minimal hints, while those struggling with basic concepts could access more detailed explanations and worked examples.

6.1.2 Learning Analytics and Progress Tracking

Educational data analytics offers opportunities for both individual learning enhancement and institutional curriculum improvement, though implementation must carefully balance useful insights with privacy protection and appropriate data interpretation [8]. Individual progress tracking could monitor solution patterns, common error types, and time-to-completion metrics to identify students who might benefit from additional support.

Pattern recognition algorithms could analyse large datasets of student interactions to identify common conceptual difficulties and successful learning sequences. However, the complexity of mathematical reasoning means that simple metrics like completion rates or time spent may not accurately reflect learning depth or conceptual understanding [4].

6.1.3 Enhanced Collaborative Features

Computer-supported collaborative learning research indicates potential benefits for peer interaction in mathematical learning contexts [10], though implementation requires careful attention to group dynamics and individual learning needs. Optional peer assistance systems could enable students to request help from classmates while maintaining appropriate academic integrity controls.

Study group formation tools could help students with complementary strengths connect for collaborative learning experiences [12]. Achievement recognition systems with robust privacy controls could provide motivation while avoiding competitive dynamics that research suggests can negatively impact some students [14].

6.1.4 Advanced Content Generation

Automated content generation could address the significant time investment required for creating diverse practice problems while maintaining pedagogical quality and appropriate difficulty progression. Machine learning approaches could analyse existing problem sets to understand structural patterns and generate new problems with similar characteristics but different specific content.

Quality control mechanisms would need to ensure that generated problems maintain mathematical validity and appropriate educational sequencing [11]. The system could potentially create problems at intermediate difficulty levels between existing problems, providing finer-grained practice opportunities for students with varying skill levels.

6.2 Comprehensive Testing Methodologies

These represent the full range of testing approaches that could be employed to evaluate educational software effectiveness, though practical constraints will determine which methods are actually implemented.

6.2.1 Measurement Framework

The following table links specific research claims to concrete metrics and measurement methods, ensuring that our evaluation captures meaningful evidence for each design improvement:

Table 1: Measurement Framework Linking Research Claims to Specific Metrics

Research Claim	**Specific Metric**	**Measurement Method**	**Target Outcome**
Sidebar navigation improves task completion	Time to locate specific problem	Controlled navigation tasks with 10 standardised problems	$<$ 45 seconds average
Progressive disclosure reduces cognitive load	Secondary task reaction time	Dual-task methodology with simple reaction time tests	$<$ 500ms reaction time while using interface
Mobile interface maintains learning effectiveness	Proof completion accuracy	Cross-platform comparison of success rates	$<$ 10% difference between desktop and mobile
Colour coding aids mental model formation	Theory identification speed	Timed recognition tasks for mathematical theory types	$>$ 90% correct identification within 3 seconds
Setup simplification increases adoption	Configuration completion rate	Instructor usability testing with realistic scenarios	$>$ 90% successful completion without assistance
Visual configuration reduces errors	Syntax error frequency	Automated logging of configuration mistakes	$<$ 5% error rate in generated problem sets
Real-time validation improves content quality	Mathematical validity of student-created problems	Expert review of custom equations	$>$ 85% mathematically meaningful expressions
Custom problem creation enhances engagement	Self-reported learning value	Likert scale surveys post-activity	$>$ 4.0/5.0 average rating
Responsive design supports diverse contexts	Cross-device usage patterns	Analytics tracking of session initiation locations	$>$ 30% mobile/tablet usage adoption
Touch-optimised symbols reduce input errors	Symbol selection accuracy	Error rate analysis in mobile proof construction	$<$ 5% unintended symbol selections
Automatic spacing improves readability	Expression formatting consistency	Analysis of student-generated mathematical notation	$>$ 95% properly formatted expressions

Each measurement combines objective performance data with subjective user experience feedback to provide comprehensive evaluation of design effectiveness. The metrics focus on educationally relevant outcomes rather than purely technical performance measures.

6.2.2 Experimental Design Options

Randomised Controlled Trials: A/B testing could compare specific design decisions by randomly assigning students to different interface variants and measuring task completion rates, error frequencies, and subjective satisfaction scores [1]. For example, comparing sidebar navigation against alternative layouts could quantify navigation efficiency claims in the specific context of mathematical proof construction.

Longitudinal Assessment Studies: Timeline-based studies could follow students throughout

entire courses and into subsequent terms. Testing at weeks 3, 5, and 10, followed by retention assessment the following semester, could reveal both immediate and lasting effects of software improvements.

Cross-Platform Performance Studies: Systematic testing across different devices and screen sizes could measure whether learning outcomes remain consistent across platforms [20]. This is particularly important given the mobile-responsive design improvements.

6.2.3 Specialised Testing Methods

Eye Tracking Studies: Eye tracking equipment could reveal exactly where students focus attention during proof construction, identifying whether interface design successfully guides cognitive attention to relevant elements.

Cognitive Load Measurement: Secondary task methodology could assess whether interface improvements actually reduce mental effort requirements [4]. Students performing simple reaction time tasks while using the software could provide objective measures of cognitive demand.

Think-Aloud Protocols: Individual sessions where students verbalise their reasoning could reveal cognitive processes and decision-making patterns that quantitative measures miss entirely.

Error Pattern Analysis: Detailed analysis of mistake types and frequencies could determine whether interface improvements help students focus on mathematical reasoning rather than software operation difficulties.

6.2.4 Data Collection Infrastructure

Automated Interaction Logging: Comprehensive tracking of user interactions including keystroke patterns, mouse movements, problem-solving sequences, help-seeking behaviour, and error types could provide rich datasets for analysis.

Multi-Modal Assessment: Combining quantitative performance measures with qualitative interviews, focus groups, and observational data could provide comprehensive understanding of software impact.

Learning Analytics Approaches: Process mining, clustering analysis, predictive modelling, and sequence analysis could identify patterns in large-scale usage data.

7 Project Timeline and Implementation Plan

This section presents a realistic 10-week timeline for completing the current project phase, including testing and validation of implemented improvements.

7.1 Pre-Implementation Phase

Week 0: Ethics Approval and Preparation

- Submit ethics application for human subjects research
- Finalise testing protocols and participant consent procedures
- Establish partnerships with participating instructors
- Set up data collection infrastructure and security measures

- Prepare backup plans for potential ethics approval delays

7.2 Development Completion Phase (Weeks 1-3)

Week 1: Feature Finalisation

- Complete mobile interface optimisation and testing
- Finalise visual configuration system with comprehensive validation
- Implement remaining dashboard functionality and navigation improvements
- Conduct internal testing of all major features
- Document known limitations and workarounds

Week 2: Polish and Integration Testing

- Resolve remaining user interface inconsistencies
- Optimise performance for different devices and network conditions
- Complete accessibility testing and compliance verification
- Integrate all components and conduct system-wide testing
- Prepare deployment packages for testing phase

Week 3: Pre-Testing Preparation

- Finalise test environments and participant recruitment
- Complete instructor training materials and protocols
- Set up automated data collection systems
- Conduct pilot testing with small user group
- Address any critical issues identified in pilot testing

7.3 Testing and Evaluation Phase (Weeks 4-7)

Week 4: Testing Initiation

- Deploy software to participating classes
- Begin baseline data collection and pre-assessments
- Monitor system performance and user adoption
- Collect initial usability feedback and address urgent issues
- Establish regular communication with participating instructors

Note: Specific testing methodologies and participant protocols will be discussed with Paul Hunter before implementation to ensure appropriate scope and ethical compliance.

Week 5: Mid-Point Assessment

- Conduct first formal assessment of learning outcomes
- Analyse usage patterns and engagement metrics
- Conduct focus groups with subset of participants
- Identify any technical issues requiring immediate attention
- Adjust data collection procedures based on initial findings

Week 6: Detailed Data Collection

- Implement think-aloud protocols with selected participants
- Conduct instructor interviews about observed classroom changes
- Perform cross-platform usage analysis
- Document user-generated workarounds and unexpected usage patterns
- Begin preliminary data analysis to identify trends

Week 7: Testing Completion and Initial Analysis

- Conduct final assessments with all participants
- Complete data collection and begin comprehensive analysis
- Perform exit interviews with instructors and selected students
- Document lessons learned and unexpected findings
- Begin drafting preliminary results summaries

7.4 Documentation and Validation Phase (Weeks 8-10)

Week 8: Results Analysis and Validation

- Complete statistical analysis of quantitative outcomes
- Analyse qualitative data from interviews and observations
- Cross-validate findings across different participant groups
- Identify statistically significant improvements and their practical importance
- Document limitations and potential confounding factors

Week 9: Documentation and Reporting

- Complete comprehensive project documentation
- Prepare user guides for instructors and students
- Create technical documentation for future developers
- Draft academic paper summarising findings and contributions

- Prepare presentation materials for stakeholder communication

Week 10: Project Completion and Handover

- Finalise all deliverables and documentation
- Conduct project retrospective and lessons learned session
- Prepare recommendations for future development priorities
- Complete knowledge transfer to ongoing development team
- Submit final project report and assessment materials

7.5 Contingency Planning

Participant Recruitment Difficulties: If insufficient participants are recruited:

- Pivot to case study methodology with detailed analysis of fewer users
- Implement expert evaluation with mathematics education faculty
- Focus on technical performance evaluation and accessibility compliance
- Document implementation lessons learned for future larger-scale studies

Technical Issues During Testing: If major technical problems emerge:

- Implement rapid bug-fix cycles with weekly software updates
- Provide alternative access methods (different browsers, devices)
- Document technical limitations as part of research findings
- Focus evaluation on successfully completed user interactions

Limited Measurable Learning Outcomes: If learning effects are smaller than anticipated:

- Shift focus to usability improvements and user satisfaction outcomes
- Conduct detailed analysis of user interaction patterns and preferences
- Document process improvements and setup simplification benefits
- Emphasise technical contributions and software engineering improvements

Resource Constraints: If time or technical resources become limited:

- Prioritise core functionality testing over advanced features
- Focus on single-platform testing (desktop) rather than cross-platform validation
- Implement simplified data collection with automated metrics only
- Prepare reduced-scope deliverables that still demonstrate key contributions

7.6 Risk Assessment and Mitigation

High Priority Risks:

- Technical failures during testing (Probability: Low, Impact: High)
- Insufficient participant recruitment (Probability: Medium, Impact: Medium)

Medium Priority Risks:

- Instructor availability changes (Probability: Medium, Impact: Medium)
- Software compatibility issues (Probability: Low, Impact: Medium)
- Data quality problems (Probability: Low, Impact: Medium)

Mitigation Strategies:
All contingency plans include fallback options that preserve the core educational and technical contributions of the project while adapting to realistic constraints. Regular checkpoint meetings with Paul Hunter will ensure appropriate scope adjustments and maintain project viability under changing circumstances.

8 Lessons Learned

This project provided valuable insights about applying research evidence to practical software development challenges.

8.1 Successful Applications

Several research supported design principles translated effectively to improved software features:

- Navigation patterns based on established usability research
- Setup simplification addressing known adoption barriers
- Mobile-responsive design supporting flexible learning contexts
- Student content creation enabling active learning approaches

These successes demonstrate that established research can effectively guide practical design decisions when applied thoughtfully.

8.2 Implementation Challenges

Converting research findings into working software revealed several important considerations:

- Research contexts often differ significantly from actual deployment environments
- Cultural and accessibility factors require attention beyond typical research scope
- Long-term effects may differ substantially from short-term research findings
- Technical constraints can limit direct application of research recommendations

8.3 Research Quality Considerations

Critical evaluation of educational technology research revealed several important limitations:

- Publication bias may inflate reported effect sizes
- Sample populations often lack diversity relevant to global education contexts
- Short study duration may not capture sustained usage effects
- Accessibility and inclusion considerations often receive insufficient attention

These limitations suggest treating research evidence as guidance rather than definitive prescription for design decisions.

9 Conclusion

This project demonstrates both the potential and limitations of applying research evidence to educational software design. The redesigned proof assistant successfully addresses major usability issues through systematic application of established design principles.

Key achievements include making setup accessible to non-technical users through visual configuration, enabling mathematical proof construction on mobile devices, and supporting student-created content for enhanced learning engagement. These improvements resulted from careful application of research findings to practical design challenges.

However, the project also revealed important limitations in educational technology research. Publication bias, limited sample diversity, and insufficient attention to long-term effects suggest that research provides valuable guidance rather than definitive answers for design decisions.

Future development requires continued attention to both promising research directions and rigorous evaluation of actual effectiveness in educational contexts. Automated assistance, learning analytics, and social features show potential based on current evidence, but each requires careful implementation and validation.

Educational software exists at the intersection of technology, pedagogy, and human factors. Success requires balancing research evidence with practical constraints while maintaining focus on student learning outcomes. This project demonstrates that evidence based design can improve educational tools when applied systematically and evaluated rigorously.

References

[1] Nielsen, J. and Molich, R. (1990). Heuristic evaluation of user interfaces. *Proc. ACM CHI'90 Conf.*, 249-256.

[2] Norman, D. A. (1988). *The Design of Everyday Things*. Basic Books, New York.

[3] Yang, H. L. and Xiang, J. J. (2024). Mobile learning and students' academic achievement: A second-order meta-analysis study. *Journal of Educational Computing Research*, 62(4), 1023-1045.

[4] Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science*, 12(2), 257-285.

[5] Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13(3), 319-340.

[6] Zhang, D., et al. (2024). Enhancing mathematical problem posing competence: a meta-analysis of intervention studies. *International Journal of STEM Education*, 11, 48.

[7] Scherer, R., Siddiq, F., and Tondeur, J. (2019). The technology acceptance model (TAM): A meta-analytic structural equation modeling approach to explaining teachers' adoption of digital technology in education. *Computers & Education*, 128, 13-35.

[8] Clark, R. C., Nguyen, F., and Sweller, J. (2016). *Efficiency in Learning: evidence based Guidelines to Manage Cognitive Load*. John Wiley & Sons.

[9] Koedinger, K. R., Corbett, A. T., and Perfetti, C. (2013). The Knowledge-Learning-Instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive Science*, 36(5), 757-798.

[10] Chen, J., Wang, M., Kirschner, P. A., and Tsai, C. C. (2018). The role of collaboration, computer use, learning environments, and supporting strategies in CSCL: A meta-analysis. *Review of Educational Research*, 88(6), 799-843.

[11] Anderson, L. W., et al. (2001). *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. Longman, New York.

[12] Vygotsky, L. S. (1978). *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press.

[13] Bandura, A. (1977). *Social Learning Theory*. Prentice Hall, Englewood Cliffs, NJ.

[14] Deci, E. L. and Ryan, R. M. (2000). The "what" and "why" of goal pursuits: Human needs and the self-determination of behavior. *Psychological Inquiry*, 11(4), 227-268.

[15] Kirschner, P. A., Sweller, J., and Clark, R. E. (2006). Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist*, 41(2), 75-86.

[16] Moreno, R. and Mayer, R. E. (2004). Personalized messages that promote science learning in virtual environments. *Journal of Educational Psychology*, 96(1), 165-173.

[17] Paas, F., Renkl, A., and Sweller, J. (2003). Cognitive load theory and instructional design: Recent developments. *Educational Psychologist*, 38(1), 1-4.

[18] van Merriënboer, J. J. and Sweller, J. (2010). Cognitive load theory in health professional education: Design principles and strategies. *Medical Education*, 44(1), 85-93.

[19] Mayer, R. E. (2014). *The Cambridge Handbook of Multimedia Learning*. Cambridge University Press.

[20] Sung, Y. T., Chang, K. E., and Yang, J. M. (2015). How effective are mobile devices for language learning? A meta-analysis. *Educational Research Review*, 16, 68-84.

A Screenshots and Interface Examples

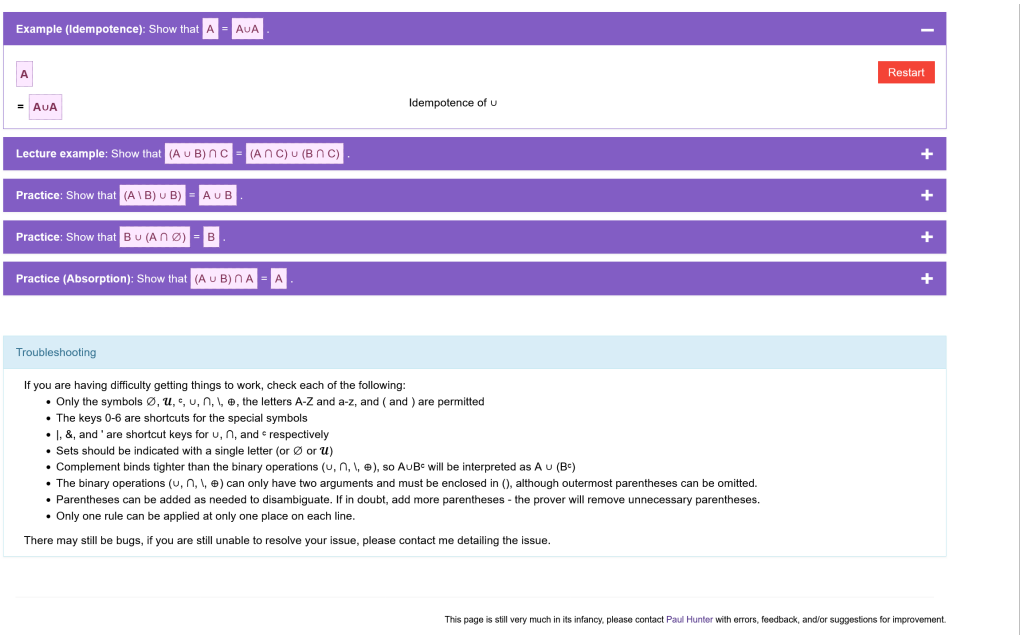


Figure 1: Original proof assistant interface showing navigation limitations and design issues identified through user feedback

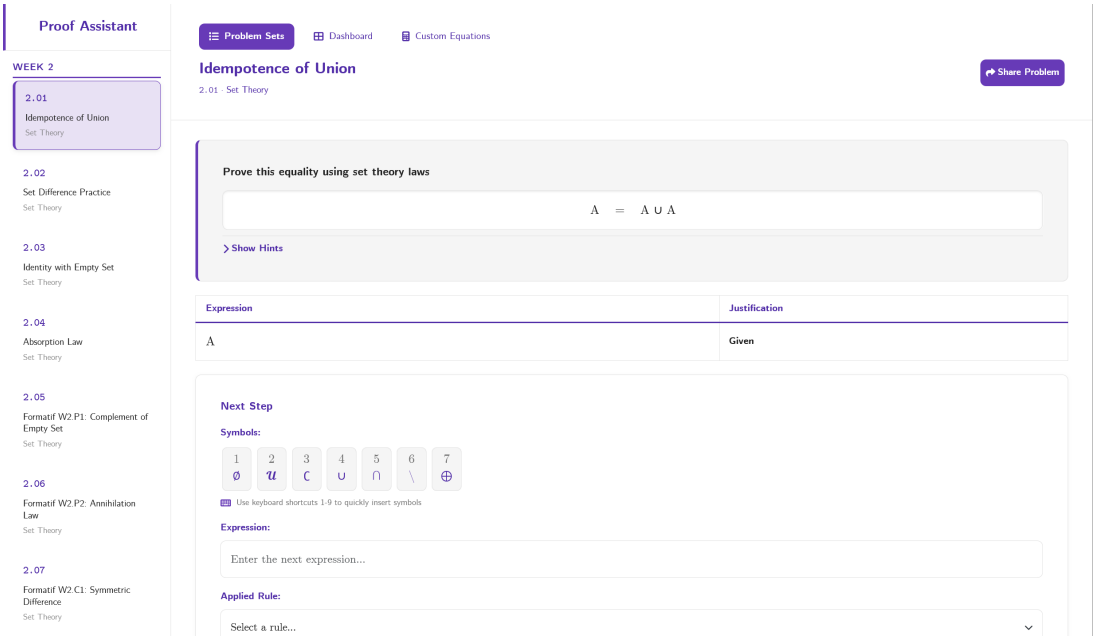
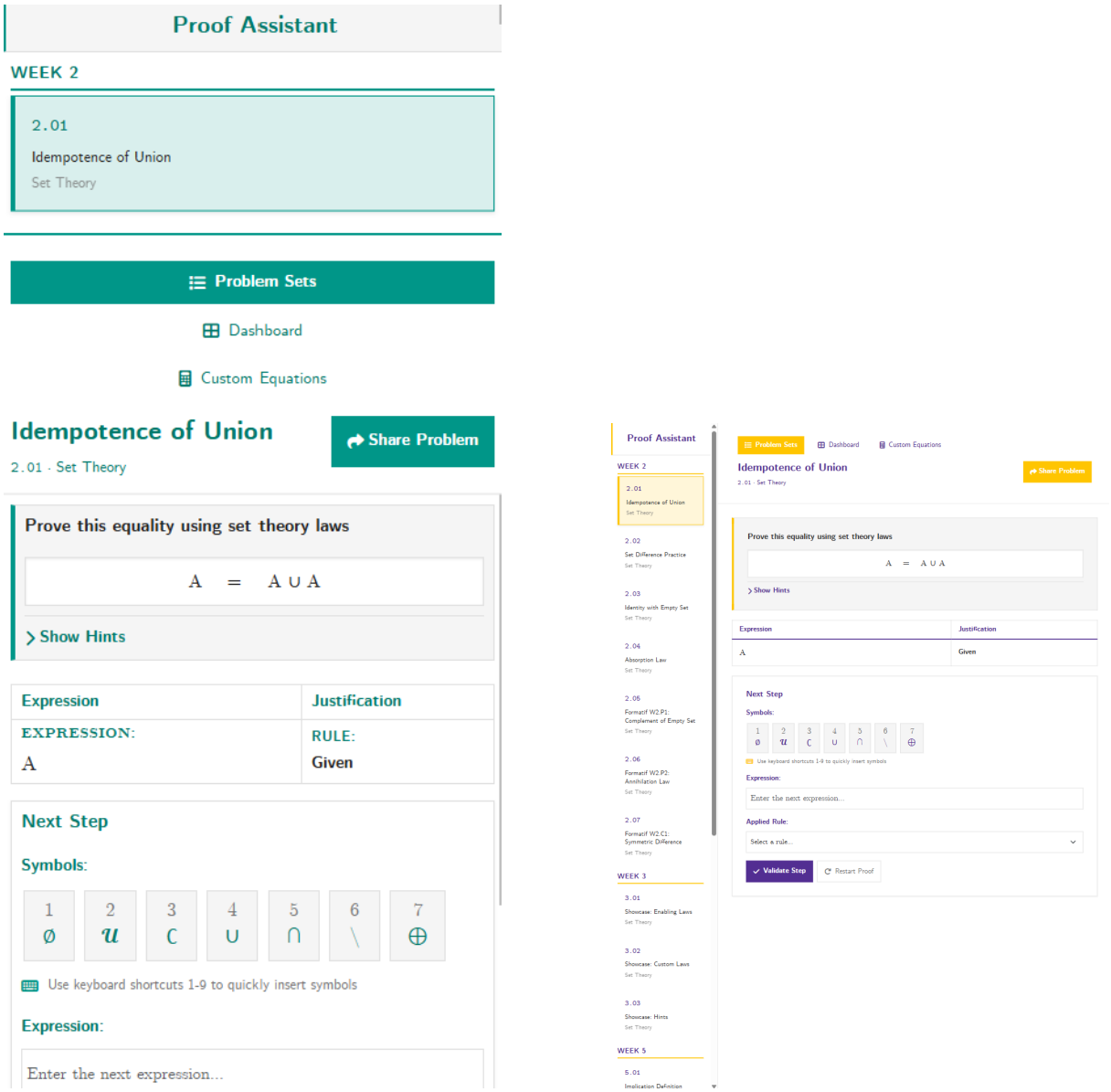


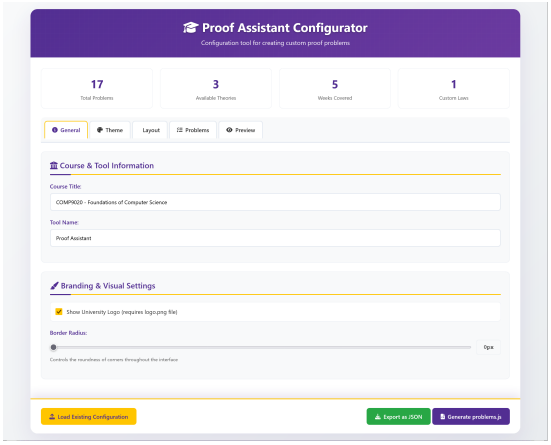
Figure 2: Redesigned interface implementing evidence based navigation patterns and progressive disclosure principles



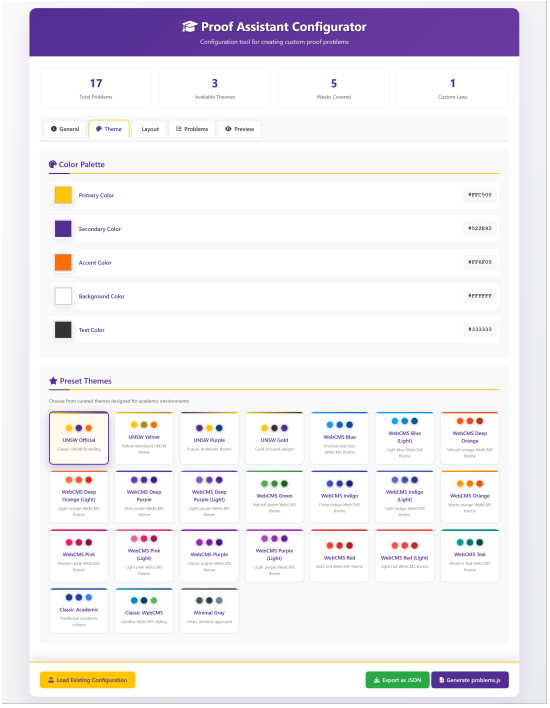
(a) Mobile responsive design

(b) iPad optimised interface

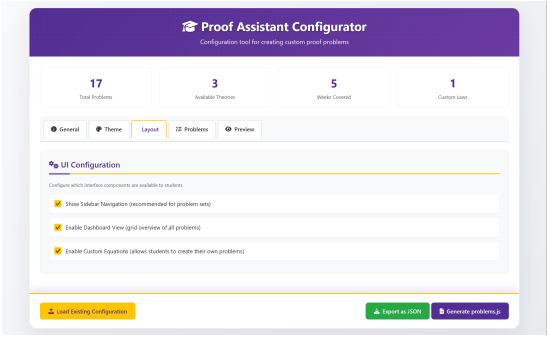
Figure 3: Mobile-first responsive design addressing mobile learning research findings



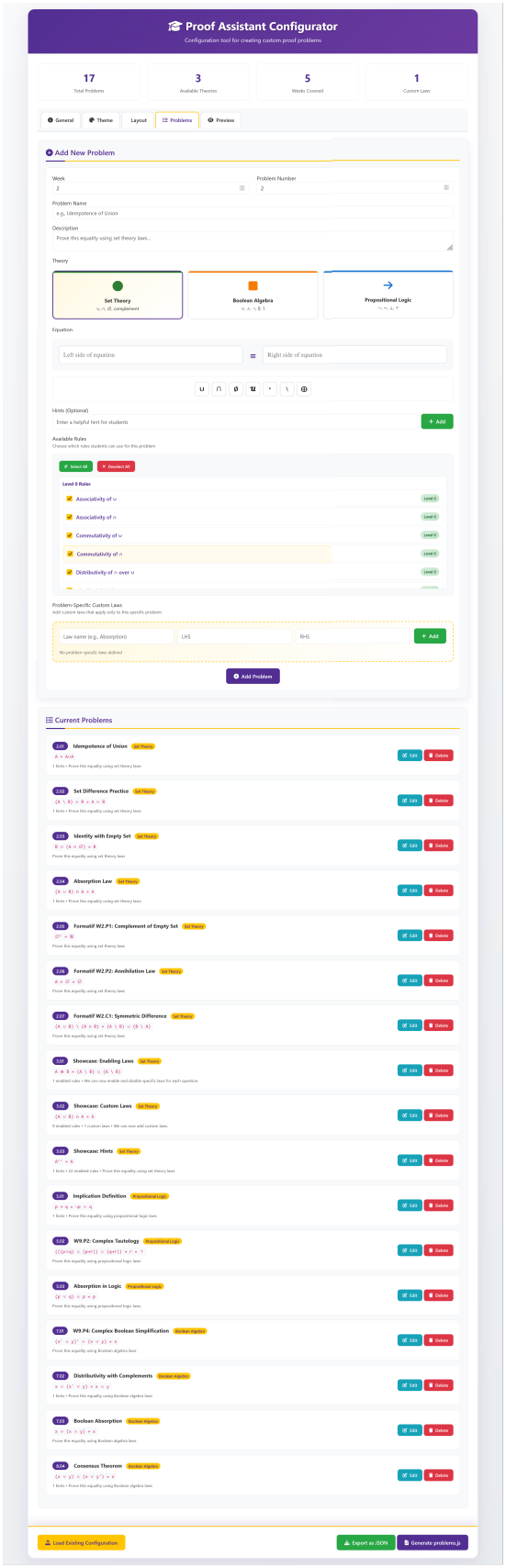
(a) Visual configuration step 1



(b) Visual configuration step 2



(a) Visual configuration step 3



(b) Visual configuration step 4

Figure 5: Visual configuration system reducing setup complexity based on adoption barrier research

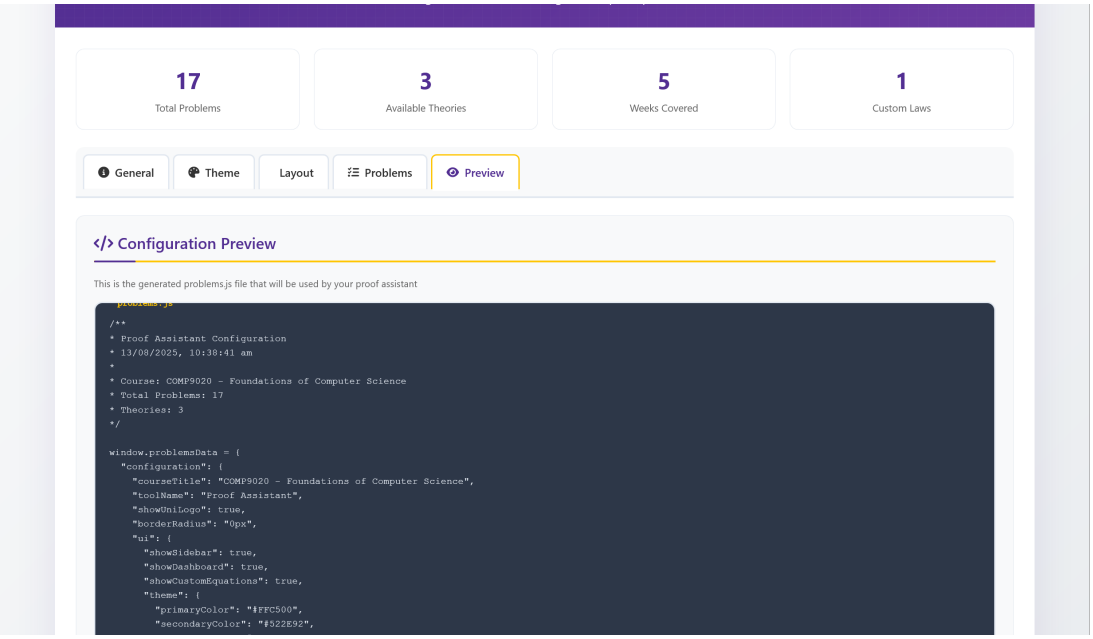
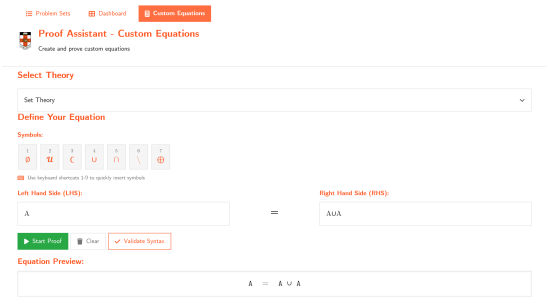
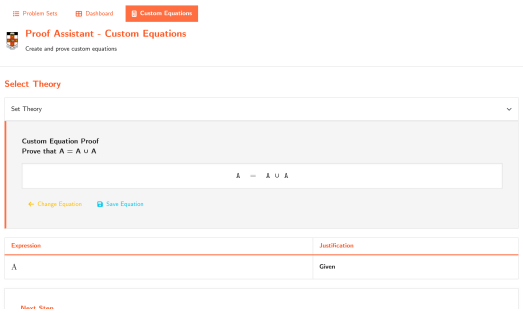


Figure 6: Final configuration step showing real-time validation and export options

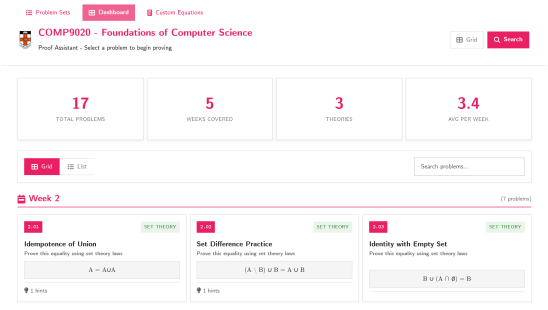


(a) Custom equation builder interface

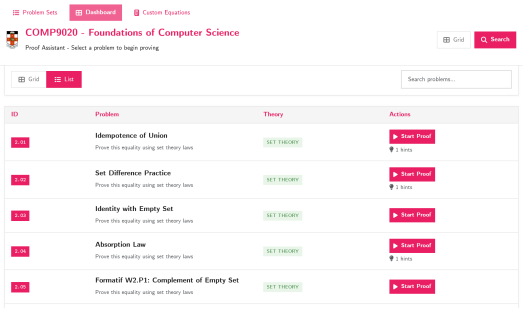


(b) Symbol palette and validation

Figure 7: Custom equation creation supporting problem-posing research principles



(a) Grid-based dashboard view



(b) List-based dashboard view

Figure 8: Dashboard interface implementing established principles for educational data visualisation