Michigan Department of Transportation
RC-1645

Final Report
Bridge Design System Analysis and Modernization

Tim Colling, PhD, PE, PI
Center for Technology & Training
Michigan Technological University
1400 Townsend Drive
Houghton, Michigan 49931

Chris Gilbertson, PhD, PE, Co-PI
Center for Technology & Training
Michigan Technological University
1400 Townsend Drive
Houghton, Michigan 49931

Gary Schlaff, Co-PI
Center for Technology & Training
Michigan Technological University
1400 Townsend Drive
Houghton, Michigan 49931

Mike Pionke
Center for Technology & Training
Michigan Technological University
1400 Townsend Drive
Houghton, Michigan 49931

September 27, 2016
Final Report

Michigan Technological University
1400 Townsend Drive
Houghton, MI 49931

Technical Report Documentation Page

1. Report No. RC-1645	**2. Government Accession No.** N/A	**3. Recipient's Catalog No.**
4. Title and Subtitle Bridge Design System Analysis and Modernization		**5. Report Date** September 27, 2016
		6. Performing Organization Code N/A
7. Author(s) Tim Colling; Chris Gilbertson; Gary Schlaff; Mike Pionke		**8. Performing Organization Report No.** N/A
9. Performing Organization Name and Address Michigan Technological University Center for Technology & Training 1400 Townsend Drive Houghton, Michigan 49931		**10. Work Unit No.** N/A
		11. Contract or Grant No. Contract 2013-0506
12. Sponsoring Agency Name and Address Michigan Department of Transportation (MDOT) Research Administration 8885 Ricks Road P.O. Box 33049 Lansing, Michigan 48909		**13. Type of Report and Period Covered** Final Report, 11/1/2013 to 9/30/2019
		14. Sponsoring Agency Code N/A

15. Supplementary Notes

Conducted in cooperation with the U.S. Department of Transportation, Federal Highway Administration. MDOT research reports are available at www.michigan.gov/mdotresearch.

16. Abstract

The Bridge Design System (BDS) is an in-house software program developed by the Michigan Department of Transportation's (MDOT) Bridge Design Unit. The BDS designs bridges according to the required specifications, and outputs corresponding design drawings and calculations. It has been the primary design tool for MDOT's bridges over the last several decades. Because of the BDS's longevity of use and development, MDOT has experienced a high level of comfort, familiarity, and efficiency with it. However, components of the BDS have been added and removed over the years, and little associated documentation exists today. The code itself has seen nearly 60 years of evolution in the Fortran programming language. Migration to another software system is likely to require significant changes to MDOT business processes and may require multiple software systems rather than the unified design system of the BDS. Also, long-term viability of the BDS would require documentation of the existing architecture and operation of the system as well as development of a plan for future compatibility and functionality of the software. Therefore, the Center for Technology & Training at Michigan Tech was contracted to document, analyze and propose modernization options for the BDS. This report describes the tools used to conduct this assessment and the results of this task.

17. Key Words	**18. Distribution Statement** No restrictions. This document is available to the public through the Michigan Department of Transportation.

19. Security Classif. (of this report) Unclassified	**20. Security Classif. (of this page)** Unclassified	**21. No. of Pages**	**22. Price** N/A

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

Disclaimer

Table of Contents

List of Figures

List of Tables

EXECUTIVE SUMMARY

As early as 1956, the Design Division of the Michigan Department of Transportation (MDOT) began developing Fortran computer programs to aid in bridge layout and superstructure/substructure design. In 1970, these programs were combined into a single system that is now known as the Bridge Design System (BDS). In 1984, MDOT's Design Division approved funds to complete the documentation of the BDS. Along with the documentation generated by this effort, MDOT added a series of forms to facilitate data input to the BDS. These forms became the basis for a graphical user interface (GUI) for the BDS. Over time, the BDS and its GUI have been updated with new features. However, documentation efforts have not been maintained at previous levels. To support the long-term viability of the BDS, MDOT contracted with the Center for Technology & Training (CTT) at Michigan Technological University in 2014 to document the existing architecture and operation of the system and to develop a plan for future compatibility and functionality of the software.

To achieve the objectives of the project *Bridge Design System Analysis and Modernization*, MDOT provided the CTT project team with the source code and GUI for the BDS, along with all historical documentation available. The tools that were selected to aid in exploring and documenting the BDS source code included a source control system, a Fortran compiler (Lahey/Fujitsu), an integrated development environment (Microsoft Visual Studio), a documentation generation program (Doxygen), and a high-level programming language (Python).

During the forensic investigation of the BDS, the CTT project team made non-functional changes to a copy of the BDS source code in order to enable investigation of features via Visual Studio and the Lahey compiler. These diagnostics identified obsolete language components and proprietary language extensions. The copy of the BDS source code was then separated into multiple files (known as modules) based on program unit boundaries.

The program architecture, process flow and functionality of the BDS were mapped out by creating call and caller diagrams, logic diagrams, and a hyperlinked table of variables. Design equations in the BDS were verified and cross-referenced against equations in *AASHTO LRFD Bridge Design Specifications* (7th edition), *AASHTO Standard Specifications for Highway Bridges* (17th edition), and the *Michigan Design Manual – Bridge Design*. Logic diagrams produced during this project detail the relationships between BDS modules. Call and caller diagrams supplement the logic diagrams; these examine the referential relationships between functions and track the flow of values between functions. Call and caller diagrams were generated by Doxygen and incorporated into Doxygen's HTML output. The CTT project team created worksheets cross-referencing BDS equations with their referenced specifications. They also

grouped cross-references into three tiers to prioritize their review with a focus on Load and Resistance Factor Design (LRFD) specifications as the top priority.

The *BDS Reference Guide* was created by using Python scripts to add markup to the BDS modules so that Doxygen could process them into cohesive HTML documentation. This HTML documentation contains the diagrams, tables, equation cross-references, and other content created up to this point. The overall organizational structure of the *BDS Reference Guide* serves to accommodate the documentation needs of software engineers and structural engineers.

An analysis of the BDS GUI and corresponding content from versions of the *BDS User's Manual* published in 1987 and 1997 culminated in a new amalgamated user's manual. The manual defines the constraints, expected values and required units (metric and/or standard) for each field of the GUI, and includes the addition of helpful technical illustrations.

The CTT project team interviewed 13 MDOT staff who are regularly involved with the BDS to help identify the strengths and weaknesses of the BDS from a user's standpoint. These interviews answered questions related to the design and maintenance of the BDS and assessed the users' general opinions of the system.

The tasks that comprised the *Bridge Design System* project achieved the following results: The *BDS Reference Guide* combined Doxygen's HTML output with the other documents generated by this project. It contains the *Engineering Technical Reference, Software Technical Reference*, and other generated documentation. The *BDS User's Manual* documented the history, scope and limitations of the BDS; the installation of the GUI; file options and preliminary data windows of the GUI; and the design input tabs of the GUI.

From the interviews with MDOT staff, the CTT project team identified potential areas of improvement to the BDS, including suggested modifications to the BDS and requested development of new time-saving features.

The CTT project team prepared a Software Risk Assessment describing the results of the forensic work with the BDS. This assessment also considered the development history of Fortran and the BDS, the institutional- and code-based risks associated with these developments, and opportunities to simplify program structure and improve code practice. The Software Risk Assessment yielded a series of recommendations for ensuring sustainable maintenance of the BDS in relation to modern computing environments:

1. Institute source control for the BDS code and documents.
2. Use an integrated development environment (IDE) for future development work.

3. Expand institutional knowledge of the BDS within MDOT so that staff retain sufficient institutional knowledge to be informed consumers of the services necessary to support the BDS.
4. Eliminate obsolete language components.
5. Modernize code practices.
6. Eliminate proprietary compiler extensions to enhance software portability.
7. Develop an automated test suite that is used in the development process.
8. Make the BDS callable from the GUI (create "Run" button).
9. Create a mixed-language program with a dynamic-link library (DLL).
10. Upgrade the BDS interface to be more visually representative of input.
11. Decouple distinct design or analysis functions where appropriate so that these functions can be run independently from one another.
12. Investigate and mitigate computational areas of concern.

To determine the best process for resolving and/or modernizing the BDS, the CTT project team evaluated each of these recommendations using six criteria—efficiency, scope, obsolescence, maintainability, interdependence, and usability and function. Based on this evaluation, the CTT project team organized the recommendations for modernizing the BDS code into a seven-phase implementation process. As time and budget allowed, the CTT project team began addressing some of the foundational recommendations while reserving the more complex, larger scope, or second-level recommendations for future work.

The recommendations can be implemented in the following seven-phase process:

Phase 1 – Modernize Development Tools
The first phase introduces modern tools for managing the development of the BDS software and reduces the time spent on all of the following phases. This introduction consists of small-scope items that can be accomplished within six months, easily fitting into the scope and budget of the BDS modernization project. The CTT project team accomplished the work required in this phase during the initial project contract.

Phase 2 – Reduce Global Risks to BDS Code
Phase 2 ensures the future operation of the BDS software while lowering the overall effort required for future development. The Phase 2 tasks address items that are not technically complex, but are typically global and frequent; because these addressed items are global and frequent, they represent a significant scope. The CTT project team accomplished the work required in this phase during the initial project contract.

Phase 3 – Combine GUI and BDS Code

Phase 3 establishes a mixed-language program that combines the BDS and GUI into one construct, allowing them to run outside of a DOS shell. Combining the BDS and GUI enables future improvements that will enhance the interactivity of the BDS and GUI. The CTT project team accomplished the work required in this phase during the initial project contract.

Phase 4 – Redesign GUI and Output Reports

The fourth phase involves revision of the GUI based on identified risks and meetings with MDOT staff. Revising the GUI results in a BDS interface that visually represents the input and that has the ability to run discrete design/analysis parts of the BDS independently.

Phase 5 – Modernize Code Practices

As an ongoing effort, Phase 5 remedies risks that are globally distributed, complex to correct and isolate, and extensive in scope. These remedies would be completed on a per-module basis as pieces of the software are edited or new features are added. The CTT project team accomplished some of the work required in this phase during the initial project contract.

Phase 6 – Update Engineering Calculations

Phase 6 involves investigation and mitigation of computational areas of concern. The Phase 6 tasks address locations identified by the CTT project team as needing further analysis regarding the suitability of a calculation or needing updates to meet current design specifications from MDOT or the American Association of State Highway and Transportation Officials (AASHTO).

Phase 7 – Develop New Enhancements

The seventh phase represents ongoing development that would impact the overall process of maintaining and modernizing the BDS. The scope, budget, and time frame will be determined as the ongoing maintenance requirements and BDS update needs evolve.

The *Bridge Design System* project concluded with several key developments that will ensure sustainable maintenance of the BDS in relation to modern computing environments. First, updates made to the BDS code guarantee that the program will work with modern software tools, including modern compilers, IDEs, and Doxygen. Second, the CTT project team verified and cross-referenced equations used in the BDS code with the most recent standard design specifications. Third, new processes enabled creation and maintenance of the *BDS Reference Guide*, a document that guides both engineers and software programmers through the use and applications of the BDS; in addition, the CTT project team updated the *BDS User’s Manual*, which serves as a guide for navigating the GUI of the BDS. Fourth, the CTT project team provided training that relates to the BDS and MDOT’S business process for bridge design. Finally, a software risk assessment identified code risks and critical concerns to the ongoing development and maintenance of the BDS; the CTT project team addressed recommendations resulting from the seven-phase BDS enhancement process as part of this project and is continuing this effort as part of a subsequent project, *Bridge Design System Ongoing Modernization and Support*.

1.0 INTRODUCTION

The Bridge Design Unit of the Michigan Department of Transportation (MDOT) developed an in-house software program called the Bridge Design System (BDS), which has been the primary tool used to design most of MDOT's bridges over the last several decades. The development of this system began as early as 1956, with significant features being added throughout the 1970s and 1980s using internal State of Michigan staff resources (Figure 1).

Incremental updates to the BDS have been made since 1995 to keep the program up-to-date with the current bridge design specifications. However, most documentation after this date has been in the form of internal code comments rather than external user's manuals or other documentation. The core of the BDS code used multiple versions of the Fortran programming language (varying depending on when the original code was written), and contained approximately 44,000 lines of code that called nearly 1,600 entry points in 90 subprograms. Portions of the BDS code were old by software standards but, nonetheless, still functional. In fact, the BDS still appears to be serving the Bridge Design Unit well. Fortran was developed to perform complex engineering calculations, so it was a natural choice for the original developers of the BDS. Fortran remains a viable programming language that reliably performs engineering calculations, and it is easy for structural engineers and software engineers to understand. The BDS source code also has an accompanying graphical user interface (GUI) that provides input and output functions for the users of the system.

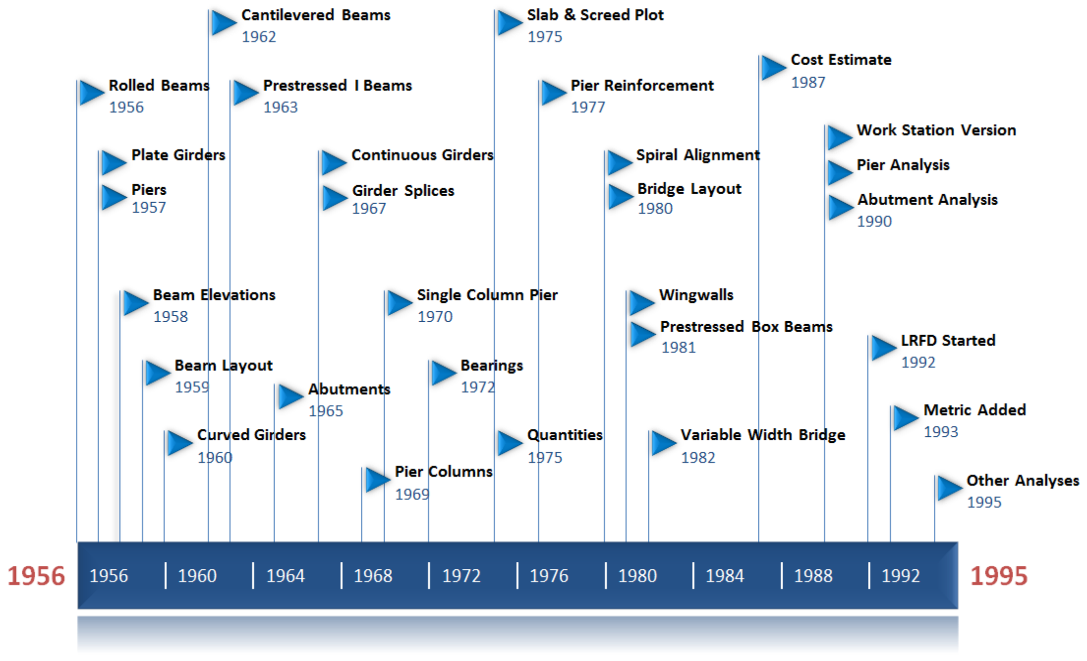


Figure 1: Development History of the BDS

The BDS is a tool that was specifically developed to fit the MDOT Bridge Design Unit's business process for designing structures and producing the necessary design drawings and calculations. Because the BDS was custom-made for MDOT, MDOT's Bridge Design Unit experiences a high level of comfort, familiarity and efficiency with the program. MDOT has considered using third-party software to assist or replace parts of the BDS's functionality; however, migration to another software system is likely to require significant changes to existing business processes and may involve multiple software systems rather than the unified design that is currently available in the BDS.

The MDOT Bridge Design Unit has further benefited from the fact that, over the last several decades, the two primary software developers for the BDS have been long-term, easily accessible employees of the State of Michigan. The institutional knowledge that these primary developers had was an irreplaceable asset in terms of assisting in long-term support and operation of the BDS. Recently, however, several developers of the BDS have retired, leaving a potential knowledge and support gap. While documentation for the BDS exists from the 1990s, external user documents have not been kept current since 1997, rendering the existing user documentation dated.

MDOT also maintained an archive of materials pertaining to the BDS, including two versions of a *BDS User's Manual*, lists of notations and routines, Seminar Notes for BDS training sessions, Developer Notes, and hand calculations. Two editions of the *BDS User's Manual* existed in the archive: a 1987 and a 1997 edition. The Seminar and Developer Notes date from the mid-1950s to the mid-1980s; they were a resource for training MDOT staff on new or updated portions of the BDS. The Developer Notes appear to be commentary and hand checks related to updated portions of the code, which were checked against the specifications or previous versions of the BDS. Neither the Seminar nor Developer Notes seem to have been formatted with the intent for the material to be distributed. Rather, they were formatted as notes for a presenter or a person familiar with the code and contain important and useful information for assessing the content of the BDS. Eight of these documents were digital, while 33 of these were hard copies.

2.0 OBJECTIVES

The goal of this project was to support the long-term viability of MDOT's BDS by documenting the existing architecture and operation of the system and by developing a plan for future compatibility and functionality of the software. Specifically, this would be accomplished through the following tasks:

- Documenting the existing program architecture, process flow and functionality of the BDS.
- Identifying design equations referenced in the BDS source code and cross-referencing them against current design specifications and institutional assumptions.
- Quantifying the risk created by the programming language of the BDS and its GUI related to existing and future compatibility with modern desktop computing hardware and software.
- Developing options to mitigate risk.
- Developing a plan for future maintenance of the BDS that includes contingencies for adapting to operating system, software and staffing changes.
- Enhancing functionality of the BDS to develop additional features or routines that will create operational efficiencies when creating bridge designs.
- Updating user documentation to reflect the current state of the BDS.
- Developing training resources that can be used to support and train MDOT bridge designers on newly added functions and calculations of the BDS, and using these resources to train existing MDOT staff.

Completing these tasks would establish a starting point for MDOT in prioritizing maintenance enhancements to the BDS, which would retain and protect the institutional investment that has been dedicated to the BDS and the business processes that have developed within MDOT's Bridge Design Unit. Developing documentation and training materials would increase the transparency of the bridge design process, which would assist in training new staff and in identifying technical areas that may require updates to maintain currency with future design procedures. The project would also develop a long-term plan that would guide the reduction of risk to the program and assist in scheduling future enhancement and maintenance activities.

3.0 SCOPE

The scope of this project was limited to investigating risks to the BDS code, proposing solutions related to modernization of the code, and undertaking initial tasks to implement those proposed solutions. The project did not specifically address risks associated with computations, design assumptions, or functionality of the BDS with regard to design specifications issued by the American Association of State Highway and Transportation Officials (AASHTO) or MDOT. Within the overall project's scope, each specific task had a defined scope that provided a narrow focus while achieving the project's objectives.

4.0 METHODOLOGY

4.1 Materials Received from MDOT

MDOT provided the CTT project team with the BDS source code, which is a single Fortran file (MVA30613.FOR), and the BDS GUI, which is comprised of 93 Visual Basic files.

The CTT project team also received numerous historical printed and electronic documents relating to the BDS. These documents aided understanding of the design and function of the BDS. The types of documents received included:

- WordPerfect/PDF explanations of changes and design considerations in the BDS
- BDS user's manuals from 1987 and 1997
- List of notations and their descriptions
- List of routines organized by subroutine
- Seminar Notes describing BDS training sessions that took place between the 1950s and the 1980s
- Developer Notes and reference materials from external sources, presumably used in development of portions of the BDS
- Hand calculations and review notes for small portions of the analysis for Load Factor Design (LFD) and LRFD designs, along with PDFs that show organization and test cases

The project team digitized these archival documents pertaining to the BDS and included them in the *BDS Reference Guide*.

4.2 Tool Selection

Several software tools were used extensively for the process of investigating and documenting the BDS. These tools are listed in the following sections.

4.2.1 SOURCE CONTROL

Source control is a system that manages access and changes to source code. Source control automatically records the history of code changes made whenever any part of the source code is modified by logging what changed, who changed it, the reason for the change, and when the change was made. This functionality is similar to the "track changes" function in Microsoft Word. Source control serves as an archive of a program's development and can be used to roll back the source code to a point in time before a change was made. Source control allows many people to work on the same program simultaneously.

Using a source control system for this project allowed an unaltered version of the BDS source code to be maintained while the CTT project team documented all subsequent changes in a working copy. A locally installed version of TortoiseSVN provided CTT software engineers, structural engineers and technical writers with the simultaneous ability to access and update the source code in a source code repository (the container that holds the program code within source control) without conflicts.

The source control repositories contained all utilities, code, and scripts used for the project. These included:

- Line printer to PDF conversion utility, which the CTT project team modified to allow the BDS to print to PDF
- BDS GUI
- Fortran 95 version of the BDS
- Utility for splitting a single Fortran program into smaller program units
- Utility for changing Fortran 77 line continuations into Fortran 90 style
- Python scripts used to process Fortran files into Doxygen comments
- Bridge design concepts guide (*Engineering for Programmers*) for software engineers or other non-structural engineers
- Logic diagrams
- *BDS User's Manual*
- BDS equation worksheets.

4.2.2 FORTRAN COMPILER

The CTT project team purchased the most current version (7.6.1) of the Lahey/Fujitsu Fortran compiler and installed it as a component of Microsoft Visual Studio. The BDS was built by MDOT with an older version (5.7) of the Lahey/Fujitsu Fortran compiler. Using the current version of the Lahey compiler ensured that the latest bug fixes, user experience, enhanced developer features, and performance improvements were available. The more recent compiler was required to guarantee compatibility between the executable code produced by the compiler and current versions of Windows.

4.2.3 IDE

Microsoft Visual Studio served as the integrated development environment (IDE) for this project. Modern IDEs are graphical form-based programs that provide a complete facility for software development. IDEs provide a single environment in which all development occurs. They typically provide many features for composing, debugging, and packaging software in one program rather than relying on separate programs to provide these functions. IDEs can either include stand-alone source control or interface with an outside source control system. Most modern IDEs offer an intelligent code completion feature that speeds code production and reduces errors. Modern IDEs typically have class and/or object browsers and diagramming tools for object-oriented software development.

4.2.4 DOXYGEN

The CTT project team selected the program Doxygen (1.8.7) to help create flowcharts and documentation for the BDS because of its ability to output HTML documents that map and hyperlink code quickly and consistently. Output from Doxygen is created and formatted based on a configuration file that instructs it on how to read a program's source code; user-defined tags to the source code give further control over the format and content of the output.

4.2.5 PYTHON

Python (3.4.1) is a programming language that was used to search, modify and package the BDS source code in conjunction with Doxygen. Python was chosen because of its extensive ability to search for and modify strings of code and for its compatibility with Doxygen.

4.3 Investigation and Modification of BDS Source Code

4.3.1 INVESTIGATION OF EXISTING FORTRAN CODE

The CTT project team identified BDS calculations that had associated comments referencing equations within specific design specifications. They checked these calculations for consistency with the current versions of the equations and conditionals presented in the following references:

- *AASHTO LRFD Bridge Design Specifications,* U.S. Units, 7th edition, 2014
- *AASHTO Standard Specifications for Highway Bridges,* 17th edition, 2002
- *Michigan Design Manual – Bridge Design*
- MDOT Bridge Design Guides
- *MDOT Standard Plans*

The most effective way to assess the large number of referenced equations within the BDS was to limit focus to the context of an individual entry point in order to review a greater percentage of the code given constraints in time and budget. Specific equations could then be examined in more detail on a case-by-case basis as justified by the standard evaluation. While equations were assessed based on information provided within the equation's entry point, the CTT project team also checked basic conditionals, such as those that limit equations to specific ranges or that determine appropriate equations based on relative relationships between other variables. While the CTT project team evaluated key components of an equation that were defined in other entry point, they did not concurrently verify program flow.

Variables calculated outside of a referenced equation's entry point were assumed to be accurate and were not verified. The evaluation of the accuracy of reference equations also excluded consideration of interactivity among different sections of the BDS, call order, unexpected behavior within GO TO or loop structures, and other issues of program flow. For the purposes of this study, the CTT project team also presumed that the appropriateness of a particular equation and the assumptions behind its use had been verified by the original developers and validated through years of use.

The CTT project team used a tiered approach for strategically evaluating equations in order to complete as many referenced equations as possible with the allocated time and budget. The written structure of the BDS source code identified clearly defined program unit boundaries, which helped organize the BDS into what will be hereafter referred to as "modules" (such as prestressed beams, rolled beams, and abutments). These modules contain equations for both the LRFD methodology and the Standard Specifications methodology; references to the Standard Specifications were considered a lesser priority than references to the LRFD methodology. The CTT project team, with MDOT approval, established tiers for the referenced equations based on the design methodology (LRFD or Standard Specifications) and based on the overall significance of a module to the BDS.

4.3.1 CHANGES TO THE BDS SOURCE CODE

The original source code files were preserved as received and placed in a source control repository. A duplicate copy of the source code, also placed in the source control repository, served as a working copy that could be modified for the purposes of this project. For the remainder of this report, the term "source code" refers to the working copy of this code rather than the original copy provided by MDOT.

Several non-functional changes to the BDS source code were necessary in order to use the capabilities of Visual Studio and the new version of the Lahey/Fujitsu Fortran compiler in the forensic exploration of the code. These capabilities include keyword highlighting, finding references, locating definitions, parameter expansions, and setting and stopping on breakpoints. The visual presentation of the code in the IDE also improved efficiency because the IDE changes how the code is displayed when working with it. The primary changes made to the BDS source code in order to benefit from these capabilities were:

- A PROGRAM MVA statement was added to the beginning of the file and all program unit END statements were changed to complete form: END [class [name]], where [class] is either PROGRAM, FUNCTION, SUBROUTINE, or BLOCK DATA, and [name] is the name of the program unit

- Language level was raised from Fortran 77 (a few sections were in older versions) to Fortran 2015; resulting syntax errors were corrected.
- Source code was changed from fixed to free format.
- Source code was changed to use new style line continuation.
- Blanks in names and keywords were removed.
- Declaration of assumed-length CHARACTER type functions was adjusted.
- Comments and blank lines from between subprograms were moved into program elements.
- COMMON blocks were replaced with USE [module] structures. This added two files, common.f95 and free_common.f95, to the project.
- The single BDS source code file was separated into 93 files based on program unit boundaries.
- Data initialization from the BLOCK DATA subprogram was moved into the corresponding modules in common.f95 and free_common.f95.

4.3.2 CREATION OF LOGIC DIAGRAMS

The CTT project team mapped out program architecture, process flow, and functionality of the BDS by creating call/caller diagrams, logic diagrams, and a hyperlinked table of variables. The CTT project team also linked program architecture and process flow documents to their respective source code locations in order to provide structural engineers and software engineers with an easy means of understanding the BDS operation.

Flowcharts called logic diagrams were created to illustrate the major algorithms, workflow, and processes of the BDS. They helped the CTT project team to visualize the BDS's functionality and to understand the bridge design process better. The logic diagrams at the highest levels of the BDS represent design elements as a single flowchart element. At this level, an element like Beam Design can be analyzed in terms of how it fits into the workflow of the BDS. Therefore, a logic diagram for Beam Design represents the major elements and decision points within Beam Design, where flowchart elements detail the Rolled Beam, Prestressed, and Continuous routines. This process of drilling down into smaller logical units continued until logic diagrams were no longer useful or necessary to describe the algorithm or process. Parts of the BDS exclusively used for reading input or writing output were not diagrammed. This process was repeated for each of these high-level units.

The creation of logic diagrams relied on one of two methods. Software engineers manually reviewed sections of the source code using tools in the IDE to determine flow and function.

They created many logic diagrams from scratch using Microsoft Visio. Alternatively, the software engineers created a few logic diagrams using Code Visual to Flowchart software to draft the flow diagram directly from the source code; these drafts were then edited in Visio to produce the final, accurate logic diagram. All Visio drawings were converted to PDF for inclusion with Doxygen's HTML documentation.

Logic diagrams were supplemented by the two types of diagrams created by Doxygen:

- Call diagrams showing what parts of the BDS are *used by* a given part of BDS source code. In other words, a call diagram illustrates the functions that each subprogram (subroutine, function, entry point) directly or indirectly calls.
- Caller diagrams showing what parts of the BDS are *using* a given part of BDS source code. In other words, a caller diagram depicts the functions by which that function is called.

Call and caller diagrams represent top-level relationships among BDS modules. Call and caller diagrams are basic program analysis tools. They increase understanding of the program by examining the referential relationship between functions and by tracking the flow of values between functions.

4.3.3 CROSS-REFERENCING OF DESIGN EQUATIONS

A complete list of referenced specifications in the BDS was generated and sorted by modules and further divided by entry points. Each reference within the BDS was then assessed against the most current specification. The most current *AASHTO LRFD Bridge Design Specifications* available at the time of the project proposal was the 6th edition with 2014 interim updates. The 7th edition became available at the onset of this task and, thus, became the reference against which the BDS was compared. The CTT project team also verified BDS references against the *AASHTO Standard Specifications for Highway Bridges,* 17th edition, and the *Michigan Design Manual—Bridge Design*.

The CTT project team used a variety of search parameters to identify references to specifications that were included in the original BDS source code as comments. The original references assumed many forms, with some addressing a particular section of the specification while others vaguely noted a page number; many did not include a reference to the specification itself. The CTT project team compiled and reviewed a list of potential references to determine if the referenced section agreed with the computations in the BDS. They also inserted Doxygen reference tags into the source code (the @aashto tag was used) to reference the specification, section or equation number, and a descriptive title of the referenced specification section.

Because of time and budget limitations, referenced design equations in the BDS code were categorized by module into one of three tiers (see Table 1; file name for the corresponding module is in italic text). LRFD sections of the BDS received prioritization over Standard Specifications sections. The assessment of specification equations referenced in the BDS was organized through a collection of worksheets.

Table 1: Tiers for Completion of Design Equations
Tier 1
• *beam* (rolled beams) • *cont* (continuous) • *main* (main program) • *pre* (prestressed beams) • *stif* (stiffener shear capacity)
Tier 2
• *abut* (abutment design) • *colu* (column design) • *hamm* (hammerhead pier design) • *pier* (pier design) • *pile* (pile reinforcement) • *pres* (earth pressures) • *reinfo* (reinforcement)
Tier 3
• *barspa* (bar spacing for pile caps and footings) • *bear* (bearing design) • *curt* (curtain wall design) • *prld* (loading of continuous structures) • *prop* (properties for continuous and prestressed beams) • *quan* (quantities) • *retu* (return wall footing on piles) • *retw* (return walls) • *slab* (slab and screed) • *spli* (splices in continuous sections) • *spre* (reinforcement for spread footings) • *stub* (stub abutments) • *wall* (abutment walls)

The CTT project team developed worksheets for each entry point containing one or more references. The entry points tended to be organized by design topics and sometimes contained several related equations, generally from the same section of the specifications. Each worksheet includes four sections: a summary of the AASHTO or MDOT equations, a list of variables associated with these equations along with existing definitions, a copy of the Fortran code comprising the entry point to illustrate context within the BDS, and commentary from the reviewing engineers.

- The **AASHTO Equations** section contains referenced equations and precursory equations located within the same entry point in the BDS. The equations were reformatted from the linear Fortran layout to mathematical notation to make them easier to read. An engineer manually reviewed the reformatted equations and compared them against the equations found in the specifications. If an area requiring further assessment was found, the reviewer highlighted that portion of the equation. If a more specific reference to the specifications was available, the reviewer added that information to the worksheet, highlighted it, and later updated it in the *BDS Reference Guide*. While conducting the assessment, some unreferenced equations within the BDS were found. References were added to these previously unreferenced equations both in the worksheet and in the BDS source code.
- The **Variable Definitions** section displays variables used in the referenced equations and their definitions as documented in the original BDS subroutine Notations file. Many variables contained no definition or contained a definition that did not appear appropriate under the context of a particular entry point. While these definitions were presented as found, their modification in the worksheet reflects the appropriate definition within the context of the entry point. Highlighting designates new definitions, while instances of strike-through font identify definitions not related to the entry point examined. The *BDS Reference Guide* includes a new Notations file that reflects the updated definitions.
- The **Context within BDS** section contains a copy of the Fortran code from the entry point being reviewed. The information presented in this section was not modified, but the equations presented in the AASHTO Equations section were highlighted to make them easier to find when conducting a review.
- The **Comments** section contains a summary of the reviewing engineer’s findings. This section also contains a presentation or summary of the content from the referenced specification and an assessment of how the BDS compares to the specification content. Inconsistencies or areas of concern are indicated in bold.

4.4 Update of the BDS Documentation

The BDS documentation needed to serve the overlapping needs of three specific audiences:

- Bridge designers who wish to know more about the internal operations of the BDS
- Software engineers who maintain the BDS
- General users of the BDS

To accommodate the different intended audiences, the CTT project team developed distinct documents:

- *BDS Reference Guide*
  - *Engineering Technical Reference*
  - *Software Technical Reference*
- *BDS User's Manual*

4.4.1 BDS REFERENCE GUIDE

In order for Doxygen to correctly interpret the BDS source code and organize the output documentation, the BDS source code first had to be marked up with custom Doxygen tags and then processed by Python scripts.

4.4.1.1 Markup of BDS Source Code

Markup of the BDS source code was essential for guiding Doxygen to compile HTML documentation as desired. The CTT project team added markups in the form of tags—special types of comments that do not affect the operation of the BDS. They used the following tags for directing Doxygen output:

- **@file** – Indicates that a comment block contains documentation for the file in which it occurs.
- **@brief** – Starts a one-paragraph brief description. This tag was added to each program unit and some entry points. This tag processed the larger or more important program units of the BDS into smaller units. A logic diagram for the main section and for the subsection are present in each of these smaller units.
- **@details** – Starts a detailed description (sometimes several paragraphs long). Where they could be identified, program narratives from the document *Documentation for Bridge Design Program* (June 20, 1997) were added to the corresponding program unit. This information is then displayed in the Detailed Description in the corresponding module and entry point as found in the reference guide.
- **@link** – Creates a link to an object (a file, class, or member) with user-specified link text.

- **@mainpage** – Replaces the default front page generated by Doxygen.
- **@ref** – Creates a reference to a named section, subsection, page, or anchor.
- **@section** – Creates a named section.
- **@seminarnotes** – Links to a PDF of notes from BDS training sessions run by MDOT.
- **@programmernotes** – Starts a section of notes for software developers.
- **@engineernotes** – Starts a section of notes for structural engineers.
- **@aashto** – A brief description of specifications cited in the program.

4.4.1.2 Processing of the BDS Source Code for Doxygen

The default Doxygen parser does not support some older Fortran language items that are not part of the current language standard. In particular, the parser misinterpreted obsolete string formatting conventions as Doxygen markup, and the obsolete string formatting conventions had to be replaced. The Python scripts accomplished other modifications to the BDS source code, and the Python scripts were re-run to update the documentation as needed.

The changes made to the BDS source code by the Python scripts are not permanent and do not appear in the working copy of the code. These scripts were stored in the same source control repository as the BDS. The Python scripts performed the following file processing:

1. **Converting ENTRY to SUBROUTINE:** The Fortran parser in Doxygen was not able to include ENTRY point structures in the call diagrams. Code was written to convert ENTRY points into internal subroutines that allowed a complete call diagram to be generated by Doxygen. By the end of the project this step was no longer required in Python as ENTRY points were converted to subroutines within the working copy of the BDS (see Phase 5 work plan, Section 4.10.5 of this report).

2. **Converting code following certain @brief Doxygen tags into internal subroutines:** This allows large program units to be documented and charted as a group of smaller units.

3. **Adding variables in notation.f95 to common.f95 documentation:** Comments on variable names from notation.f95 that matched a variable name in common.f95 were added to the documentation of that variable name. When there were multiple variables in common.f95 modules that had the same name, they were all given the same comments from the corresponding variable in notation.f95.

4. **Adding hyperlinks to the USE statements of common.f95 modules:** A hyperlink to the file where a USE statement of a common.f95 module occurs was added to the documentation of that module.

5. **Creating notation variables table**: A table of the variables in notation.f95 was created and added to the documentation of notation.f95 and to its own HTML page. The table contains the variable name, variable description, and references to each common.f95 module that has a variable with the same name.

6. **Processing statistics:** A separate HTML page containing statistics from the file processing performed by the Python processor was created.

Figure 2 outlines the multi-step process of modifying the BDS source code and producing the *BDS Reference Guide*. The process was executed by several Python scripts controlled by an outer Python script that has various options telling it where to read and write files. A temporary copy of the BDS source code was created and modified; following this, a Python script called Doxygen to process the source code. HTML documentation of the BDS source code was created as output from Doxygen.

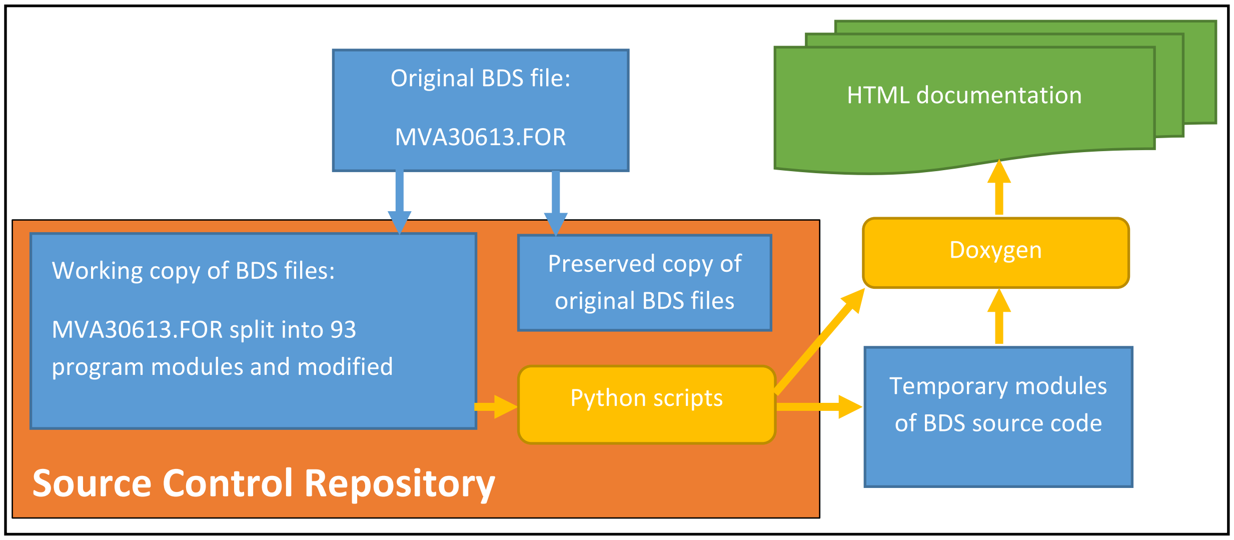

Original BDS file:
MVA30613.FOR
HTML documentation
Working copy of BDS files:
MVA30613.FOR split into 93 program modules and modified
Preserved copy of original BDS files
Doxygen
Python scripts
Temporary modules of BDS source code
Source Control Repository

Figure 2: Process for Creating HTML Documentation from Original BDS Source Code

4.4.1.3 Addition of Ancillary Documents

Much of the content generated from the forensic investigation and documentation of the BDS—such as logic and call diagrams and cross-references of design equations—was mutually beneficial to structural engineers and software engineers. This documentation was combined into a single document—the *BDS Reference Guide*—organized with tabs to separate the Engineering Technical Reference from the Software Technical Reference while allowing access to both. The centralizing source for the *BDS Reference Guide* was Doxygen's HTML output of the marked-up BDS source code since it contained the functionality to add and hyperlink documents related to the Technical References.

4.4.2 BDS USER'S MANUAL

A systematic assessment of the BDS GUI was conducted by reviewing each field through the GUI's front end. When more insight into a field was required, the GUI's code was examined with Microsoft Visual Studio. Screenshots were taken of each tab, and all internal documentation from the GUI was transcribed into a table. This content was paired with any matching content found in the earlier versions of the BDS user's manuals. The constraints, expected values and required units (metric and/or standard) for each field were defined. The CTT project team updated any relevant technical illustrations from old documentation into vector format to be more legible; meanwhile, they created new technical illustrations for fields that required additional explanation. This content was made into a print-formatted *BDS User's Manual* for those interested in using the BDS GUI and was kept separate from content related to the BDS source code.

4.5 Documentation of User Business Processes

The CTT project team determined the strengths and weaknesses of the BDS based on interviews with current BDS users. The narrow focus of these interviews was intended to help the CTT project team better understand:

1. How MDOT structural engineers use the BDS
2. MDOT's methodology for maintaining the BDS
3. Engineering and management concerns regarding the current limitations of the BDS
4. Desired enhancements to the BDS from a user's perspective
5. MDOT's long-term plans for the BDS.

In order to help the CTT project team identify the business processes that MDOT has built around the BDS, MDOT selected 13 individuals for interviews. These individuals were organized into three groups based on their roles in relation to the BDS: Bridge Design Squad Leaders, BDS Support Engineers & Software Evaluation Team, and Bridge Design Management. Each group was interviewed as a panel, which was more efficient than conducting 13 separate interviews.

During the interviews, each group was asked 12 questions:

- Did MDOT evaluate commercial bridge software as full or partial replacement of the BDS?
- Who supports the BDS? Do you have any concerns with this?
- Did MDOT consider outsourcing BDS support?
- What percentage of MDOT bridge in-house design is completed by the BDS?

- What are the current limitations of the BDS? Can the BDS for example do refined analysis of the steel beams when the skew is greater than 30 degrees?
- How do you overcome the current limitations of the BDS?
- How useful is the BDS in the analysis mode?
- Are there any features of the BDS that are obsolete and no longer in use? For example, the Quantities subroutine?
- Does the BDS get reviewed for consistency, where applicable, with the load rating group "Policies and Modeling Preferences"?
- What other in-house written applications do you use?
- Is there anything you would change in the BDS (money notwithstanding)?
- Would you like to add load-rating capability to the BDS?

4.6 Support for MDOT Bridge Designers

The CTT project team met with MDOT staff responsible for making changes regarding the modernization of the BDS software. The CTT project team provided training in the use of source control software, specifically checking in and out the BDS source code and downloading the *BDS Reference Guide*.

4.7 Software Risk Assessment

An evaluation of the existing BDS source code and Visual Basic GUI sought to identify any operational risks to the long-term viability of the software due to the changing state of practice for PC hardware and software. The suitability of design methods or equations was not addressed by the risk assessment.

The forensic investigation of the BDS, the documentation process, and the creation of the logic diagrams involved an extensive examination of the BDS source code both from an overall structural viewpoint and line by line. The diagnostic capabilities of Visual Studio and the compiler helped to identify obsolete language components and proprietary language extensions. Opportunities to simplify program structure and improve code practice were noted as well. Assessments were made of the business processes MDOT uses to support the BDS as well as the viability of the programming languages and tools used to deliver the BDS.

These results were used to gauge potential institutional and code-based risks to the BDS's long-term viability. The risk assessment also noted areas that were not necessarily considered outright risks, but could be opportunities for potential areas of improvement.

4.8 BDS Enhancement Phases

In response to the findings of the Software Risk Assessment (see Section 5.4.1), the CTT project team outlined a work plan for modernizing the BDS software and carried out as much of the work as was feasible within time and budget constraints.

To determine the best process for resolving the recommendations of the Software Risk Assessment and/or for modernizing the BDS, the CTT project team evaluated each recommendation using the following criteria:

- **Efficiency**—*Benefits to be gained from completing the recommendation early*: This could include benefits derived from the recommendation that make future development work easier or that otherwise have global benefit. Recommendations that add efficiency were prioritized due to the impact they would have on the budget during the life of the project.
- **Scope**—*Relative amount of work necessary to complete the recommendation based on its extent and complexit*y: Small-scope recommendations were selected to be completed under the current project as time and budget allowed. Large-scope projects were noted for future work plans.
- **Obsolescence**—*The recommendation's associated risk to the future operation of the software and the immediacy of that risk*: While no immediate threats to the operation of the BDS were discovered during the risk assessment, most issues identified in the recommendations would become acute concerns if not addressed in three to six years.
- **Maintainability**—*The recommendation's associated risk to the future of the software*.
- **Interdependence**—*Degree to which recommendations are related to each other*: Recommendations that have a high degree of interdependence were grouped together in order to be completed at the same time.
- **Usability and Functionality**—*User experience through changing functions or interface components*: This includes adding or changing software functionality, and will impact MDOT's business processes.

Based on the evaluation of each recommendation, the CTT project team developed a seven-phase approach for enhancing the BDS:

Phase 1 – Modernize Development Tools

- Phase 1 should be completed before any significant development work is started on the BDS because it provides efficiency and security for further phases.

Phase 2 – Reduce Global Risks to BDS Code

- Phase 2 should be completed after Phase 1 to take advantage of the efficiencies associated with the use of modern development tools.

Phase 3 – Combine GUI and BDS Code

- Phases 3, 6 and 7 can be completed at any time after Phase 2. These phases—Phases 3, 6, and 7—can occur before, after, or simultaneously, relative to one another based on MDOT's prioritization of needs. Phase 3, however, is a precursor to Phase 4, which involves a redesign of the GUI and the output reports.

Phase 4 – Redesign GUI and Output Reports

- Although Phase 4 should be completed after Phase 3, it is not otherwise constrained by the remaining phases. Some concurrent activity should be expected between Phase 6 and Phase 4.

Phase 5 – Modernize Code Practices

- Ongoing modernization takes place in Phase 5 and will be most effective and efficient if conducted concurrently as individual modules of the BDS are updated or improved.

Phase 6 – Update Engineering Calculations

- Phase 6 can be run concurrently with any other phase after Phase 2 is complete.

Phase 7 – Develop New Enhancements

- Phase 7 can be run concurrently with any other phase after Phase 2 is complete.

The organization of these seven phases maximizes efficiency for BDS modernization efforts.

Figure 3 shows the seven phases and their general order of operation.

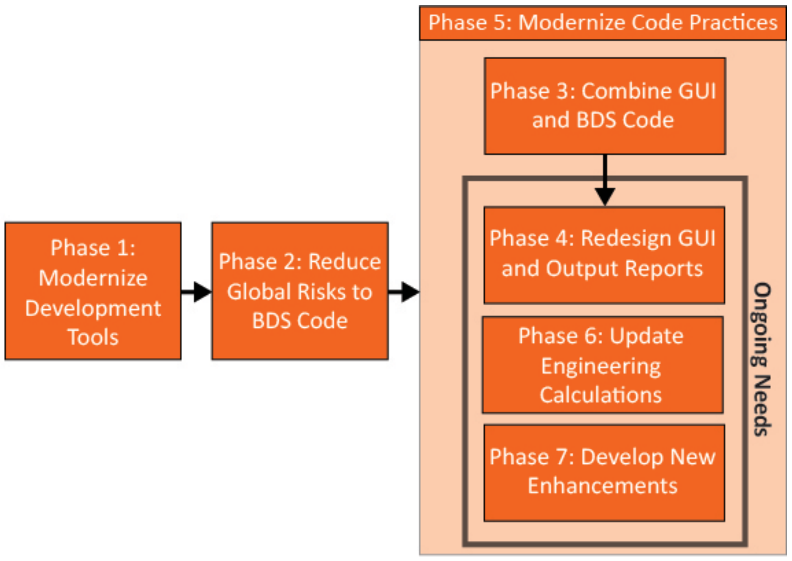

Phase 5: Modernize Code Practices
Phase 1: Modernize Development Tools
Phase 2: Reduce Global Risks to BDS Code
Phase 3: Combine GUI and BDS Code
Phase 4: Redesign GUI and Output Reports
Phase 6: Update Engineering Calculations
Phase 7: Develop New Enhancements
Ongoing Needs

Figure 3: Proposed BDS Enhancement Phases

The following subsections outline each phase.

4.8.1 PHASE 1: MODERNIZE DEVELOPMENT TOOLS

Modernizing the BDS depends upon implementation of modern tools for managing the development of software. Thus, the CTT project team categorized into Phase 1 those risk assessment recommendations that reduce the time spent on all of the following phases by boosting software engineer productivity, consequently reducing the chance of errors or catastrophic code loss and providing a method for multiple software engineers to work on the project simultaneously. Phase 1 recommendations isolate relatively small-scope items that could be accomplished in six months. These items fit into the scope and budget of the existing BDS modernization project; the results are detailed in section 5.5.1 in the Discussion of Results.

4.8.2 PHASE 2: REDUCE GLOBAL RISKS TO BDS CODE

Recommendations addressed by Phase 2 pertain to issues that have developed over time as programming practices and Fortran language standards have changed. These are recommendations that should be addressed before further development or modernization occurs on the BDS. While not technically complex, these recommendations are typically global (not confined to any single area of the program) and occur frequently. As such, they represent a significant amount of work. Phase 2 items address recommendations with a moderately-sized scope (that is, correctable in a short amount of time; e.g., months, not years) that ensure the future operation of the BDS while lowering the overall effort necessary for support and future development. These items fit into the scope and budget of the existing BDS modernization project; the results are detailed in section 5.5.2 in the Discussion of Results.

4.8.3 PHASE 3: COMBINE GUI AND BDS CODE

Phase 3 creates a mixed-language program that combines the BDS and GUI into one construct, allowing them to run outside a DOS shell. This enables future improvements that will enhance the interactivity of the BDS and GUI. This phase includes three tasks:

- *Task 1: Create an Object Model for Major Bridge Design Elements*

  This task creates a callable library of bridge design elements. This task also creates object models that encapsulate each of the major bridge design elements. The emphasis of this task is to define each design element in terms of its attributes (variables) and methods. All variables are limited in scope to those design modules that use them.

- *Task 2: Rewrite MAIN Subroutine*

  This task rewrites the MAIN subroutine so that it is able to call design elements of the BDS separately after the changes have been made to the GUI (see Phase 4). This task also implements all the long-term changes defined in Phase 5 during the process of rewriting MAIN.

- *Task 3: Create a Mixed-Language Program with a Dynamic-Link Library (DLL)*

  This task combines the Visual Basic GUI and the BDS Fortran program into a single project and converts the BDS from a console application to a DLL.

Some of the task items fit into the scope and budget of the existing BDS modernization project while others will be addressed in the subsequent project *Bridge Design System Ongoing Modernization and Support*; both the results and further needs are detailed in section 5.5.3 in the Discussion of Results.

4.8.4 PHASE 4: REDESIGN GUI AND OUTPUT REPORTS

Phase 4 addresses recommendations related to the design, testing, and creation of a new GUI and output reports for BDS calculations (Figure 4). Phase 4 also includes modifications to the BDS code that are specific to the GUI interactivity or output functions.

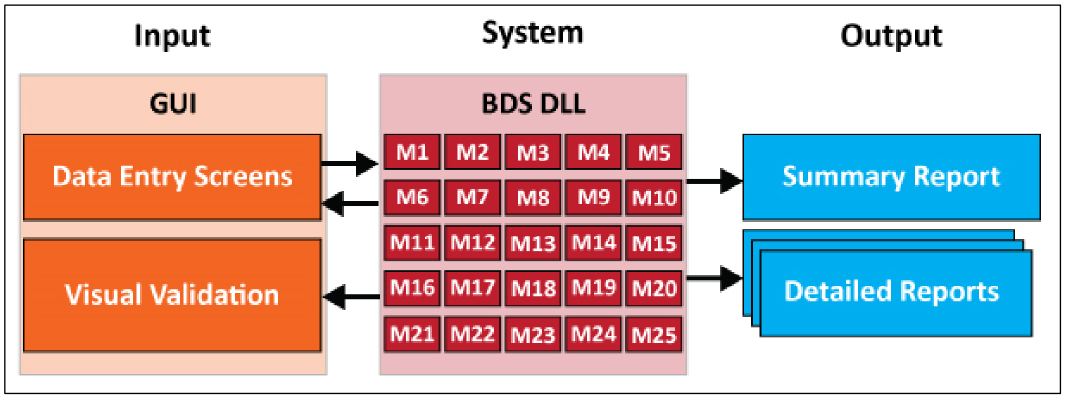


Figure 4: Proposed GUI/BDS Workflow

The scope for Phase 4 depends upon several requirements meetings, which would determine an accurate estimate of the time and budget for this phase. Phase 4 items were beyond the scope and budget of the existing BDS modernization project but will be addressed in the subsequent project *Bridge Design System Ongoing Modernization and Support*; an outline for Phase 4 is detailed in section 5.5.4 in the Discussion of Results.

4.8.5 PHASE 5: MODERNIZE CODE PRACTICES

Phase 5 addresses recommendations that were not handled during the previous phases. Phase 5 encompasses modernization recommendations that ensure the future operation of the software while lowering the effort necessary for support and development. These recommendations are global, complex to correct and isolate, and extensive in scope. As pieces of the software are edited or new features are added, recommendations would be completed on a per-module basis. The CTT project team believes that, in many cases, the extent of the fix necessary to correct the issues outlined in Phase 5 may require almost as much work as a rewrite of the module.

Three tasks cover these recommendations:

- *Task 1: Eliminate Obsolete Language Components*

  This task replaces non-block DO constructs, replaces real and double precision DO variables and DO loop control expressions with integer value parameters, and replaces ENTRY statements with internal SUBROUTINEs or MODULE PROCEDUREs. It also corrects non-standard array elements and substrings by using integer instead of decimal valued indices.

- *Task 2: Eliminate Mixed-Mode Arithmetic*

  This task eliminates the use of mixed-mode arithmetic wherever practical.

- *Task 3: Replace Logical IF and GO TO Flow Control*

  This task replaces logical IF statements and GO TO statements when possible.

Some of these tasks fit into the scope and budget of the existing BDS modernization project while others are an ongoing effort being addressed in the subsequent project *Bridge Design System Ongoing Modernization and Support*; both the results and further needs for responding to Phase 5's recommendations are detailed in section 5.5.5 in the Discussion of Results.

4.8.6 PHASE 6: UPDATE ENGINEERING CALCULATIONS

Phase 6 begins the process of following up on calculations flagged for further review, verifying their suitability, and documenting the results or identifying corrective actions that may be necessary through changes to the BDS code. Development of design documentation occurs as sections of the BDS are updated. Phase 6 is a long-term commitment and requires significant MDOT staff time to identify the necessary corrective action for each calculation of concern. Phase 6 items were beyond the scope and budget of the existing BDS modernization project but will be addressed in the subsequent project *Bridge Design System Ongoing Modernization and Support*; an outline for Phase 6 is detailed in section 5.5.6 in the Discussion of Results.

4.8.7 PHASE 7: DEVELOP NEW ENHANCEMENTS

Phase 7 represents ongoing development that would impact the overall process of maintaining and modernizing the BDS but has no scope, budget, or time frame. Examples of ongoing development include the addition of beam types or keeping specific calculations up-to-date with the changing AASHTO standards. Phase 7 items were beyond the scope and budget of the existing BDS modernization project but will be addressed in the subsequent project *Bridge Design System Ongoing Modernization and Support*; an outline for Phase 7 is detailed in section 5.5.7 in the Discussion of Results.

5.0 DISCUSSION OF RESULTS

5.1 Investigation and Modification of BDS Source Code

The changes made to investigate and document the BDS source code should not affect the operation of the BDS. It was verified that the modified BDS source code compiles with Lahey 7.6.1 and successfully runs with the test cases provided by MDOT.

5.1.1 FLOWCHARTS OF MAIN PROGRAM

By investigating the existing Fortran code and by examining the logic diagrams showing detailed relationships within BDS modules, the CTT project team was able to produce flowcharts containing decision points where the BDS directs program flow (Figure 5).

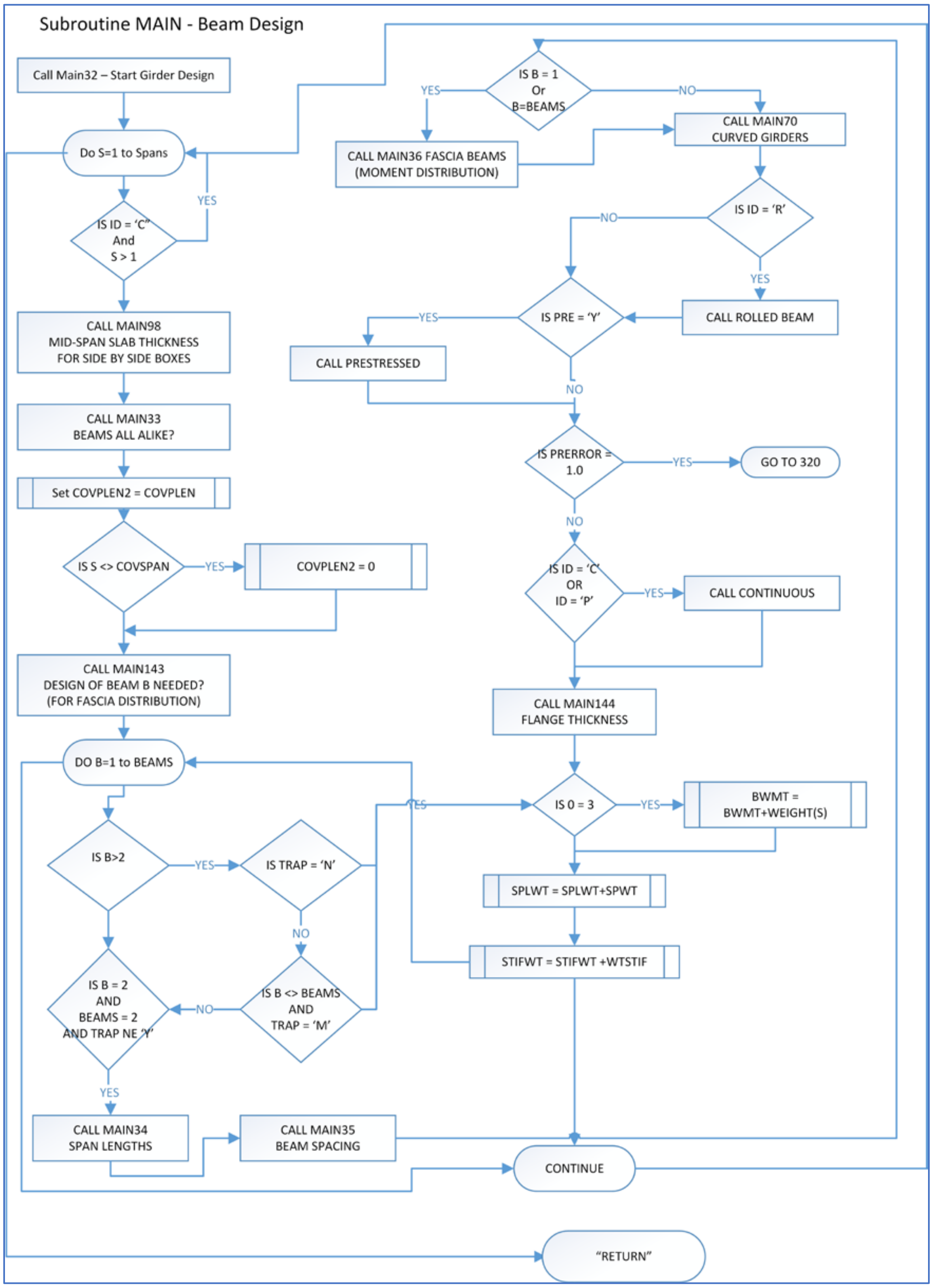


Figure 5: Logic Diagram for Beam Design (A Component of Main)

Logic diagrams were created for 38 BDS modules (see Table 2), resulting in 208 diagrams total.

Table 2: BDS Modules with Logic Diagrams

abut_analysis	curtain	pier_design	rolled_beam
abutment	deflections	pierld	rpress
abutquan	distributions	pierpile	slab_and_screed
abutwall	elevation	pilesreinf	splice
beam_layout	hammer	plates	spreadreinf
bearing	joint	pressures	stiffener
c_mom	main	prestressed	stubabut
cmoms	moment	properties	wallret
column	MVA	quantities	
continuous	pier_analysis	retwall	

Because Doxygen automatically generated these flowcharts, the flowcharts do not contain any of the logic that takes place within the code. The individual cells within each call diagram hyperlink to the corresponding module or entry point, allowing for easier navigation of the BDS source code. The *BDS Reference Guide* contains a total of 4,632 call/caller diagrams, which were generated by Doxygen (Figure 6 and Figure 7).

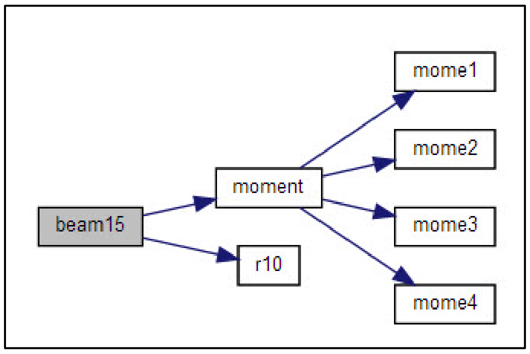


Figure 6: Call Diagram for beam15, A Subroutine of Rolled Beam

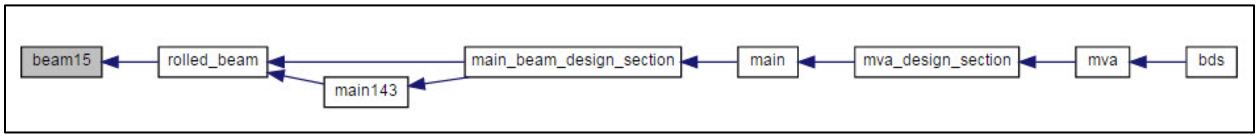


Figure 7: Caller Diagram for beam15

5.1.2 CROSS-REFERENCING OF DESIGN EQUATIONS

Worksheets were created for all AASHTO LRFD references within Tier 1 (69 worksheets). Time and budget allowed for the creation of an additional 44 worksheets within the remaining tiers.

Ongoing support of the BDS will require revisiting these worksheets. Additional worksheets will need to be generated for references not addressed within this project and for sections of the code that are updated to the most current AASHTO specifications.

Individual worksheets, located in the *BDS Reference Guide*, contain detailed information comparing the BDS source code equations to those found in the referenced specifications. Instances of bold font denote areas where deviations between the BDS and referenced specifications occurred. A summary table in the *BDS Reference Guide* provides an overview, which can be used as a punch list of areas in the BDS that should be further investigated.

5.2 Update of the BDS Documentation

The CTT project team created two separate documents: the *BDS Reference Guide* and the *BDS User's Manual*.

5.2.1 BDS REFERENCE GUIDE

The *BDS Reference Guide* is an electronic document containing BDS documentation pertinent to structural engineers and software engineers (Figure 8 through Figure 12). It contains the *Engineering Technical Reference, Software Technical Reference*, and other BDS documentation including the historical notes provided by MDOT.

The *Engineering Technical Reference* contains:

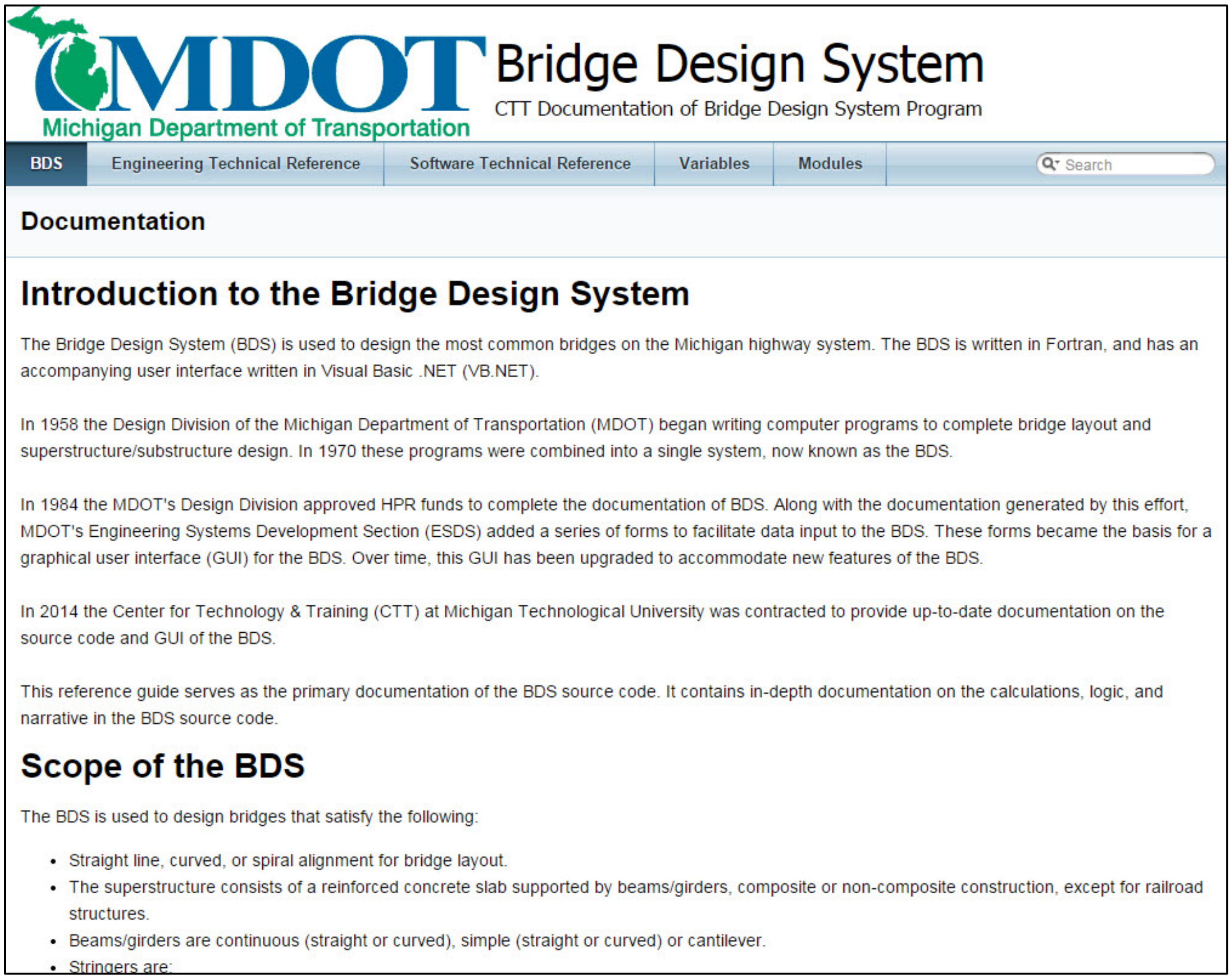
- Engineer Notes: Worksheets developed for the assessment of the referenced equations
- AASHTO References: Referenced equations within the BDS; Programmer Notes and Engineer Notes can also be accessed through this area
- Seminar Notes: PDFs of Seminar and Developer Notes from the historical documentation
- Variable Notes: Variables and their corresponding definitions (same list as shown under the *Software Technical Reference* tab)

The *Software Technical Reference* contains:

- Programmer Notes: Links to the logic diagrams
- File Statistics: Explanation of how program units and variables were analyzed for documentation
- Variable Notes: Variables and their corresponding definitions (same list as shown under the *Engineering Technical Reference* tab)

Other BDS documentation includes:

- Variables: Variable groups (COMMON blocks) with links to the detailed documentation of the variables in the groups and the referencing modules
- Modules: List of modules, with links to the detailed documentation of each module (including call/caller diagrams) and links to the BDS source code for the module.


MDOT
Michigan Department of Transportation
Bridge Design System
CTT Documentation of Bridge Design System Program
BDS
Engineering Technical Reference
Software Technical Reference
Variables
Modules
Search
Documentation

Introduction to the Bridge Design System

The Bridge Design System (BDS) is used to design the most common bridges on the Michigan highway system. The BDS is written in Fortran, and has an accompanying user interface written in Visual Basic .NET (VB.NET).

In 1958 the Design Division of the Michigan Department of Transportation (MDOT) began writing computer programs to complete bridge layout and superstructure/substructure design. In 1970 these programs were combined into a single system, now known as the BDS.

In 1984 the MDOT's Design Division approved HPR funds to complete the documentation of BDS. Along with the documentation generated by this effort, MDOT's Engineering Systems Development Section (ESDS) added a series of forms to facilitate data input to the BDS. These forms became the basis for a graphical user interface (GUI) for the BDS. Over time, this GUI has been upgraded to accommodate new features of the BDS.

In 2014 the Center for Technology & Training (CTT) at Michigan Technological University was contracted to provide up-to-date documentation on the source code and GUI of the BDS.

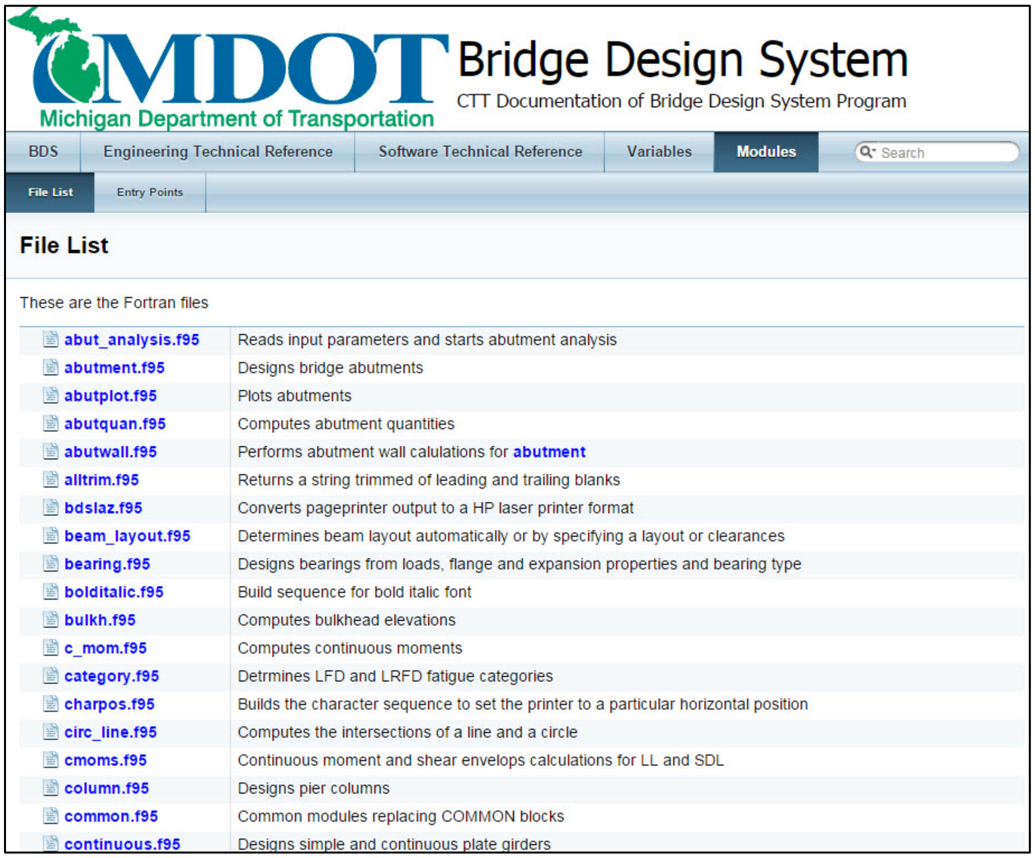
This reference guide serves as the primary documentation of the BDS source code. It contains in-depth documentation on the calculations, logic, and narrative in the BDS source code.

Scope of the BDS

The BDS is used to design bridges that satisfy the following:

- Straight line, curved, or spiral alignment for bridge layout.
- The superstructure consists of a reinforced concrete slab supported by beams/girders, composite or non-composite construction, except for railroad structures.
- Beams/girders are continuous (straight or curved), simple (straight or curved) or cantilever.
- Stringers are:

Figure 8: Front Page of the BDS Reference Guide

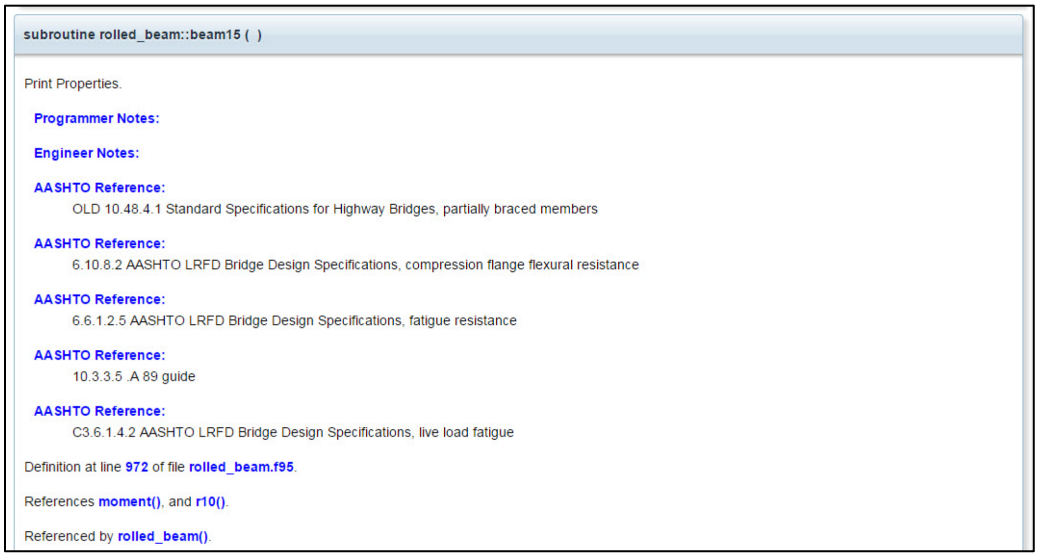

Michigan Department of Transportation
Bridge Design System
CTT Documentation of Bridge Design System Program
BDS
Engineering Technical Reference
Software Technical Reference
Variables
Modules
Search
File List
Entry Points

File List

These are the Fortran files

abut_analysis.f95	Reads input parameters and starts abutment analysis
abutment.f95	Designs bridge abutments
abutplot.f95	Plots abutments
abutquan.f95	Computes abutment quantities
abutwall.f95	Performs abutment wall calulations for **abutment**
alltrim.f95	Returns a string trimmed of leading and trailing blanks
bdslaz.f95	Converts pageprinter output to a HP laser printer format
beam_layout.f95	Determines beam layout automatically or by specifying a layout or clearances
bearing.f95	Designs bearings from loads, flange and expansion properties and bearing type
bolditalic.f95	Build sequence for bold italic font
bulkh.f95	Computes bulkhead elevations
c_mom.f95	Computes continuous moments
category.f95	Detrmines LFD and LRFD fatigue categories
charpos.f95	Builds the character sequence to set the printer to a particular horizontal position
circ_line.f95	Computes the intersections of a line and a circle
cmoms.f95	Continuous moment and shear envelops calculations for LL and SDL
column.f95	Designs pier columns
common.f95	Common modules replacing COMMON blocks
continuous.f95	Designs simple and continuous plate girders

Figure 9: Links to Documentation for Each BDS Module



subroutine rolled_beam::beam15 ( )

Print Properties.

Programmer Notes:

Engineer Notes:

AASHTO Reference:
OLD 10.48.4.1 Standard Specifications for Highway Bridges, partially braced members

AASHTO Reference:
6.10.8.2 AASHTO LRFD Bridge Design Specifications, compression flange flexural resistance

AASHTO Reference:
6.6.1.2.5 AASHTO LRFD Bridge Design Specifications, fatigue resistance

AASHTO Reference:
10.3.3.5 .A 89 guide

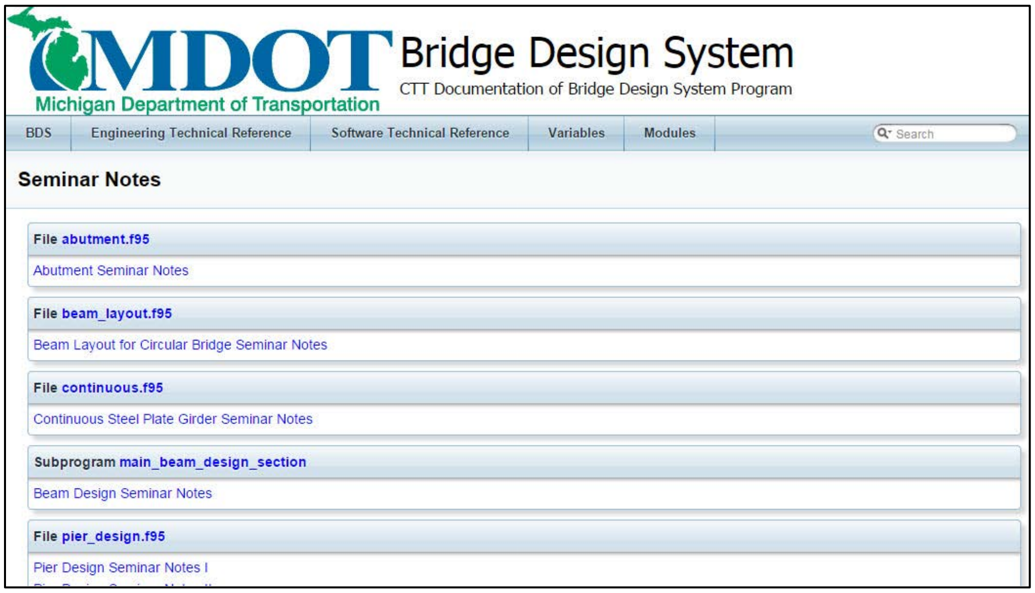
AASHTO Reference:
C3.6.1.4.2 AASHTO LRFD Bridge Design Specifications, live load fatigue

Definition at line 972 of file rolled_beam.f95.

References moment(), and r10().

Referenced by rolled_beam().

Figure 10: Reference Guide Documentation for the beam15 Entry in the Rolled Beam Module


MDOT
Michigan Department of Transportation
Bridge Design System
CTT Documentation of Bridge Design System Program
BDS
Engineering Technical Reference
Software Technical Reference
Variables
Modules
Search

Seminar Notes

File abutment.f95

Abutment Seminar Notes

File beam_layout.f95

Beam Layout for Circular Bridge Seminar Notes

File continuous.f95

Continuous Steel Plate Girder Seminar Notes

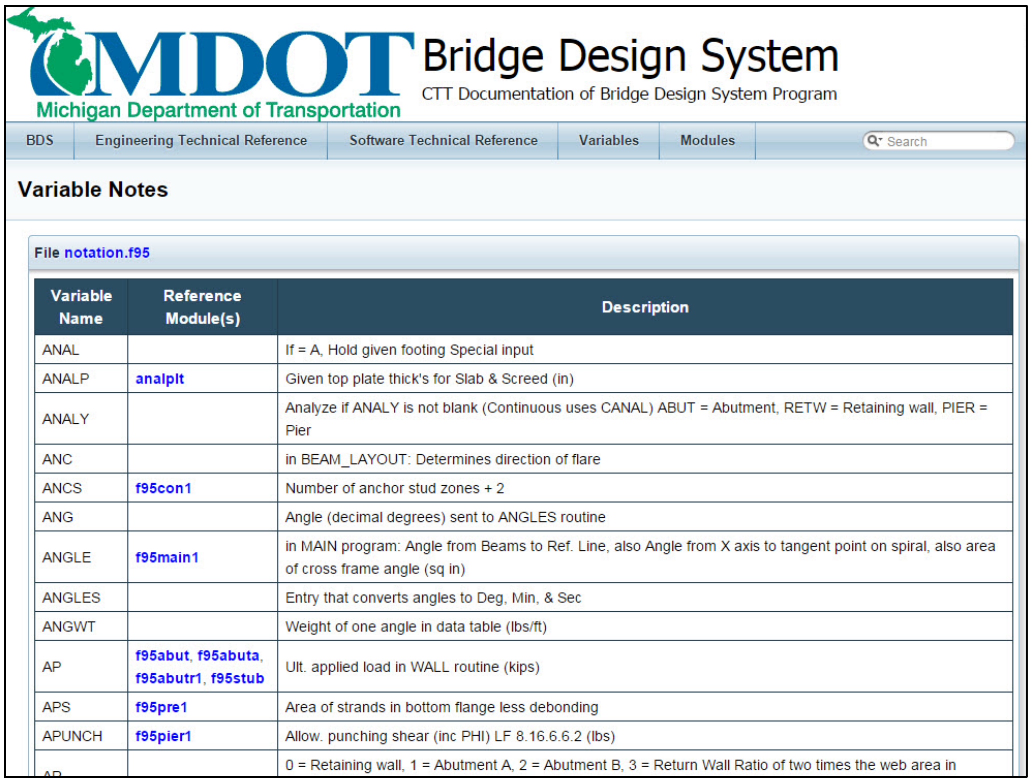
Subprogram main_beam_design_section

Beam Design Seminar Notes

File pier_design.f95

Pier Design Seminar Notes I

Figure 11: Seminar Notes Listed in the BDS Reference Guide


MDOT
Michigan Department of Transportation
Bridge Design System
CTT Documentation of Bridge Design System Program
BDS
Engineering Technical Reference
Software Technical Reference
Variables
Modules
Search

Variable Notes

File notation.f95

Variable Name	Reference Module(s)	Description
ANAL		If = A, Hold given footing Special input
ANALP	analplt	Given top plate thick's for Slab & Screed (in)
ANALY		Analyze if ANALY is not blank (Continuous uses CANAL) ABUT = Abutment, RETW = Retaining wall, PIER = Pier
ANC		in BEAM_LAYOUT: Determines direction of flare
ANCS	f95con1	Number of anchor stud zones + 2
ANG		Angle (decimal degrees) sent to ANGLES routine
ANGLE	f95main1	in MAIN program: Angle from Beams to Ref. Line, also Angle from X axis to tangent point on spiral, also area of cross frame angle (sq in)
ANGLES		Entry that converts angles to Deg, Min, & Sec
ANGWT		Weight of one angle in data table (lbs/ft)
AP	f95abut, f95abuta, f95abutr1, f95stub	Ult. applied load in WALL routine (kips)
APS	f95pre1	Area of strands in bottom flange less debonding
APUNCH	f95pier1	Allow. punching shear (inc PHI) LF 8.16.6.6.2 (lbs)
		0 = Retaining wall, 1 = Abutment A, 2 = Abutment B, 3 = Return Wall Ratio of two times the web area in

Figure 12: Variable Notes as Displayed in the Reference Guide

Since a printed version of the *BDS Reference Guide* would be over 4,000 pages, the reference guide is expected to remain in HTML format as opposed to PDF or hard copy.

5.2.1.1 Python Scripts and Doxygen Configuration File

The CTT project team developed a series of Python scripts that seamlessly integrates any updates made to the Doxygen markups in the BDS source code into the *BDS Reference Guide*. When a working copy of the BDS is requested from source control, these scripts are automatically placed in the "PythonScripts" folder. These Python scripts can handle the multi-step process of modifying[1] the BDS source code into a compatible form for Doxygen, and then calling Doxygen to create new HTML documentation. To execute each step in this process, an outer Python script (*bdsgendocs.py*) with various options can be run from the Windows command prompt, or a shortcut on the desktop can be created to run the script.

Executing *bdsgendocs.py* creates updated HTML documentation for the *BDS Reference Guide* from the BDS source code. *Bdsgendocs.py* is typically executed on the command line with the "PythonScripts" folder as the current directory. *Bdsgendocs.py* must be given the location of the BDS source code as its first command line argument. When the script completes, updated documentation will be created at the destination folder specified by the Doxygen configuration file. The root of the site is the index.html file in the "…\html\" folder.

In addition to creating HTML documentation during this process, the Python scripts also self-test to ensure that their output is correct. If there is an error in creating the output, a warning message will appear.

Several command line arguments exist for customizing the location of input and output files as well as degree of documentation processing. Documentation creation was tested on Windows 7 (64-bit systems). The version of Doxygen used was 1.8.7 while the version of Python used was 3.4.1. The computer creating the documentation must have Python installed and Doxygen installed. Additionally, to generate call/caller diagrams the "dot" tool (part of the Graphviz graph visualization package) must also be installed.

5.2.2 BDS USER'S MANUAL

The *BDS User's Manual* covers the major operations of the BDS GUI. It includes history, scope and limitations of the BDS; installation and overview of the GUI; file options and preliminary

[1] The working copy of the BDS source code **is not changed** when creating documentation. The copy Doxygen works with is a temporary copy intended only for Doxygen use.

data windows; and design input tabs. The vast majority of the *User's Manual* documents the 13 design input tabs:

- Alignment (27)
- Special Alignment (14)
- Superstructure (20)
- Special Superstructure (31)
- Substructure (9)
- Special Abutment/Misc. (22)
- Special Pier (16)
- Special Rolled Beam (12)
- Special Continuous Plate Girder (7)
- Special Prestressed Beam (13)
- Hold Beam Offsets (1)
- Hold Bolster Elevations (1)
- Autodraw Plot (no fields)

Documentation of the input tabs focuses on explaining fields (or groups of fields when multiple fields have similar documentation). Altogether, 173 fields or groups of fields were documented; the number per tab is indicated in parentheses in the list above.

5.3 Documentation of User Business Processes

After analyzing the results of the three group interviews, the following was observed:

- MDOT engineers had strong preferences for maintaining the BDS as the long-term main bridge design software while using other commercial software for checking purposes or in the few cases that the BDS cannot design. However, these programs do not communicate with each other, and could not replace the BDS. MDOT management has not made a final decision regarding the long-term plans for the BDS.
- Engineers and management uniformly view the BDS as a "black box" and would like it to be easier to understand and maintain.
- The BDS has been primarily written and maintained by structural engineers, which has resulted in a product that is harder to understand and maintain than if it had been written and maintained by professional software engineers.
- Maintaining the BDS is preferred to be a group effort (in-house or outsourced) to avoid being too reliant on one single individual from a long-term perspective. Furthermore,

the size of the BDS is too large for one individual to maintain all standards and code effectively.

- MDOT would like to add more features to the BDS. Two features in particular were mentioned in multiple interviews: having the BDS capable of designing new types of prestressed concrete beams (in particular Indiana's Bulb Tee beams); and incorporating partial rebounding of beams in deck replacement projects.
- Management and engineers both prefer to keep bridge design and bridge rating separate via the use of multiple software applications.
- Maintenance and testing have been significantly better for the BDS than for the other stand-alone applications developed in-house. MDOT management seems to prefer to take these stand-alone applications out of service, while MDOT engineers would like to keep them available to users at some level.
- MDOT management does not want to allocate resources to maintain or expand the existing autodraw applications. They perceive that these applications are not widely used and expanding them does not align with the future trend in automated drafting. None of the structural engineers interviewed felt strongly about keeping these applications either. MDOT is more interested in the development of new applications, such as 3-D drafting.
- Many MathCAD and Excel spreadsheets have been developed for particular uses by individual engineers in the different design squads. It would be preferred if these were inventoried, consolidated, verified against current code, and identified by staff for future maintenance.

5.4 Software Risk Assessment

The Software Risk Assessment describes institutional and code-based risks associated with the BDS source code and GUI. It outlines the development history of Fortran and the BDS, followed by an analysis of the risks associated with these developments. It concludes with a series of recommendations for ensuring sustainable maintenance of the BDS in relation to modern computing environments:

- Use a source control system
- Use an integrated development environment (IDE)
- Eliminate obsolete language components
- Modernize code practices
- Eliminate proprietary extensions

5.4.1 FINDINGS

Interim Report 1 (submitted July 16, 2015) contained a Software Risk Assessment outlining major code-based and institutional risks to the long-term viability of the BDS. Findings from the Software Risk Assessment report were:

- The BDS evolved alongside the Fortran language, with newer modules following the newer standards of the language and older sections remaining written in the version that was current at the time.
- Fortran is a programming language specifically suited to engineering and scientific computational programming. Fortran is a mature and stable language that is well understood and has significant resources available for its support.
- Maintenance of the BDS should not be considered a one-time effort but is, instead, an ongoing initiative to preserve the significant investment that has been expended to create the program.
- Institutional risks to the software result from the practices used to develop and maintain the software. The following institutional risks were identified:
    - Lack of source control of the BDS has resulted in:
        - Lack of documentation and retention of changes in the BDS source code.
        - Difficulty coordinating branches of the BDS source code with its main development.
        - Limitations on the number of developers that can modify the BDS source code at one time.
    - The software development environment limits productivity and increases the risk of errors.
    - A loss of institutional knowledge about the BDS has been occurring within MDOT as key developers have retired from the workforce.
- Code-based risks to the viability of software program result from the maintenance and update of the programming language and tools used to produce the software as time passes. The following code-based risks were identified:
    - Changes to the Fortran language standards.
    - Changes to programming practice.
    - Evolution and support of Fortran compilers.

- Evaluation of the bridge design specification calculations in the BDS identified several areas where further investigation can be used to verify that the correct variables or formulas are being used. The impacts of these areas have not been assessed to determine their significance.

5.4.2 RECOMMENDATIONS

Based on the Software Risk Assessment, the business process interviews, and the review of select engineering calculations in the BDS, the CTT project team made the following modernization and risk mitigation recommendations (see Interim Report 1):

1. Institute source control for the BDS code and documents.
2. Use an integrated development environment (IDE) for future development work.
3. Expand institutional knowledge of the BDS within MDOT so that staff retain sufficient institutional knowledge to be informed consumers of the services necessary to support the BDS (see Section 6.0 of Interim Report 1).
4. Eliminate obsolete language components:
    a. Replace H edit descriptors.
    b. Replace non-block DO constructs.
    c. Correct non-standard array elements and substrings.
    d. Replace real and double precision DO variables and DO loop control expressions.
    e. Replace alternate return from subprograms.
    f. Correct warnings with respect to assumed/variable length character functions and variables.
    g. Correct non-standard array specifiers and length selectors for character variables.
    h. Replace computed GO TO statements.
    i. Replace DOUBLE PRECISION type declarations with REAL *[kind-selector]* declarations.
    j. Correct various warnings with respect to non-standard intrinsic procedures.
5. Modernize code practices:
    a. Separate the program into source files based on program unit boundaries.
    b. Replace the deprecated language features COMMON and BLOCK DATA with MODULEs.
    c. Replace ENTRY statements with internal SUBROUTINEs or MODULE PROCEDUREs that can access shared data in the module.
    d. Change “fixed” source form to “free” source form.

    e. Use IMPLICIT NONE and declare all variables and functions (the program is using Implicit Typing).
    f. Eliminate use of mixed-mode arithmetic.
    g. Replace logical IF and GO TO flow control with the more powerful block IF structure.
    h. Use generic names for intrinsic functions.
6. Eliminate proprietary compiler extensions to enhance software portability.
7. Develop an automated test suite that is used in the development process.
8. Make BDS callable from GUI (create "Run" button).
9. Create a mixed-language program with a dynamic-link library (DLL).
10. Upgrade the BDS interface to be more visually representative of input.
11. Decouple distinct design or analysis functions where appropriate so that they can be run independently from one another.
12. Investigate and mitigate computational areas of concern.

5.5 BDS Enhancement Phases

The CTT project team divided the 12 recommendations outlined in Section 5.4.2 into seven phases for implementation based on the efficiency, scope, obsolescence, maintainability, interdependence, and usability and functionality of each recommendation. The organization of the seven-phase approach for enhancing the BDS balances the drive for new functionality with the need to reduce risk and long-term maintenance concerns.

5.5.1 PHASE 1: MODERNIZE DEVELOPMENT TOOLS

The recommendations covered during Phase 1 were:

1: Institute source control.

2: Institute an integrated development environment (IDE).

3: Provide technical assistance to MDOT staff in working with the IDE and source control.

5a: Split code into modules.

5b: Remove COMMON blocks.

5d: Change "fixed" source form to "free" source form.

6: Eliminate proprietary extensions and consider new compiler.

7: Build an automated test suite.

8: Build a "Run" button that makes the BDS callable from the GUI.

As part of this project's BDS modernization effort, the CTT project team completed all tasks required to address these recommendations. The completion of Phase 1 has allowed for the implementation of modern tools for managing the development of the BDS, hence providing both efficiency and security for later phases of the project.

By implementing a source control system prior to making any broad changes to the source code, the CTT project team minimized code loss, enabled revisions and rollbacks of source code, and provided the capability for several people to work on the BDS at the same time. Similarly, an IDE provided tools to increase software developers' productivity, making this recommendation a critical step before undertaking major revisions. Automation of a testing suite allowed for rapid, minimal-effort quality-control checks on source code to determine if changes have impacted the BDS's operation.

Splitting the BDS into individual files by program unit boundaries (modules), removing the COMMON blocks, and changing from "fixed" to "free" source form allowed for the use of modern development tools and modern language syntax. This is the first step toward allowing individual design modules of the BDS to be run separately. Elimination of proprietary extensions removed compiler-specific dependencies and permitted use of other industry-standard compilers.

The Lahey/Fujitsu Fortran compiler has a long-standing history (since 1967), and currently appears to be a viable, supported compiler. However, there are problems with the debugging capabilities in the Visual Studio environment, and interactions with Lahey's technical support services during the project were cumbersome as support responses were slow or generic. Additionally, the compiler support forums were unavailable during the project.

The CTT project team selected the Intel Visual Fortran Compiler for tests with the BDS source code because it has much better integration with Microsoft Visual Studio, it supports the 2003 and 2008 Fortran standards, and it has assumed long-term viability.

The two versions of the code (one containing the proprietary Lahey functions and the other standard Fortran commands) were tested side by side to evaluate any differences in output that may occur between the Lahey and Intel compilers. Both versions compiled and ran without issue. The output from these tests was evaluated using file comparison software, and differences in output values were further investigated by the project team. A total of 215 test cases were run and results of the output compared.

Implementing a "Run" button simplifies the BDS operation by removing the need for executing the BDS from a command line in a DOS shell.

5.5.2 PHASE 2: REDUCE GLOBAL RISKS TO BDS CODE

The recommendations covered during Phase 2 were:

4a: Replace H edit descriptors.

4e: Replace alternate return from subprograms.

4f: Correct warnings with respect to assumed/variable length character functions and variables.

4g: Correct non-standard array specifiers and length selectors for character variables.

4h: Replace computed GO TO statements.

4i: Replace DOUBLE PRECISION type declarations with REAL [*kind-selector*] declarations.

4j: Correct various warnings with respect to non-standard intrinsic procedures.

5e: Use IMPLICIT NONE and declare all variables and functions.

5h: Use generic names for intrinsic functions.

As part of this project's BDS modernization effort, the CTT project team completed all tasks required to address these recommendations. The completion of the recommendations in Phase 2 has removed the risk these items pose to the future maintainability of the BDS. The largest concern was that compiler manufacturers might remove vestigial support for these obsolete language components at some time in the future. Correcting these issues has not only simplified the process of checking for the correctness of the new code, but also allows portability of the code to a wider array of compilers.

5.5.3 PHASE 3: COMBINE GUI AND BDS CODE

The recommendation covered during Phase 3 is:

9: Create a mixed-language program with a dynamic-link library (DLL).

As part of this project's BDS modernization effort, the CTT project team completed some of the tasks items required to address this recommendation; the outstanding task items are being addressed in the subsequent project *Bridge Design System Ongoing Modernization and Support*.

The current graphical user interface (GUI) delivers data to the BDS for execution in a one-way data stream (Figure 13). The GUI feeds the BDS with a single batch of data, which necessitates completing all input parameters before running the BDS. Thus, the GUI provides no feedback as calculations are made or visual design cues to assist designers in verifying or modifying their input to the BDS.

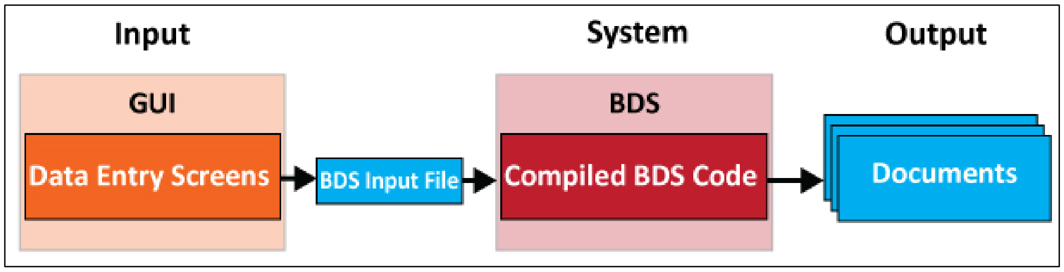


Figure 13: Current GUI/BDS Workflow

While Phase 1 recommends that the BDS code be split into separate files in order to make the code easier to handle, this does not allow each module to be executed separately as an individual bridge design element. Phase 3 begins the structural changes necessary to make the modules into callable libraries. These changes are the first step in enhancing the BDS GUI and improving how the BDS and GUI interact. This phase has been broken down into three tasks:

Object Models for Major Bridge Design Elements: Currently, there are over 2,000 variables defined in the Fortran BDS. Almost all of these variables are global in scope in that every module can access and modify their values. Since a global variable can be used or changed by any part of the program, it is difficult to determine the coupling between variables and the various design modules. This results in hidden dependencies between variables that can be broken or easily forgotten. Creating a callable library of bridge design elements aids in understanding dependencies and in limiting the scope of all variables to only those design elements that use them. This task lays the groundwork for moving the design modules into separate libraries.

Rewrite of the MAIN Subroutine: By rewriting the MAIN subroutine and implementing long-term changes from Phase 5, this task lays the groundwork for the process to call design elements of the BDS separately after the GUI has been redesigned and upgraded (see Phase 4).

Mixed-Language Program with a DLL: By combining the Visual Basic GUI and the BDS Fortran program into a single project and by converting the BDS from a console application to a DLL, this task removes the need for a DOS shell to run the BDS.

5.5.4 PHASE 4: REDESIGN GUI AND OUTPUT REPORTS

The recommendations covered during Phase 4 are:

10: Upgrade BDS interface to be more visually representative of input.

11: Decouple distinct design or analysis functions where appropriate so they can be run independently from one another.

Tasks necessary to address these recommendations will be part of the CTT project team's subsequent project *Bridge Design System Ongoing Modernization and Support*.

The GUI delivers data to the BDS for execution in a one-way data stream; likewise, output is delivered in a one-way, non-modifiable, non-interactive format. These one-way data streams greatly limit interactivity and the usefulness of the input and output functions of the BDS.

Redesigning the GUI and output reports for BDS calculations enhances their interactivity with the BDS itself. BDS users can be assisted in making design choices by displaying Intermediate BDS calculations in the GUI. A redesigned GUI also provides an opportunity for verification of or feedback on data that have been entered; for example, a schematic of the bridge cross-section showing the number and spacing of the beams (Figure 14) could be displayed in the GUI to help verify data and catch dimension and location errors.

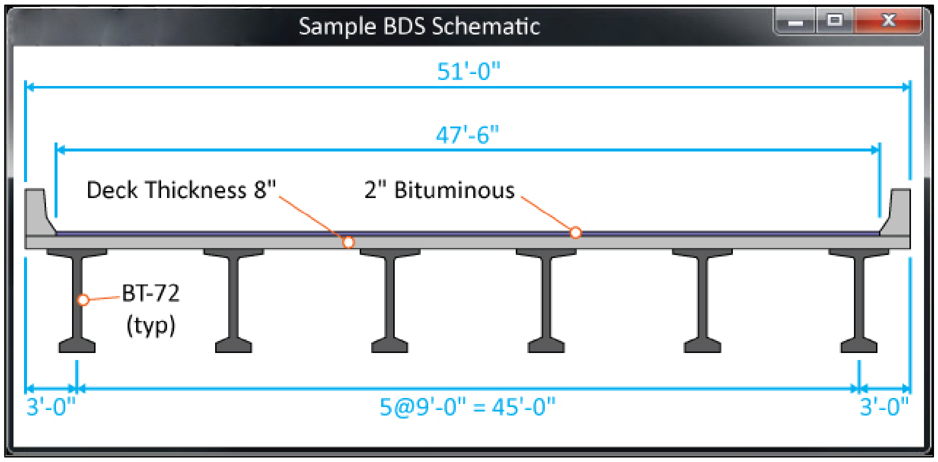


Figure 14: Sample BDS Schematic

Possible Phase 4 enhancements include:

- Creation of GUIs for each module.
- Ability to run single modules independently.
- Feedback to the GUI from the BDS displaying intermediate calculations or verifying input.
- Visualization of input and output.
- Ability to customize hard-coded specifications and assumptions.
- Modernization of input and output (currently, the GUI and BDS are executed separately, and BDS output files need extra steps or processing in order to be viewed; all these steps could be controlled from within the GUI). Tasks include:

    - Simplify input: The output file could be written by the GUI as structured data (i.e., XML or SQLite) using .NET libraries and read using the same libraries by BDS. This would be an improvement over the current GUI output file, which requires over 4,000 lines of code in the BDS to parse.
    - Design reports: Manage the output files from the BDS (as appropriate) from within the GUI. (Conversion of printer files to a portable format like PDF via a .NET report management component is straightforward. Similarly, tabular data could be exported to spreadsheet format.) Create view, print, export, and save functions for the output files.
    - Format for CAD viewers: Format BDS output for a CAD program (like Bentley View or DWG TrueView). Launch the viewer or its plugin from the GUI.
- Incorporate structural analysis documentation (SAD) and hand calculations.

This phase could be split into two sub-phases: developing requirements and design/mockup of the GUI and developing an estimate for the implementation of the design.

5.5.5 PHASE 5: MODERNIZE CODE PRACTICES

The recommendations covered during Phase 5 are:

4b: Replace non-block DO constructs

4c: Correct non-standard array elements and substrings

4d: Replace real and double precision DO variables and DO loop control expressions

5c: Replace ENTRY statements with internal SUBROUTINEs or MODULE PROCEDUREs

5f: Eliminate use of mixed-mode arithmetic

5g: Replace logical IF and GO TO flow control with the more powerful block IF structure.

As part of this project's BDS modernization effort, the CTT project team completed some of the tasks items required to address these recommendations; the outstanding task items are being addressed in the subsequent project *Bridge Design System Ongoing Modernization and Support*.

Handling issues during a rewrite is the most efficient method to deal with these recommendations due to their complexity. The Phase 5 recommendations can be covered on a per-module basis through three discrete tasks: eliminate obsolete language components, eliminate mixed-mode arithmetic, and replace logical IF and GO TO flow control.

Obsolete Language Components: The BDS uses some programming techniques that were necessary prior to Fortran 77, which introduced block IF and END IF statements, with optional ELSE and ELSE IF clauses, and the SELECT-CASE control structure. The absence of these structured programming features leads to dependence on the GO TO statement, the primary criticism of which is that programs using GO TO statements are harder to understand than alternative constructions. Because one can implement arbitrary control flow with the GO TO statement, it is impossible to know exactly what has preceded the execution of a specific part of the program. This complexity means that understanding a piece of the program using GO TO statements often requires understanding all of it, so the effort needed to modify or enhance the functionality of such programs can be out of proportion to the changes being sought. The GO TO statement remains in use in certain situations, but alternatives are generally used when available.

Mixed-Mode Arithmetic: The common definition of mixed-mode arithmetic is when an expression contains both reals and integers. Using such expressions is error-prone. A combination of operator precedence rules and integer valued intermediate results can introduce truncated, whole-number values where fractions are expected.

Modernizing the coding practices also simplified the structure of the BDS which, in turn, makes future changes or updates to the code easier.

5.5.6 PHASE 6: UPDATE ENGINEERING CALCULATIONS

The recommendation covered during Phase 6 is:

12: Investigate and mitigate computational areas of concern.

Tasks necessary to address this recommendation will be part of the CTT project team's subsequent project *Bridge Design System Ongoing Modernization and Support*.

During the forensic investigation of the existing BDS code, the CTT project team evaluated calculations containing specification references. They identified locations where further investigation into the suitability of the calculation should be verified or updated to meet the current design specifications. An example of a calculation to investigate further is shown below.

cont144	BDS does not calculate stud fatigue shear resistance or nominal stud shear resistance according to AASHTO LRFD 7th edition. Constant values are assigned which may approximate AASHTO values given limitations to stud properties within the BDS; this was not checked.

In this case, stud resistance values were not calculated but rather assigned a constant value. This may or may not be appropriate given design constraints based on assumptions made elsewhere in the BDS or MDOT policy regarding use of shear studs. Assessing whether it is appropriate to assign a constant value to stud fatigue shear resistance may involve updating the GUI to allow a design engineer to enter a value for the shear resistance instead of having a value hard-coded in the BDS.

5.5.7 PHASE 7: DEVELOP NEW ENHANCEMENTS

A program that is the size and complexity of the BDS should be in a constant state of development in order for it to remain modern and keep pace with users' changing needs. Phase 7 presents a placeholder to represent the ongoing maintenance and updates of the BDS to include new functionality. Tasks necessary to address these ongoing maintenance and update needs will be part of the CTT project team's subsequent project *Bridge Design System Ongoing Modernization and Support*.

5.5.8 IMPACTS OF PHASING AND SCHEDULE

Figure 15 depicts the phasing and schedule of the BDS enhancement phases.

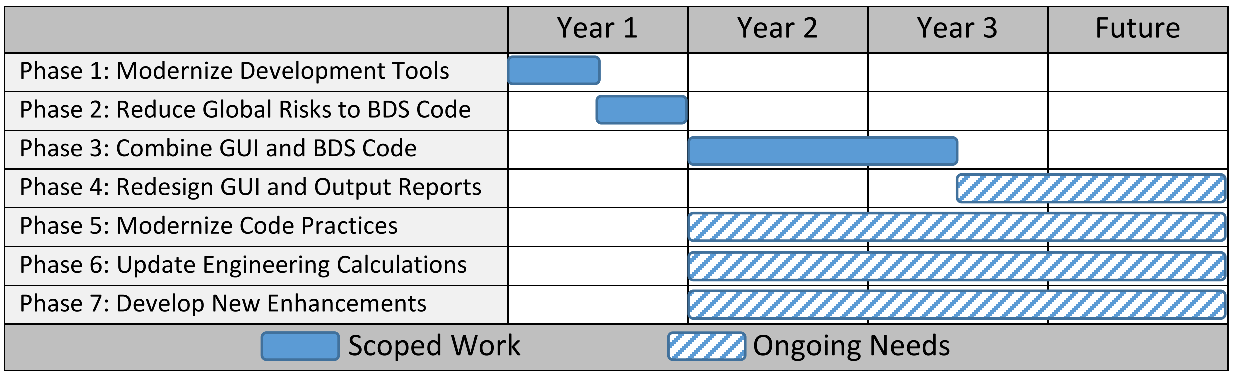


Figure 15: Phasing and Schedule of BDS Enhancement Phases

6.0 CONCLUSIONS

6.1 Existing Fortran Code

The BDS is an impressive structural engineering tool representing decades of hard work by MDOT employees. It provides a significant institutional value to MDOT by being tailored to the specific business process and needs of MDOT's Bridge Design Unit. It also has a great commercial value if properly modernized and supported.

The size and complexity of the BDS has grown beyond the ability of a single person to fully understand without years of full-time experience with it. The process of completing the forensic investigation of the BDS was very time-consuming for the CTT project team, which was comprised of structural engineers and software engineers who were using modern tools and, as a team, were well-versed in structural engineering and Fortran programming. It is apparent that this initial work to understand the inner workings of the BDS has a significant value, is difficult to achieve, and should be maintained as an ongoing initiative.

The processes that were set up for creating and maintaining the *BDS Reference Guide* created a robust framework for continued modernization, documentation and support work for years into the future. These new processes are easily maintainable over time and should be made part of the standard operating procedure for working with the BDS.

Non-functional but broad-in-scope changes to the BDS source code were necessary to allow the program to work with modern software tools, including modern compilers, IDEs, and Doxygen. These changes will need to be maintained in order to generate the BDS documentation. The modified working copy of the BDS that the CTT project team produced will replace the production version of the BDS that MDOT currently maintains. This is the most cost-effective method for future maintenance, but requires thorough testing to verify that the changes did not inadvertently affect the operation or results of the BDS. MDOT should run the “original” Lahey-compiled version of the BDS in parallel with the new Intel-compiled version until they are satisfied with the results.

6.2 Cross-Referenced Design Equations

This study identified 174 entry points in the BDS that contained one or more equations referenced to the LRFD specifications. The budget for this project allowed completion of worksheets that summarized AASHTO or MDOT equations, listed variables associated with these equations along with existing definitions, contained a copy of the Fortran code comprising the entry point, and had a commentary from the reviewing engineers. These worksheets addressed 113 of these references—the 69 worksheets in Tier 1 (highest priority) and an additional 44 worksheets from Tiers 2 and 3. Calculations in the BDS were compared to the equations and conditionals presented in the appropriate specifications. The study also checked basic conditionals, such as those that limit equations to specific ranges or that determine appropriate equations based on a relative relationship between other variables. Variables calculated outside of the referenced equation’s entry point were assumed accurate and were not verified. Equations were assessed based on information provided within the equation’s entry point; interactions with other sections of the BDS were not assessed. Ongoing support of the BDS will require revisiting these worksheets. Additional worksheets will need to be generated for references not addressed within this project and for sections of the code that are updated to the most current AASHTO specifications.

6.3 BDS Manuals

This project produced two major sources of documentation: the HTML-based *BDS Reference Guide* and a printable *BDS User's Manual*. The *BDS Reference Guide* should be considered a living document that should be updated any time a change is made in the program or a new investigation into the functioning of a component of the BDS is completed. Tying the structure and format of the *BDS Reference Guide* to the structure of the BDS provides an impetus to keep the documentation current when code changes occur.

6.4 User Business Processes

The interviews with MDOT staff and managers who interact with the BDS provide a positive picture of the BDS as an innovative design tool that provides engineers with a high degree of confidence. MDOT's reviews have determined that none of the commercially available programs could replace the BDS because the BDS specifically fits MDOT's business process for design and designs the entire structure rather than just one component.

Interviews also identified areas where staff and managers felt the BDS could be modified for better meeting their design needs for complex, large, or detailed-scope designs. Interviewees also had a number of requests for new time-saving features.

The interviews also revealed a strong concern regarding the level of staff support for the BDS not being adequate for long-term sustainability of the program. This concern also emerges in the risk assessment.

The CTT project team will continue to provide necessary training to MDOT staff in the use of the BDS and its related components as part of the CTT project team's subsequent project *Bridge Design System Ongoing Modernization and Support*.

6.5 Software Risk Assessment

The Software Risk Assessment identified several areas that will become critical concerns if not addressed in the near future. Primarily these issues are related to changing practices in coding and the Fortran language. The risk assessment also identified areas that have the potential to develop into risk items based on institutional practice.

Through the forensic investigation of the BDS and interviews with MDOT staff, this study found that the GUI interactions with the BDS could be improved to expand the program's power and usability. While this does not necessarily constitute a risk, failing to take advantage of these interactions represents a lost opportunity.

Currently, the GUI and BDS are executed separately, and BDS output files need extra steps or processing in order to be viewed. The GUI writes a file to be read by the BDS as input; but, it takes over 4,000 lines of code in the BDS to read and parse that input file. However, all of these steps could be controlled from within the GUI. The GUI could write the BDS input as structured data (i.e., XML or SQLite) using .NET libraries, and the BDS could read the files using the same libraries. By interactively executing the BDS Fortran program from within the GUI, the BDS could eventually have a library of BDS components callable by .NET programs. The BDS could output files within the GUI; printer files could be converted to PDF via a .NET report management component in a straightforward fashion; tabular data could be exported to spreadsheet format; and functions could also include viewing, printing, exporting, and saving output files. Finally, increasing the interactivity of the BDS and its GUI can allow outputs destined for a CAD program to be formatted for display in a viewer like Bentley View or DWG TrueView, and this format can be made accessible by launching the viewer or its plugin from the GUI.

6.6 Value of *Bridge Design System Analysis and Modernization* Project

The BDS is a pivotal asset of MDOT's Bridge Design Unit. The CTT's forensic evaluation, modernization, and optimization efforts that comprised the *Bridge Design System Analysis and Modernization* project protect the significant intellectual investment that has been made in the creation of the BDS. The documentation that resulted from these efforts—the *BDS Reference Guide* and the *BDS User's Manual*—captures the historical development of the BDS and accommodates the information needs of software engineers and structural engineers. The CTT established source control for the BDS through Assembla, a set of task and code management tools for software engineers to manage work requests and maintain source code. This created a structure for future support and development of the BDS. The CTT project team recommended seven enhancement phases for ensuring modernization and optimization of the BDS and undertook initial BDS enhancement tasks. MDOT, in collaboration with the CTT, will address the outstanding enhancement phases in the subsequent project *Bridge Design System Ongoing Modernization and Support*.

APPENDIX A: IMPLEMENTATION AND LEGACY SUPPORT PLAN FOR THE BRIDGE DESIGN SYSTEM

The Michigan Department of Transportation (MDOT) Bridge Unit staff has decades of institutional experience working with the Bridge Design System (BDS), which has led to a high degree of confidence and understanding regarding what the BDS does and appropriate applications for its use. This technical familiarity and understanding allows bridge designers to anticipate limitations of the design methodology and to expect processes that require additional analyses for challenging or unique designs. Most importantly, the BDS contains significant MDOT-specific design efficiencies that may not be present with off-the-shelf systems.

It is crucially important to document the years of institutional knowledge that went into the development and design of the BDS, thus ensuring the viability of the BDS for years into the future. The MDOT project OR14-29 *Bridge Design System Analysis and Modernization* provides a platform for updates and improvements to ensure that the BDS is a viable software resource. Updated documentation provides tools that allow for better understanding of the BDS design process.

This project further identified major code-based and institutional risks to the long-term viability of the BDS and provided recommendations to reduce these risks. The risk-reduction strategy, which was organized into seven phases, balanced the need for efficiency in software coding activity with impacts on the Bridge Design Unit's business processes. The first two risk-reduction phases as well as portions of the third and fifth phases were addressed in a supplemental work plan as part of the original project.

The efforts described in the Work Plan will address the remaining phases in the risk reduction strategy for the BDS:

Phase 3 – Combine GUI and BDS Code

Phase 4 – Redesign GUI and Output Reports

Phase 5 – Modernize Code Practices

Phase 6 – Update Engineering Calculations

Phase 7 – Develop New Enhancements

These phases of strategic risk reduction will be addressed by the subsequent project *Bridge Design System Ongoing Modernization and Support* through six action items based on an understanding of the scope of work. Key MDOT staff and the Center for Technology & Training (CTT) project team can anticipate a high level of involvement on this project. A successful end product will depend upon MDOT's significant guidance concerning expectations for a redesigned graphical user interface (GUI), output reports, AASHTO Specification mitigation prioritization, and new enhancements. The CTT project team, in collaboration with key MDOT staff, will ensure that the needs and desires of the BDS users are met to the extent imposed by budget and programing limitations.

Successful implementation of the modernization of MDOT's Bridge Design System will require efficient and effective communication with the entire bridge design and support team. The CTT has established a source repository and work request ticketing system (Assembla) which allows the entire project team, both MDOT and MTU to collaboratively advance the modernization efforts. MDOT will establish priorities for the implementation of updates to the BDS which will be reevaluated as part of regular monthly status update meetings. Priorities will be evaluated based on need, scope, and budget. MDOT has expressed their first priority as the need to bring the BDS up to the current AASHTO LRFD specifications. The collaborative process for modernization of each identified priority is outlined below:

- MDOT evaluates priorities during monthly project team meetings according to need, scope, and budget
- Assembla tickets are created for new priorities (available for MDOT and CTT staff to review progress)
- CTT will work toward resolving tickets
- CTT will test BDS using the automated test suite and submit results to MDOT
- MDOT will review test suite results and determine release viability
- MDOT marks ticket as resolved and issues updated BDS

Action Items

While Phases 3, 4, 6, and 7 each have direct correlations to the six action items detailed below, Phase 5 is not addressed as a specific action item. Modernizing code practices is an ongoing task that will be completed concurrently with any major development or modification of the BDS code. The work required to address these changes will only become apparent as the requirements for a module rewrite are specified. Time and cost estimates related to modernization will be included as part of a specific rewrite or modification request. As such, Phase 5 is not included as a specific action item but rather interspersed within each other action item.

ACTION ITEM 1: COMBINE GUI AND BDS CODE

The current GUI delivers data to the BDS for execution in a one-way data stream, limiting the interactivity and usefulness of the BDS input functions. The GUI feeds the BDS with a single batch of data, which necessitates completing all input parameters before running the BDS. In turn, the BDS provides no feedback to the GUI as it makes calculations, nor does it offer visual design cues that can assist bridge designers in verifying or modifying their input to the BDS.

MDOT recognizes the value of a more interactive BDS and GUI structure. To achieve a more enhanced GUI and better interactivity between the BDS and GUI, MDOT's support is vital for three sub-items:

Item 1A: Create an "Object model" for Major Bridge Design Elements

> Almost all of these variables defined in the Fortran BDS are global in scope, which means that any part of the program can access and modify these values. This results in both coupling between variables and various design modules as well as hidden dependencies between variables that can be broken or easily forgotten. For this action item, the CTT project team, on behalf of MDOT, will create a callable library of bridge design elements, outlining dependencies and limiting the scope of the variables to only those design elements that use them. This action item will also create an object model for each of the major bridge design elements in order to define each design element in terms of its attributes (variables) and methods. Completion of this action item lays the groundwork for moving the design modules into separate libraries.

Item 1B: Rewrite MAIN Subroutine

> The CTT project team, on behalf of MDOT, will rewrite the MAIN subroutine. This rewrite lays the groundwork and structure for the BDS to call design elements of the BDS separately when the GUI has been redesigned and upgraded in Action Item 2.

Item 1C: Create a Mixed-language Program with a Dynamic-linked Library (DLL)

> The CTT project team, on behalf of MDOT, will combine the Visual Basic GUI and the BDS Fortran program into a single mixed-language project and will convert the BDS from a console application to a DLL. Completion of this action item removes the need for a DOS shell to run the BDS.

ACTION ITEM 2: REDESIGN GUI AND OUTPUT REPORTS

The BDS currently delivers output in a one way, non-modifiable, non-interactive format, also limiting the interactivity and usefulness of the BDS output functions.

MDOT will greatly benefit from improved BDS output functions. Therefore, it is important that MDOT support the design, testing, and creation of a new GUI and output reports for BDS calculations as well as the modification of the BDS code that are specific to the GUI interactivity or output functions.

MDOT staff who use the BDS will need to express their specific needs and desires for an updated GUI and output reports before detailed sub-items can be fully defined. Action Item 2 efforts could, for example, range from having intermediate calculations in the BDS displayed by the GUI in order to assist users in making design choices, to enabling verification or feedback in the redesigned GUI in order to help users catch dimension and location errors as they enter data. Other enhancements that might appeal to MDOT users include:

- Creation of GUIs for each module
- Ability to run single modules independently
- Feedback to the GUI from the BDS that displays intermediate calculations or verifies input
- Visualization of input and output
- Ability to customize hard-coded specifications and assumptions
- Modernization of input and output (for example, controlling input and output functions solely from within the GUI):
    - Simplify Input: The output file could be written by the GUI as structured data, i.e., XML or SQLite, using .NET libraries and read using the same libraries by BDS.
    - Design reports: Manage the output files from the BDS (as appropriate) from within the GUI. Conversion of printer files to a portable format like PDF via a .NET report management component is straightforward. Similarly, tabular data could be exported to spreadsheet format. Create view, print, export, and save functions for the output files.
    - CAD viewers: Format BDS output for a CAD program (like Bentley View or DWG Trueview) to display instead in a viewer. Launch the viewer or its plugin from the GUI.
- Incorporate Structural Analysis Documentation (SAD) and hand calculations.

Estimating the time or budget for this action item is difficult due to the wide range of possible changes to the GUI and, consequently, a highly variable scope. By splitting this action item into three sub-items, an accurate estimation of time and budge becomes possible for the first two sub-items; the third sub-item, however, is dependent on the outcome of the first two sub-items. These sub-items are:

Item 2A: Gather requirements

The appropriate MDOT staff, as determined by the MDOT project manager, will need to meet twice with the CTT project team in order to identify user requirements for a redesigned GUI and output reports. The MDOT project manager will receive a report, compiled by the CTT project team, that documents the identified requirements of a new GUI and new output functions.

Item 2B: Develop design and scoping documents

The MDOT project manager will be able to prioritize and select from the requirements of a new GUI and output reports identified that were identified in Action Item 2A. The CTT project team will prepare design documents and budgets to implement the selected requirements. The MDOT project manager will receive these design documents and proposed budgets.

Item 2C: Implement selected development

MDOT will be able to prioritize and direct development efforts based on design and cost documents developed in Item 2B to the extent supported by the budget. If development exceeds the budgeted amount for this action item, the timeline for delivery will be extended to future program years.

ACTION ITEM 3: UPDATE ENGINEERING CALCULATIONS

As part of OR14-29 *Bridge Design System Analysis and Modernization*, the CTT project team identified specific lines of BDS code that referenced AASHTO specifications. The CTT project team evaluated these BDS equations against the *AASHTO LRFD Bridge Design Specifications* 7th edition and noted any assumptions made or deviations from the AASHTO specifications as areas of concern for further evaluation.

Action Item 3 likely be a long-term, ongoing operation. In this action item, the MDOT project manager will receive the results of a detailed evaluation of BDS variables and associated recommendations. To compile this report, the CTT project team will investigate the previously-identified areas of concern in greater detail. By tracing variables through the BDS using the modern development tools incorporated in the original project, the CTT project team will determine how variables are set within the BDS and, therefore, their appropriateness when used in the referenced specification equations. Differences that are due to design limitations imposed by other sections of the BDS will be noted because any changes made to those variables will affect both sections.

This action item will require significant MDOT staff time to mitigate those problems and offer comments on the suggested mitigation actions for each calculation of concern. The programming and engineering work for these mitigations cannot accurately be estimated at this time due to the uncertainty of the corrective action necessary for each item. MDOT will prioritize the calculations to be verified and work with CTT project team to document the results to the extent supported by the project budget. The MDOT project manager will receive an enumeration of BDS code changes identified by this action item that should be considered for development as part of Item 4C.

ACTION ITEM 4: MAINTENANCE AND NEW ENHANCEMENTS

A program the size and complexity of the BDS should be in a constant state of development in order for it to remain modern and keep pace with users' changing needs. Consequently, technical support and routine maintenance are a critical component of BDS's success and continued acceptance by its MDOT users. The insights that MDOT can provide for prioritizing technical support and routine maintenance are invaluable.

Action Item 4 will provide ongoing maintenance and update of the BDS to include new functionality, such as additional beam types, or to update specific calculations in order to keep pace with changing AASHTO standards. Action Item 4 includes future, not-yet-defined development that would impact the overall process of maintaining and modernizing the BDS; at this time, Action Item 4 has no defined scope, budget, or time frame. Nonetheless, this action item can be divided into three sub-items:

Item 4A: User Technical Support

The MDOT project manager will receive a list of action items that reflect user requests for small maintenance items and functional enhancements of the BDS. To compile this action item list, the CTT project team will respond to user requests for small maintenance items and functional enhancements as time and budget permits . A CTT software developer will prioritize user requests and develop responses to those deemed reasonable, cost effective, and important for users' success. The MDOT project manager will be able to consider the requested developments.

Item 4B: AASHTO standards

MDOT, in conjunction with the CTT project time, will be able to evaluate the significance of updates to the *AASHTO LRFD Bridge Design Specifications* and any impacts they may have on the BDS. This action item will further identify, document, and scope any BDS code changes necessary to comply with these updates. MDOT will be able to prioritize these changes and work with CTT project team, who will make changes as necessary to the extent supported by the project budget. A procedure will be developed to document specification-update-related changes to the BDS so that it is clear these changes have been made.

Item 4C: New Enhancements

This action item addresses future development of new enhancements as the needs arise. These future needs may be identified in one of four ways: from the mitigation recommendations following further investigation of the engineering calculations (Action Item 3), through technical support (Item 4A), as the result of changes arising from new AASHTO standards (Item 4B), or as directed by the MDOT project manager. MDOT will be able to prioritize these changes and work with CTT project team in order to implement the changes as necessary to the extent supported by the project budget.

This sub-item also launches a pilot program that can assist in developing an efficient method to conduct new enhancement code changes. The goal of the pilot will be to establish a best practice for interactions between MDOT staff and the CTT project team for successfully adding new enhancements to the BDS. It is anticipated that this pilot program will add to the BDS the capability of designing with Indiana Bulb Tees.

ACTION ITEM 5: BDS TRAINING MATERIAL AND DELIVERY OF TRAINING

This action item will offer two single-day training workshops for approximately 10 to 25 MDOT Bridge Unit staff already familiar with the BDS as well as other participants at the discretion of the MDOT BDS Program Manager. The workshop will provide an overview of the *Bridge Design System Analysis and Modernization* project, updates made to the BDS software, and the associated documentation materials. Per the determined budget, the CTT project team will send three members to Lansing to present the material. It is assumed that MDOT staff will be able to secure the necessary conference space and invite attendance of their selected participants. The CTT project team will prepare the necessary slides, handouts, and examples for the training.

ACTION ITEM 6: PERIODIC PROJECT MEETINGS, REPORTING AND PROJECT MANAGEMENT

Item 6A: Kickoff meeting

A kickoff meeting is necessary to establish priorities and ensure the needs of the MDOT Design Staff are met in the ongoing modernization efforts.

Item 6B: Monthly progress meetings

This project, to be successful, requires a high level of interaction between the MDOT staff and the CTT project team. To facilitate this interaction, monthly meetings, conducted by the CTT project team, will review progress on current tasks and will set priorities for future tasks. The monthly meetings will include the appropriate MDOT staff as determined by the MDOT project manager as well as members of the CTT project team. It is assumed that the monthly meetings will be less than two hours in length and will be conducted by phone.

Conclusion

The newly commencing project *Bridge Design System Ongoing Modernization and Support* encompasses the action items of this implementation plan. As such, the close collaboration between MDOT and the CTT project team outlined in this plan will be a necessary component of the new project, *Bridge Design System Ongoing Modernization and Support*. Through their continued collaboration, both MDOT and the CTT project team will be able to preserve the legacy and ensure the modernization of the BDS.